





IBM Operational Decision Manager on Cloud

Welcome to the IBM Knowledge Center for Operational Decision Manager on Cloud. Here you can find information about creating and deploying business rules for your applications through the cloud version of Operational Decision Manager. Updated: June 2019







Get started

- Overview
- First steps
- Tutorials
-  Videos
- Product legal notices
- Additional notices

Find support

-  Online Education
- Troubleshooting and support
-  Detailed system requirements
-  FAQ

Stay current

- What's new
-  Known limitations
-  Technotes
-  Recommended reading
-  Facebook
-  Twitter
-  Blog

Overview

IBM® Operational Decision Manager on Cloud is a web-based platform for making business rule solutions that automate the application of complex decisions.

[What's new](#)

Ongoing development ensures that Operational Decision Manager on Cloud continues to meet your needs for creating and implementing business rule applications.

[Introduction](#)

Operational Decision Manager on Cloud is a comprehensive decision automation platform for capturing, automating, and governing rule-based business decisions.

[Cloud workflow](#)

This example introduces you to the collaborative process for creating and deploying a rule application in Operational Decision Manager on Cloud.

[Operational Decision Manager](#)

Operational Decision Manager delivers a proven development system for capturing policies in solutions that automate the application of decisions.

[Key components](#)

Operational Decision Manager on Cloud includes the main components in Operational Decision Manager.

[Cloud environments and user roles](#)

The components for creating and using rule applications are kept in cloud environments. Access to the environments is controlled through user roles.

[Express](#)

IBM Operational Decision Manager on Cloud Express covers the entire lifecycle of a decision service in one environment.

[Hybrid cloud environments](#)

Complement your on-premises or cloud development system with Operational Decision Manager on Cloud.

[First steps](#)

Before you begin, you must gain access to the cloud portal. If you want to create decision services, you must install Rule Designer. When you finish setting up, you can go through the tutorials to learn how to use Operational Decision Manager on Cloud.

[Usage reports](#)

You can view the use of your decisions and artifacts through a web-based dashboard. It displays performance data in graphs that enable you to readily understand your rule application traffic.

Related information:

[Operational Decision Manager](#)

What's new

Ongoing development ensures that Operational Decision Manager on Cloud continues to meet your needs for creating and implementing business rule applications.

The latest release introduces a dashboard for monitoring the use of artifacts and decisions. It replaces the monthly usage reports.

Users of Express can also add an additional environment for integration testing or production, and Mac users can use Rule Designer on macOS Mojave.

To see all the new features, consult the sections below. You can skip ahead to a section by clicking its shortcut link:

Sections:

- [General](#)
- [Rule Designer](#)
- [Decision Center](#)

General

Usage dashboard

Operational Decision Manager on Cloud now includes a dashboard for monitoring artifacts in Decision Center and decisions in Rule Execution Servers. The dashboard is accessible to cloud administrators, and monitors usage over a selected period of time. The dashboard replaces the monthly usage reports. [!\[\]\(faf942dc3e59ce8eb64b4ac481eca7e0_img.jpg\) Learn more...](#)

Adding an environment to Express

Express comes with a single environment for developing and deploying decision services. Now, you can add an additional environment for integration testing or production. Taking Express all the way to three environment requires switching to the full version of Operational Decision Manager on Cloud. [!\[\]\(d3102649f02e825ddb76dc3de0190154_img.jpg\) Learn more...](#)

Checking your user role

Now, you can check your user role in your portal profile. Your role determines your access to the various features, environments, and components in the portal. Check the role to see what is available to you. If you have more than one account with different roles, you can check the role in the profile to determine which account you are working in. [!\[\]\(95b425611cbd2b8716a140cf67c81822_img.jpg\) Learn more...](#)

Rule Designer

MacOS Mojave support

You can now use Rule Designer on macOS Mojave (version 10.14). [!\[\]\(3342c215b2a8b663596a81468d5dc314_img.jpg\)](#)

Testing rulesets in Rule Designer

Now you can verify that rulesets behave as expected in Rule Designer by defining scenarios and running a testing configuration. Rule Designer displays the results in its Console view. [!\[\]\(1f56542a42e2413e44a2b2023033aa2e_img.jpg\) Learn more...](#)

Upgrading Rule Designer

Each release comes with its own version of Rule Designer. When you upgrade to the latest release, you must reinstall the rule editor. To see which release you are working with, look for its number in the bottom right corner of your instance of Operational Decision Manager on Cloud. [!\[\]\(5a351309c3b87e4420622c1f0e57efc0_img.jpg\) Learn more...](#)

Decision Center

Configuration settings available in the Business console

You can set some configuration options to customize Decision Center behavior or display in the Business console. You must have the permission manager role to edit the configuration settings. [!\[\]\(9f3852d68d41e1e95bc4ec10e81aba4b_img.jpg\) Learn more...](#)


Exporting and importing test suites and simulation artifacts


Export test suites and simulation artifacts with a project from the Business console. The export function generates a compressed file that you can import back into the console later or use with a different instance of Decision Center. A few limitations apply, but you get a working, portable file that's ready for validation. [!\[\]\(206536f97fdb267876a3a10ea42b0254_img.jpg\) Learn more...](#)

Grid columns for displaying properties

Select and order the columns used to display the properties of rules, decision tables, ruleflows, and variable sets in the Decision Artifacts and Queries tabs of decision service branches in the Business console. [!\[\]\(241407ae374027aec4b030ca93d07b05_img.jpg\) Learn more...](#)

Generating reports on your projects and decision services

You can now generate reports on decision services, projects, or queries. Displayed in the HTML format, the reports show the content and properties of deployed project elements. You can use the reports in auditing your repository or doing other resource management operations.  [Learn more...](#)

You can also use the new Decision Center REST API endpoints to see the contents of your repository.  [Learn more in Explore section...](#)


Creating and editing advanced rule artifacts in the Business console

Now you can create and edit advanced rule artifacts in the Business console. Developers can use technical rules to tap constructs and features such as loops and explicit ILOG Rule Language (IRL) mapping, and functions to share procedure code between elements in a ruleset. (**Note:** The Business Action Language (BAL) is not supported.)


 [Learn more about technical rules...](#)

 [Learn more about functions...](#)

Locking decision tables in the Business console

In Rule Designer, you can apply different types of locks to a decision table to prevent edition by others. Now, you can apply those locks to decision tables in the Business console when you publish your decision service.  [Learn more...](#)

Sharing projects in different decision services or branches

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in other decision services, or in other branches of the same decision service. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that you need to maintain only one BOM and one vocabulary, and changes made to the BOM and vocabulary are automatically propagated to all the projects that use them.  [Learn more...](#)

[Change history](#)

See what's new in past releases of Operational Decision Manager on Cloud.

[Deprecated features](#)

These features are deprecated or removed from Operational Decision Manager on Cloud.

Parent topic: [Overview](#)

Change history

See what's new in past releases of Operational Decision Manager on Cloud.

Releases:

- [2018.12](#)
- [2018.10](#)
- [2018.06](#)
- [2017.10](#)
- [2017.06](#)
- [2017.03](#)
- [2016.12](#)

2018.12

Features introduced in December 2018.

General

Subscription selection: Now you can switch between your instances of Operational Decision Manager on Cloud without logging out. The new feature enables you to move quickly among subscriptions. [!\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\) Learn more...](#)

Decision Center

Decision modeling in the Business console: The decision model service makes business experts autonomous in developing rule applications. They can create, maintain, and deploy decision services, and edit a model directly by using a relatively nontechnical language. A diagram-based development system clearly shows the flow of data and decisions through a decision service. [!\[\]\(5361750c22c4e047a52f4eac1ec2d4cc_img.jpg\) Learn more...](#)

Decision governance through REST API: The Decision Center REST API now includes decision service management through the decision governance framework. [!\[\]\(870f5d5e9c0d57485634be3ecf52f3ca_img.jpg\) Learn more in the Govern section...](#)

Running diagnostics in the Business console: Administrators can now run diagnostics on their systems from the Business console. They can get information on versioning, data sources, extensions, verbalizers, and databases, and use it to troubleshoot issues. [!\[\]\(4fe57c3593bf1b21d272ae7ac8dfaf77_img.jpg\) Learn more...](#)

Webhook notification from Decision Center: You can use webhooks to send notifications from Decision Center to an external application of your choice. Get real-time notifications for events that are related to decision governance, RuleApp deployment, and changes to business rules. [!\[\]\(0d5ec72f61334709c3fc9450209b754f_img.jpg\) Learn more...](#)

Customer survey in the Business console: An in-application survey now solicits feedback from users of the Business console. A pop-up window opens on your screen, giving you an opportunity to rate the service by selecting a score. (Note: The survey opens in English when it does not support the user's locale.) The system is set to check in with you every 90 days. Your time is valuable, and you are under no obligation to participate. Even if you close the pop-up without responding, taking a couple of minutes to provide input directly to the Operational Decision Manager on Cloud team is appreciated. Your feedback is taken seriously. By taking advantage of this feature, you can contribute to improving Operational Decision Manager on Cloud. [!\[\]\(b792654f2cef9719eabeb6c5be00811e_img.jpg\) Learn more...](#)

Rule Designer

Move to Eclipse 4.7.3: Rule Designer now runs on Eclipse 4.7.3. The move brings changes to the component for creating and deploying rule applications. [!\[\]\(2bae76de5ebbd5c4d7d47162f1673734_img.jpg\) Learn more...](#)

Rule Execution Server

REST API methods for Rule Execution Server diagnostics: You can use new methods in the REST API to get Rule Execution Server diagnostics. These methods are available in the **/utilities** tab in the REST API test tool in the Rule Execution Server console. [!\[\]\(84f47badaad7772cd95667a7c387a639_img.jpg\) Learn more...](#)

Ruleset selection for RuleApp archives: When you package a RuleApp archive for downloading from the Rule Execution Server console, you can now choose to include only selected rulesets. Previously, all rulesets in the RuleApp were included in the package downloaded. [!\[\]\(28f72b996fc97883dfd9d4e8b1b16b4e_img.jpg\) Learn more...](#)

Tutorials

Getting started with decision modeling: This tutorial shows the basics of creating a decision model service. You model and author a decision service from the Decision Center Business console. [!\[\]\(aff7c69c44a5e015f18c35867ef3f5c3_img.jpg\) Learn more...](#)

2018.10

Features introduced in October 2018.

General

Operational Decision Manager on Cloud Express®: The new Operational Decision Manager on Cloud

Express covers the entire lifecycle of a decision service in one environment. It includes all the components for creating a decision service, developing it collaboratively, and running it for a client application. Unlike the complete Operational Decision Manager on Cloud, Express does not include separate environments that are dedicated to integration testing and production applications. [!\[\]\(125d701e9425b54c764340b5671b38cd_img.jpg\) Learn more...](#)

Updated password policy: The constraints for composing passwords have been updated. Also, the duration for passwords is now 90 days. [!\[\]\(21199eb166cc97331a0c54c649195dcc_img.jpg\) Learn more...](#)

2018.06

Features introduced in June 2018.

General

Customer-defined cloud portals: Now you can configure the environments in your cloud portal to better match your development process. New customers determine the makeup of their cloud environments during the initial provisioning. Existing customer can buy new environments to change their current setup. The provisioning of additional environments is done by IBM®. [!\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\) Learn more...](#)

Usage reports for decisions and managed artifacts: Reports are now sent on the usage of decisions and managed artifacts. You can use the reports to track your services and manage your subscription compliance. The reports are sent monthly, and cover daily usage. You can request to receive your reports more frequently. For information on metrics: [!\[\]\(c694a3ff3b077d76910920a6a1593ab4_img.jpg\) Learn more...](#)

New password rules and duration: The constraints for composing passwords have been updated to remain in compliance with rules set by the US Federal Risk and Authorization Management Program (FedRAMP). The rules also include a shorter duration for cloud portal passwords, which now expire in 60 days instead of 90 days. When a password created under the old 90-day expiration period is changed, the new password becomes subject to the new expiration period. [!\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\) Learn more...](#)

Decision Center

Using Decision Center with multiple subscriptions: Now you can attach the Decision Center of one Operational Decision Manager on Cloud subscription to the execution runtime environments of another subscription of Operational Decision Manager on Cloud. This can increase the redundancy of execution environments in cases where higher availability is required, and a choice has been made to have an instance deployed in the same or multiple data centers.

You can use a similar setup in a hybrid environment where the runtime environment is running on premises or another cloud. The Rule Execution Server console in the cloud environment must be securely exposed on the public Internet or through an outbound VPN. Ask your IBM representative for more information. For information on hybrid environments: [!\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\) Learn more...](#)

Offline updating of decision tables from the Business console: Now you can work on a decision table offline, and then import it into the Business console. You start by exporting the decision table from the Business console as an Excel spreadsheet. After making changes, you import the spreadsheet back into the Business console to update the corresponding decision table. [!\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\) Learn more...](#)

Managing servers in the Business console: As a permission manager, you can now use the Business console to manage the list of servers that are available through Decision Center for deployments, tests, and simulations. You can also restrict access to specific groups. [!\[\]\(fe3aebe81acea8d45108cd2768939da7_img.jpg\) Learn more...](#)

Tutorials

Getting started in Operational Decision Manager on Cloud: This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal. [!\[\]\(a8f9309f944226d1420f5fed22e2b6e6_img.jpg\) Learn more...](#)

Debugging a decision service in Rule Designer: This tutorial provides a methodology for debugging a decision service in Rule Designer. [!\[\]\(248b91fcdac4810ffd15cf33fb6aec6f_img.jpg\) Learn more...](#)

2017.10

Features introduced in October 2017.

General


Customer survey: An in-application survey now solicits feedback from users of Operational Decision Manager on Cloud. A pop-up window opens on your screen, giving you an opportunity to rate the service by selecting a score. (Note: The survey opens in English when it does not support the user's locale.) The system is set to check in with you every 90 days. Your time is valuable, and you are under no obligation to participate. Even if you close the pop-up without responding, taking a couple of minutes to provide input directly to the Operational Decision Manager on Cloud team is appreciated. Your feedback is taken seriously. By taking advantage of this feature, you can contribute to improving Operational Decision Manager on Cloud. [!\[\]\(d3e32d099174a7c248ec1f564ee4f69c_img.jpg\) Learn more...](#)

Password restrictions: The restrictions for composing passwords for the cloud portal have been updated to conform with the current guidelines from the Federal Risk and Authorization Management Program (FedRAMP). These restrictions do not apply to SAML accounts. [!\[\]\(40770d9ed6ed4f1222ebf89a1396e8b2_img.jpg\) Learn more...](#)


Rule Designer

Automatic exception handling in rule conditions: The behavior of the following BAL expressions is now the same as the other expressions in rule conditions:

- If there is no ... then ...
- If there is at least one ... then ...


The objects are counted only when the overall where clause is true. False and unknown objects are ignored.  [Learn more...](#)


Rule Execution Server console


Deleting resources and libraries cleanly: When you remove a Java™ XOM resource or library, it might be referenced by other artifacts or it might reference other artifacts. Now you can remove all the links that point to the resource or library, as well as all the links that point from the library, in the Rule Execution Server console in just one go, giving you a clean delete.  [Learn more...](#)


Decision Center

Decision Center REST API: You can use the Decision Center REST API to build, test, and deploy decision services. You can set up automated continuous deployment that uses the programming language of your choice.  [Learn more...](#)

Exporting and importing projects in the Business console: Administrators and configuration managers can now export the working branch of a decision service to a compressed file, and import a project into the Decision Center Business console from a compressed file. The working branch can contain more than one project.  [Learn more...](#)

Multipage display in the Business console: The Business console now shows the contents of its tabs (Decision Artifacts, Queries, Tests, Simulations, Deployment, and Snapshots) on one or more pages. The new feature improves the browsing of projects, especially when you have numerous artifacts.  [Learn more...](#)


Deleting decision services: Administrators can now delete decision services directly through the Business console. This operation used to be limited to the Enterprise console in Decision Center.  [Learn more...](#)

Embedded managed Java XOM in Decision Center: You can now use the Decision Center REST API to generate a RuleApp archive that includes an embedded managed Java XOM.  [Learn more...](#)


2017.06

Features introduced in June 2017.

Cloud portal

Service credentials for client applications: You use service credentials to authenticate a client application when it calls a decision service that is running in the cloud.  [Learn more...](#)

Decision Center tutorial

Merging branches in the Business console: This tutorial shows you how to merge changes between branches in a decision service in the Decision Center Business console. You can develop rules in one branch, and then transfer your changes the same rules in other branches.  [Learn more...](#)


Troubleshooting and support

Support for Decision Center: Find out how to get support for Operational Decision Manager on Cloud. Customer support is available for users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.  [Learn more...](#)


2017.03


Features introduced in March 2017.

Rule Execution Server

Running on Java 8: Rule Designer runs on Java 8. You must use JDK 8 and Eclipse 4.4 to run Rule Designer on an existing instance of Eclipse.  [Learn more...](#)

Decision Center Business console

Using custom test data providers: You can generate scenario files for custom test data providers in the Business console. You select the tests and expected execution details, and then download a custom data provider template. You can then use the template with various data sources, such as XML files or a database.  [Learn more...](#)

Comparing versions of a variable set: You can see the differences between versions of a variable set by comparing snapshots of the versions in the Business console.  [Learn more...](#)

2016.12

Features introduced in December 2016.

IBM Cloud™ Product Insights

Enablement for IBM Cloud Product Insights: *(No longer available.)* Decision Center is now enabled for IBM Cloud Product Insights. This Bluemix® service replaces the experimental service IBM Cloud Hybrid Connect. IBM Cloud Product Insights serves as a window onto your running inventory and runtime usage metrics.

Rule Execution Server

OpenAPI support for running decision services: Generating OpenAPI descriptions, or Swagger, is part of the Decision Connect capability. It makes it easier to integrate decision service APIs into calling applications. The descriptions are compatible with IBM API Connect® and other API management products. [!\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\) Learn more...](#)

Including resources and libraries in the deployed RuleApp archive: When you deploy a RuleApp archive by using the Rule Execution Server console, resources and libraries can be included in the deployed archive. [!\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\) Learn more...](#)

Decision Center Business console

Comparing and merging branches: You can now use the compare and merge function in the Business console to manually merge the contents of branches, releases, and activities in a decision service. [!\[\]\(3211b5d1d968fc1665909b34f9f16010_img.jpg\) Learn more...](#)

Managing resources in decision service projects: Upload files of any type to a decision service project in the Decision Center repository through the Business console. You can then find the files in the Resources folder of the project, where you can view, edit, create, or download files. [!\[\]\(6059a5aa8b4ca7bb793408023d6c6e42_img.jpg\) Learn more...](#)

Updating dynamic domains: You can now update dynamic domains in the Business console. If a domain is defined in an Excel file, you can download and upload the file from the Model tab in the Business console. [!\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d_img.jpg\) Learn more...](#)

Filtering columns in decision tables: Searching through a long decision table for a specific value can be slow, tedious work. The new filtering feature for decision tables can shorten your search time by showing only the rows that contain your filter values. [!\[\]\(6a9b39b98eb945faa14c645ec99e4eaa_img.jpg\) Learn more...](#)

Direct links to console content: Any URL in the Business console is accessible externally. You can bookmark these URLs in your browser, or reference them from an external source, and navigate directly to a specified branch, release, activity, or project element. You can now see the differences between versions of a variable set by comparing snapshots in the Business console. [!\[\]\(9c2e8d1b5bd77cb5c9f83b7a9cff79fd_img.jpg\) Learn more...](#)

Decision Center tutorials

Creating ruleflows: You set the order for firing rules in one or more ruleflows, and in doing so, list the rules that are run in your decision service. This tutorial shows you how to create and implement ruleflows. [!\[\]\(f60b7a900783ac3fd531bfd9c111be6d_img.jpg\) Learn more...](#)

Creating simulations: Use the simulator in the Business console to validate your rules with real or representative data. The simulator produces reports that you can use to refine decision services. Follow this tutorial to learn how to make simulations. [!\[\]\(f1c5da15572e3e09d343161be98f508d_img.jpg\) Learn more...](#)

Creating decision tables: *(No longer available)* Don't write the same rule over and over again to apply different condition and action values. Instead, create a decision table that forms a series of rules by applying the same rule statement to a table of values. Learn how in this tutorial on decision tables.

Parent topic: [What's new](#)

Deprecated features

These features are deprecated or removed from Operational Decision Manager on Cloud.

- Two-factor authentication
- Templates for action rules and decision tables
- Decision tree artifacts
- Ruleset completeness analysis

Parent topic: [What's new](#)

Introduction

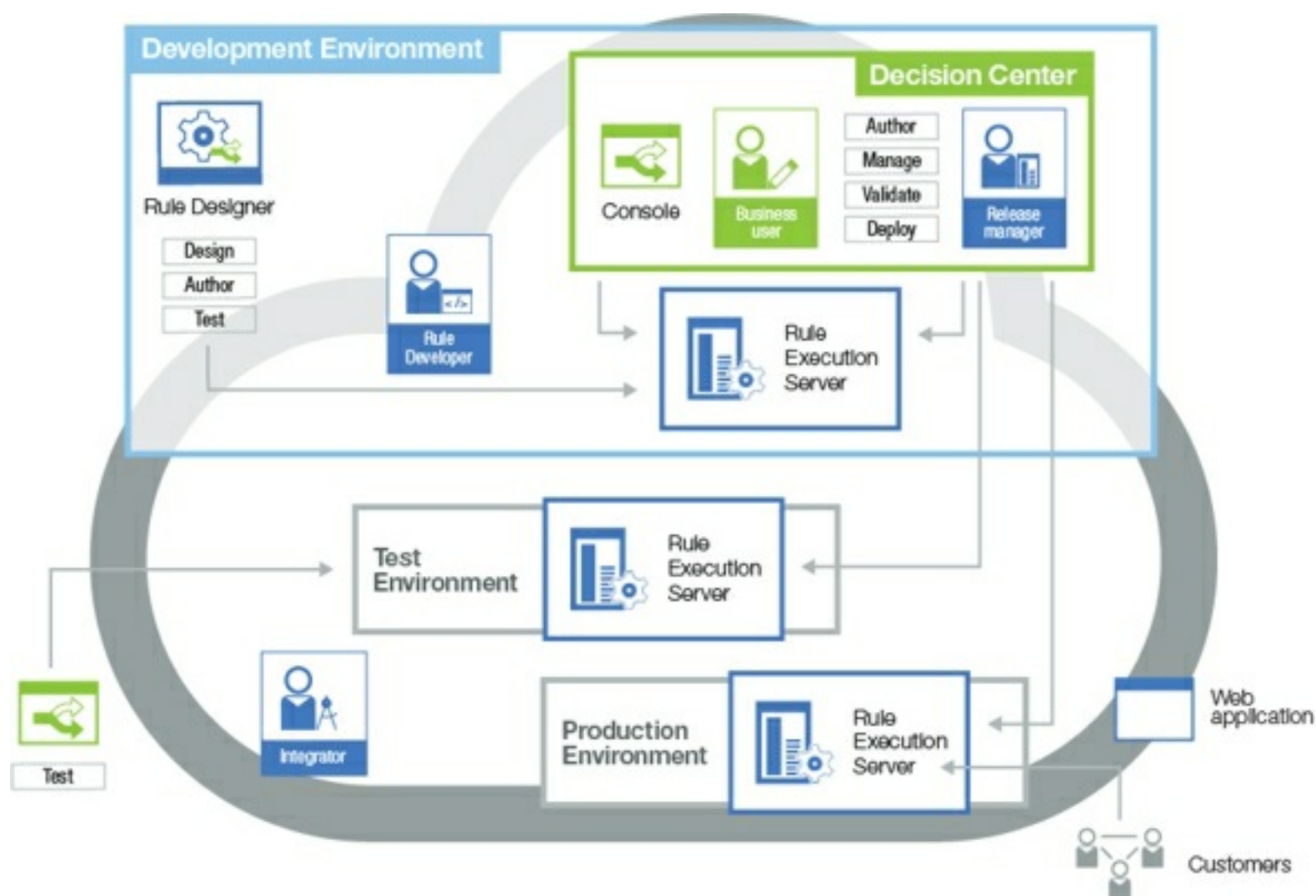
Operational Decision Manager on Cloud is a comprehensive decision automation platform for capturing, automating, and governing rule-based business decisions.

Both technical and business users can use its components to create solutions for key operations such as authorizing loans, applying promotions, detecting cross-sell opportunities, and processing insurance. You can implement decisions immediately to keep your applications continuously up to date and aligned with the changing business objectives of your organization.

Your instance of Operational Decision Manager on Cloud is ready to use when you log in to it for the first time. You get a dedicated version of Operational Decision Manager that is set up, configured, and provisioned for you. You have immediate access to tools for designing, developing, and deploying business rule applications.

Collaborative development

As the following diagram shows, you capture the policies of your organization in business rules through a collaborative service that supports developers, analysts, and business users.



You start by creating the business rules in Rule Designer, an Eclipse-based component, and then further develop and maintain the rules in an online platform that supports both business and technical users. Governance features control user access and impose a workflow to ensure the proper development of your rules. When your rules are ready, you deploy them to an execution server for use with client applications.

Two solutions to choose from

Operational Decision Manager on Cloud is available in the full version and Express. The full version gives you three environments – development, test and production – for a complete rule lifecycle solution. The Express version delivers one environment that delivers many of the features in the full version. You can add a test or production environment to an instance of Express, but you must get the full version to use all three environments together. Whichever solution you use, you can obtain additional execution servers for running your rule applications.

Maintained by IBM

IBM does all the maintenance on Operational Decision Manager on Cloud. It provides regular upgrades to ensure that you get the latest features, and it performs migration tasks to keep your data in sync with the platform. If you need an additional environment or execution server, IBM adds it for you.

Cloud workflow

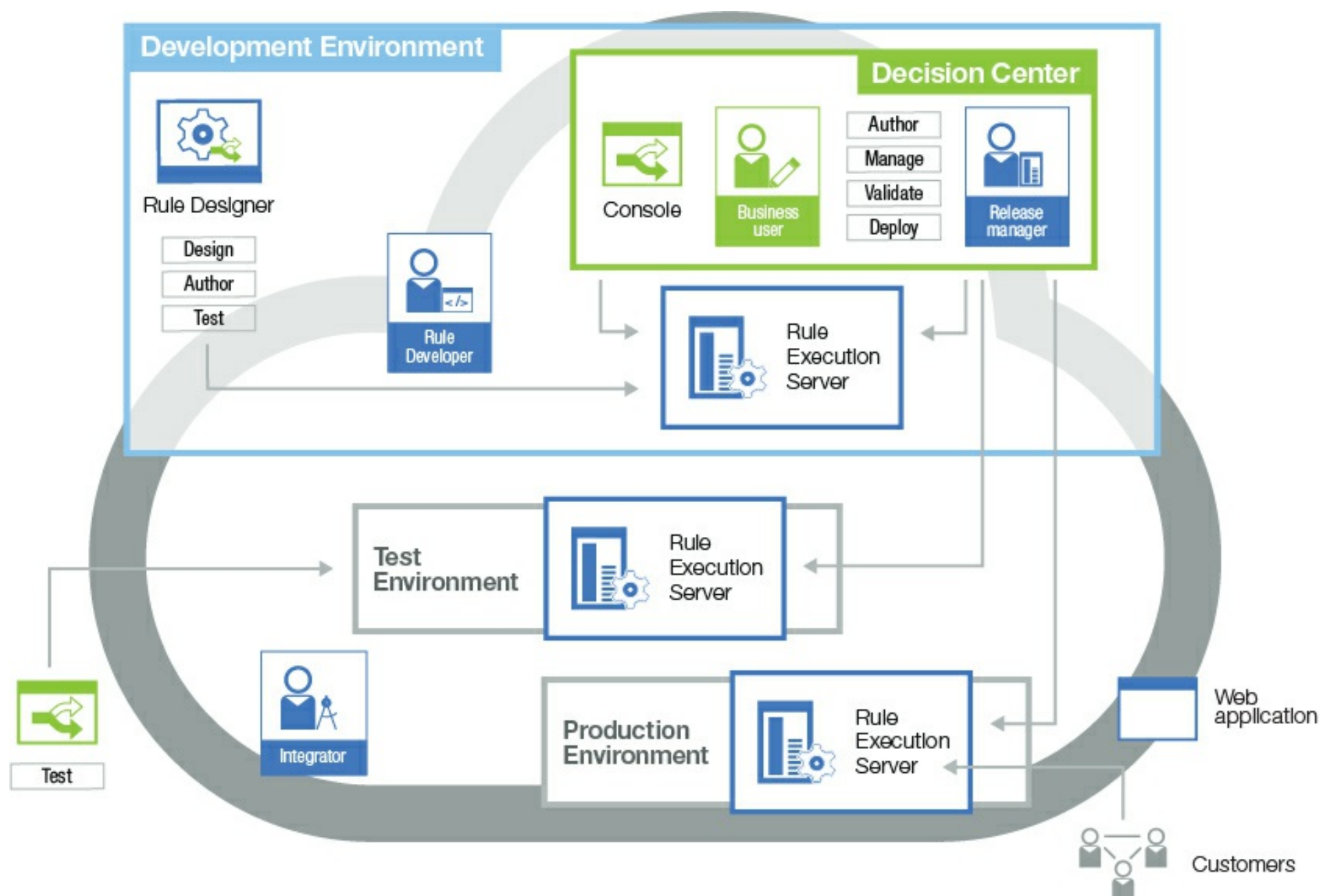
Follow the collaborative process for creating and deploying a rule application in Operational Decision Manager on Cloud.

This example takes you through the cloud workflow. A fictitious car rental company uses Operational Decision Manager on Cloud to implement rental decisions that are based on policies and pricing. A group of people collaborate to design the business rules, create and test a decision service, and deploy rule applications to runtime environments.

The example is organized into three sections:

- [Setting up the user roles](#)
- [Collaborating on the decision service](#)
- [Testing and promoting the decision service](#)

The following diagram illustrates how predefined user roles work with the product components during the lifecycle of a business rule application. The cycle includes rule development and validation, testing, and production. The collaborators work in the different cloud environments.



Setting up the user roles

In the table below, John is shown as the information technology (IT) administrator at the car rental company. He is responsible for administering user accounts in the Operational Decision Manager on Cloud portal. As the first person invited to the portal, he has the role of cloud administrator. Though he might not use the cloud components himself, he must invite other people to the portal and assign them roles so that they can use the components.

Table 1. Team members and user roles

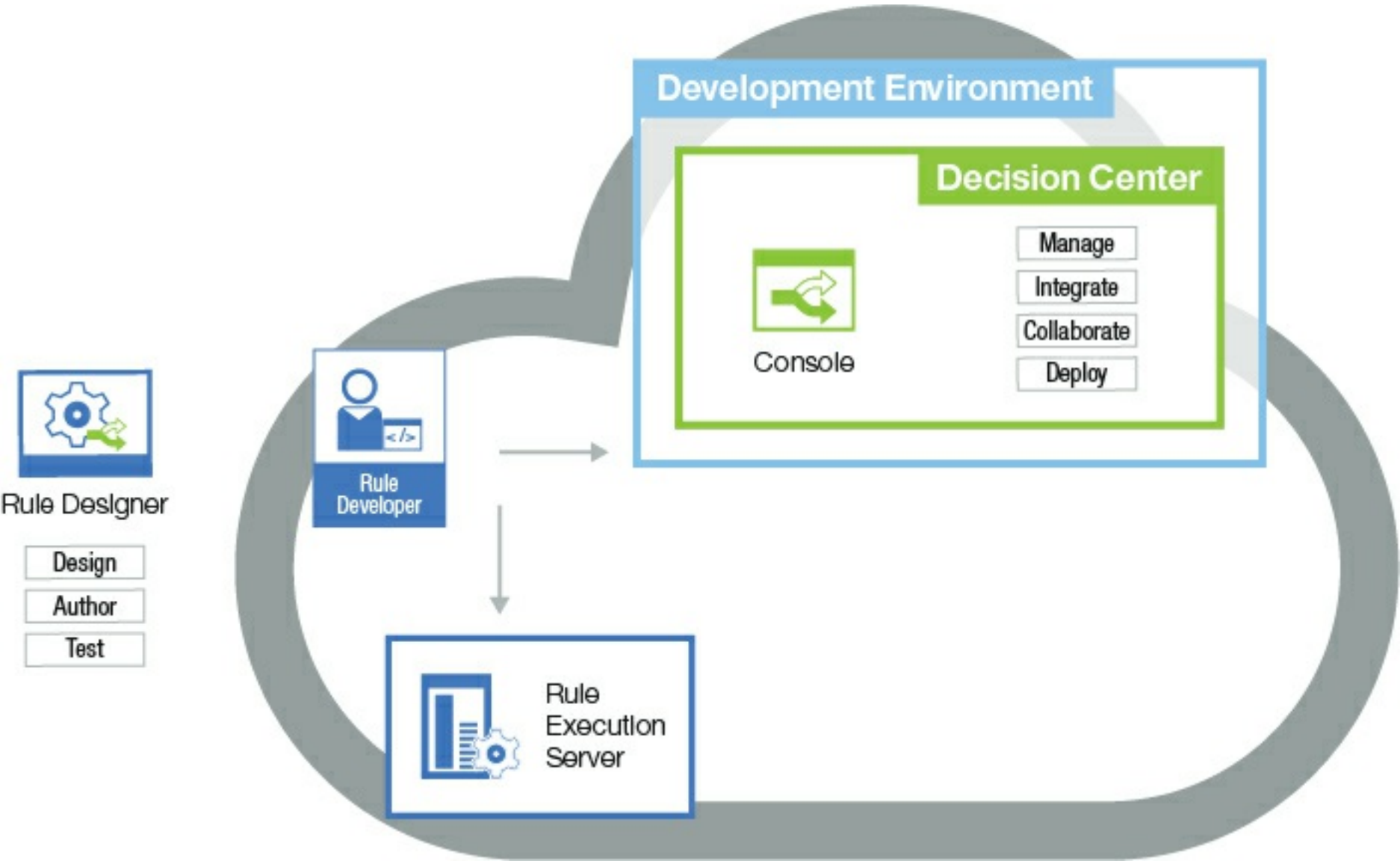
Team member	Job	User role	Component and environment access
John	IT administrator	Cloud administrator (admin) Permission manager	Operational Decision Manager on Cloud portal and Decision Center console, user accounts, groups and roles
Kim	Application developer	Rule developer	Rule Designer and Decision Center, deploys to the development and test environments
Gary	Marketing strategist	Business user	Decision Center Business console, deploys to the development environment

Frank	Analyst	Release manager	Decision Center Business and Enterprise consoles, including administrative functions, deploys to all environments
Arjun	IT operations specialist	Integrator	Decision Center Business console, deploys to development and test environments and performs decision service benchmark testing

Collaborating on the decision service

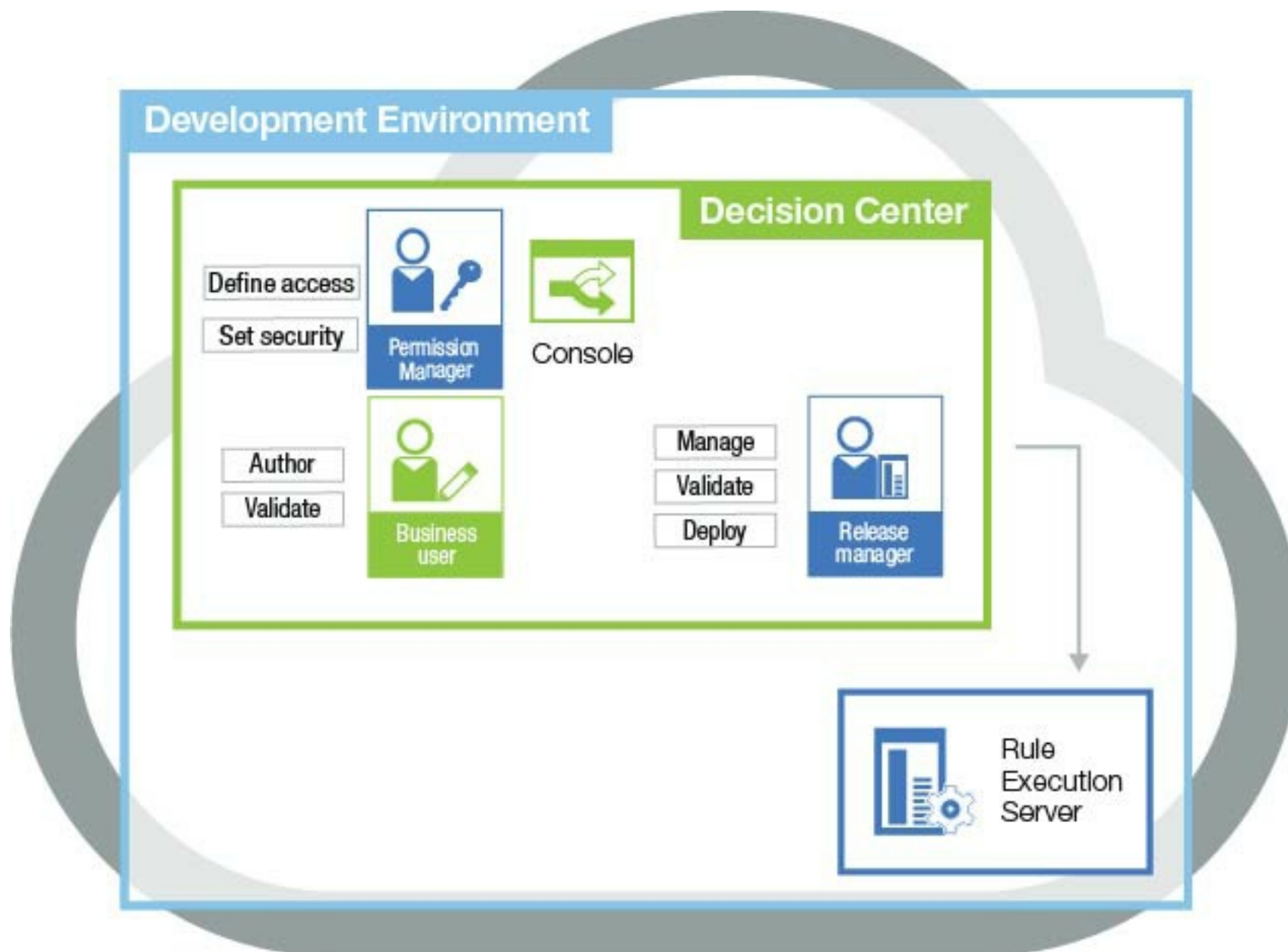
As the rule developer, Kim uses Rule Designer to create a decision service, deploy it to a runtime environment, and publish it to Decision Center. She knows Java™ and understands the company's object model. She works with software architects to develop the first version of the decision service, defining its vocabulary and writing the business rules that model the logic of the car rental business.

The following diagram shows Kim's activities. First, she creates the decision service and associated artifacts in Rule Designer. In working on the decision service, she tests it by deploying it to Rule Execution Server in the development environment. When she completes the initial version of the decision service, she publishes it to Decision Center.



As the release manager, Frank uses the decision governance framework in the Decision Center Business console. He creates a release for the decision service, and sets up change and validation activities in the release. When the team has completed the validation activities, and the rules are ready to be tested, Frank deploys the release to Rule Execution Server in the development environment.

In the following diagram, the business user, Gary, updates the decision service in Decision Center, and Frank deploys the decision service from Decision Center to Rule Execution Server.

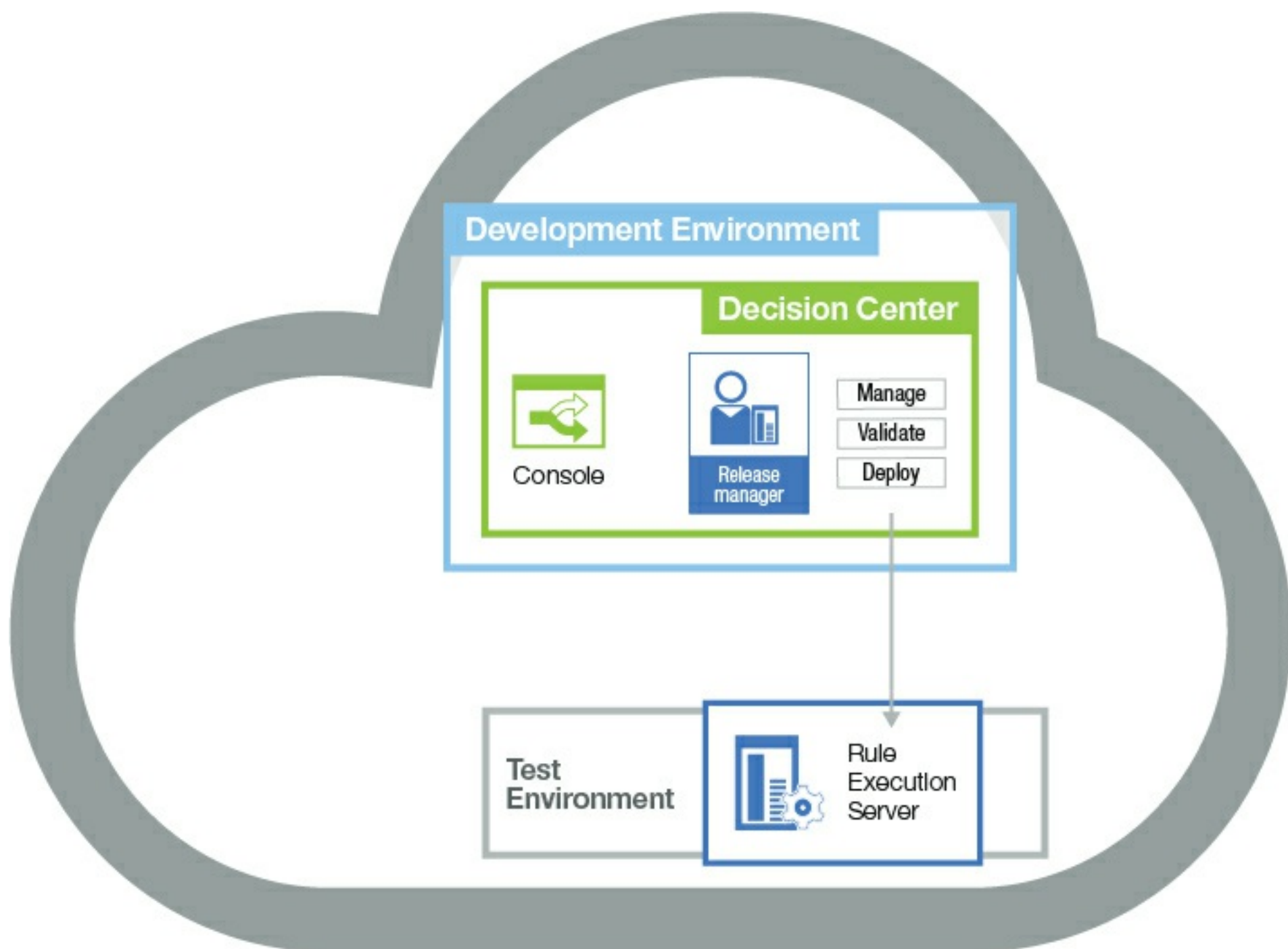


Because Gary works in the marketing department, he wants to review the pricing policies that are expressed in the rules. Gary views and edits the rules through the Decision Center Business console. Gary works in an activity branch that tracks his changes to the rules.

In addition to setting up the user accounts in the cloud portal, John also works as the permission manager in the Decision Center Business console. He sets the security and user access parameters in the administration tab. He makes sure that Gary, along with the other users who need access to the release and activity branches in the decision service, are assigned to the correct user groups.

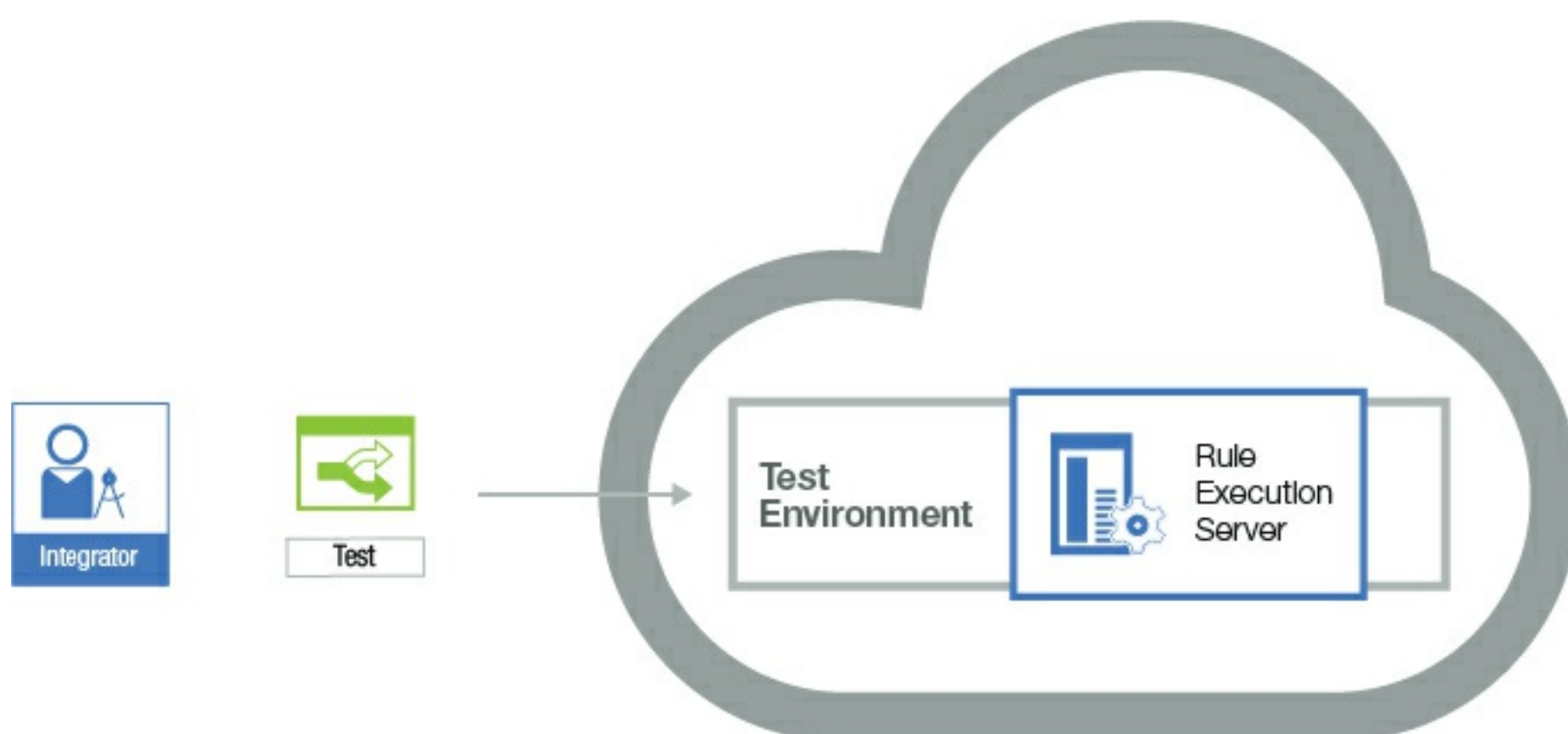
Testing and promoting the decision service

As the release manager, Frank has access to all three cloud environments. He can perform testing throughout the development lifecycle of the decision service. In the following diagram, when the team finishes the decision service in the development environment, Frank deploys it to the test environment.



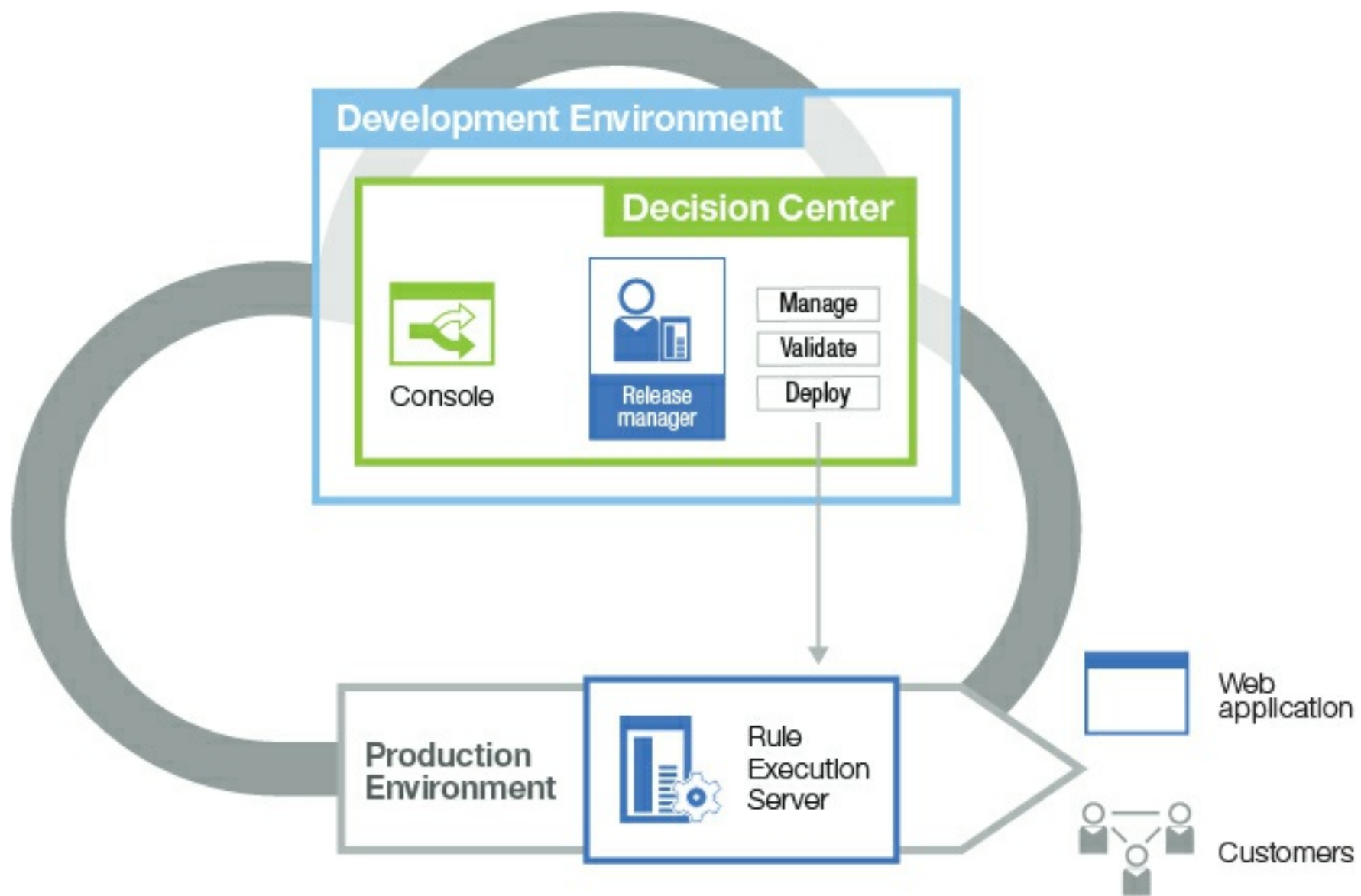
Frank works closely with Arjun, the integrator. To determine whether the decision service is running as expected, Arjun works in the Decision Center Business console and all three cloud environments. He monitors and evaluates the behavior of the decision service when it is called by a client application.

To evaluate the performance of the decision service in the test environment, Frank asks Arjun to run benchmarks that use standard software development validation procedures. Arjun adds code to a car rental web application to enable it to call the new decision service. In the following diagram, the car rental web application runs on the company application server and calls the decision service from the Operational Decision Manager on Cloud test environment.



When development is complete and Frank is satisfied with the benchmark results in the test environment, he promotes the decision service to Rule Execution Server in the production environment as the final step in the decision service lifecycle. As the release manager, Frank is the only user who can deploy a decision service to production.

The following diagram shows the promotion of the decision service to the production environment. When customers use the web application, it calls the decision service.



Gary can also tell his marketing colleagues that he can implement a new promotional pricing campaign next quarter because he can use Operational Decision Manager on Cloud to revise the pricing rules in the decision service.

Operational Decision Manager

Operational Decision Manager delivers a proven development system for capturing policies in solutions that automate the application of decisions.

Introduction

Operational Decision Manager provides cognitive technology that enables a business to respond to real-time data with intelligent, automated decisions. With Operational Decision Manager, IT and business users alike can manage the business decision logic that is used by operational systems within an organization.

Accessibility

Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with Operational Decision Manager.

Introduction

Operational Decision Manager provides cognitive technology that enables a business to respond to real-time data with intelligent, automated decisions. With Operational Decision Manager, IT and business users alike can manage the business decision logic that is used by operational systems within an organization.

Decision automation

Business agility depends on responsive, intelligent decision automation. Operational Decision Manager helps manage decisions separately from business applications, and with more flexibility and responsiveness to the changing needs of the business.

Decision services

Decision Server provides tools for designing, developing, and deploying decision services. Rule Designer is an Eclipse-based development environment in which you can develop and integrate decision services. Rule Execution Server provides the runtime environment for running and monitoring decision services.

Decision management

Decision Center is a scalable decision management application and repository with collaborative web environments for authoring, managing, validating, and deploying decision services.

Roles and activities

Each step of the rule development lifecycle requires specific skills that are held by different users. The roles of the users must be assigned at the outset of a project.

Parent topic: [Operational Decision Manager](#)

Decision automation

Business agility depends on responsive, intelligent decision automation. Operational Decision Manager helps manage decisions separately from business applications, and with more flexibility and responsiveness to the changing needs of the business.

The ability to deal with change in operational systems is directly related to the decisions that they are able to make. Every transaction, order, customer interaction, or process depends on decisions, which are in turn dependent on particular internal or external requirements and contexts. Every change therefore affects decisions, many of which are handled automatically within business systems.

Extracting decisions from your application code

Business policies are statements that are used to make decisions. These decisions can determine pricing for insurance or loan underwriting, eligibility approval for social or health services, or product recommendations for online purchases. Business policies are typically found inside application code, in the form of if-then statements. However, they can also be stored elsewhere for documentation purposes, such as in procedural manuals and other documents.



A business policy can be expressed as several business rules. Here is an example of the kind of business policy that might be familiar:

Customers who spend a lot of money in a single transaction need an upgrade.

The process of capturing rules happens in two steps. The first step consists in formalizing the vocabulary that is required to express the policy as a conceptual object model. The second step consists in representing the logic of the business policy as if-then statements.

After the vocabulary is created, the business policy can be implemented with the following business rule:

```
if
    the customer's category is Gold
    and the value of the customer's shopping cart is more than $1500
then
    change the customer's category to Platinum
```

When a business policy also has an IT policy or security policy that is embedded in it, you can combine business rule management with capabilities to handle the business policy aspects. For example, the following business policies can be handled as rules: *customers who spend a lot of money should be routed to a preferential service* or *customers who spend a lot of money require additional security on their transactions*.

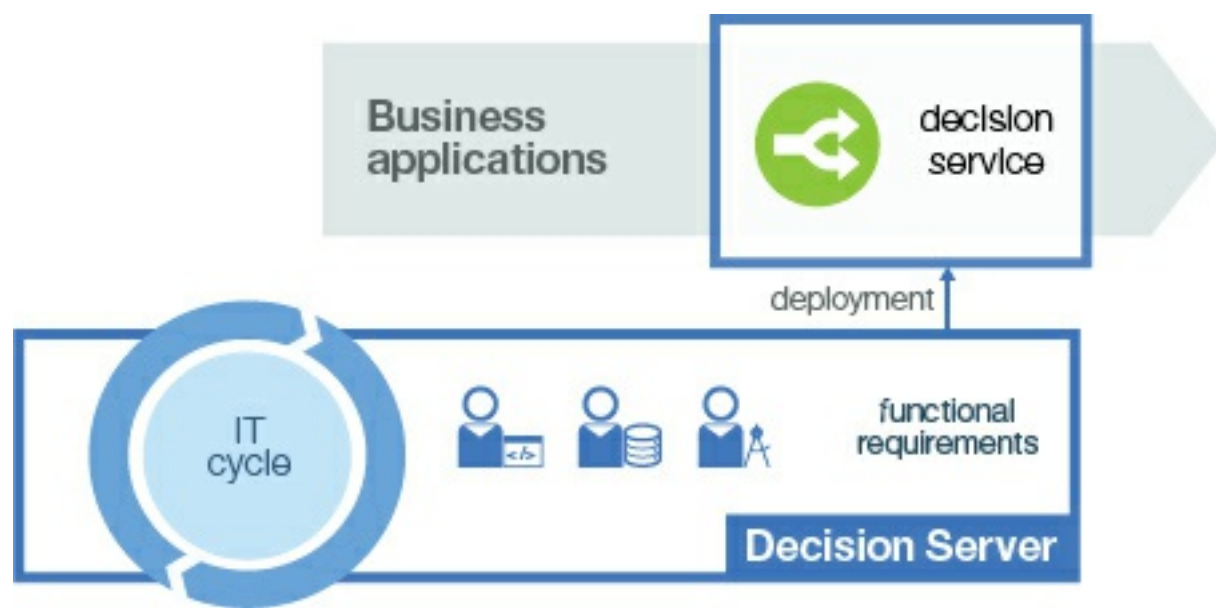
In the form of business rules, the business logic can be packaged and called from the application code as a decision service. Therefore, changes to the business policy do not require changes to the application or process code.

Managing decisions

When decision management is separate from application code, business experts can define and manage the business logic. Decision management reduces the amount of time and effort that is required to update the business logic in production systems, and increases the ability of an organization to respond to changes in the business environment.

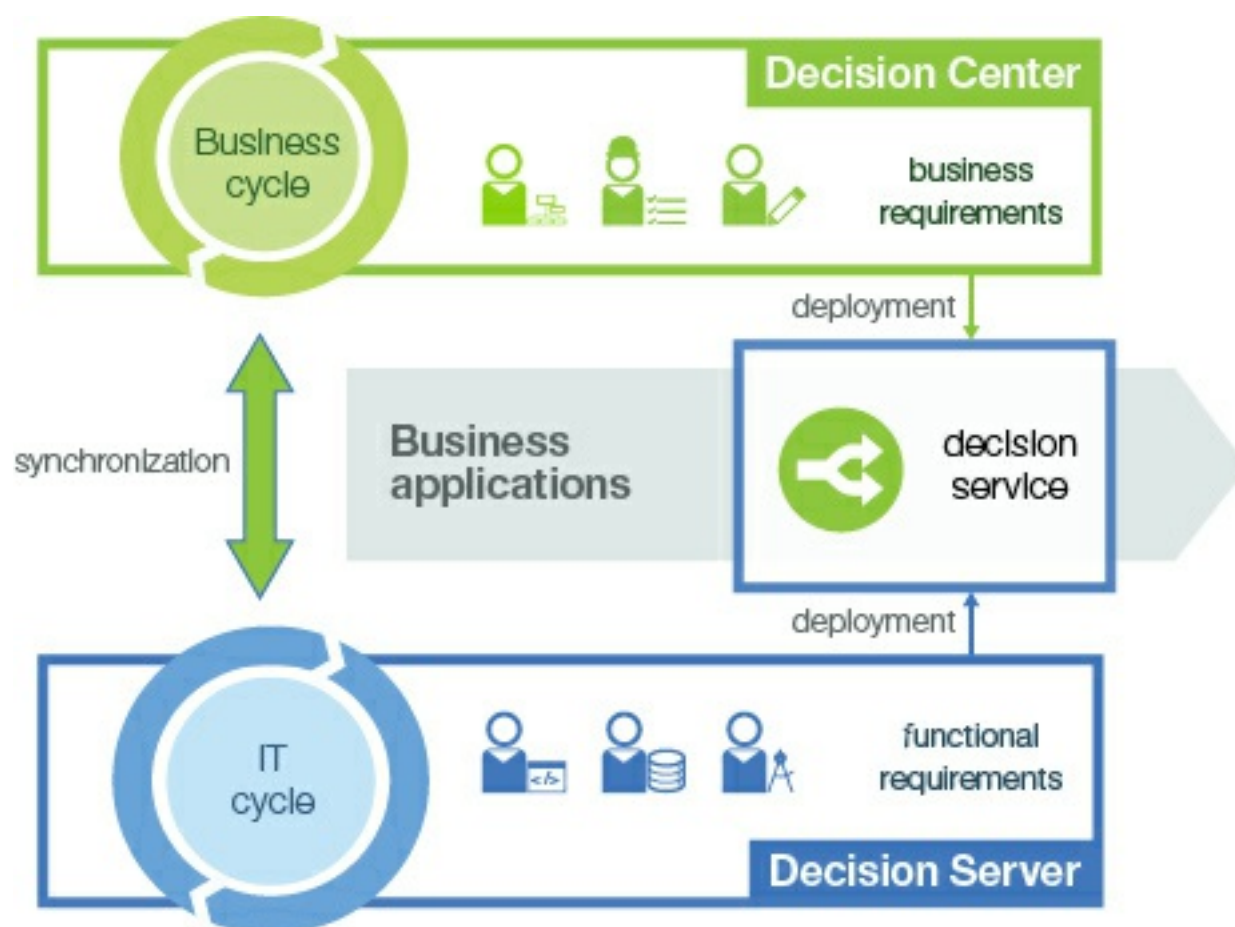
Operational Decision Manager provides an environment for designing, developing, and deploying decision services. The IT cycle consists of developing and maintaining this infrastructure. After the infrastructure is set up, distributed business teams can start collaborating through a web-based environment to create and maintain the decision logic.

Decision Server provides the runtime and development components to automate the response of highly variable decisions that are based on the specific context of a process, transaction, or interaction.



Putting decision management in the hands of business users

With Decision Center, business users can manage decisions that are directly based on organizational knowledge, with limited dependence on the IT department. Business users have a varying degree of dependence, which can range from a limited review to complete control over the specification, creation, testing, and deployment of the business logic. Business and IT functions can work in collaboration. The entire organization can align in the implementation of automated decisions. The maintenance lifecycle accelerates based on new external and internal requirements.



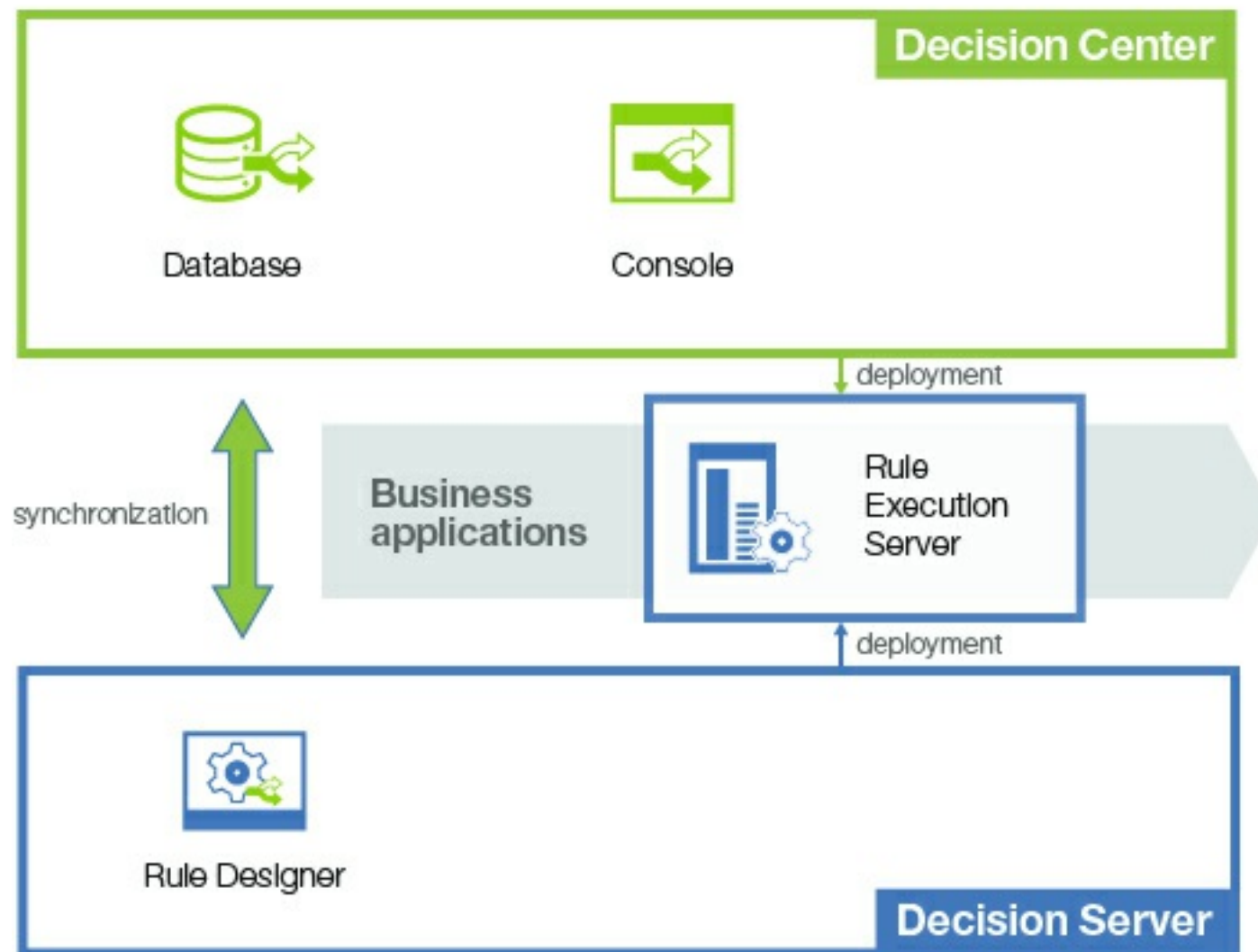
The lifecycles of decision service development and management can evolve in parallel. Decisions can evolve as required by the business context, without putting an extra load on the development of the business rule application. Each time the business rule application evolves, the decision management environment synchronizes with the development environment.

With this separation, decisions and application architecture can be managed asynchronously. For example, developers can develop a new version of a decision service in response to changing application infrastructure and core business requirements. At the same time, policy managers can work on new decisions that are delivered in response to an evolving market or a changing regulatory environment.

Operational Decision Manager components for decision management

The following figure shows the components that Operational Decision Manager provides for decision service development, rule management and authoring, and the execution environment.

IBM Operational Decision Manager



In addition to working on different timelines, developers and business users also expect to work with different tools, reflecting their different skill sets and views of the application. For example, developers are accustomed to the Java™ world. They use source-code management systems to work simultaneously on separate copies of a project without interfering with each other.

Business users do not concern themselves with the details of application development, but are interested in testing and managing decisions. Therefore, they need tools that can help them author, organize, and search for rules in the context of the overall policy.

With developers and business users that work in their own environments at their own pace, the work of these two groups must be synchronized and merged.

Finally, both developers and business users require access to a rule execution environment to deploy rules to enable testing, validation, and rollout to production of new and changed business logic.

Parent topic: [Introduction](#)

Related concepts:

[Decision services](#)

[Decision management](#)

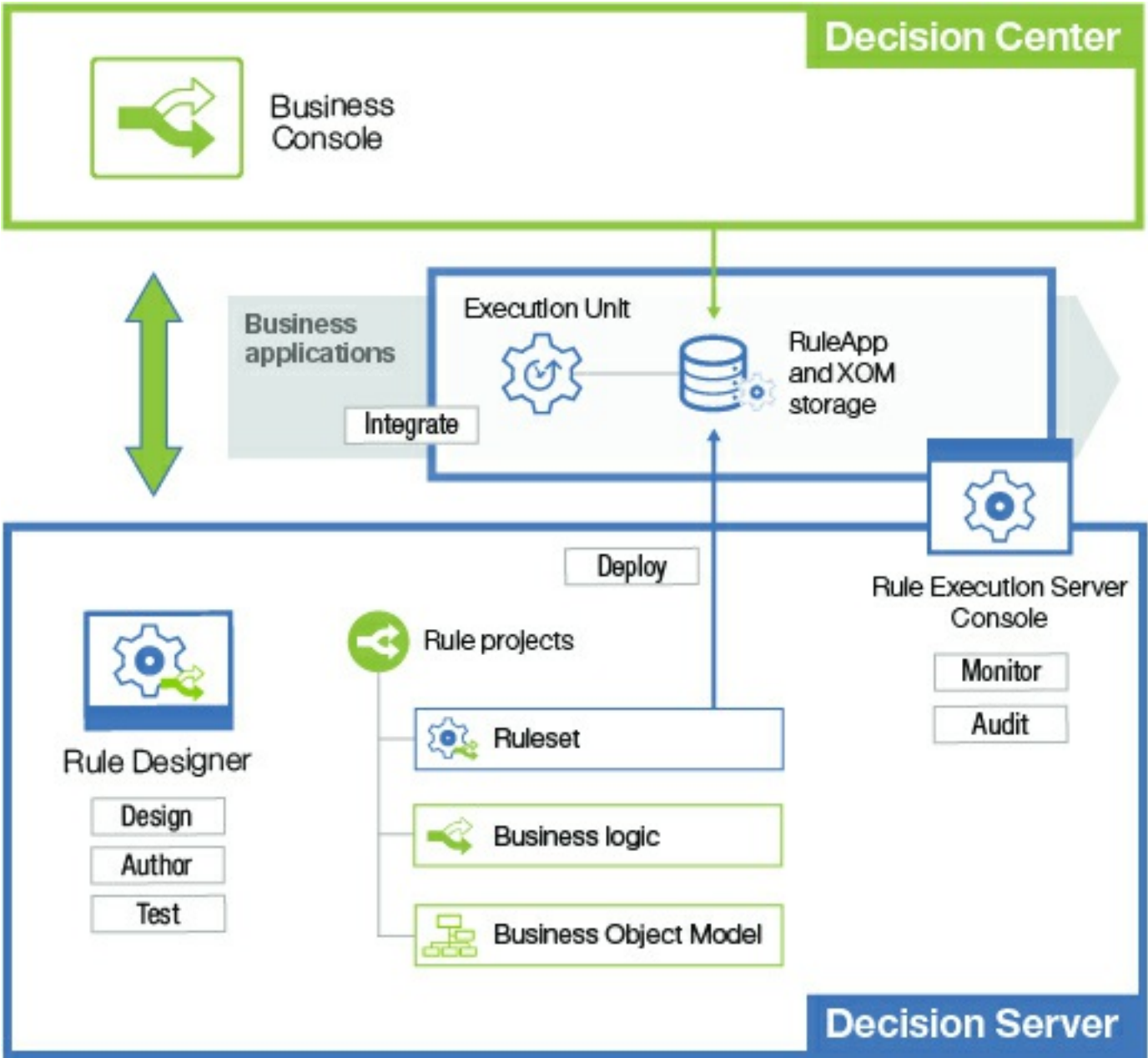
Related information:

[Roles and activities](#)

Decision services

Decision Server provides tools for designing, developing, and deploying decision services. Rule Designer is an Eclipse-based development environment in which you can develop and integrate decision services. Rule Execution Server provides the runtime environment for running and monitoring decision services.

The following diagram illustrates the different tools that you use to develop and deploy a decision service as a RuleApp, and the tasks you must do for this development.



Activity	Learn more
Get started To get started with the development tools and processes for business rules, follow the First Steps tutorial. To learn more about the architecture of business rule applications, read an overview.	Tutorials Introduction
Design In Rule Designer, you design the granularity of your decision services and the contract between client applications and the decision service. You also design the model and vocabulary for authoring business rules.	Designing projects for rule authoring
Author You create an initial set of business rules, and set up some tools to facilitate rule authoring for business users.	Authoring rules
Monitor You administer and monitor business rule applications by using the tools that are provided with Rule Execution Server.	Managing rule execution in Rule Execution Server

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision management](#)

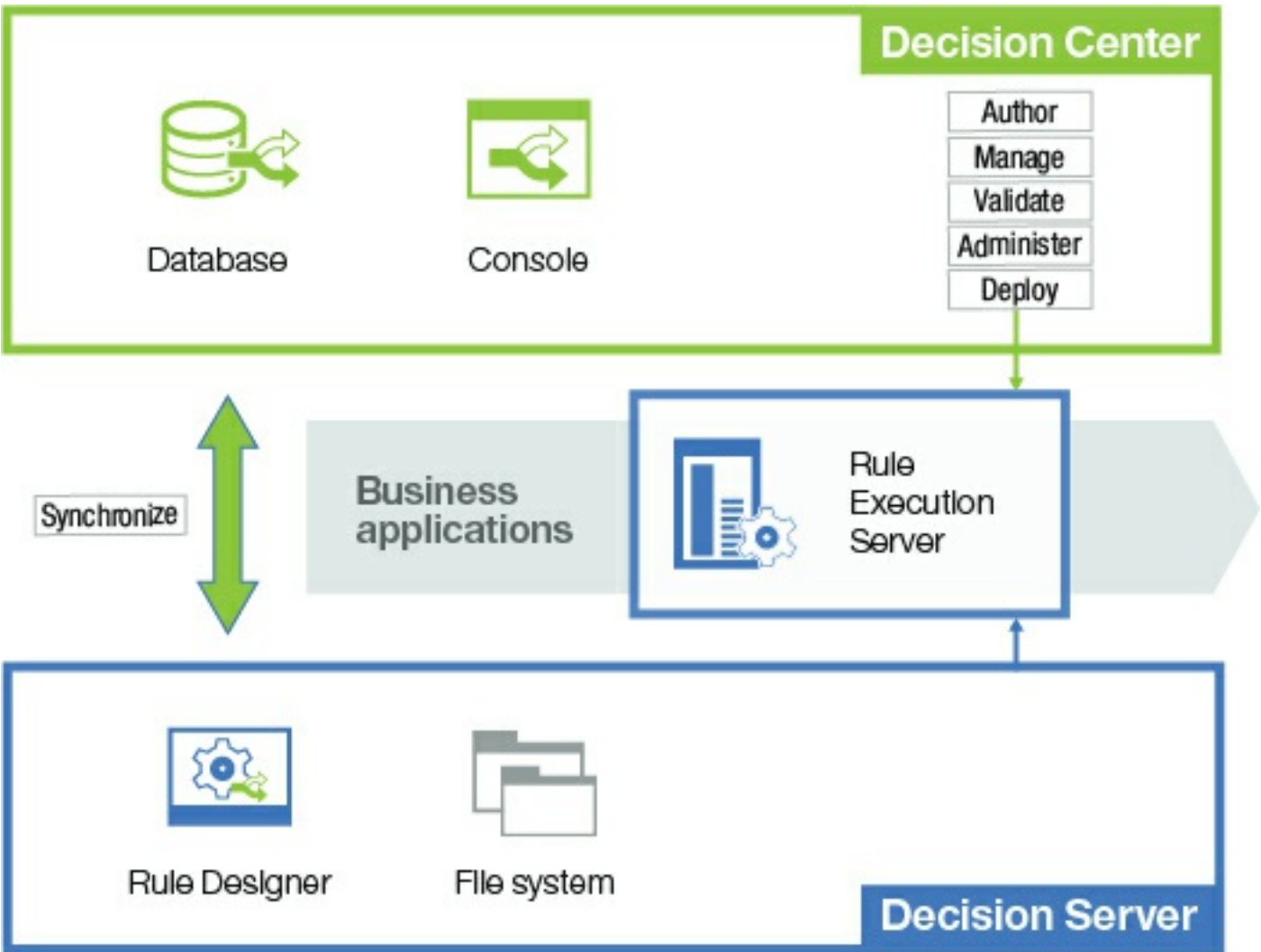
Related information:
[Roles and activities](#)

Decision management

Decision Center is a scalable decision management application and repository with collaborative web environments for authoring, managing, validating, and deploying decision services.

For decision services that are not too complex, business users can also create the underlying model on which decisions are authored.

The following diagram illustrates the main tools and tasks for decision management in Decision Center.



In the Business console, business users can work with a ready-to-use approach to change management and governance, which is based on releases and change activities. This approach is called decision governance.

Activity	Learn more
Get started To learn more about decision management in Decision Center, read the overview.	Decision Center
Configure For business users to manage automated decisions, you configure the projects and work environment. You can configure and customize the model and vocabulary that is used in the rules, and you can define permissions for different types of users. You also set up the deployment configurations. Part of the configuration can be done in Rule Designer, or in Decision Center for users with administrator privileges.	Designing projects for rule authoring
Synchronize Synchronization is the link between the IT cycle and the business cycle. When the projects are ready on the IT side, you can publish them to Decision Center. What you publish and the options you set when publishing have an impact on how rules are managed in the business side. If you publish a decision service, you can choose whether the decision service is managed with regular branches, or with the decision governance framework in the Business console.	Synchronizing and storing rules Managing changes

	nges
Manage To manage decision services, you can use the decision governance framework, a predefined release workflow that is based on change and validation activities. You can also choose to use regular branches for a customized change management workflow. The decision governance framework is only available in the Business console.	Overview: Identifying a set of rules Managing changes
Author Business users can author action rules and decision tables, and Decision Center tracks versions of rules. With the decision governance framework, you must create a change activity within a release to modify rules. Policy managers and rule authors can author business rules in the Decision Center Business and Enterprise consoles. You can integrate business rule authoring and management extensions that are developed in Rule Designer into the Decision Center consoles.	Working with rules
Validate Decision Center includes testing and simulation features that enable business users to validate the behavior of rules. Business users must be confident that business rules are written correctly and that any update does not break the business logic encapsulated in the ruleset. Business users validate the business logic against well-defined usage scenarios, by running tests and simulations against their rules.	Testing sets of rules Simulating business application results Decision Center RES T API

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision services](#)

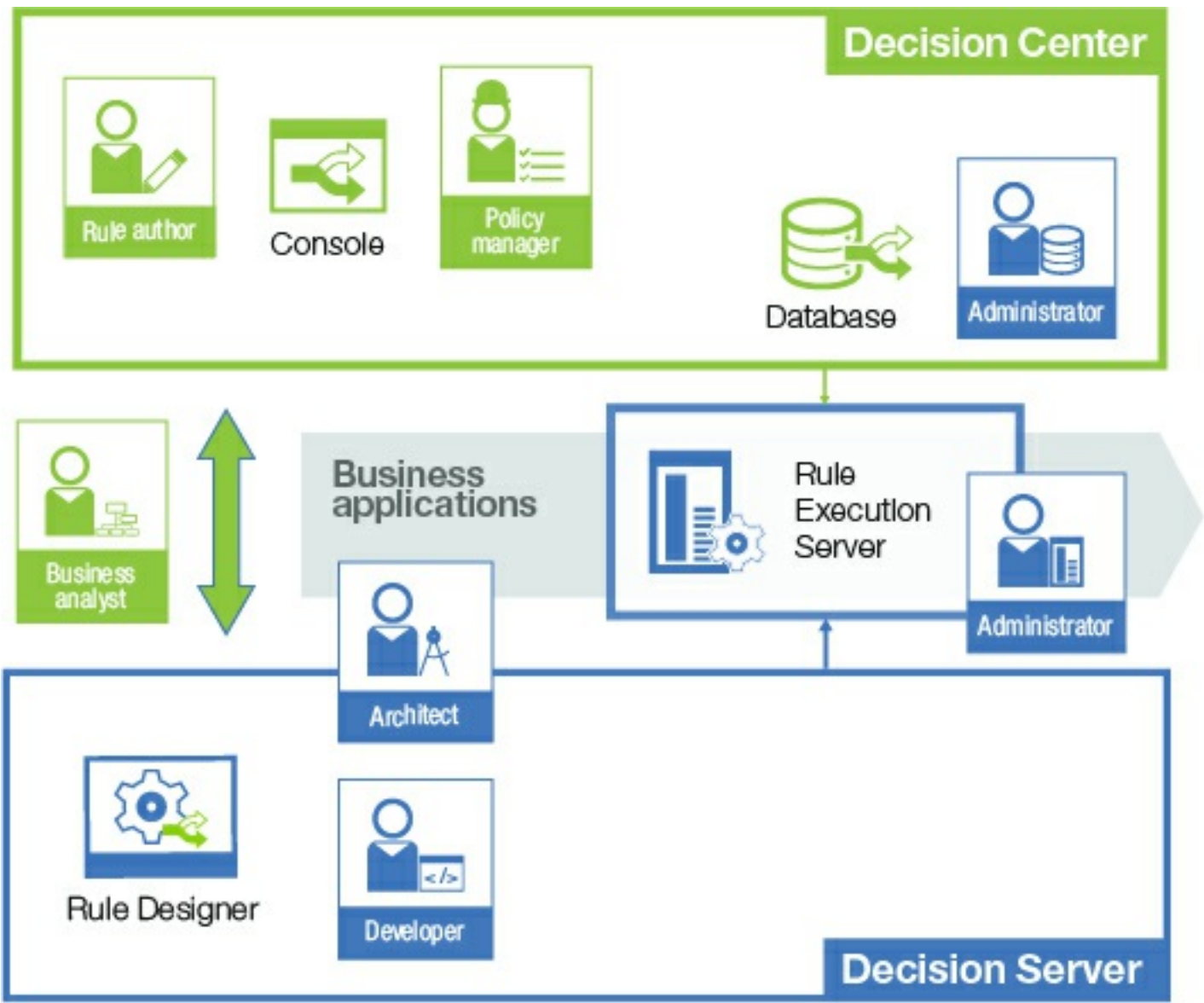
Related information:
[Roles and activities](#)

Roles and activities

Each step of the rule development lifecycle requires specific skills that are held by different users. The roles of the users must be assigned at the outset of a project.

Decision Server Rules and Decision Center

Together, Decision Server Rules and Decision Center provide a comprehensive decision management environment. The following figure shows the various development roles and their places in the management environment.



The roles can be broken down into two categories:

IT users



The architects, developers, and administrators who develop and maintain the rule application.



Business users


The business analysts, policy managers, and rule authors who develop and maintain the decision logic.

The following table lists the different types of IT and business users.

Table 1. User roles for the development of business rule applications

Role	Activities	Description
	<ul style="list-style-type: none">DesignIntegrateDeploy	<p>Architects work mainly in Decision Server to do the following tasks:</p> <ul style="list-style-type: none">Define the decision services and their interface with the calling applications.Define the project organization for the developers and business users. For example, they set up the data model for the rule vocabulary.Optimize rule execution.
	<ul style="list-style-type: none">Design	<p>Developers know the object models, APIs, and the development environment. They work mainly in Decision Server to do the following tasks:</p>

	<div>g n</div> <ul style="list-style-type: none">• A u t h o r• T e s t• I n t e g r a t e• D e p l o y	<ul style="list-style-type: none">• Develop, test, debug, and deploy decision services. They contribute to the design of the rules.• Define the client execution code to integrate decision services in to business applications.
	<ul style="list-style-type: none">• D e s i g n• A u t h o r• T e s t• S y n c h r o n i z e• M a n a g e• V a l i d a t e	<p>Business analysts work with business and IT departments, from design to integration of a software application. They work in Decision Server and Decision Center to do the following tasks:</p> <ul style="list-style-type: none">• Design a formal specification for the rules, with validation from both developers and policy managers.• Define the vocabulary for the rules.• Identify candidate business rules.• Write and organize business rules so that rule authors can maintain them.• Validate rules to make sure they produce the expected results. <p>Business analysts can share tasks with developers, but typically they do not write code.</p>
	<ul style="list-style-type: none">• M a n a g e• V a l i d a t e• A u t	<p>Policy managers own the decisions within an organization. They work mainly in Decision Center to do the following tasks:</p> <ul style="list-style-type: none">• Participate in the design of a formal specification for the rules.• Define vocabulary elements with the help of business analysts.• Manage releases and validation activities in the context of decision governance.• Create and update rules.• Review how the running of rules is orchestrated.• Report on the status of the decisions.• Test rules to ensure that they provide the intended business outcome. <ul style="list-style-type: none">• Run simulations to assess the potential impact of changes to rules.

	h o r	
	<ul style="list-style-type: none"> • A u t h o r 	<p>Rule authors work in Decision Center to do the following tasks:</p> <ul style="list-style-type: none"> • Update and sometimes create rules. • Review business rules. • Create change activities in the context of decision governance.

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision services](#)
[Decision management](#)

Accessibility

Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with Operational Decision Manager.

This product supports the following accessibility features:

Keyboard shortcuts

You can navigate through the different Operational Decision Manager environments and their documentation using keyboard shortcuts. The rule management environments, such as Eclipse, provide documentation on their accessibility features. You can also find topics that cover keyboard shortcuts in the documentation for Operational Decision Manager.

Magnification

You can use the settings of display systems to magnify development interfaces and supporting documentation.

Screen reading

You can use a screen reader with a digital speech synthesizer to read aloud what is displayed on your screen. Consult the documentation with your assistive technology for details on using it with this product suite and its documentation.

Technical documentation

The IBM® product documentation is available in an online environment that you can access by using a web browser. See [IBM Knowledge Center release notes](#).

Keyboard shortcuts

The Operational Decision Manager modules provide keyboard shortcuts. You can find information on the keyboard shortcuts in other topics and the product interfaces.

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see [IBM Accessibility](#).

Parent topic: [Operational Decision Manager](#)

Key components

Operational Decision Manager on Cloud includes the main components in Operational Decision Manager.

Decision Server

Decision Server provides development and production components for creating and running rule applications.

Decision Center

Decision Center supports collaborative development and management of decision services.

Version differences

Operational Decision Manager on Cloud has fewer features than the on-premises version of Operational Decision Manager.

Parent topic: [Overview](#)

Decision Server

Decision Server provides development and production components for creating and running rule applications.

The following Decision Server components are in Operational Decision Manager on Cloud:

- Rule Designer: An Eclipse-based development component for designing, authoring, testing, and deploying rule applications.
- Rule Execution Server: The execution component for running and monitoring rule applications.

In a rule-based solution, a client application requests a decision from a rule application. The client application can contain many decision points that use the rule application. At each decision point, business rules in rulesets express policies for decisions. A rule application is deployed to Rule Execution Server as a RuleApp. Each RuleApp contains one or more rulesets, and each ruleset corresponds to a decision.

Decision Server also interacts with Decision Center, which includes a rule repository and collaborative web consoles for business and technical users to author, manage, validate, and deploy rules.

The design and development phases of rule applications are managed through rule projects. In Operational Decision Manager on Cloud, you work primarily in rule projects called decision services. A decision service can support complex decisions that involve several rule projects. Synchronization, branching, and change management are applied to all the rule projects in a decision service hierarchy, allowing the decision service behavior to be governed and deployed consistently.

Creating

In Rule Designer, you create a rule application and set up its connection with the client application. You also create the model and vocabulary for authoring business rules.

To create a rule application, you must put in place the necessary infrastructure for editing rules and producing one or more rulesets:

- You define the business rules that make up an executable decision unit, or ruleset. The ruleset uses input and output parameters to pass data to and from the client application. You give each ruleset a unique signature of input and output parameters. In decision services, you create decision operations that define the contents and signatures of rulesets. Because you can use the same rule projects and packages in different decision operations, you can easily change a decision service to add new decision points that require different signatures.
- You define the vocabulary that is used in the business rules. In Rule Designer, you develop the business object model (BOM) that defines the elements and relationships in the vocabulary. You can define the vocabulary by mapping the BOM to the execution object model (XOM). You can also create the vocabulary by generating the BOM from the XOM, and then configuring the business vocabulary from the BOM.
- You set up a rule project hierarchy. A rule project is a type of Eclipse project that is dedicated to the development of rule applications. In a decision service, a main rule project serves as the top-level project so that the decision service behavior can be governed and deployed consistently.
- You organize rule packages in rule projects to store business rules and define a ruleflow to specify the order for executing rules.
- You set up business user validation tools by configuring and customizing tests and simulations.

Authoring

If you are responsible for creating and managing the rules, you might author most of the business rules in a project. If business users are responsible for creating and managing the rules, you set up tools to facilitate rule authoring for them. You can create the following types of business rules:

- Action rules
- Decision tables

These business rules use the Business Action Language (BAL), which is designed to have a natural language syntax. You can also create technical rules, which are based on the ILOG® Rule Language (IRL) and require programming skills.

Authoring might require you to do the following operations:

- Create business rule templates to facilitate the creation of rules that use a common pattern.
- Define vocabulary categories to filter the vocabulary elements that are available when you author business rules.

Debugging and testing

You can debug a ruleset in Rule Designer, and test that the decision service or a project implements the expected business logic:

- You debug the ruleset by using a rule engine to manage rule execution.
- You analyze rules by using constrained semantic queries, which check the consistency and completeness of individual rules and the ruleset as a whole.
- You run test scenarios on rules. You can run these tests locally in Rule Designer, without the entire Rule Execution Server component.

Integrating

You integrate the client application from Rule Designer.

After the ruleset connection is defined, you host the ruleset on Rule Execution Server and call it from the client application. Rule Execution Server is the execution component for deployed business rules. It handles the creation, pooling, and management of ruleset instances so that client applications can call the decisions as easily as possible.

To call the ruleset from the client application, you post the ruleset as a hosted transparent decision service (HTDS) that is called through web service protocols.

Deploying

A RuleApp holds a group of rulesets that are deployed together. The RuleApp also contains the input and output parameters that define the interaction with the client application, and the ruleset path needed by the client application to identify rulesets and their versions.

When a decision service is fully validated and approved, you can deploy it to Rule Execution Server for use in production. You deploy the decision service by using a deployment configuration that defines what to assemble in a RuleApp and where to deploy it. A deployment configuration can reference one or more decision operations. A decision operation includes all the settings needed to produce a ruleset, including input and output parameters, main ruleflow, and any rule extraction. The deployment configuration can be synchronized with Decision Center, and subject to change management and governance in the Business console.

Parent topic: [Key components](#)

Decision Center

Decision Center supports collaborative development and management of decision services.

Business users can work in Decision Center to apply organizational knowledge and best practices in decision services, with little dependence on developers. Decision Center includes a rule repository and web consoles for authoring, managing, validating, and deploying rules.

Rule editing

Business users work primarily in the Decision Center Business console. They use rule editors that ensure that they use the correct business vocabulary and comply with proper business rule syntax.

There are two consoles in Decision Center:

- Business console: The primary environment for change management and deployment of decision services (see [Governing rules with the Business console](#))
- Enterprise console: The environment in which administrative users do certain tasks (see [Managing business rules with the Enterprise console](#))

Synchronization

Developers and business users work on rules in different environments, and save the rules to different locations. Developers typically work in source code control, and business users save to the Decision Center repository.

To enable collaboration between the development and business cycles of rules, you must use the synchronization tool in Rule Designer to synchronize the work that is done by the different users (see [Synchronizing and storing rules](#)).

Deploying rules

When you complete the development and testing of rules, you can deploy them to Rule Execution Server, which is the runtime execution component. In Decision Center, you deploy from the Business console. You can also deploy rules by using the [Decision Center REST API](#).

Administrators can create and edit deployment configurations in decision services. These deployment configurations can then be used to deploy releases, and change activities and regular branches in the decision services. When you deploy decision services, the XOM is also deployed.

Decision governance framework

Decision Center offers a ready-to-use approach to change management and governance that is based on decision services, releases, and activities. It encourages collaboration, and enables more advanced users to check the work of other users.

The decision governance framework is the preferred way of working and is presented as such to the business users. You should be familiar with the following sections:

- [Change management](#)
- [Governance principles](#)

Validating rules

You can validate the behavior of rules or assess the effects of rule changes. The Decision Center Business console can run tests and simulations:

- Tests: Compare expected results with the actual results from applying rules to usage scenarios.
- Simulations: Run rules against representative data to determine how changes to the rules affect key performance indicators.

You can also run tests with the [Decision Center REST API](#).

Parent topic: [Key components](#)

Version differences

Operational Decision Manager on Cloud has fewer features than the on-premises version of Operational Decision Manager.

Both the cloud and on-premises versions of Operational Decision Manager include collaborative components. However, the following on-premises features are not available in the cloud version (note that some of these features are also no longer available in the on-premises version):

- Decision Server Insights
- Classic rule projects
- Classic rule engine
- Rule Solutions for Office
- Decision Warehouse
- Decision Validation Services (DVS)
- Customization of Decision Center and Decision Server
- Java™ API
- Ant scripts
- .NET platform
- Sample server and samples console
- Operational Decision Manager tutorials
- B2X or XOM methods that make outbound calls to a database or a service
- COBOL and PL/I support
- Access to the server operating system
- Access to the application server console
- Repackaging of EARs (no customization)
- Rule model extension
- Custom value editor
- Custom data provider for dynamic domains
- Custom data provider for testing and simulation
- Monitored Transparent Decision Service

Parent topic: [Key components](#)

Cloud environments and user roles

The components for creating and using rule applications are kept in cloud environments. Access to the environments is controlled through user roles.

Each Operational Decision Manager on Cloud portal contains cloud environments. The standard cloud environments are development, test, and production.

Each environment covers part of the decision service lifecycle, and has the appropriate Operational Decision Manager components for the work. For example, you create decision services in the development environment, which contains Rule Designer, Rule Execution Server, and Decision Center.

Access to the environments and their components is controlled through user roles, which are assigned based on the type of work. The role of business user, for example, can limit a user to modifying decision services in the Decision Center Business console.

The following table associates the user roles with the different cloud environments. The first environment is for creating decision services, the second is for testing decision services, and the third is for making decision services available to client applications.

Table 1. Table shows how the access of cloud user roles can be limited to environments and components.

User role	Development environment			Test environment	Production environment
	Rule Designer	Decision Center	Rule Execution Server	Rule Execution Server	Rule Execution Server
Rule developer	X	X	X	-	-
Release manager	X	X	X	X	X
Business user	-	X	X	X	-
Integrator	-	X	X	X	X
Permission manager	-	X	-	-	-

Cloud environments

Operational Decision Manager on Cloud comes with cloud environments for developing, testing, and calling decision services.

User roles

Access to the environments and components is controlled through user roles.

Cloud environments

Operational Decision Manager on Cloud comes with cloud environments for developing, testing, and calling decision services.

Development Environment(s)

Rule developers, release managers, business users, integrators, and permission managers use this workspace to collaborate on the development of decision services. It includes Decision Center, the main component for collaborative development and governance. The workspace also includes at least one instance of Rule Execution Server, the execution environment for running rulesets deployed from decision services. These activities can include functional tests and simulations in Decision Center, or a sample application that calls a ruleset. A development version of a production application can be used to obtain feedback from domain users during the development phase. Rule developers create decision services in Rule Designer, and then publish the decision services to Decision Center.

Test Environment(s)

Business users and integrators use one or more instances of Rule Execution Server in this workspace to test rulesets from decision services during or after development, and within the governance framework that is defined by the release manager. Tests can include nonfunctional tests for performance, system integration, or other user acceptance requirements. For example, a nonproduction web application can be used as a production application to run a ruleset on simulated data to validate the results. In this workspace, you can use a load testing harness to stress a ruleset, and measure its performance and scalability.

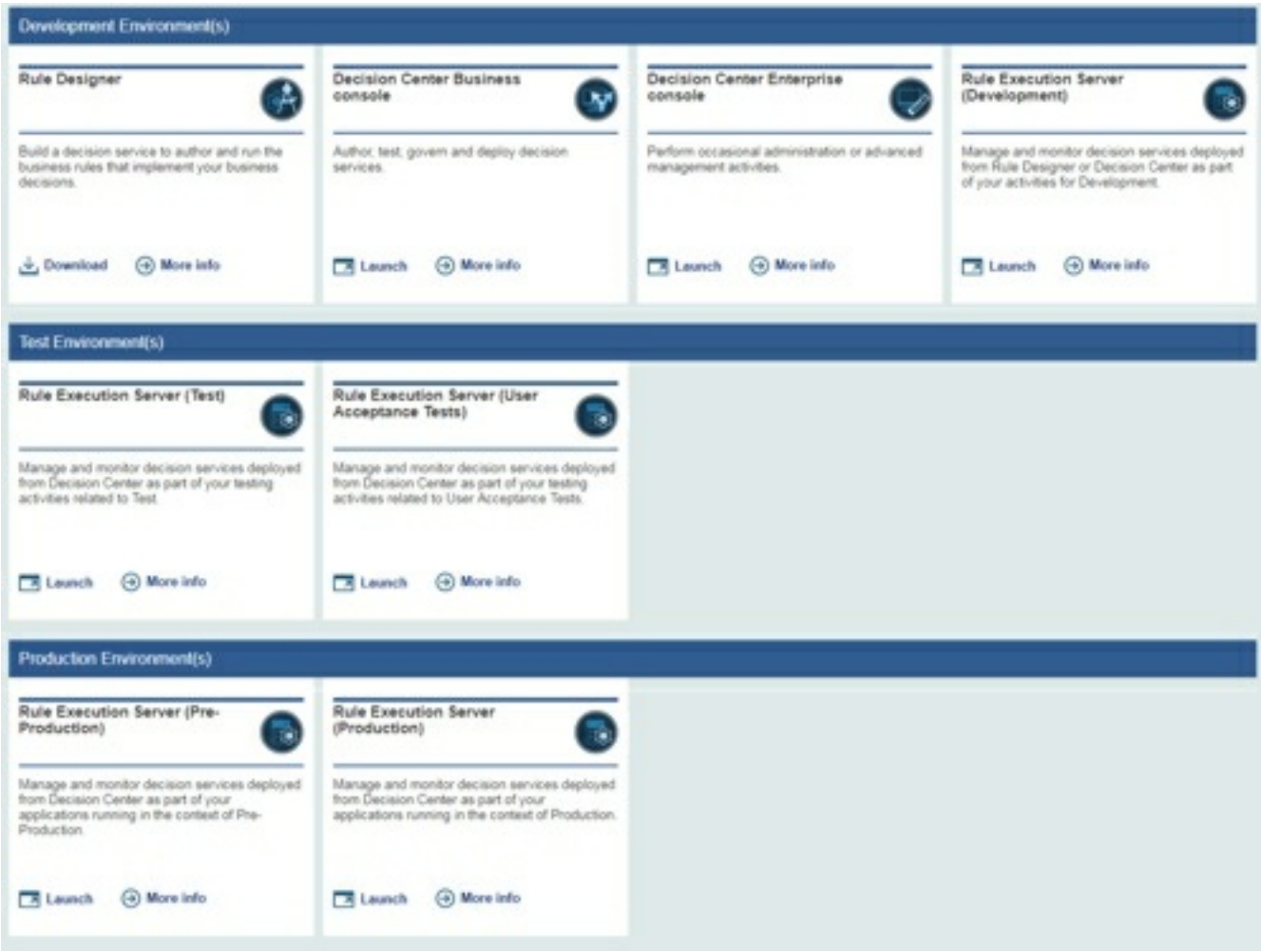
Production Environment(s)

Release managers use one or more instances of Rule Execution Server in this workspace to run rulesets from decision services for production applications. This workspace enables an application to call a ruleset to implement a decision-based business function. This workspace is not for development or tests. It is restricted to release managers, who are ultimately responsible for validating, completing and deploying a ruleset to production. Some user roles can view what is deployed, but some roles cannot access this workspace.

Adding execution environments

You can add additional instances of Rule Execution Server to your environments. During the provisioning of your cloud portal, you determine how many execution servers are needed for your work. You can also purchase more execution servers for an existing portal by contacting your IBM® representative.

The following image shows a cloud portal that has five instances of Rule Execution Server:



Parent topic: [Cloud environments and user roles](#)

User roles

Access to the environments and components is controlled through user roles.

Operational Decision Manager on Cloud has two groups of users.

Cloud group

At the cloud level, there is only one user role:

Administrator

The administrator manages the list of users and assigns roles in the Operational Decision Manager on Cloud portal. The administrator also creates service credentials for authenticating client applications. Each cloud portal must have at least one administrator, and only an administrator can remove another administrator. Most users do not have this role.

ODM group

The following user roles collaborate on modeling, authoring, governing, deploying, and integrating decision services:

Rule developer

- Works primarily in Rule Designer and the development environment.
- Creates the model of a decision service.
- Uses the Rule Designer component to convert the knowledge from the business domain into IBM® ODM artifacts.
- Makes the initial version of the business rule artifacts, including action rules, decision tables, and ruleflows.
- Runs the decision service locally or in the cloud development environment until the expected results are achieved.
- Publishes the decision service from Rule Designer to Decision Center.
- Collaborates with the release manager and business users in authoring and governance activities.
- Collaborates with the integrator to integrate the decision service into an application.
- Creates deployment configurations in Rule Designer for the development, test, and production environments.
- Can deploy a decision service to the development environment, but cannot deploy to the test or production environment.

Release manager

- Works primarily in the Decision Center Business console component.
- Orchestrates the lifecycle of a decision service, and is responsible for the deployment of a decision service release to production.
- Follows a staged progression from development to production.
- Creates development branches or releases.
- Defines change and validation activities for rule developers, business users, and integrators.
- Assigns ownership for work, reviews, and approvals.
- Can create deployment configurations in the Business console for the development, test, and production environments.

Business user

- Works primarily in the Decision Center Business console component.
- Implements and maintains some or all of the business rule artifacts that are in a decision service.
- Runs functional tests and simulations in the development environment to validate the changes that are made for a release.
- Can deploy a decision service to the test environment to validate changes in a test application. Can also deploy to the development environment.
- Can participate in the review or approval process for other business users or integrators.

Integrator

- Works primarily in the Decision Center Business console component, and uses other development, integration, and test tools.
- Builds the client applications that call a decision service in the development, test, and production environments.
- Sets up performance and reliability test applications.
- Can be involved in the validation activities that are defined by the release manager.
- Can deploy decision services to the development and test environments.
- Can view and use the decision services that are deployed in the production environment.

Permission manager

- Works primarily in the Decision Center Business console.
- Implements security on decision services.
- Creates groups, sets the permissions, adds users to the groups, and sets the groups on decision services.
- Can create deployment configurations in the Business console for the development, test, and production environments.
- Can deploy decision services from any cloud environment.

- Can hold all the ODM roles when, for example, a team is small.

The following table associates the ODM roles with some of the key activities.

The last column uses the following abbreviations:

- Dev: development environment
- Test: test environment
- Prod: production environment

Table 1. Table summarizes the user roles.

ODM role	Creates branches	Edits rules	Edits vocabulary (XOM/BOM)	Deploys to cloud environments (Dev/Test/Prod)
Rule developer	X	X	X	Dev
Release manager	X	X	X	Dev/Test/Prod
Business user	X	X	-	Dev/Test
Integrator	X	X	X	Dev/Test/Prod
Permission manager	X	X	X	Dev/Test/Prod

Parent topic: [Cloud environments and user roles](#)

Related concepts:
[Governance principles](#)

Related information:
[Roles and activities](#)

Express

Operational Decision Manager on Cloud Express® covers the entire lifecycle of a decision service in one environment.

It includes all the components for creating a decision service, developing it collaboratively, and running it for a client application. Unlike the complete Operational Decision Manager on Cloud, Express does not include separate environments that are dedicated to integration testing and production applications.

Tip: You can add a test or production environment to your Express instance. To add both environments, you must purchase the full IBM® Operational Decision Manager on Cloud. The contents of an Express instance can be migrated to the full version.

Introduction

Express is intended for non-mission critical rule applications. These applications have fewer than 500 artifacts and do fewer than a million executions per month. They address simple problems such as input validation and project assignment. These applications do not do the complex, load-heavy decision-making found in, for example, loan processing or order management.

The Express environment is similar to the development environment in the complete Operational Decision Manager on Cloud. The functionality of the components is the same as the functionality of the components in the complete version.

Express includes the user roles and permissions of the full version. However, every user has all the permissions in Express. The roles and permissions are included to facilitate migration to the full version, where they control user access.

Client applications can use the execution server in the Express environment. However, it is better to keep development and production in separate environments. The execution servers in the complete Operational Decision Manager on Cloud meet the specific development, testing, and production demands of rule applications.

Express is fully provisioned by IBM, and includes the following components:

Rule Designer

You use Rule Designer to create decision services. Based on Eclipse, Rule Designer comes with features for designing and assembling decision services. You start in Rule Designer, and typically publish decision services to Decision Center for final development.

Decision Center Business console

You work with colleagues to fully develop and maintain decision services in the Business console. This component includes testing and simulation features to validate rules, and the decision governance framework for managing projects.

Decision Center Enterprise console

Express includes the Enterprise console primarily for file management. It includes development features for advanced users, but development work is usually done in the Business console.

Rule Execution Server console

You test and run rulesets from decision services in Rule Execution Server as part of the development process. When completed, the rulesets can be used by production client applications. Both development and production share this execution server.

Comparison

Express is the small-solution option to the full IBM Operational Decision Manager on Cloud. If your development needs change, you can upgrade easily from Express to the full version with minimal changes to your existing projects.

The following table compares the two offerings by feature:

Table 1. Comparison table

Feature	Express	IBM Operational Decision Manager on Cloud
Decision service creation and development	Yes	Yes
Online collaboration	Yes	Yes
User access security	Yes	Yes
Number of users	Limited (5 concurrent users)	Depends on the conditions of the subscription
Workflow governance	Yes	Yes
System/environments	1	3+
Role-based access permissions	No	Yes

permissions		
Production workload	Limited (1 million executions per month)	Very high (licensed in units of 1 million executions)
Number of rules	Performance reduced above 100	High
Amount of data	Limited (500 artifacts)	High
High availability	No	Yes
Service availability guaranty	None	99.93%

Cloud Services terms

For the Cloud Services terms for Express, see [IBM Operational Decision Manager on Cloud Express](#).

Parent topic: [Overview](#)

Hybrid cloud environments

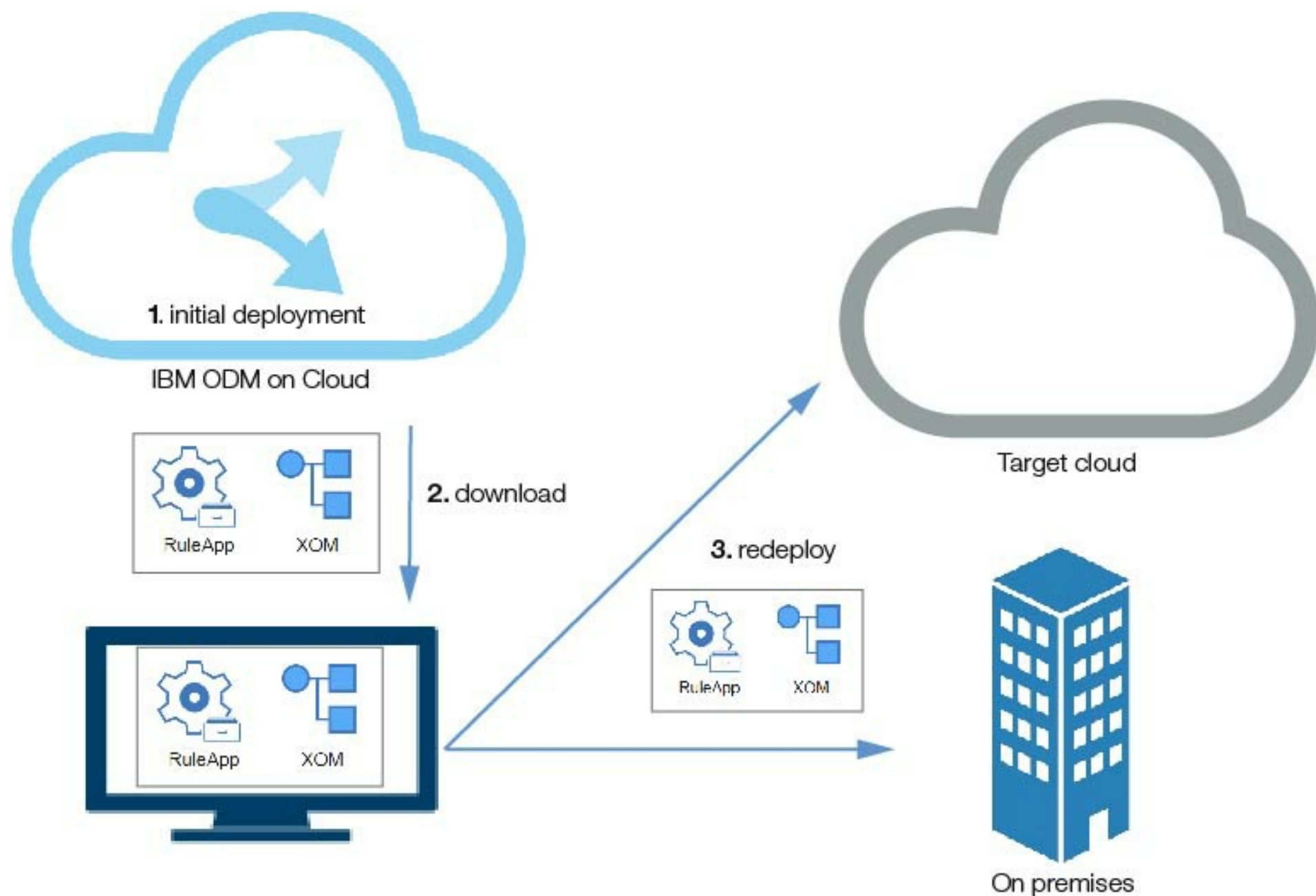
Complement your on-premises or cloud development system with Operational Decision Manager on Cloud.

Hybrid cloud environments bring together complementary on-premises and cloud systems. These hybrid solutions offer a flexible allocation of systems that delivers high levels of availability, performance, and security.

Operational Decision Manager on Cloud supports hybrid cloud environments that bring Operational Decision Manager on Cloud components together with compatible Operational Decision Manager components that are outside the Operational Decision Manager on Cloud environment, for example, on a private cloud or an enterprise on-premises system.

In a typical hybrid scenario, you create a decision service project in Rule Designer on your computer, and publish or deploy it to Operational Decision Manager on Cloud, where it can be further developed and tested. Then, you download and redeploy the executable assets (RuleApp and XOM) to an instance of Decision Server that is running on premises or in a cloud. You can redeploy the asset through a REST API or the Rule Execution Server console.

The following diagram shows the process for developing a decision service in a hybrid cloud environment:



Parent topic: [Overview](#)

Related tasks:
[Deploying to a hybrid cloud environment](#)

Related reference:
[IBM Operational Decision Manager on Cloud Compatibility](#)

First steps

Before you begin, you must gain access to the cloud portal. If you want to create decision services, you must install Rule Designer. When you finish setting up, you can go through the tutorials to learn how to use Operational Decision Manager on Cloud.

[Accessing your cloud portal](#)

Your cloud administrator invites users to your portal. You must initialize your account to work in the portal. The portal provides two types of user authentication, and service credentials for client application authentication.

[Preparing to create decision services](#)

You use Rule Designer to create decision services for Operational Decision Manager on Cloud. You must install the component in an existing version of Eclipse.

Parent topic: [Overview](#)

Accessing your cloud portal

Your cloud administrator invites users to your portal. You must initialize your account to work in the portal. The portal provides two types of user authentication, and service credentials for client application authentication.

[Selecting subscriptions](#)

You can switch between instances of Operational Decision Manager on Cloud without logging out.

[Inviting users to the portal](#)

The cloud administrator invites users to the cloud portal, and assigns them roles.

[Removing accounts from the portal](#)

The cloud administrator can remove accounts from the portal.

[Managing your portal account](#)

You must activate your account to use the cloud portal. You provide a password for your account, and update it regularly.

[User authentication](#)

To access the cloud portal and its components, you use basic or SAML authentication.

[Client application authentication](#)

You use service credentials to authenticate a client application when it calls a decision service that is running in Operational Decision Manager on Cloud.

Parent topic: [First steps](#)

Selecting subscriptions

You can switch between instances of Operational Decision Manager on Cloud without logging out.

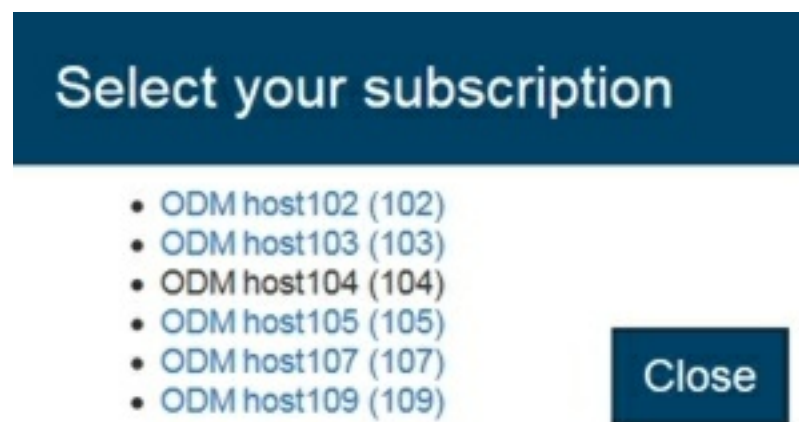
About this task

If your organization has more than one subscription for IBM® Digital Business Automation offerings on the cloud, you can switch between the instances through the cloud portal. This feature can enable you to check the consistency of your assets across the subscriptions. For example, you can check that the latest version of a decision service deployed in Operational Decision Manager on Cloud is referred from a process that you defined in Business Process Manager on Cloud, or that the same version of a decision service is deployed on all your Operational Decision Manager on Cloud subscriptions.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`. The variable `<vhostname>` is the name of the virtual host machine that holds your cloud portal.
2. Click your account name at the top of the home page.
3. Click **Your Subscriptions**. A list shows the instances that you can access. The open subscription is not hyperlinked.

The following image is an example. Your list probably does not contain the subscriptions shown in this example.



4. Click the link of the subscription that you want to open. The cloud portal of the linked subscription opens in your browser.

Parent topic: [Accessing your cloud portal](#)

Inviting users to the portal

The cloud administrator invites users to the cloud portal, and assigns them roles.

About this task

The cloud administrator makes the cloud portal available to other users. The administrator sends an invite to each user, and assigns roles to the users. The roles limit the users' access to cloud environments and components.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`. The variable `<vhostname>` is the name of the virtual host machine that holds your cloud portal.
2. Click **Admin** to open the user management page.
3. Click **Invite New Users** to open the invite dialog.
4. Enter the email addresses of the people you want to invite. Enter the addresses in the following format: `user@example.com`. Separate the addresses by pressing the Comma, Space, or Enter key.
5. Click **Send Invite** to send the invitations. The user management page displays the invited users.

Tip: If the user management page does not display the invited users automatically, refresh the web page.

6. Assign roles to the users by hovering over their names on the user management page and using the drop-down menus to select roles. You can select from the following roles:
 - Cloud role:
 - Administrator
 - Operational Decision Manager component roles:
 - Rule developer
 - Release manager
 - Business user
 - Integrator
 - Permission manager

Results

Your users now have accounts and roles that allow them to work in the cloud portal.

Parent topic: [Accessing your cloud portal](#)

Removing accounts from the portal

The cloud administrator can remove accounts from the portal.

About this task

You can remove user or service accounts from your instance of the cloud portal. You might do this to remove people who no longer use the portal, to clear out temporary accounts, or to replace an existing account.

Procedure

1. On the Operational Decision Manager on Cloud main page, click **Admin**.
2. Hover over the name of a user to be removed until you see a minus sign.
3. Click the minus sign to remove the user.

Results

The user can no longer log in to the portal.

Parent topic: [Accessing your cloud portal](#)

Managing your portal account

You must activate your account to use the cloud portal. You provide a password for your account, and update it regularly.

[Activating your account](#)

You must activate your account to access your cloud portal.

[Updating your profile](#)

Use your user profile to provide your name, and to select your preferred language.

[Resetting your password](#)

You can reset your password through the cloud portal or an expiration notice. You can also reset a forgotten password.

Parent topic: [Accessing your cloud portal](#)

Activating your account

You must activate your account to access your cloud portal.

Before you begin

Check your email for an invitation from `noreply@odm.ibmcloud.com`. The invitation contains a link and instructions to activate your account. If the invitation is not in your main inbox, check your spam mail.

About this task

You create a password when you activate your account. Your email address and the password serve as your login credentials. To activate your account, complete the following steps.

Procedure

1. Click the link in the email invitation. The Operational Decision Manager on Cloud welcome page opens in your default browser.

Tip: To check that you are using a compatible browser, see [System requirements](#).

2. Enter your name and create a password.
3. Click **Activate**.

Results

The main page of the cloud portal opens. It displays the **Applications** tab, which shows the cloud environments that are available to your user role. If you are the cloud administrator, you can also see the **Admin** tab for managing user roles. If you are not the cloud administrator, you must contact your administrator to change your assigned role.

Your password expires in 60 days. You receive a reminder by email to renew the password before it expires. The expiration notice includes a link to change your password. You can also change your password through the main page.

Parent topic: [Managing your portal account](#)

Updating your profile

Use your user profile to provide your name, and to select your preferred language.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`.
2. Click your name, and then click **User profile**. The profile editor opens. It shows your name, default language, and user role:



3. Update your profile. You can change your name and the language.

Important: The profile editor does not show your login name, which is your email address. To change your login name, contact your cloud administrator, who must create another account for you that uses the new login name.

4. Click **Update** to save your changes.

Tip: If your user profile language is set to **Browser language (default)** and you change the language of your browser, you must log back in to the portal for the change to take effect.

Parent topic: [Managing your portal account](#)

Resetting your password

You can reset your password through the cloud portal or an expiration notice. You can also reset a forgotten password.

About this task

These password services are available for user accounts that use basic authentication in the cloud portal. You cannot use them with SAML authentication or service credentials.

Changes to the password policies are applied to existing accounts when the passwords are changed. For example, an existing password becomes subject to a new expiry period when the password is changed or it expires under its original expiry period.

The following restrictions apply to the composition of your password:

- Minimum number of characters: 8
- Minimum number of alphabetic characters: 2 (at least 1 uppercase and 1 lowercase)
- Minimum number of numeric characters: 1
- Minimum number of special characters: 1 (special characters: _-|@.,?V!~#\$%^&*(){}[]=)
- Validity: maximum 90 days
- History of used passwords: 10

Example password: MyNewPa\$sw0rd

Login failure

You are locked out of your cloud instance for 30 minutes if you exceed five failed login attempts. After a password expires, you get three attempts to log in to reset your password. If after three attempts you still cannot log in, you can click **Forgotten your password?** to reset your password.

Procedure

To reset your password, do one of the following procedures:

- In the cloud portal, click your name and **Change password**. A dialog opens to confirm your current password and to enter a new one.
- Reset your password through the expiration notice that is sent to you when your password is about to expire. The notice contains a link that takes you to a dialog to change your password.

For a forgotten password, follow these steps:

1. In the login window of Operational Decision Manager on Cloud, enter your user name and click **Continue**. Your user name is your email address.
2. Click **Forgot your password?** A message directs you to check your email for information on how to reset your password.
3. Click the link in the email message to change your password. The link remains valid for one hour.
4. Enter a new password, and click **Set Password**. The main portal page opens.

Parent topic: [Managing your portal account](#)

User authentication

To access the cloud portal and its components, you use basic or SAML authentication.

When you provision your cloud portal, you select a type of user authentication:

Basic authentication

The user logs in to the cloud portal directly. The user has a user account in the portal, and logs in with an email address and a password that meets the security requirements of the portal.

SAML authentication

The user signs in to the cloud portal through a non-portal login service. Security Assertion Markup Language (SAML) is an XML standard for exchanging single sign-on information. SAML delegates authentication to a third party. When a customer subscribes to Operational Decision Manager on Cloud through SAML, authentication is typically delegated to the customer's organization. The organization stores and manages its own user credentials. There is no duplication of credentials between IBM® and the customer.

Important:

If you log in to the cloud portal by using SAML authentication, you cannot run decision services or download archive files for a hybrid cloud environment. You need basic authentication for these tasks.

Using SAML

To use SAML, you must submit metadata for your login service. IBM uses the metadata to delegate authentication to your service. SAML support is configured per portal instance for the users of the instance, and for their email domain.

When the cloud administrator first logs in to the cloud portal:

1. The administrator enters an email address.
2. Operational Decision Manager on Cloud determines the type of authentication:
 - Basic authentication: The account administrator is prompted for a password.
 - SAML authentication: The administrator is redirected to another login page.
3. If the authentication is successful, the account administrator is logged in to Operational Decision Manager on Cloud.

Authorizing users

The cloud administrator invites users to Operational Decision Manager on Cloud, and assigns them roles. For SAML authentication, Operational Decision Manager on Cloud stores only the user roles and not the user passwords, which stay in users' login service.

Authenticating client applications

You must use basic authentication to connect client applications to the cloud portal. You cannot use SAML to authenticate a client application. Service credentials are specifically designed for authenticating client applications (see [Service credentials for client applications](#)).

Parent topic: [Accessing your cloud portal](#)

Client application authentication

You use service credentials to authenticate a client application when it calls a decision service that is running in Operational Decision Manager on Cloud.

Operational Decision Manager on Cloud provides two options for authentication:

- **User accounts:** These accounts provide credentials that users enter to sign in to the cloud portal and its components. While client applications can also login credentials of user accounts, this practice is not recommended because user passwords and any applications that use them must be updated regularly.
- **Service credentials:** These accounts provide credentials for authenticating calling applications. They include an ID that is associated with a function, and a highly secure password. Because of its high security, the password does not require regular updates and is well suited for authenticating client applications.

Because service credentials are linked to functions, and not to real users, they do not have to be changed when a user leaves a project or a company. SAML users, who log in to the cloud portal through the login systems of their organizations (see [Authentication](#)), must use service credentials for applications that need to authenticate with the cloud.

Tip: If you are using the trial version of Operational Decision Manager on Cloud, you can use the login credentials of your user account in a client application to call rules from the cloud portal. You can also request service credentials for this operation.

Service credential basics

Service credentials comply with basic authentication. They use a functional ID for the user name, and a long, machine-generated password to foil brute force attacks by hackers, for example:

- Functional ID: `custval.fid@t100`
- Password: `8xcFS90S60EGcvj0coppPDH9+/iBx9aDrjhD8zwn`

When you create a set of service credentials, you enter an alias (for example, `custval`). The cloud service generates a functional ID from the alias by adding an extension that stands for functional ID (`fid`) and your instance of the cloud portal (`t100`).

Tip: For the functional ID, use an alias that associates the service credentials with a decision service or a client application. For example, you can use the alias `custval` for a decision service that validates customers.

Important points:

- Service credentials are only used to authenticate calling applications. They have no cloud role, and cannot be used by a user or Rule Designer to log in to the cloud portal. If you try to log in by using service credentials, you get an error message: A functional user is not allowed to perform this operation.
- Only cloud portal administrators can create service credentials. The administrators give the service credentials to the developers of the client applications.
- You cannot update an existing set of service credentials. You must replace the set with another set.
- You cannot use the same alias in more than one set of service credentials. You must delete the first set before you can create another set that uses the same alias.
- When you delete a set of service credentials, the cloud portal no longer recognizes it. The client application needs a new set to connect to the cloud portal.

[Getting a set of service credentials](#)

You create service credentials for a client application.

[Deleting a set of service credentials](#)

You delete service credentials so that a client application can no longer use them to connect to the cloud.

Parent topic: [Accessing your cloud portal](#)

Getting a set of service credentials

You create service credentials for a client application.

About this task

This procedure shows you how to create service credentials, and share them with the developers of a client application.

Important: For the trial version, you can get service credentials by sending an email message to supportodmoncloud@us.ibm.com. Include your name, the name of your organization, the URL of your cloud instance, and your login name (email address).

Procedure

1. Log in to the Operational Decision Manager on Cloud portal as an administrator. You must be a cloud administrator to do this procedure.
2. Click **Admin > Service Credentials**. A table displays the service credentials in your instance of Operational Decision Manager on Cloud.
3. Click the **Create Credentials** button to open the Create service credentials dialog.
4. Enter a functional ID alias, for example, custval. You do not have to enter a description, but it can help later in determining the purpose of the credentials.

Functional ID alias (required)
custval
Description (optional)
Functional ID for client application

Tip: For the functional ID, use an alias that associates the service credentials with a decision service or a client application. For example, you can use the alias custval for a decision service that validates customers.

5. Click the **Create** button. The Your service credentials dialog opens. It shows the functional ID and password that your instance of Operational Decision Manager on Cloud can now recognize when a client application uses the credentials to call a decision service that is running in Operational Decision Manager on Cloud:

Functional ID: custval.fid@t100
Password: 8xcFS9OS60EGcvj0coppPDH9+/iBx9aDrjhD8zwn
Description: Functional ID for client application

6. Click the **Copy to Clipboard** button. After a moment, the following message appears when the operation is successful:

The credentials were copied to the clipboard.

Important: You must copy the credentials before you close the dialog. When you close the dialog, the table of service credentials shows the functional ID, but the password is hidden. You cannot reopen the dialog to obtain the same password. If you do not save the credentials, you must generate a new set of service credentials.

7. Paste the credentials into a text file. The credentials look similar to the following example:

```
Functional ID: custval.fid@t100
Password: 8xcFS9OS60EGcvj0coppPDH9+/iBx9aDrjhD8zwn
Description: Functional ID for client application
```

Results

You can now give the service credentials to the developers of the client application. The developers must integrate the credentials into the application to enable it to call a decision service.

Parent topic: [Client application authentication](#)

Deleting a set of service credentials


You delete service credentials so that a client application can no longer use them to connect to the cloud.

About this task

You learn how to delete a set of service credentials. When you no longer need a set of service credentials or you want to delete it for security reasons, you can remove it from the cloud portal. Also, when you want to change the password in a set of service credentials, you must first delete the set, and then generate a new set with the same functional ID.

Important: After you remove a set of service credentials, the cloud no longer recognizes the credentials. The client application can no longer use the credentials to authenticate with the cloud when the application calls a decision service.

Procedure

1. Log in to the Operational Decision Manager on Cloud portal as an administrator. You must be a cloud administrator to do this procedure.
2. Click **Admin** > **Service Credentials** to open the table of service credentials.
3. Hover over the row that contains the set of service credentials, and click the **Delete** button  to open the delete dialog:



4. Ensure that the functional ID in the dialog box matches the ID in the set of service credentials that you want to delete.
5. Click the **Delete** button.

Results

When the table refreshes, it no longer displays the deleted set of service credentials.

Parent topic: [Client application authentication](#)

Preparing to create decision services

You use Rule Designer to create decision services for Operational Decision Manager on Cloud. You must install the component in an existing version of Eclipse.

Installing Rule Designer

You add Rule Designer to an existing Eclipse environment.

Connecting to a proxy server

You can configure Rule Designer to connect to your Operational Decision Manager on Cloud portal through an HTTP proxy server.

Inbound and outbound connectivity

Operational Decision Manager on Cloud supports authorized inbound and outbound connectivity. Except for Rule Designer, connectivity is based on API.

Parent topic: [First steps](#)

Installing Rule Designer

You add Rule Designer to an existing Eclipse environment.

Before you begin

You use Rule Designer to create decision services. To install Rule Designer, you must have the appropriate JDK and Eclipse for your operating environment already installed on your computer. Consult the Rule Designer download dialogue box in the development environment of your instance of Operational Decision Manager on Cloud. It provides links to resources for downloading Java, Eclipse, and sample projects.

Important: Each release comes with its own version of Rule Designer. You must reinstall the rule editor when you upgrade to the latest release. Starting with Operational Decision Manager on Cloud 2019.06, you can find the release number in the bottom right corner of the portal homepage.

Check the Java™ version of your Eclipse installation, and change it if necessary. In Eclipse, click **Help > Installation Details**. On the Configuration tab, check the version of Java in the `java.version` system property. You must use the SDK listed in the Rule Designer download dialogue box.

To change your Eclipse to the correct version of Java:

1. Close Eclipse.
2. Open `eclipse/eclipse.ini`, and add or edit the following two lines at the beginning of the file:

```
-vm
<SDK_InstallDir>/bin
```

<SDK_InstallDir> is the location of the Java instance on your computer.

3. Restart Eclipse.

Make sure Eclipse and Java both use the same bit value, either 32 or 64 bits.

Tip: If you get an error while installing Rule Designer, install the Eclipse Modeling Tools.

About this task

In your Eclipse installation, you configure a repository for Operational Decision Manager on Cloud and use it to install Rule Designer.

Procedure

1. In Eclipse, click **Help > Install New Software**.
2. In the Install dialog, click **Add**.
3. In the Add Repository dialog, enter the repository details that are available from the Operational Decision Manager on Cloud portal.

To obtain the repository details:

- a. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`.
- b. On the **Applications** tab, in the Rule Designer box, click **Download**.
- c. From the Rule Designer download window, copy the Eclipse update site URL and paste it in the **Location** field of the Add Repository dialog of your Eclipse. Click **OK**.

4. Select the package **IBM ODM on Cloud**, and click **Next**.
5. Click **Next** again.
6. Accept the license terms.
7. Click **Finish**.
8. Click **OK** to accept unsigned content.
9. Click **Yes** to restart Eclipse. Make sure your workspace is selected, and then click **OK**.
10. In the wizard that opens, enter the URL for your Operational Decision Manager on Cloud portal in the format `https://<vhostname>.bpm.ibmcloud.com/`. You can test the connection to the cloud portal by clicking **Connect** and providing your Operational Decision Manager on Cloud credentials.
11. Click **Finish**. The wizard configures the connections to your Decision Center and Rule Execution Server instances on the cloud.

Tip: You can run the connection configuration wizard anytime from the Rule Designer menu bar by clicking **File > Configure IBM ODM on Cloud connection**. For example, you might want to connect as a different user to Operational Decision Manager on Cloud or to a different cloud instance.

12. When Rule Designer opens, switch to the Rule perspective if it does not open by default.

Parent topic: [Preparing to create decision services](#)

Connecting to a proxy server

You can configure Rule Designer to connect to your Operational Decision Manager on Cloud portal through an HTTP proxy server.

About this task

You enable an HTTP proxy server for Rule Designer by adding the server to your operating system internet options and to your Eclipse preferences for Rule Designer.

Procedure

1. Specify the proxy server in your operating system's internet options. For instance, on Windows 7:
 - a. Open Internet Explorer.
 - b. Open **Tools > Internet options > Connections**. For your type of network, do as follows:
 - Dial-up or private network: Select your configuration and click **Settings**.
 - Local area network: Click **LAN**.
 - c. Select **Use a proxy server**.
 - d. Enter your proxy server address in Address, and your proxy port in Port.
 - e. Save your changes.
2. Add the proxy server to your Eclipse preferences.
 - a. Start Rule Designer.
 - b. Click **Preferences > General > Network Connection**.
 - c. Set Active Provider to **Manual**.
 - d. Edit the HTTP and HTTPS proxy entries so that they correspond to your proxy server.
 - e. Save your changes.

Results

Rule Designer is now configured to connect to your proxy server.

Parent topic: [Preparing to create decision services](#)

Inbound and outbound connectivity

Operational Decision Manager on Cloud supports authorized inbound and outbound connectivity. Except for Rule Designer, connectivity is based on API.

The the types of connectivity use the following security protocols:

- Basic authentication: cloud user name and password
- Service credentials: a highly secure, multicharacter ID for connecting client applications
- SAML security: deferred login to your enterprise security system

Theses sections cover the authorized connectivity for Operational Decision Manager on Cloud.

Inbound connectivity

Rule Designer

- Use a version of Rule Designer that is entitled to connect to Operational Decision Manager on Cloud. Use basic authentication or SAML security.
- Rule Designer can synchronize projects to Decision Center in Operational Decision Manager on Cloud.
- Rule Designer can deploy to Rule Execution Server in Operational Decision Manager on Cloud.

Deploying to Rule Execution Server

- Deploy from Decision Center in an on-premises Operational Decision Manager installation to Rule Execution Server in Operational Decision Manager on Cloud.
- Deploy from Decision Center in another cloud service to Rule Execution Server in Operational Decision Manager on Cloud.

Invocation of decision services

The invocation of decision services deployed in Operational Decision Manager on Cloud requires basic authentication or service credentials.

Invocation of Decision Center API

The invocation of the Decision Center API deployed in Operational Decision Manager on Cloud requires basic authentication or service credentials.

IP whitelisting

For connectivity from known IP addresses and to avoid unauthorized connections, you can use IP whitelisting on a per-tenant basis. For a client tenant, you provide an IP address range or IP addresses that are authorized to connect to the tenant.

Outbound connectivity

Deploying to another Rule Execution Server

- From Decision Center in Operational Decision Manager on Cloud, you can deploy to an external Rule Execution Server by providing the required authentication credentials for the target server.
- From the Decision Center in Operational Decision Manager on Cloud, you can deploy to any Rule Execution Server outside the cloud by providing the required basic authentication.
- In the Operational Decision Manager eXecutable Object Model (XOM) code, you can use transport layer security to create connections to any external server and invoke any methods. You must ensure that the server is accessible, and that it has the required credentials. The server should have a CA-signed certificate to avoid certificate management. If the server's certificate is self-signed, your XOM code needs to set up a truststore with that certificate.

Parent topic: [Preparing to create decision services](#)

Usage reports

You can view the use of your artifacts and decisions through a web-based dashboard. It displays data in graphs that enable you to readily understand your rule application traffic.

Dashboard

Directly monitor the traffic of your artifacts and decisions.

Monthly reports

Sent reports provide data on the use of artifacts and decisions.

Dashboard

Directly monitor your usage statistics on Operational Decision Manager on Cloud.

Operational Decision Manager on Cloud includes a dashboard for monitoring artifacts in Decision Center and decisions in your Rule Execution Servers. The dashboard is accessible to cloud administrators, and displays usage data for a chosen period of time.

The dashboard is included in new instances of Operational Decision Manager on Cloud, and becomes available in existing instances when they are upgraded to the latest release. Until an existing instance is upgraded, you continue to receive monthly usage reports for it (see [Monthly reports](#)).

Metrics

The dashboard displays information on two types of metrics:

Decisions

This metric shows the number of calls to a decision service in Rule Execution Server during the selected time period. For example, if an application calls a decision service 10,000 times per hour in one day, the metric shows 240,000 calls for that day.

Artifacts

This metric shows the number of unique versioned artifacts in Decision Center. The artifacts are counted only once across all branches, releases and activities, in a given instance. If you use duplicates of artifacts in different instances, the artifacts are counted separately for each instance. The metering service does not see the duplication.

Types of artifacts:

- Business artifacts (BAL rules, decision tables, decision trees, technical rules, functions)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- Business object models (BOMs)
- Vocabulary
- Business-to-Exchange (B2X) artifacts
- Resources
- Simulation artifacts (metrics, key performance indicators, report formats, input data, and simulations)
- Test suites
- Decision validation service (DVS) artifacts (scenario suites for testing and simulation)
- Business Action Language (BAL) templates
- Smart views

Time window

You set the period of time that is monitored by the dashboard. You select the dates to start and end a monitoring period. The dashboard collects the data for the metrics from Operational Decision Manager on Cloud.

Using the dashboard

You access the dashboard through the Operational Decision Manager on Cloud homepage.

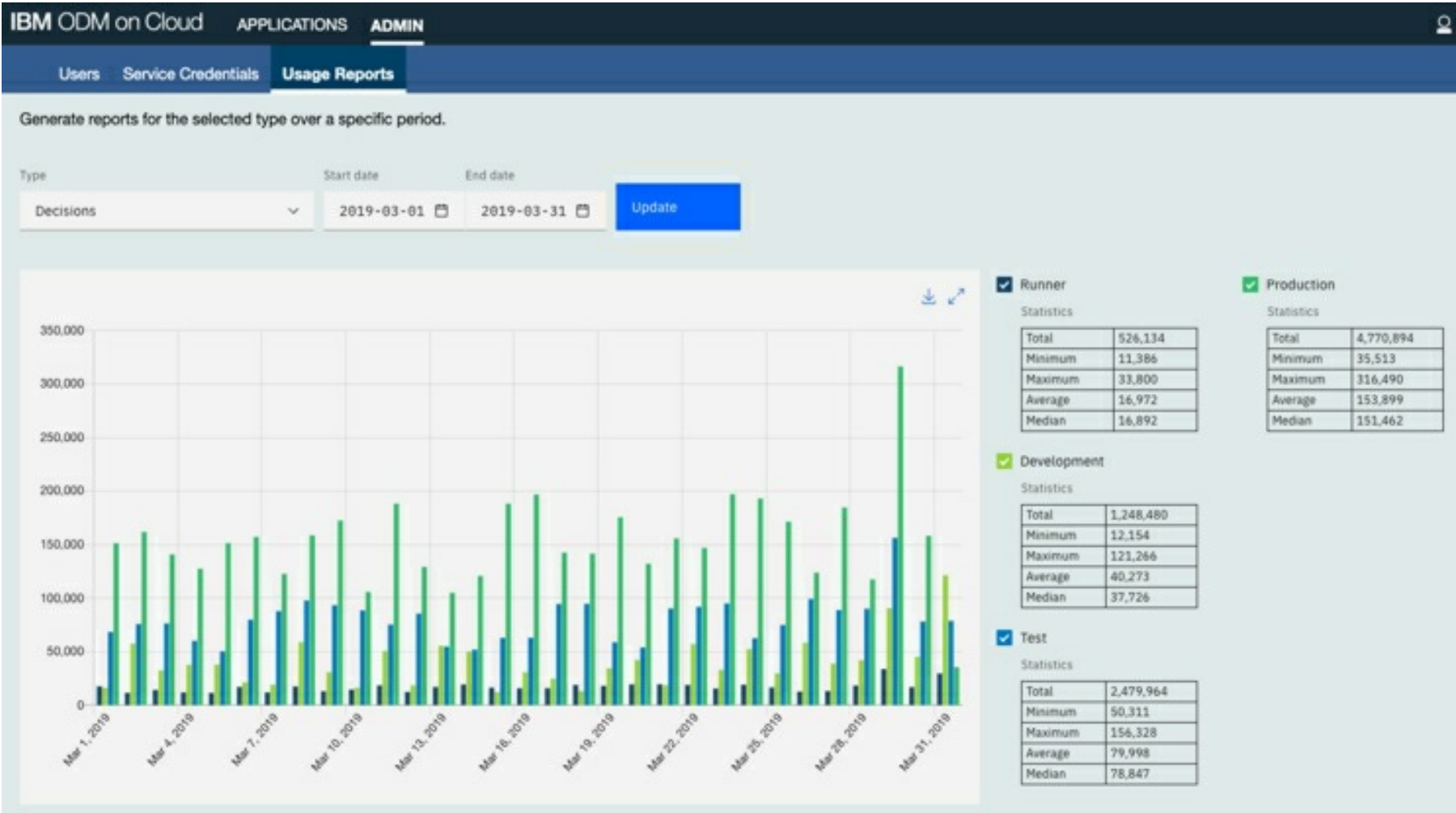
1. Open your instance of Operational Decision Manager on Cloud as a cloud administrator. The homepage displays the Applications window with the different environments and their components.
2. Click **Admin** to display the cloud administrative features.
3. Click **Usage Reports** to open the dashboard. You are presented with an empty report page.
4. Under Type, click **Select report type** to open its drop-down menu, and select a metric, for example, **Decisions**:



5. Use the **Start date** and **End date** calendars to define the reporting period similarly to the following image:

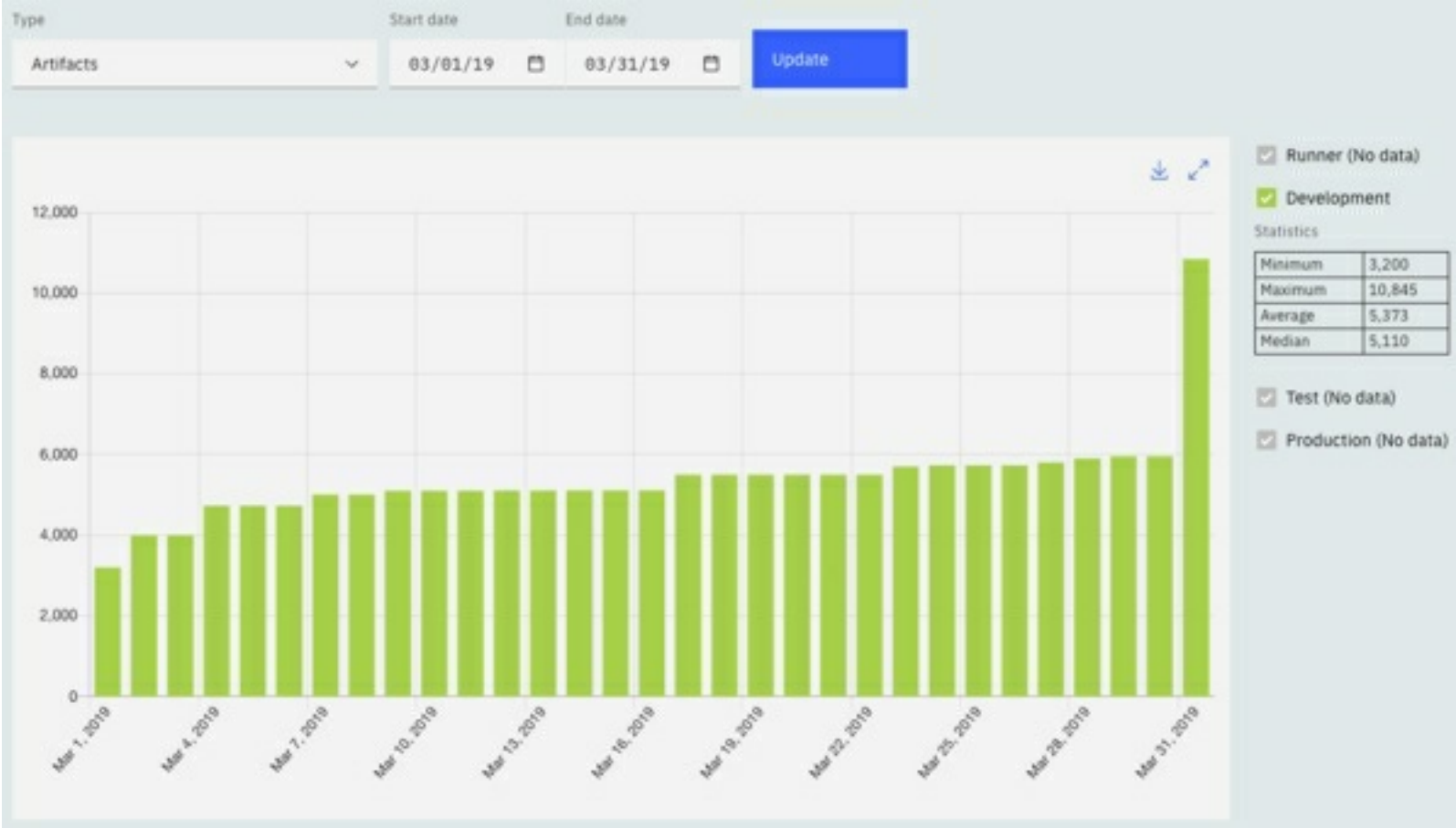




6. Click **Show** to produce a report. Similar to the following image, the dashboard displays the results in a bar chart, and includes data from the decision runner (Runner) and the Rule Execution Server environments:



The overall statistics are displayed to the right of the dashboard. By default, all the environments are shown. To hide data for a metric, click the box next to its name to deselect it.

7. Change Type to **Artifacts** and click **Update**. Similar to the following image, the dashboard displays data on the artifacts in Decision Center:



8. To see the chart in full screen, click the **Expand** button . The dashboard enlarges the graph, and hides the display parameters and overall statistics.
9. To download the report, click the **Download** button . The download dialog of your browser opens. Use the dialog to download a Comma Separated Values (CSV) file of the report to your computer.

Tip: A CSV file uses a simple format for storing tabular data. You can open it by using a program that stores data in tables.

10. To refresh the data in the chart, click **Update**. The dashboard adds any new data for the metrics and the chosen time period.

Attention: The dashboard does not retain data when you navigate away from it in your web browser. You must reset its display parameters when you return to it.

Monthly reports

Sent reports provide data on the use of artifacts and decisions.

New instances of Operational Decision Manager on Cloud come with the usage dashboard (see [Dashboard](#)). For existing instances, IBM continues to send monthly usage reports until the instances are upgraded to the latest release of Operational Decision Manager on Cloud.

The report shows the traffic of rule applications for the previous month. Typically, the report goes to the administrators on record for the instance, but it can be directed to a different recipient. Like the dashboard, the reports provides usage data on artifacts and decisions.

The following image is an example of a usage report. The example is for the full version of Operational Decision Manager on Cloud, and shows usage data for the artifacts, decision runner, and the development (dev), production (prod), and test (test) environments. The dates are formatted day-month-year, and the total shown for artifacts is the largest number for the month. The last column shows the sum total of decisions for each day in the month, and the total at the base of each column of decisions is the sum total for the month for a specific environment.

	A	B	C	D	E	F	G
1	Day	Artifacts	Decision Runner Decisions	Decision Server (dev) Decisions	Decision Server (prod) Decisions	Decision Server (test) Decisions	Total Decisions
2	01/03/2019	3000	11323	37388	108857	78125	238693
3	02/03/2019	3200	17525	16250	151407	68414	256796
4	03/03/2019	4000	11469	57444	161828	75679	310420
5	04/03/2019	4000	14230	32517	140828	76327	267902
6	05/03/2019	4729	11902	37726	127523	60104	241984
7	06/03/2019	4729	11386	37880	151462	50311	255768
8	07/03/2019	4729	17030	21391	157219	79866	280235
9	08/03/2019	5007	11933	19207	122763	87888	246798
10	09/03/2019	5007	17397	58924	158692	97851	337871
11	10/03/2019	5100	12967	30821	172595	93454	314937
12	11/03/2019	5100	14558	16271	105765	88584	230278
13	12/03/2019	5100	18498	50821	188264	75240	337923
14	13/03/2019	5105	12355	18584	129202	85543	250789
15	14/03/2019	5105	16892	55501	104865	54764	237127
16	15/03/2019	5110	19374	50148	120793	51868	247293
17	16/03/2019	5110	16394	12154	188265	62900	284823
18	17/03/2019	5110	15876	30764	196803	62892	311445
19	18/03/2019	5500	15914	24850	142611	94598	283473
20	19/03/2019	5500	18883	13106	141546	94827	273862
21	20/03/2019	5500	18034	34542	175612	58763	292451
22	21/03/2019	5500	19474	42084	132061	53864	252983
23	22/03/2019	5500	19555	18694	155874	90227	289850
24	23/03/2019	5500	18808	57160	146960	91914	320342
25	24/03/2019	5700	15566	32998	197179	95137	346580
26	25/03/2019	5728	19074	52504	193058	62577	332941
27	26/03/2019	5728	16573	29739	171433	74952	298425
28	27/03/2019	5728	12684	58401	123897	99170	299880
29	28/03/2019	5800	13101	38886	184685	88878	331350
30	29/03/2019	5900	18338	42007	117456	90033	273734
31	30/03/2019	5955	16900	45280	158245	78164	304544
32	31/03/2019	5955	16848	47984	187587	77845	336219
33	Total	5955	490861	1122026	4715335	2400759	8887716

Tutorials

The Operational Decision Manager on Cloud tutorials cover the key features in the development components. They demonstrate how to create decision services and deploy them to a production environment.

Note: Additional projects can be found in the [ODMDEV Repositories](#).

Introduction to Operational Decision Manager on Cloud

Get a head start in using the cloud portal.

[Getting started in Operational Decision Manager on Cloud](#)

This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.

Rule Designer tutorials

You use Rule Designer to create decision services and publish them to the Operational Decision Manager on Cloud portal for further development and deployment.

[Creating a decision service in Rule Designer](#)

This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.

[Debugging a decision service in Rule Designer](#)

This tutorial provides a methodology for debugging a decision service in Rule Designer.

Decision Center Business console tutorials

You use the Business console to develop and maintain decision services. You can create new rule artifacts, update existing ones, and deploy decision services to Rule Execution Server.

[Getting started with decision modeling](#)

This tutorial shows the basics of creating a decision model service, in which you model and author a decision service, all from the Business console.

[Creating a ruleflow in the Business console](#)

This tutorial shows you how to create a ruleflow in a decision service in the Business console.

[Creating a simulation in the Business console](#)

This tutorial shows you how to create a simulation in a decision service in the Business console.

[Merging branches in the Business console](#)

This tutorial shows you how to merge changes between branches in a decision service in the Business console.

Miniloan Service

Most of the tutorials are based on the Miniloan Service decision service. Setting it up is essential to using the tutorials.

[Preparing and removing the tutorial project](#)

You use the Miniloan Service decision service in most of the tutorials. You import it into Decision Center, and when you no longer need it, you delete it.

[Next >](#)

Getting started in Operational Decision Manager on Cloud

This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.

Operational Decision Manager on Cloud is a complete system for creating and using rule applications. It enables you to express your organization's decisions and policies in business rules, and to make the rules available to your production applications. Users can work together in the portal to develop and maintain the rules in decision services.

In this tutorial, you work in the cloud portal. You update the rules in a decision service, and then test the decision service. After the decision service passes the test, you deploy the rules from the decision service to different execution servers. Finally, you look at a sample application that uses the rules that were deployed from the decision service.

Important:

- You can do this tutorial in Operational Decision Manager on Cloud Express. However, only the parts of the tutorial on the development environment apply to Express. The parts that refer to the test and production environments only work in the full version of Operational Decision Manager on Cloud.
- You do not create a decision service in this tutorial. To learn how to create a decision service, see [Creating a decision service in Rule Designer](#).
- This tutorial does not cover the decision governance framework, the system for coordinating and tracking work on projects in Decision Center. For more information about the framework, see [Governance principles](#) and the video [Using the Decision Governance Framework in the Decision Center Business Console](#).

Learning objectives

You do the following tasks:

- Import a decision service.
- Create a branch in a decision service.
- Edit a decision table and create an action rule.
- Test a decision service.
- Deploy a RuleApp from a decision service.
- Explore the contents of a RuleApp in an execution server console.
- Call a ruleset from a client application by using REST.

Time required

1 hour

[Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work on the Miniloan Service decision service, which is used by a fictitious loan company to determine whether borrowers are eligible for loans. When the company updates its loan policies, you reflect the changes in the business rules in the decision service.

The rules in the decision service are kept in two folders:

- Eligibility: Contains the rules that determine whether a loan can be approved.
- Validation: Contains the rules that make preliminary checks to determine whether data is rejected immediately.

You update the decision service in the cloud portal, which has three different work environments. You modify and test the decision service in the development environment, and deploy the rules from the decision service to execution servers in the development, test, and production environments. Finally, you make the rules available to a sample client application, which uses the rules to process data.

Note: This tutorial is also available on GitHub. You can find it at [Getting started in IBM Operational Decision Manager on Cloud](#).

Audience

The tutorial is for users who want to work on decision services in the cloud portal of Operational Decision Manager on Cloud.

Prerequisites

You need the following items to do this tutorial:

- Access to a cloud portal instance of Operational Decision Manager on Cloud.
- Access to ODMDEV on GitHub: <https://github.com/ODMDev>

Lessons in this tutorial

[Task 1: Preparing to do the tutorial](#)

You download the project files for the tutorial, and gain access to the Operational Decision Manager on Cloud portal. Then, you import the Miniloan Service decision service, and create a branch in the service for your changes.

[Task 2: Touring the decision service](#)

You look at some of the artifacts in the Miniloan Service decision service.

[Task 3: Creating and editing rules](#)

You update a decision table, and create an action rule to implement policy changes.

[Task 4: Testing and deploying the decision service](#)

You test the changes to the decision service, and then deploy the rules in decision service to an execution server.

[Task 5: Testing in the execution server](#)

You test the rules from the decision service in the execution server. When you finish, you deploy the rules to the test environment for integration testing, and then deploy them to the production environment for use with client applications.

[Task 6: Calling the rules from a client application](#)

You have deployed the rules from the decision service. Now, you can set up a client application to call the rules from the execution server in the development environment.

Task 1: Preparing to do the tutorial

You download the project files for the tutorial, and gain access to the Operational Decision Manager on Cloud portal. Then, you import the Miniloan Service decision service, and create a branch in the service for your changes.

About this task

You update the Miniloan Service decision service in the Decision Center Business console in the development environment in the cloud portal. The decision service holds the business rules and related artifacts for developing and deploying a rule application that is used by a client application. The Business console contains editors for creating and editing rules, and tools for testing decision services and deploying their rules to the execution servers in the development, test, and production environments in the cloud portal.

To start the tutorial, you must first download a group of projects from GitHub. Step 1 explains how to download the files.

To use the Business console, you must have access to the cloud portal. Step 2 explains how to obtain access to the cloud portal if you do not have it.

When you have access to the cloud portal, you can work on the Miniloan Service decision service. The decision service must be imported into Decision Center through the Business console. You look for the decision service in the Business console in step 3, and if you need to import it, you follow the instructions in step 4.

Finally, when Miniloan Service is in the Business console, you create a branch for your work in the decision service. Your branch isolates your changes from the other branches in the decision service.

Step 1: Downloading the tutorial files

You download the source files for the tutorial from Github and install them on your computer. These files are required to do the tutorial.

Procedure

1. Go to the GitHub repository for the tutorial: [Getting started in IBM® Operational Decision Manager on Cloud](#)
2. Download the contents of the repository to a directory on your computer. The files are downloaded in a compressed file that is named `odm-cloud-getting-started-master.zip`.
3. Open the downloaded file `<InstallDir>/odm-cloud-getting-started-master.zip`. `InstallDir` is used throughout the tutorial to refer to your directory for the GitHub files.
4. Extract the contents of the compressed file to the `InstallDir` directory. You get a new directory, named `odm-cloud-getting-started-master`, which contains the following items:
 - `miniloan.zip`: This compressed file contains the Miniloan Service decision service and the `miniloan-xom` Java object model. You import this file into Decision Center through the Business console.
 - `Miniloan Service`: This decision service contains the rule artifacts for approving loans. The decision service was created in Rule Designer.
 - `miniloan-xom`: This Java object model describes the classes that are used in the decision service.
 - `miniloan-server`: This client application uses REST to call the rules that are deployed from the decision services.
 - Documentation for the tutorial

Step 2: Gaining access to the cloud portal

Each instance of the Operational Decision Manager on Cloud portal has an administrator, who invites people to the portal and assigns user roles.

Before you begin

You need a user role that gives you access to the cloud portal and the Business console. You can do this tutorial with the business user role or greater. Check with your administrator. If you already have access to the cloud portal, you can skip this step and go to step 3.

About this task

The administrator gives people access to the portal by sending them invitations from the portal. The invitations enable people to connect to the portal for the first time.

The administrator also assigns user roles, which determine which components can be accessed by the users. For more information, see [Cloud environments and user roles](#).

For this tutorial, it is recommended that you use one of the following user roles:

- **Business user**: Works primarily in the Business console. Can update and create rules, but cannot deploy them.

- Release manager: Can create new releases, update and test rules, and deploy decision services.

To gain access to the cloud, follow the instructions for your type of portal:

Client portal

If your organization has an instance of the cloud portal, you must contact the administrator of the portal to gain access. For more information, see [Accessing your cloud portal](#).

Sample portal

If you want to use the trial version of the cloud portal, you must contact IBM Support to be invited to the portal. For more information, see [IBM ODM on Cloud Free Trial](#).

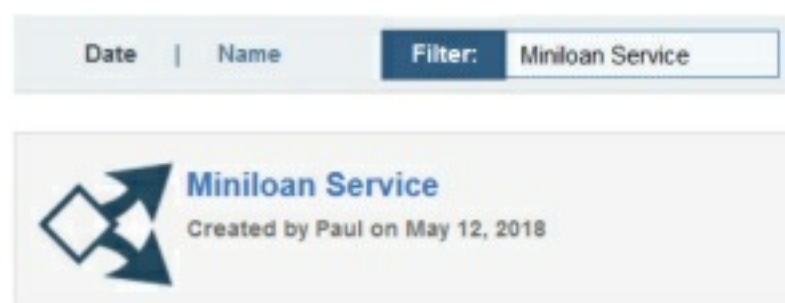
Step 3: Checking the availability of the decision service

You determine whether the Miniloan Service decision service is already in the Business console of the development environment in the cloud portal.

Procedure

1. Sign in to the cloud portal. The portal displays the resources that are available to your cloud user role.
2. Launch the Decision Center Business console in the development environment. The console opens to its home page.
3. Click **LIBRARY** to open the list of decision services that are currently in Decision Center.
4. Enter Miniloan Service in the filter to look for the decision service.

Decision Services




If the Miniloan Service decision service is in the library, skip step 4 and go to step 5.

Step 4: Importing the decision service

You import the decision service into Decision Center.

Procedure

1. In the Business console **LIBRARY** tab, click the **Import Decision Service** button .
2. Click **Choose**, and navigate to the miniloan.zip file in the directory <InstallDir>/odm-cloud-getting-started-master, which you created in step 1.
3. Select the file, and click **Open**.

Note: Do not select **Use Decision Governance Framework**. You do not use this feature in this tutorial.

4. Click **Import**. The decision service is added to the library in the Business console.


Step 5: Creating a branch

You create a branch to isolate your changes to the decision service.

About this task

The main branch in the decision service contains the rule artifacts in their original form. To update the decision service, you create a branch based on the main branch, and work in the new branch.

Procedure

1. Click Miniloan Service to open the decision service.
2. Open the **Branches** tab, and expand the main branch. Look at the names of the existing branches. When you name your branch, do not reuse the name of an existing branch.
3. Click the **New Branch** button .
4. Enter a name for your branch, for example, My Branch. Remember not to reuse the name of an existing branch. You can personalize your branch by using the initials of your name.

You can also enter a goal for your branch, for example:

Make policy changes in getting started tutorial.

5. Select **main** as the parent branch, and then click **Create**. The Business console duplicates the artifacts of the main branch in a new branch that you can modify.

Note: In the rest of the tutorial, My Branch is used as the name of the branch.

What to do next

In the next task, you explore the contents of the decision service before making changes.

[< Previous](#) | [Next >](#)

Task 2: Touring the decision service

You look at some of the artifacts in the Miniloan Service decision service.

About this task

Companies set policies for their operations. These policies can address such matters as pricing, eligibility, and product configuration. You use Operational Decision Manager on Cloud to develop business rules that express these policies. The rules state the conditions for applying the policies, and the resulting actions. Client applications then call the rules to implement the policies.

In this task, you look at some of the artifacts in your branch of the Miniloan Service decision service.

Step 1: Opening your branch

You open your branch and display its decision artifacts.

Procedure

1. Click **My Branch** in the Branches tab of the Miniloan Service decision service.
2. Click **Decision Artifacts** to see the rule artifacts. By default, only the rules are displayed.
3. Hover over **Types(1/5) X** and click the X to display all the artifacts in the decision service. The labels indicate the types of artifacts, and some show the number of artifacts:



Step 2: Exploring the ruleflow

You open the ruleflow to determine how it works.

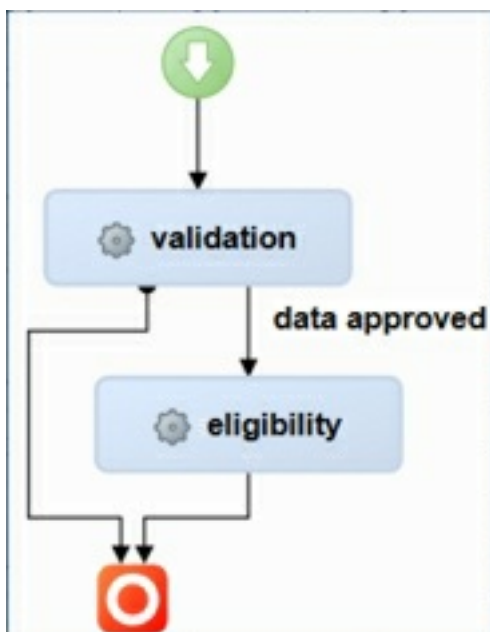
Procedure

1. Hover over **miniloan** in the list of decision artifacts.
2. Click the ruleflow to open it in the preview window.

Results

A decision service keeps its decision points in projects and folders. The Miniloan Service decision service contains one project, and two folders for different decision points. The rule in the validation folder determines whether incoming loan data is valid, and the rules in the eligibility folder apply various conditions for eligibility.

The ruleflow states the sequence for applying the rules:



The ruleflow starts by running the rule in the validation folder. If the data from a loan request passes the rule, the ruleflow directs the data to the eligibility rules. However, if the validation decision point does not approve the data, the ruleflow skips the eligibility decision point, and the decision service stops processing the loan request. For more information, see [Editing ruleflows](#).

Step 3: Exploring an action rule

You open an action rule to determine how it works.

About this task

There are two types of rules in the decision service:

- Action rules: An action rule states a policy in a natural language. It uses conditions and actions to check the data and apply a response.
- Decision tables: A decision table holds several action rules that share the same policy statement but use different variables. The variables are listed in condition and action columns, and each row of the table forms a complete action rule.

You start by looking at an action rule.

Procedure

1. In the list of decision artifacts, click **eligibility** to open the folder in the preview window.
2. Click **minimum credit score** to open the action rule in the preview window.

Results

An action rule associates one or more actions with one or more conditions. When the conditions are met, the actions are triggered.

The rule statement in the minimum credit score rule looks as follows:

```
if
  the credit score of 'the borrower' is less than 200
then
  add "Credit score below 200" to the messages of 'the loan' ;
  reject 'the loan' ;
```

As data for the borrower enters the decision service, the action rule tests the credit score variable. If the variable is less than 200, the prescribed action is to reject the loan and work stops on the loan. If the variable is equal to or more than 200, the loan is further processed. For more information, see [Action rules](#).

Step 4: Exploring a decision table

You open a decision table to determine how it works.

Procedure

1. In the list of decision artifacts, click **eligibility** to open the folder in the preview window.
2. Click **repayment and score** to open the decision table in the preview window.

Results

A decision table provides a way to view and manage sets of similar rules. Each row in a decision table represents a complete rule. The rules in the table share the same rule statement, but the condition and action values vary. The first two columns (debt to income and credit score) contain the condition variables, and the next two columns (message and rejected) contain the action variables.

The repayment and score decision table looks as follows:



	debt to income		credit score		message	loan approve
	min	max	min	max		
1	0 %	30 %	0	200	debt-to-income too high compared to credit score	<input type="checkbox"/>
2	0 %	30 %	200	800		<input checked="" type="checkbox"/>
3	30 %	45 %	0	400	debt-to-income too high compared to credit score	<input type="checkbox"/>
4	30 %	45 %	400	800		<input checked="" type="checkbox"/>
5	45 %	50 %	0	600	debt-to-income too high compared to credit score	<input type="checkbox"/>
6	45 %	50 %	600	800		<input checked="" type="checkbox"/>
7	≥ 50 %		0	800	debt-to-income too high compared to credit score	<input type="checkbox"/>

When the decision table checks a loan request, it rejects the request if the borrower must pay back an annual amount that is too high for the borrower's credit score. Otherwise, the table does not reject the loan. For more information, see [Decision tables](#).

Step 5: Running a test suite

You add a test suite and run it to verify that the decision service works correctly before changes are made.

Procedure

1. Click **Tests > Test Suites** to open the subtab.
2. Click the **New Test Suite** button . The dialog for creating a new test suite opens.
3. Click **Choose**, and navigate to <InstallDir>/odm-cloud-getting-started-master/Miniloan Service. InstallDir is your directory for the extracted files from the GitHub repository, as shown in task1, step2.
4. Select miniloan-test.xlsx, and then click **Open**.
5. Make sure that the selected server is **Testing And Simulation Runtime**, which is the default server for running tests and simulations.
6. Click the **Save and Run** button .
7. Enter the information that is shown in the following image:

Create New Version

A new version of this element will be created.

You can add a comment to associate with this version which can be viewed in the timeline.

Testing the summer release.

Create New Version

8. Click **Create New Version**, and then click **OK** in the Run Test Suite dialog. The console runs the test and switches to the Reports tab. A new report indicates a successful test by displaying a check mark in the Status column:

<input type="checkbox"/>	Name	Operation	Status	Run Date	Test Suite
<input checked="" type="checkbox"/>	Report - 2018-05-23_02-24-56-632	Miniloan ServiceOperation		5/23/18, 4:24 PM	Test Suite (latest)

9. Click the report under Recent Report. The report opens, and displays details that include the following summary:



10. Close the report.

What to do next

In the next task, you search for the decision table, update it, and create an action rule.

[< Previous](#) | [Next >](#)

Task 3: Creating and editing rules

You update a decision table, and create an action rule to implement policy changes.

About this task

The loan company wants the following changes to be implemented in the decision service:

- Increase the minimum credit score from 200 to 300 for all new loan requests.
- Reject loans that are shorter than six months.

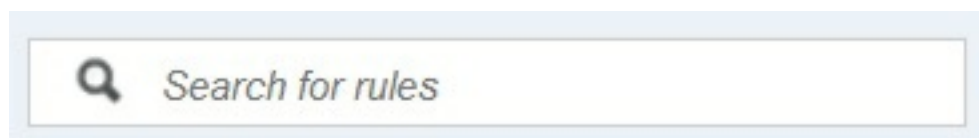
To implement the changes, you use the search function to locate the table that applies the minimum credit score. After you change the score, you create an action rule to implement the minimum loan duration.

Step 1: Searching for rules

To make the first change, you use the search function to find all the rules that change the credit score.

Procedure

1. In your branch, click inside the search field to be able to enter a search string:



2. Type score, and press Enter. The results show all the action rules and decision tables in which the word score occurs. You look at the rules, and conclude that you must edit the repayment and score decision table.
3. Click **repayment and score** in the list of search results. The decision table opens in the preview window.

Step 2: Modifying the decision table

You update the decision table. For more information, see [Decision table editor](#).

Procedure


1. Click **Edit**. The decision table editor opens.
2. Double-click 200 in row 1 of the **credit score** column, and replace it with 300. Do the same in row 2.
3. Click **Save**.
4. Enter the following comment, and then click **Create New Version**.

Changed credit score to 300

Step 3: Creating an action rule

You introduce the second policy change by creating an action rule. For more information, see [Building rules using the Intellirule editor](#).

Procedure

1. In **My Branch**, click **Decision Artifacts**, and then click **validation** to open the folder.
2. Click the **Create** button , and click **New Rule**.
3. Enter check duration as the name of the rule, and click **Create**.
4. Enter the following rule statement in the rule editor. If your web browser supports it, you can copy the rule to the rule editor. Otherwise, type in the rule or build it with the completion menu.

```
if
  the duration of 'the loan' is less than 6
then
  add "The duration of the loan is too short" to the messages of 'the loan';
  reject 'the loan';
```

The rule checks the duration of a loan. If the duration is shorter than six months, the rule adds a rejection message to the loan and rejects the loan.

5. Click **Save**.
6. Enter the following message, and then click **Create New Version**.

Added rule for summer policy

What to do next

In the next task, you test and deploy the decision service.

[< Previous](#) | [Next >](#)

Task 4: Testing and deploying the decision service

You test the changes to the decision service, and then deploy the rules in decision service to an execution server.


About this task

After you update the decision service, you want to make sure that it still works correctly. In this task, you run the decision service against a test scenario file. When the decision service produces the expected results, you deploy its rules to an execution server.

Step 1: Running a test

You run a test to see whether the decision service generates expected results. For more information, see [Testing sets of rules in the Business console](#).

Procedure

1. In **My Branch**, click **Tests** > **Test Suites** to open the subtab for running tests.
2. Hover over the Test Suite name that you added in task 2 and click the **Run** button .
3. Click the generated report when the run finishes and the status shows a check mark. The report shows a 100% success rate. The results produced by the decision service match the expected results.



4. Click **Close** to close the report.


Step 2: Deploying the decision service rules


You deploy the rules in the decision service to the execution server in the development environment to further test the service. For more information, see [Deploying from the Business console](#)

Procedure

1. Click the **Deployments** tab to see its contents. The tab contains the deployment configurations that are available to your user role.

The cloud portal has three standard environments: development, test, and production. You can only see the environments that are available to your user role. For example, the business user has access to only the development and test environments, and can only see the deployment configurations for these two environments.

2. Click the **Development** deployment configuration to open it. Go through the tabs in the configuration:
 - **General**: Provides an overview of the deployment configuration, including the name, type, RuleApp name, and base version number.
 - **Operations**: Lists the decision operations to deploy. Here, **Miniloan Service Operation** is selected for deployment. Decision operations define how the rules are used in specific rulesets for deployment. Hover over the decision operation name to see its content.
 - **Targets**: Lists where the rules can be deployed. The environment for this deployment is the development environment.
 - **Ruleset Properties**: Defines the versioning policy for each deployment. You use the default settings.
 - **Groups**: Lets the administrator choose which groups can deploy using this deployment configuration.
3. Click the **Deploy** button  in the upper right corner of the window. A dialog box opens with a summary of the deployment configuration. The Target list gives you the option of deploying to the sample server or creating a RuleApp archive.
4. Leave **Server Development Environment** selected as the target, and click **Deploy**. A message opens with the status of the deployment.
5. Click **OK** in the deployment status message. The Reports subtab opens in the Deployments tab.

6. Click the report for the deployment. The report opens and shows a summary of the deployment. It shows the target server, the configuration name, the ruleset with the rules, the deployment time, the version of the ruleset, and a link to the deployment snapshot, which can be redeployed.
7. Click the **Close** button  in the upper right corner of the window to close the report.

What to do next

In the next task, you test the rules the execution server, and then deploy the rules to the test and production environments.

[< Previous](#) | [Next >](#)

Task 5: Testing in the execution server

You test the rules from the decision service in the execution server. When you finish, you deploy the rules to the test environment for integration testing, and then deploy them to the production environment for use with client applications.

About this task

When you deployed the rules from the decision service, they were sent as a *ruleset* in a RuleApp to Rule Execution Server in the development environment. Now, you test the ruleset in the execution server. When the ruleset passes the test, you deploy it to the test environment for integration testing, and then to the production environment for use with client applications. For more information, see [Deploying decision services](#).

Note: The information in the Rule Execution Server consoles in your cloud portal might vary from the information shown in this tutorial. For example, your consoles might display more RuleApps and different version numbers.

Step 1: Viewing the deployed RuleApp

You find the RuleApp from the decision service in the execution server, and look at its contents.

Procedure

1. Return to the cloud portal, and launch the Rule Execution Server (Development) console in the development environment.
2. Click the **Explorer** tab in the console.
3. In the Navigator, expand RuleApps, and then /Miniloan/1.0. Your RuleApp contains the ruleset Miniloan_ServiceRuleset:



The RuleApp might contain more than one ruleset. You can find yours by looking at the creation dates.

4. Click **Miniloan_ServiceRuleset** to open the ruleset in the Ruleset View. The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties:



Step 2: Testing the ruleset by using the REST API

You test the ruleset to determine whether it executes correctly. You use the REST API to run the test on manually entered data.

Procedure

1. Click **Retrieve HTDS Description File** in the Ruleset View. The file retrieval page opens.
2. Select **REST** as the service protocol type, and **OpenAPI-JSON** as the format.
3. Click **Test**. The Hosted Transparent Decision Service (HTDS) opens.
4. Enter the following data as the run request:

```
{
  "loan": {
    "amount": 500000,
    "duration": 240,
```

```

    "yearlyInterestRate": 0.05
  },
  "__DecisionID__": "Test",
  "borrower": {
    "name": "Joe",
    "creditScore": 600,
    "yearlyIncome": 80000
  }
}

```

Your run request should look as follows:

```

1 {
2   "loan": {
3     "amount": 500000,
4     "duration": 240,
5     "yearlyInterestRate": 0.05
6   },
7   "__DecisionID__": "Test",
8   "borrower": {
9     "name": "Joe",
10    "creditScore": 600,
11    "yearlyIncome": 80000
12  }
13 }

```

5. Click **Execute Request**. You get the following server response:

```

{
  "__DecisionID__": "Test",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}

```

The rule application produces the expected results. The loan is rejected, and a message explains that the dept-to-income ratio is too big.

6. Close the HTDS window.

Step 3: Deploying to the test environment


Satisfied that the rules work as expected, you return to the Business console to deploy the rules to the test environment. The integrator runs integration tests on the rules in the test environment. These tests determine the stability and reliability of the rules under load before they are promoted to the production environment for use with client applications.

Procedure

Note:

If you are a business user, you cannot view the test and production environments, and cannot do steps 3 and 4.

1. Return to the Business console, and open your branch in the Miniloan Service decision service.
2. Open the **Deployment** tab. In Configurations, click **Test**.
3. Open the **Targets** tab. The Test configuration deploys the rules in the decision service to Rule Execution Server in the test environment.

4. Click the **Deploy** button . The deployment dialog opens. It shows the Server Test Environment as the target of the deployment.
5. Click **Deploy**, and click **OK** in the deployment status box. The list of reports opens in the Deployment tab. The new report shows that the rules have been deployed successfully:

<input type="checkbox"/>	Name	Status	Run By	Run Date	Configuration
<input type="checkbox"/>	 Report 2018-04-12_08-27-20-137		userodmcloud@mail.com	4/12/18, 10:27 AM	Development
<input type="checkbox"/>	 Report 2018-04-12_10-00-14-588		userodmcloud@mail.com	4/12/18, 12:00 PM	Test

Report 2018-04-12_10-00-14-588

6. Return to the cloud portal, and launch the Rule Execution Server (Test) console in the test environment.
7. Open the **Explorer** tab in the console.
8. In the Navigator, expand RuleApps, and then /Miniloan/1.0. Your RuleApp contains the ruleset Miniloan_ServiceRuleset:



9. Click **Miniloan_ServiceRuleset** to open the ruleset in the Ruleset View. The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties:

 /Miniloan/1.0/Miniloan_ServiceRuleset/1.0	
Name	Miniloan_ServiceRuleset
Version	1.0
Creation Date	Apr 3, 2018, 4:22:29 PM GMT+2
Display Name	Miniloan Service Operation
Description	
Rule engine	Decision Engine - 1.40.6
Status	 enabled
Debug	 disabled

The integrator can now do nonfunctional and integration testing on the rules before they are deployed to the production environment.

Tip: The test shown in step 2 works in Rule Execution Server in the test environment.

Step 4: Deploying to the production environment

When the integrator finishes testing the rules from the decision service, the rules can be deployed to the production environment. Client applications then call the rules from this environment to process data.

About this task

You must have the release manager role to deploy the rules to the production environment. If you have this role, you can see the deployment configuration for the production environment in the Deployments tab. You can use it as you did the deployment configuration for the test environment. However, there is no need to deploy the Miniloan RuleApp to the production environment. The deployment configuration for the production environment is included to provide a complete list of configurations for release managers.

For more information on calling the rules from this environment, see [Integrating decision services](#).

What to do next

In the next task, you set up a client application to call the rules from the execution server in the development environment.

[< Previous](#)

Task 6: Calling the rules from a client application

You have deployed the rules from the decision service. Now, you can set up a client application to call the rules from the execution server in the development environment.

Before you begin

This task describes how an application uses the rules from the decision service. It also explains how to set up and run the application. Setting up the application is optional because it requires some technical knowledge. To set up the application, you must be familiar with Maven, and Java application servers such as IBM WebSphere Liberty and Apache Tomcat.

About this task

When you deployed the rules to Rule Execution Server, you sent a ruleset in a RuleApp to the execution server. The Miniloan client application can now call the ruleset from the server.

When the application opens, it displays default information:

The screenshot shows a web application interface with two main sections: 'Borrower' and 'Loan'. The 'Borrower' section has three input fields: 'Name' (Joe), 'Yearly Income' (80000), and 'Credit Score' (600). The 'Loan' section has three input fields: 'Amount' (500000), 'Duration' (240), and 'Yearly Interest Rate' (0.05). Below these sections is an 'Execution' section with a 'Validate Loan' button.

In the Execution section, the parameters are set to call the ruleset from Rule Execution Server in the development environment:

The screenshot shows the 'Execution' section of the application. It contains four input fields: 'Server' (vhost1010.stage.bpm.ibmcloud.com), 'User' (paulreleasemanager@gmail.com), 'Ruleset' (Miniloan/Miniloan_ServiceRuleset), and 'Password' (masked with asterisks). There is also a 'Show trace' checkbox and three radio buttons for 'Dev', 'Test', and 'Prod'.

The application uses the REST API with JSON for input and output. When the **Validate Loan** button is clicked, the application processes the default data and produces the following results:

The screenshot shows a yellow banner with the word 'Rejected'. Below it is a JSON response:

```
{
  "__DecisionID__": "dc22f36e-77db-4d08-b692-b9ff565845fc0",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}
```

Optional: Installing the client application on a custom server

The following steps are for any integrator or other user who wants to set up a web application to use the rules from the decision service.

Before you begin

You need the following items to do the installation:

- WebSphere Liberty application server from [Download the latest stable WebSphere Liberty runtime](#)
- Source files from the odm-cloud-getting-started GitHub repository
- Maven from [Apache Maven Project](#)

Step 1: Creating the server

You create a server to run the Miniloan Service sample application.

Procedure

1. Download the Liberty application server.
2. Decompress the Liberty file to a directory on your computer. The directory is referred to as <WLP_Home> in the following steps.
3. From an administrator command prompt, create your server by running the following command:

```
<WLP_HOME>\bin>server create testGettingStarted
```

The command creates the folder <WLP_HOME>\usr\servers\testGettingStarted, which contains the server configuration and steps.

4. To create the profile, launch the server by using the following command:

```
<WLP_HOME>\bin>server start testGettingStarted
```

You can access the started server at <http://localhost:9080>.

5. To stop the server, use the following command:

```
<WLP_HOME>\bin>server stop testGettingStarted
```

Step 2: Setting up the sample application

You use Maven to build the sample web application, and then you add the application to the application server.

Procedure

1. Go to the <InstallDir>/odm-cloud-getting-started-master/miniloan-server directory. InstallDir is your directory for the extracted files from the GitHub repository that is listed in the prerequisites.
2. In a command prompt, call `mvn clean install`. The command builds the miniloan-webapp.war file in the target directory.
3. Copy <InstallDir>/odm-cloud-getting-started-master/miniloan-server/target/zip to <WLP_HOME>/usr/servers/testGettingStarted/apps. The folder contains the .war file for the sample application: miniloan-webapp.war
4. Declare the sample application by adding the following line in <WLP_HOME>/usr/servers/testGettingStarted/server.xml:

```
<!-- Miniloan application -->
<webApplication id="miniloan-webapp"
location="miniloan-webapp.war"
name="miniloan-webapp"/>
</application>
```

5. Save the file.

Step 3: Running the sample application

You run the sample application.

Procedure

1. Launch the server by using the following command:

```
<WLP_HOME>\bin>server start testGettingStarted
```

A message tells you when the build is complete. The build might take a few minutes to finish.

2. Enter the following URL in your web browser to open the application:

```
http://localhost:9080/miniloan-webapp/
```

The Miniloan application opens in your browser, and displays the default values.

- 3. Click **Execution** to configure the connection with your decision service. Enter the information specific to your cloud portal, for example:

Borrower

Name:

Joe

Yearly Income:

80000

Credit Score:

600

Loan

Amount:

500000

Duration:

240

Yearly Interest Rate

0.05

Execution

Server:

vhost1010.stage.bpm.ibmcloud.com

User

paulreleasemanager@gmail.com

☐ Show trace

Ruleset:

Miniloan/Miniloan_ServiceRuleset

Password

☒ Dev ☐ Test ☐ Prod

✓ Validate Loan

- 4. Click **Validate Loan** to process the default values. The application processes the request and returns the following message:

Rejected

```
{
  "__DecisionID__": "dc22f36e-77db-4d08-b692-b9ff565845fc0",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}
```

You can find additional information by selecting **Show trace** in the Execution section and validating the data again. The additional information shows that one rule, `eligibility.minimum_income`, is executed.

- 5. When you finish using the sample application, close the application in your browser and stop the Liberty application server by using the following command as shown in step 1:

```
<WLP_HOME>\bin>server stop testGettingStarted
```

Results

You have completed the getting started tutorial for Operational Decision Manager on Cloud. The tutorial showed you the artifacts a decision service branch, how to update and test rules, and how to deploy the rules from the decision service to an execution server. You also looked at using the rules with a client application.

[< Previous](#)

[< Previous](#) | [Next >](#)

Creating a decision service in Rule Designer

This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.

Learning objectives

You do the following tasks:

- Design a decision service.
- Orchestrate rules.
- Create an action rule.
- Update a decision table.
- Test and debug rules.
- Deploy a decision service to Rule Execution Server.
- Publish a decision service to Decision Center.
- Delete a decision service from Decision Center.

The tutorial does not cover collaborative development and the decision governance framework in the Decision Center Business console. To simplify the tutorial, you work in an ungoverned branch of the decision service in the console.

Best practices

This tutorial includes the following best practices for creating a decision service in Rule Designer:

- Start with an empty workspace when you create a decision service. [Example...](#)
- Follow the Rule Project Map to create a decision service. [Example...](#)
- Begin by defining a sequence in which to run the rules. [Example...](#)

Time required

1 hour

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

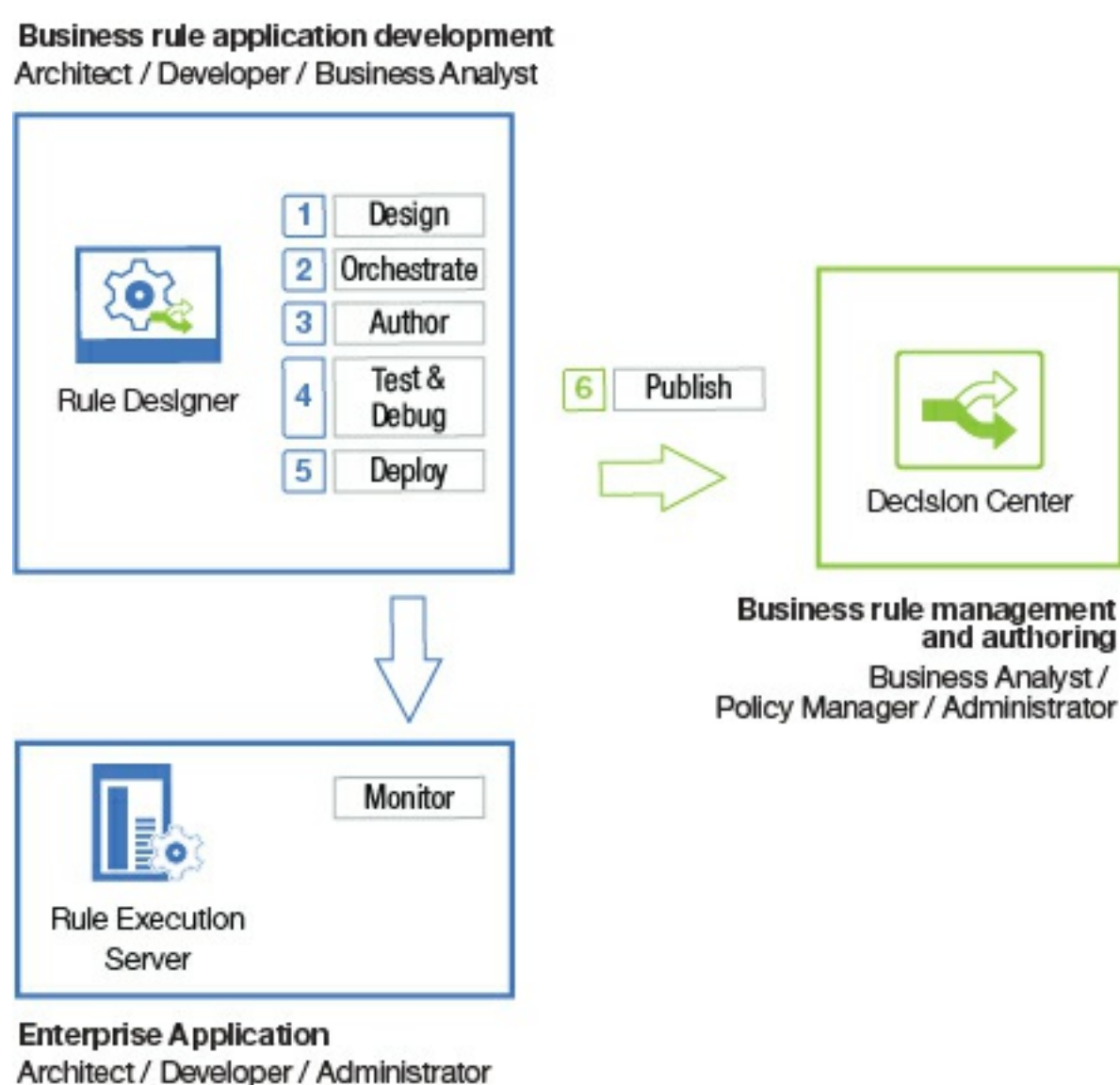
Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a decision service that determines whether a borrower is eligible for a loan. The decision service applies rule conditions for the amount of the loan, the annual income of the borrower, and the duration of the loan.

The tutorial shows you how to create and debug the decision service in Rule Designer, deploy it to Rule Execution Server, and publish it to Decision Center. Rule Designer is the editing environment for creating decision services and authoring rules, Rule Execution Server monitors and runs rules, and Decision Center is the cloud environment for collaborative rule authoring, management, and validation.

The following diagram shows the workflow for making a decision service. It goes from design to publication, and shows the users of the different components. The numbers in the diagram correspond to the tasks in the tutorial:



Audience

The tutorial introduces you to decision service development. The tutorial is primarily for release managers, and rule developers.

Prerequisites

You need the following items to do the tutorial:

- Operational Decision Manager on Cloud portal access: You must be invited to the portal and activate your account (see [Accessing your cloud portal](#)).
- Rule Designer: You must download this component from the portal and install it on your computer (see [Preparing to develop rule applications](#)). The size of the download file is about 800 MB.
- miniloanservice-projects.zip: Along with Rule Designer, download the sample project from the portal to the same directory, and extract its contents to the directory. The tutorial later refers to the directory as *<InstallDir>/miniloanservice-projects*. The size of the download file is about 13 KB.
- Rule Execution Server: You use this component in the Operational Decision Manager on Cloud portal.
- Decision Center Business and Enterprise consoles: You use these components in the Operational Decision Manager on Cloud portal.

Additional information:

- The product overview on the main goals of the product, its modules, and rule-oriented terminology (see [Introduction to Operational Decision Manager on Cloud](#))
- A knowledge of Java
- A knowledge of the Eclipse workspaces, perspectives, and views

Lessons in this tutorial

Task 1: Designing the main rule project for the decision service

You create a vocabulary so that business users can write business rules in a natural language. You use Rule Designer to create the vocabulary from a Java object model.

Task 2: Orchestrating rule running

You create a ruleflow to set the sequence in which the rules run.

Task 3: Authoring business rules

You write an action rule and import other rules for your rule project.

Task 4: Testing and debugging a ruleset

You run the ruleset by entering input data in a decision operation configuration, and then test and debug your ruleset.

Task 5: Deploying your decision service

You deploy your decision service to Rule Execution Server.

Task 6: Publishing to Decision Center

You publish your decision service to Decision Center in the Operational Decision Manager on Cloud portal, where you view your files in the Business console.

[< Previous](#) | [Next >](#)

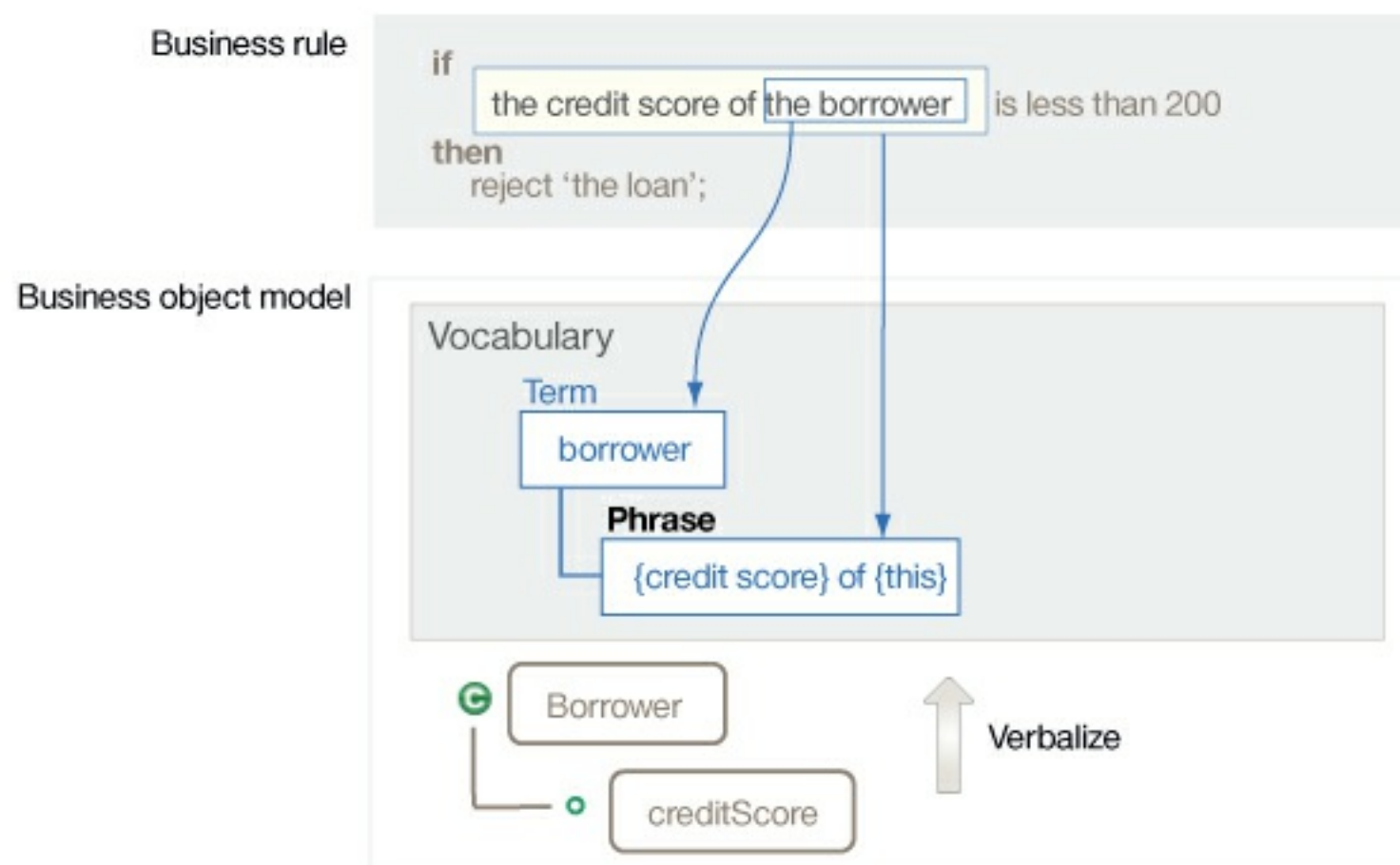
Task 1: Designing the main rule project for the decision service

You create a vocabulary so that business users can write business rules in a natural language. You use Rule Designer to create the vocabulary from a Java object model.

About this task

As a developer, you want to create a business rule vocabulary that business users can use to write and edit rules. The vocabulary must consist of terms that are familiar to the business users. The process of creating a vocabulary is called *verbalization*.

In this task, you create a business object model (BOM) that is based on an object model that is defined in a Java™ project. The following diagram shows how the classes and members of the BOM relate to the vocabulary that is familiar to the business users.



Step 1: Starting Rule Designer and importing the Java object model

Rule Designer is an Eclipse development environment for business rule applications. You can develop both rule projects and Java projects in this environment.

Before you begin

The file from which you copy rules for this tutorial is available in only American English (en_US). To use Rule Designer in American English, add the following lines to `<InstallDir>/eclipse/eclipse.ini` before `-vmargs`, where `<InstallDir>` is the installation directory for your instance of Rule Designer:

```
-nl  
en_US
```

Procedure

1. Start Rule Designer (see [Preparing to develop rule applications](#)).
2. Select a workspace. When the Workspace Launcher window opens, it shows your default workspace. If the workspace is not empty, switch to a workspace that is empty. If you want to create a workspace, change the name of the workspace in the Workspace Launcher, and click **OK**.

Tip: If you do not see the Workspace Launcher window when you start Rule Designer, you can change the workspace by clicking **File > Switch Workspace**.

3. Optional: If the connection wizard opens, connect to the cloud portal:
 - a. Enter the URL for your Operational Decision Manager on Cloud in the following format:
`https://<vhostname>.bpm.ibmcloud.com/`
 - b. Click **Connect** and enter your cloud credentials.
 - c. Click **Finish**.
4. Close the Eclipse Welcome page to see the Rule perspective. If the Rule perspective is not open, click **Window > Perspective > Open Perspective > Other**, select **Rule**, and click **Open**.

5. Click **File > Import**. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
6. Choose **Select root directory**, browse to `<InstallDir>/odm-cloud-getting-started-master`, and click **OK**. InstallDir is your directory for the files downloaded from GitHub (see [Importing and deleting the tutorial project](#)).
7. Make sure that `miniloan-xom` is selected, and deselect `Miniloan Service`.
8. Click **Finish** to import the project.

Results

The Rule perspective contains various views that you learn about in this tutorial, such as the Rule Explorer and the Decision Service Map.

The Rule Explorer shows `miniloan-xom`, which contains the execution object model (XOM) that you use in the tutorial. The XOM is the model against which you run rules. It references the application objects and data, and is the base implementation of the BOM (see [Overview: BOM and execution object model \(XOM\)](#)). The `miniloan` package holds the `Borrower` and `Loan` Java classes.

Notice the Rule Project Map tab. When you work on a decision service, the Rule Project Map tab shows the Decision Service Map, which contains the different steps for making and deploying a decision service. You can use the Decision Service Map as a guide to the creation process for the decision service. If the Decision Service Map is not displayed, click **Window > Show View > Rule Project Map** to open it.

The following image shows the Rule Project Map in the tab bar.



Step 2: Creating a rule project for a decision service

In Rule Designer, you store the business logic of your application in a decision service. A decision service can consist of a main rule project and optional standard rule projects to split the logic of your business application into different parts. Each rule project can contain rule artifacts, a BOM, a vocabulary, and a reference to the XOM. In a rule project, you can manage, build, and debug the items that comprise the business logic of your application. In this tutorial, you create only a main rule project to simplify the logic of the application.

Procedure

1. Ensure that you are in the Rule perspective, and the Rule Project Map is in the bottom tab bar.
2. In the Decision Service Map in the Rule Project Map tab, click **Create main rule project**.
3. Ensure that **Main Rule Project** is selected under Decision Service Rule Projects, and click **Next**.
4. In the **Project name** field, type `my decision service`. Use the default location.

The name `my decision service` is used throughout the tutorial. You can use a different name to avoid conflicts with other projects when you publish the decision service. For example, you can prefix `my decision service` with your initials.

5. Click **Finish**.

The Rule Explorer displays `my decision service`.

The rule project contains empty folders. You use the `rules` and `bom` folders to store your rules and BOM.

Step 3: Importing the XOM into your rule project

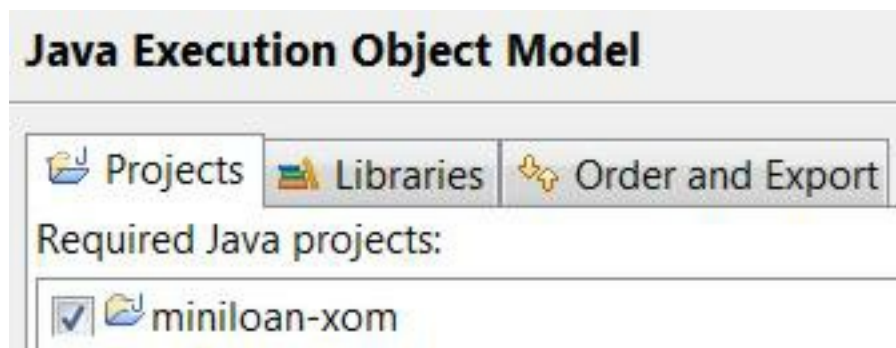
You import the XOM from the `Miniloan` Java project.

Procedure

1. In the Rule Explorer, click `my decision service`.

The Decision Service Map displays the steps to follow to design your rule project.

2. In the Decision Service Map, click **Import XOM**.
3. In the Import XOM dialog, ensure that **Java execution object model** is selected, and click **OK**.
4. In the **Projects** tab, select `miniloan-xom` and click **Apply and Close**:



Step 4: Creating the business object model

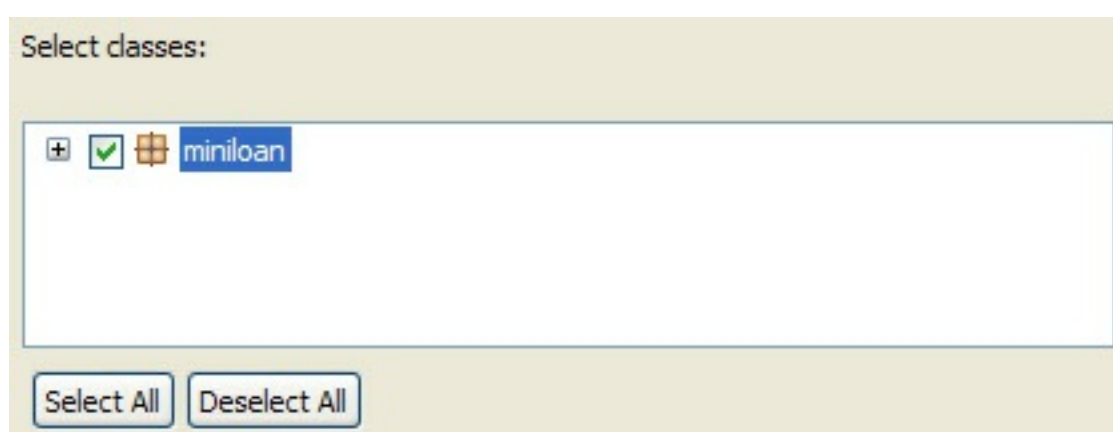
Before you can create and edit business rules, you must create a BOM. You can create a BOM manually or automatically by parsing your XOM. You use Rule Designer to parse the Java classes in the XOM, and create the BOM from their methods and properties. Then, you can write rules that use the verbalized terms that are contained in the BOM.

Procedure

1. In the Decision Service Map, click **Create BOM**.

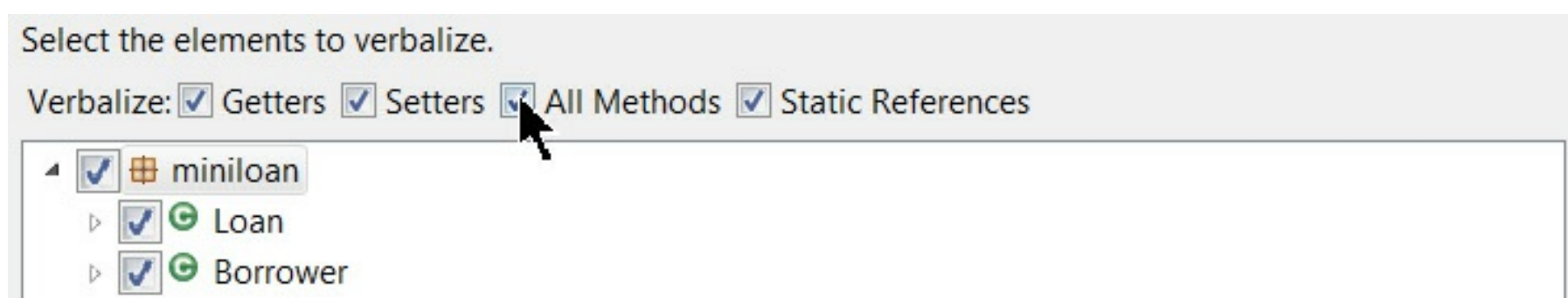
Tip: You can also right-click the bom folder in the Rule Explorer and click **New > BOM Entry**.

2. In the New BOM Entry wizard, in the **Name** field, type miniloan.
3. Ensure that **Create a BOM entry from a XOM** is selected, and then click **Next**.
4. In the **Choose a XOM entry** field, click **Browse XOM**, select platform:/miniloan-xom, and then click **OK**.
5. Under **Select classes**, select the miniloan package. Selecting the package selects all the classes in the package.



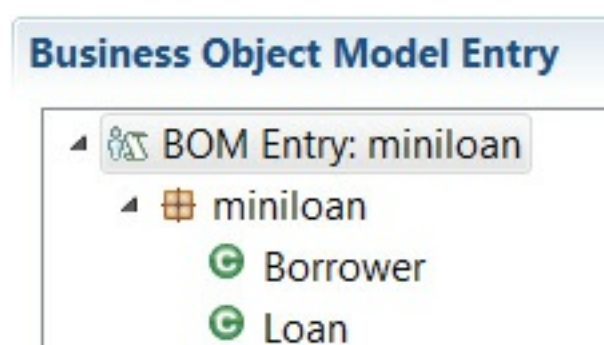
The miniloan package contains the Borrower and Loan classes.

6. Click **Next**.
7. On the BOM Verbalization page, you must select the **All Methods** check box. The **All Methods** check box ensures that all the methods are verbalized in addition to the elements already selected.



8. Click **Finish**.
9. In the Rule Explorer, double-click **bom > miniloan** to open the BOM Editor, and take a moment to look at the BOM.

In the BOM Editor, expand the miniloan entry:



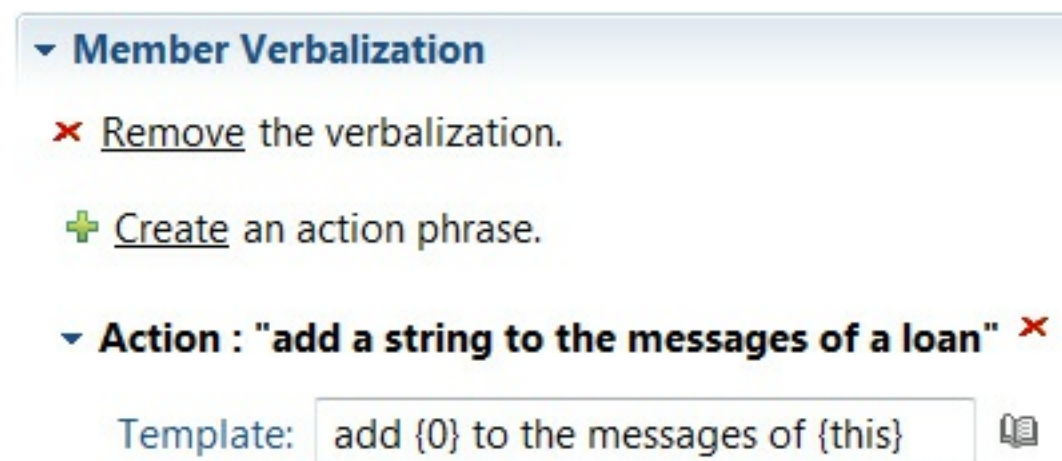
In your BOM, you now have two classes that are equivalent to the XOM classes Borrower and Loan.

10. Double-click the Loan class to open it in the BOM editor.

All the Java members and methods are converted and assigned a default verbalization.

11. In the Members section, double-click the `addToMessages(String)` method.

The BOM editor switches to the Member tab. In the Member Verbalization section, you can see that the verbalization of this method is add a string to the messages of a loan:



The verbalization is also used in the Rule Editor.

12. Close the miniloan tab to close the BOM Editor:

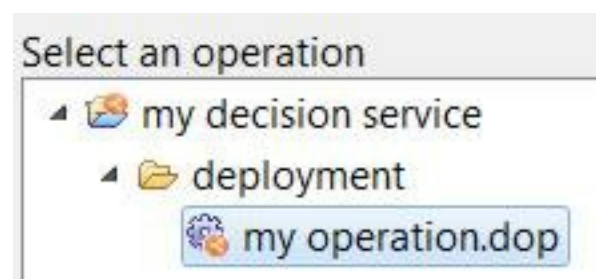


Step 5: Defining a decision operation

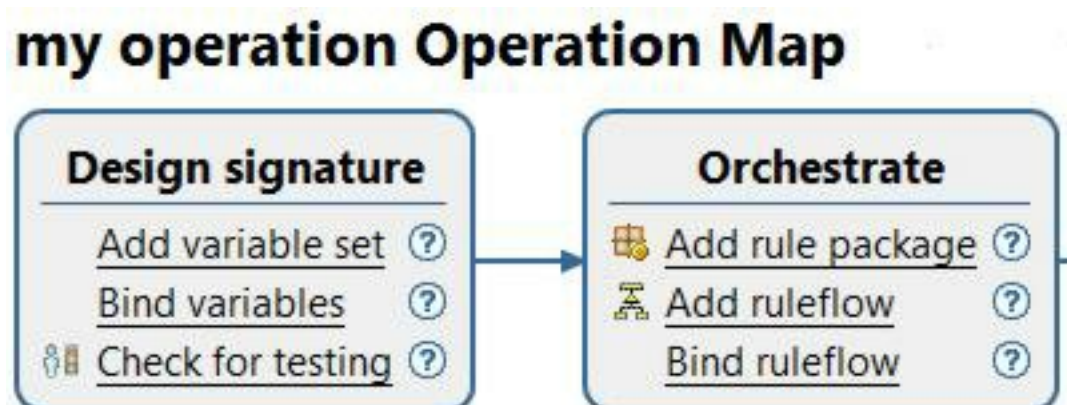
A ruleset is a runnable package that includes rule artifacts and other elements. It contains a set of rules that can be run by the rule engine. You must define the contents of the ruleset and the parameters that allow the client application and the ruleset to exchange information. A decision operation includes all the settings that are needed to define the contents of the ruleset and its parameters.

Procedure

1. In the Decision Service Map, click **Add decision operation**.
2. In the New Decision Operation wizard, in the **Name** field, type `my operation`.
3. Click **Next**. The Source Rule Project window opens.
4. Ensure that `my decision service` is selected, and then click **Finish**.
5. In the Decision Service Map, click **Go to operation map**.
6. Select **my decision service > deployment > my operation.dop**, and click **OK**.



The operation map opens and shows the different actions that you can do. The links in the map open different editors.



Step 6: Designing the operation signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation. They are equivalent to Java method parameters. They are references that you can use when you write rules.

Procedure

To enable a decision to be made on the status of a loan, you create ruleset parameters for the borrower and the loan:

- The borrower is an IN parameter. The value of the IN parameter is provided as input to the ruleset when

run.

- The loan is an IN_OUT parameter. The value of the IN_OUT parameter is provided as input to the ruleset when run, and can be modified by the ruleset and provided as output when the run completes.

1. In the operation map, click **Add variable set**.
2. In the New Variable Set wizard, in the **Name** field, type my parameters.
3. Click **Finish**.
4. Define the borrower parameter:
 - a. Click **Add**. A new row is displayed with default values.
 - b. In the **Name** column, type borrower.
 - c. In the **Type** column, click the ... button, and then double-click the **Borrower** type in the Matching types box.

The miniloan.Borrower type is displayed in the **Type** column.

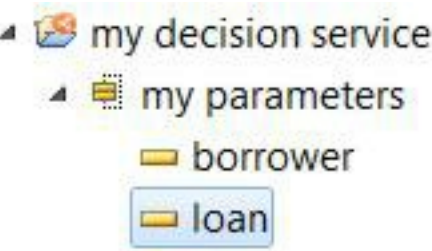
- d. In the **Verbalization** column, type the borrower.
5. Repeat step 4 to define the loan parameter:
 - Name: loan
 - Type: **Loan**
 - Verbalization: the loan

Your variable set is shown as follows:

Variable Set: my parameters



Name	Type	Verbalization	Initial Value
borrower	miniloan.Borrower	the borrower	
loan	miniloan.Loan	the loan	

6. Click the Eclipse **Save** button to save your work, and close the my parameters variable set editor.
7. In the Operation Map, click **Bind variables**.
8. Expand my parameters in the Eligible variables part of the Decision Operation Signature editor:



- a. Drag borrower to the input parameters.
- b. Drag loan to the input-output parameters.

The decision operation signature is displayed as follows:

Input Parameters		
Define the parameters required to call the execution.		
Parameter name	Verbalization	Type
 borrower	the borrower	miniloan.Borrower
Input - Output Parameters		
Define the parameters that are required, modified, and then returned by the execution.		
Parameter name	Verbalization	Type
 loan	the loan	miniloan.Loan

9. Save your work, and close my operation.

What to do next

In the next task, you create a ruleflow to orchestrate how the rules run.

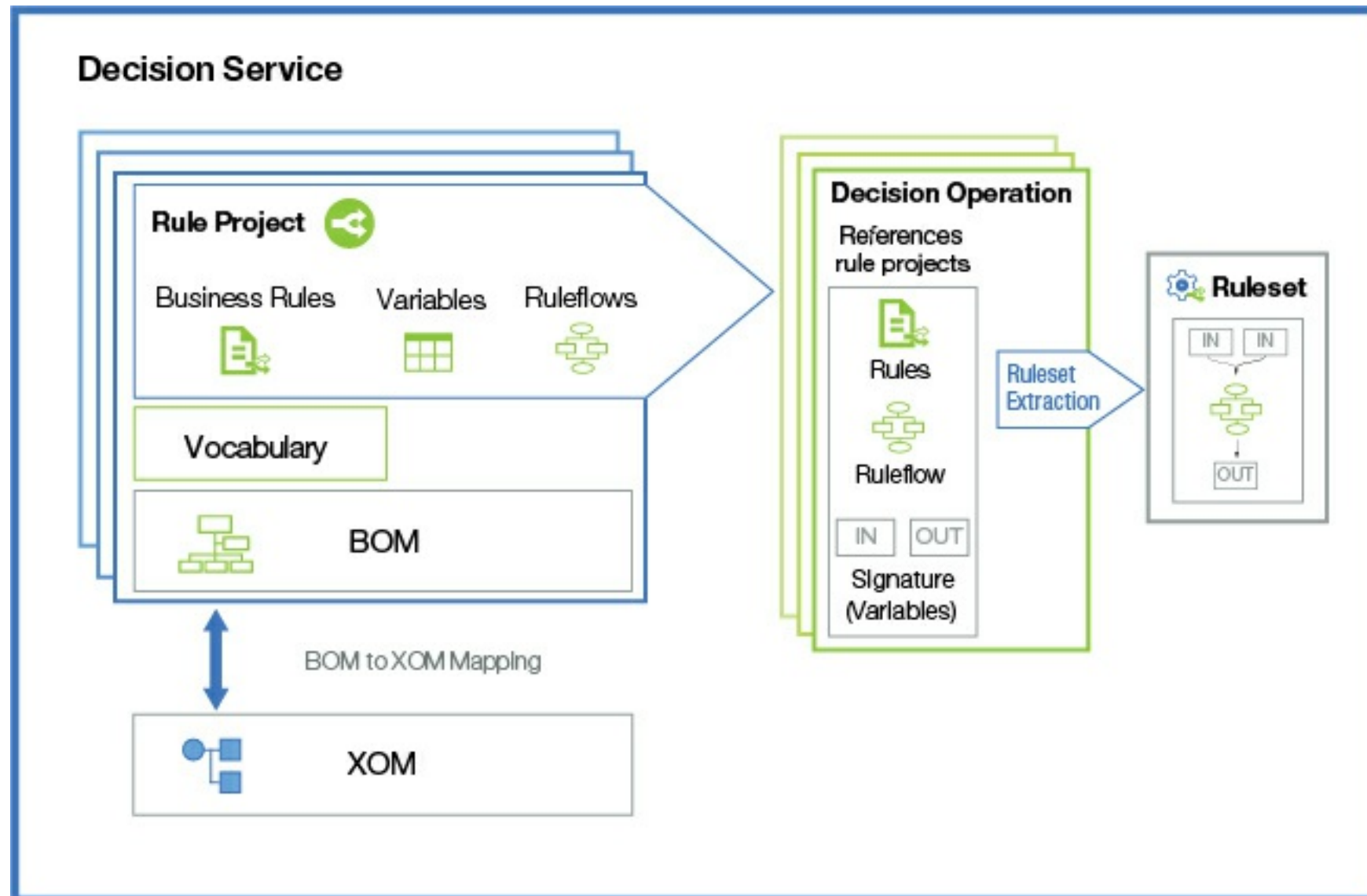
[< Previous](#) | [Next >](#)

Task 2: Orchestrating rule running

You create a ruleflow to set the sequence in which the rules run.

About this task

In Rule Designer, you use a ruleflow to orchestrate rules. The ruleflow defines the order in which the rule engine processes the rules.



Step 1: Creating rule packages

Before you define a ruleflow, you must organize your rules into packages that contain related rules. In this step, you create two rule packages, and then define the ruleflow for the packages.

Procedure

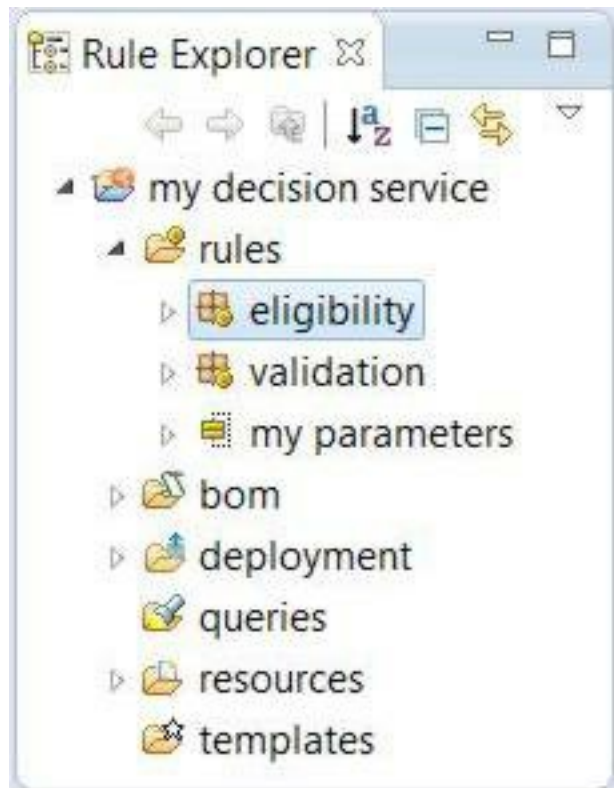
1. In the Operation Map, click **Add rule package**.

Tip: You can also right-click the my decision service/rules folder in the Rule Explorer and click **New > Rule Package**.

2. In the New Rule Package wizard, enter validation in the **Package** field, and then click **Finish**.

The validation rule package opens in the Rule Explorer.

3. Repeat steps 1 and 2 to create an eligibility rule package. Your rule project now contains two rule packages in which you can store rules.



Step 2: Creating the ruleflow diagram

You now design a ruleflow that creates a logical connection between the validation and eligibility rule packages. A ruleflow is a diagram that consists of several rule tasks that are connected by logical links. A ruleflow shows the sequence for running business rules.

Procedure

1. In the Operation Map, click **Add ruleflow**.

Tip: You can also right-click the my decision service/rules folder in the Rule Explorer and click **New > Ruleflow**.




2. In the New Ruleflow wizard, ensure that the **Source folder** field is set to /my decision service/rules, and that the **Package** field is empty.
3. In the **Name** field, enter miniloan.
4. Click **Finish**. The ruleflow editor opens.

You now construct a ruleflow in which you specify how tasks are related, that is, how, when, and under what conditions rules are run.

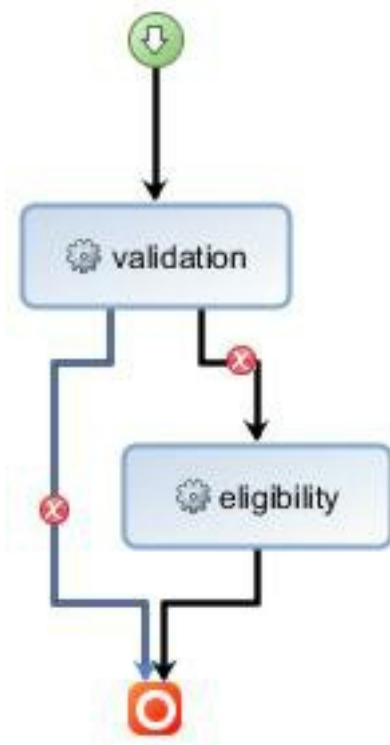
Step 3: Defining rule tasks


You add the start and end nodes, select the rule packages to include in the ruleflow diagram, and then you create the transitions between the packages.

Procedure

1. Click **Create a start node** , and then click inside the ruleflow editing window to add the node.
2. Click **Create an end node** , and then click inside the ruleflow editing window to add the node.
3. Drag the validation rule package from the Rule Explorer to the ruleflow editing window. The rule package becomes a rule task in the ruleflow.
4. Drag the eligibility rule package from the Rule Explorer to the ruleflow editing window.
5. Click **Create a transition**  and create the following transitions (shown as arrows) by clicking the first item and then clicking the second item:
 - a. Connect the start node to the validation task.
 - b. Connect the validation task to the eligibility task.
 - c. Connect the eligibility task to the end node.
 - d. Connect the validation task to the end node.

The ruleflow diagram shows errors on transitions to indicate that conditions are missing:



6. Click **Create a transition**  again to deselect the transition tool.
7. Click **Layout All Nodes**  to format the ruleflow diagram.
8. Save your work.

Step 4: Defining the main transition

You set a transition condition so that the rules in the eligibility package are run only when data is validated.

Procedure

1. Click the transition from validation to eligibility.

The Properties view shows the condition for this transition.

Tip: If you cannot see the Properties view, click **Window > Show View > Properties** to open the view.

2. In the Properties view, enter data approved in the **Label** field.
3. Ensure that **Use BAL for transition condition** is selected. The Business Action Language (BAL) is the language in which you write your conditions.
4. Click in the blank text area beneath **Use BAL for transition condition**, and then press Space to display the Content Assist box. Double-click the items to form the following statement: 'the loan' is approved.

Tip: You can also enter the statement directly in the text area.

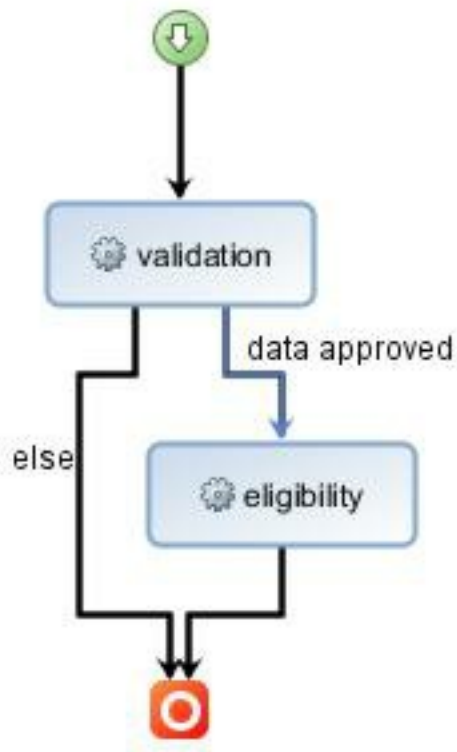
The Properties view looks as follows:



The transition from validation to the end node is automatically set to else.

5. Save your work.

Your ruleflow now looks like the following diagram:



Step 5: Defining the final action

You now have transitions between your rule packages. You can also define a final action to display a message in the console that indicates the status of the loan at the end of a rule run.

Procedure

1. Click the end node.
2. In the Properties view, in the **Final Action** section, ensure **Use BAL for action** is selected.
3. Click in the text area, and press Space to display the Content Assist box. Enter the following final action:

```
print the approval status of 'the loan' ;
```

You must add a semicolon (;) to the end of the action statement.
4. Save your work and close the ruleflow editor.

Step 6: Binding the ruleflow

To use the ruleflow at run time, you must bind the ruleflow to the decision operation.

Procedure

1. In the Operation Map, click **Bind ruleflow**.
2. In Decision Operation Overview, select **Use main ruleflow** in the Ruleflow section.
3. Click **<choose a ruleflow>** and select my decision service/rules/miniloan.
4. Click **OK**.
5. Save your work and close the decision operation editor.

What to do next

In the next task, you write an action rule.

[< Previous](#) | [Next >](#)

Task 3: Authoring business rules

You write an action rule and import other rules for your rule project.

About this task

As a developer, you write the initial business rules and design the rule templates. You want to enable business users to write and edit business rules in a web environment.

In this task, you create an action rule, and then you import the rest of the rules for your project. An action rule states the actions to take under certain conditions. You write the following rule, which is based partly on the vocabulary that you created earlier in this tutorial:

```
if
  the amount of 'the loan' is more than 1,000,000
then
  add "The loan cannot exceed 1,000,000" to the messages of 'the loan';
  reject 'the loan' ;
```

Step 1: Creating an action rule

You create an action rule that rejects any loan request that has an amount that is greater than 1,000,000.

Procedure

1. In Rule Designer, in the Author part of the Operation Map, click **Add action rule**.

Tip: You can also right-click the validation package in the Rule Explorer and click **New > Action Rule**.

2. In the **Package** field, type validation, and in the **Name** field, type maximum amount.

Source folder:	<input type="text" value="/my decision service/rules"/>	<input data-bbox="987 1394 1257 1460" type="button" value="Browse..."/>
Package:	<input type="text" value="validation"/>	<input data-bbox="987 1478 1257 1543" type="button" value="Browse..."/>
Name:	<input type="text" value="maximum amount"/>	
<hr/>		
Template:	<input type="text"/>	<input data-bbox="987 1697 1257 1762" type="button" value="Browse"/>
Type:	<input type="text" value="ActionRule"/>	

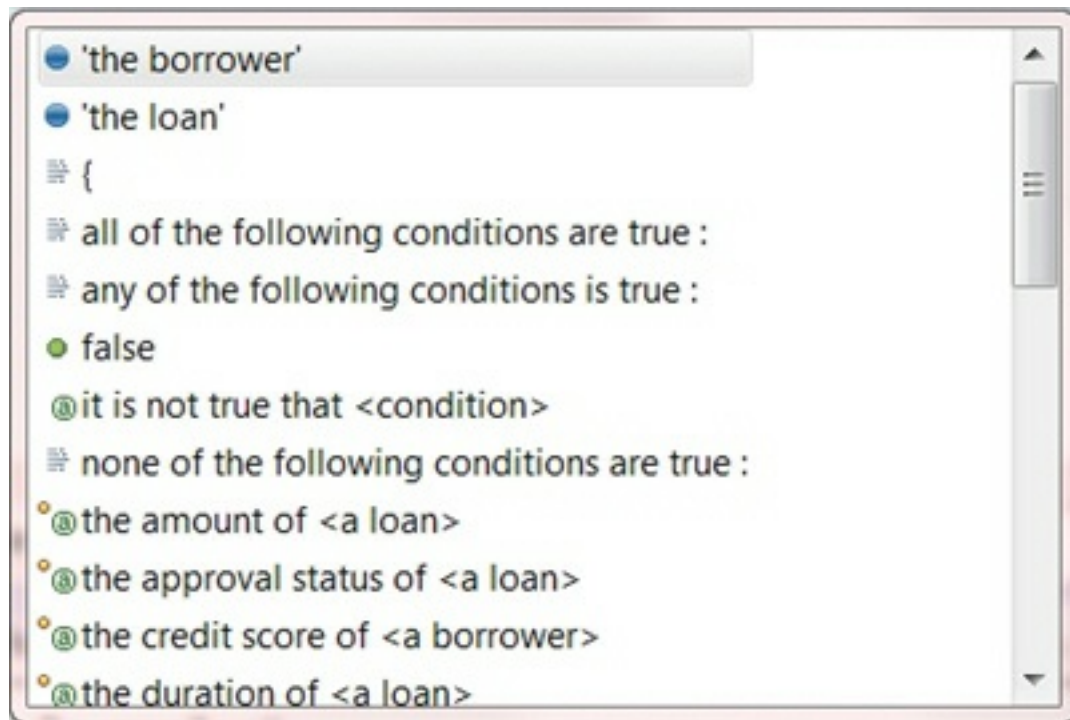
3. Click **Finish**. The Intellirule editor opens.

Step 2: Completing the action rule

You use the code completion menu in Intellirule to complete the action rule. You must create the parts of the rule, and use the completion mechanism to select phrases from a list of rule fragments to author the rule.

Procedure

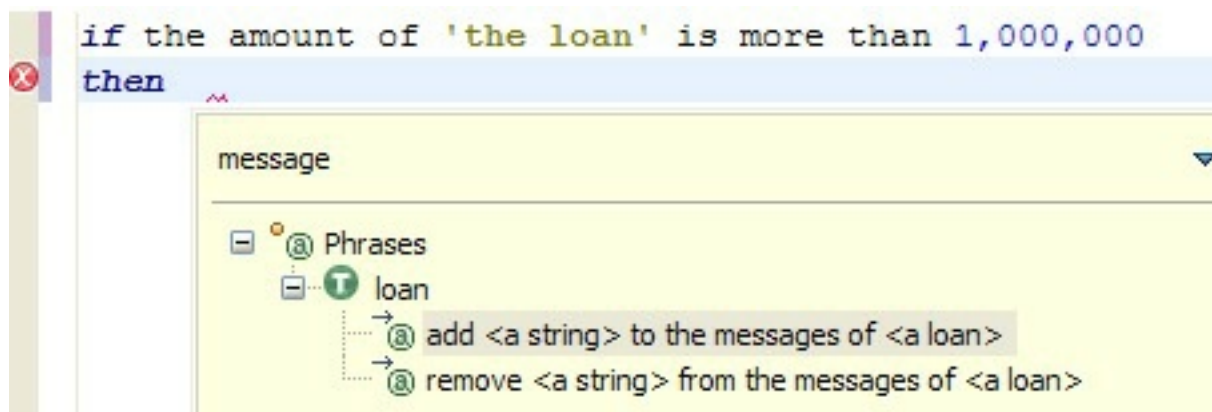
1. In the Intellirule editor, type `if`, and then press Space. The Content Assist box opens:



Select terms and phrases from the menu to compose the following expression:

```
if the amount of 'the loan' is more than 1,000,000
```

- On the next line, type then, press Space, and then press Ctrl+Shift+Space to activate the tree view of the completion menu. The tree view displays possible phrases to select for your rules, and a blinking prompt to enter text in a field.
- Type message in the text field. The tree view displays terms and phrases about messages:



- Double-click add <a string> to the messages of <a loan> to add the phrase to the rule. When the Content Assist box opens, use it to complete the phrase to create the following expression:

```
add "The loan cannot exceed 1,000,000" to the messages of 'the loan' ;
```

Important: You must add a semicolon (;) to the end of the line.

- If you are still in the Content Assist box, press Esc.
- Press Enter to create a new line, type a space, and then write the following statement. You can use the Content Assist box:

```
reject 'the loan' ;
```

- Press Ctrl+Shift+F to format the rule.
- Save your work and close the Intellirule editor.

Step 3: Importing the remaining rules

You must now add the rules that determine whether a borrower is eligible for a loan. In this step, you import the eligibility rules into your rule project.

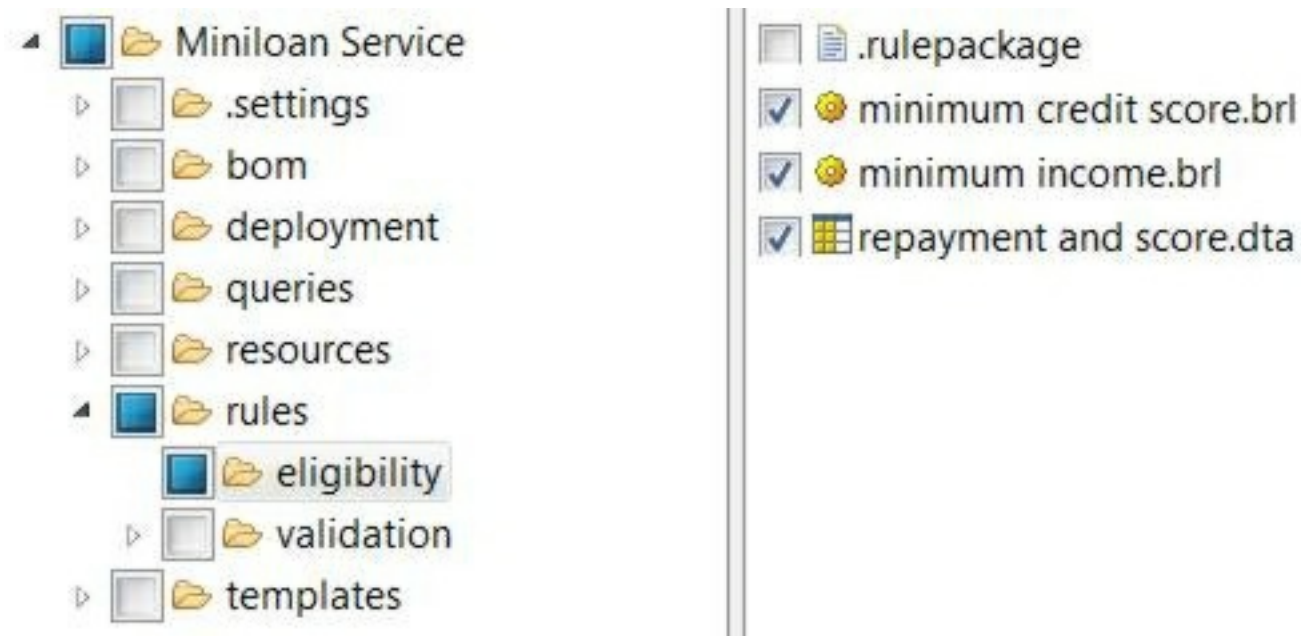
Procedure

- Click **File > Import > General > File System**, and then click **Next**.
- Next to the **From directory** field, click **Browse**. Navigate to <InstallDir>/odm-cloud-getting-started-master, select Miniloan Service, and then click **OK**.

Note: Ignore the message There are no resources currently selected for import. The Eclipse message prompts you to do the following action.

- Expand Miniloan Service/rules in the Import wizard.
- Select the check box for eligibility. You might have to click **Deselect All** before selecting eligibility.
- Click the name eligibility to display the contents of the folder.

6. Clear the .rulepackage check box, which you do not need for the tutorial. The Import wizard looks as follows:



7. In **Into folder**, ensure that my decision service is shown. If not, click **Browse**, select my decision service, and then click **OK**.
8. Under Options, select the **Overwrite existing resources without warning** option to prevent duplication:



9. Click **Finish**.

Step 4: Viewing the imported rules

You can view the imported rules in your project.

Procedure

1. In the Rule Explorer, expand eligibility and double-click the rules to review them:

minimum credit score

Rejects the loan if the credit score is too low.

minimum income

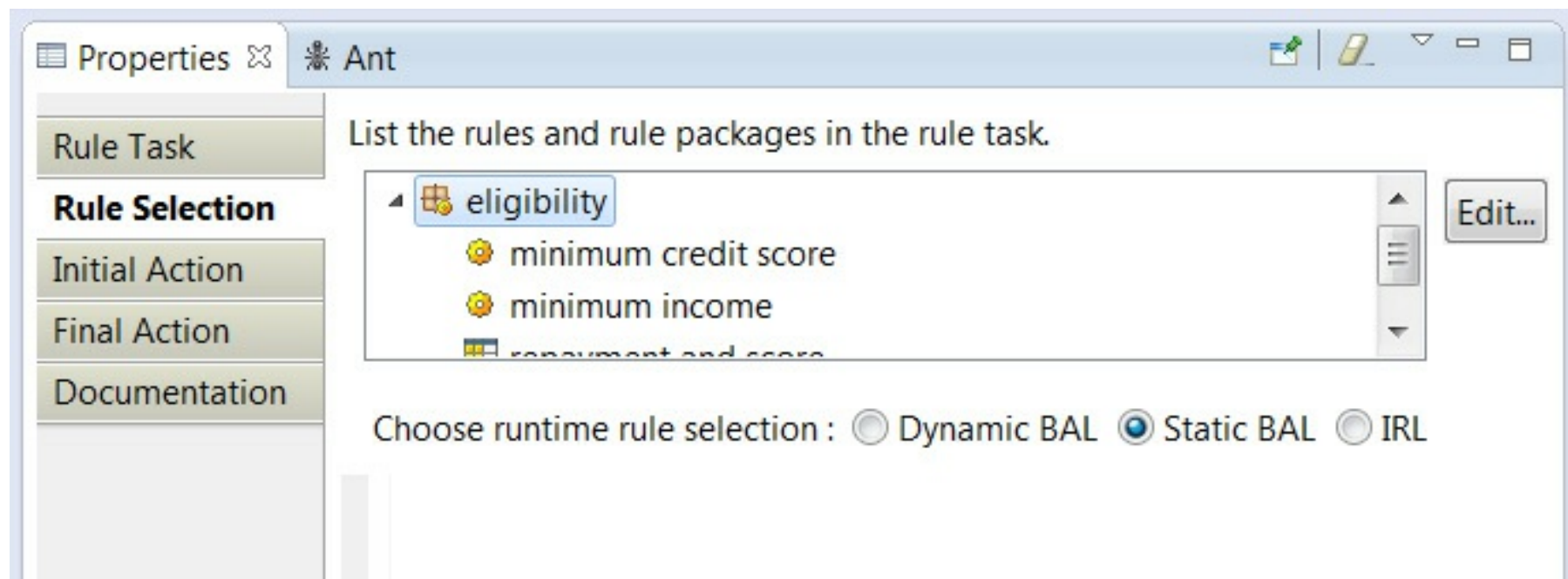
Rejects the loan if the income is too low for the yearly repayments.

repayment and score

Rejects the loan based on various values of the debt-to-income ratio and credit score. As a decision table, it represents rules that share conditions and actions. Each row in the table represents a rule. Place your cursor over the number of a row to view the corresponding rule as hover help.

Note: Rows with no actions display a warning about invalid or incomplete rows. These rows fill gaps in the table, and they are ignored when you run the project.

2. Double-click the miniloan ruleflow to open it.
3. In the Ruleflow Editor, double-click the eligibility task.
4. In the Properties view, open the Rule Selection tab, and expand the eligibility package.



By default, all the rules are used when the task is run by the rule engine.

5. Close the Ruleflow Editor.

What to do next

In the next task, you test and debug your rules.

[< Previous](#) | [Next >](#)

Task 4: Testing and debugging a ruleset

You run the ruleset by entering input data in a decision operation configuration, and then test and debug your ruleset.

About this task

You use Rule Designer to test and debug your rule project. You create a run configuration to run the decision operation in Rule Designer. Then, you insert a breakpoint that stops the run at a specific point in the ruleflow, and run the decision operation by using a debug configuration.

Step 1: Creating a run configuration

To test whether the ruleset can run, you run the decision operation in Rule Designer.

Procedure

1. In the Eclipse menu bar, click **Run > Run Configurations**.
2. Right-click **Decision Operation** and click **New**.
3. In the **Name** field, enter `Miniloan Test` for the name of the launch configuration.
4. To set the decision operation, browse to `my decision service/my operation`, and click **OK**.
5. On the Parameters & Arguments tab, click `borrower` and **Edit Value**.
6. In the Edit Parameter Value view, ensure that the Expression value is selected and enter the following text:

```
new miniloan.Borrower("Joe", 600, 8000)
```

7. Click **OK**.
8. Repeat steps 5 to 7 to set `loan` to the following value:

```
new miniloan.Loan(50000, 240, 0.05)
```
9. Click **Apply**, and then click **Run**. In the Console view, you see the results:

```
false [Too big Debt-To-Income ratio]
```

Tip:

To open the Console view, on the **Window** menu, click **Show View > Other > General > Console**, and then click **OK**.

Step 2: Inserting a breakpoint

You set a breakpoint in Rule Designer to debug rules.

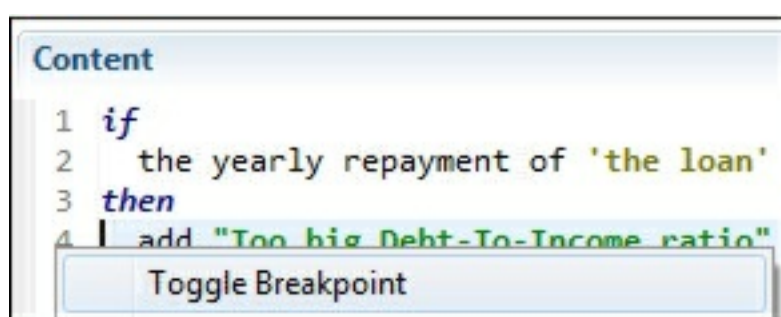
Procedure

1. In the Rule Explorer, double-click the `miniloan` ruleflow to display it in the Ruleflow Editor.
2. Select and right-click the `eligibility` task in the ruleflow diagram, and then click **Toggle Breakpoint**.

A breakpoint marker is added to the `eligibility` task.



3. In the Rule Explorer, open `eligibility` and double-click the `minimum income` rule to open it in the Intellirule editor.
4. Right-click the fourth line of the rule in the margin, and select **Toggle Breakpoint** to add a breakpoint to the first action in the rule:




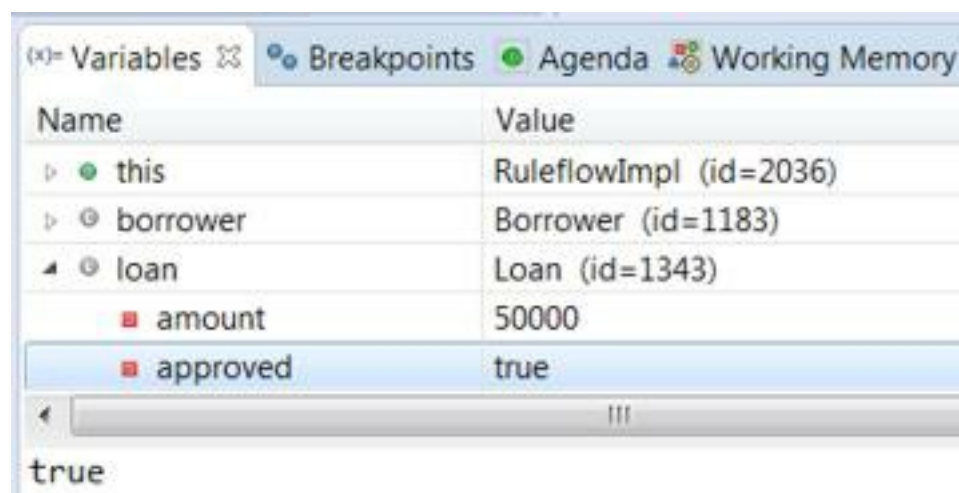
Step 3: Running the debugger

With the breakpoint in place, you can debug the run.

Procedure

1. Start the debugger:
 - a. Click **Debug > Debug Configurations**.
 - b. Select **Decision Operation > Miniloan Test**.
 - c. Click **Debug**.
 - d. When the Confirm Perspective Switch dialog opens, click **Yes** to open the Debug perspective. The debugging commands are available from the Debug view and from the **Run** menu.

Debugging stops at the beginning of the eligibility task, where you inserted the breakpoint.
2. Step through the rule code:
 - a. Click **Resume** . Debugging stops at the first action of the minimum income rule at the second breakpoint that you inserted.
 - b. In the Variables view, expand the loan object. The value of the approved attribute is true:



3. Click **Resume**  to complete the run.

When the run ends, the Console view shows the following message:

```
false [Too big Debt-To-Income ratio]
```

4. Return to the Rule perspective, and close the ruleflow.

What to do next

In the next task, you deploy a RuleApp from your decision service, and open it in the Rule Execution Server console.

[< Previous](#) | [Next >](#)

Task 5: Deploying your decision service

You deploy your decision service to Rule Execution Server.

About this task



Rule Execution Server is the running environment for business rule applications. It provides the management, performance, security, and logging capabilities that are associated with the running of rules. In Rule Designer, you create a deployment configuration in your decision service and use it to deploy a RuleApp to Rule Execution Server. You open the RuleApp to view its contents, and complete the task by removing the RuleApp.

Step 1: Creating a deployment configuration

Before you can deploy your decision service, you must create a deployment configuration that references the decision operation and lists a target server.

Procedure

1. Right-click `my decision service`, and click **New > Deployment Configuration**.
2. Enter `my deployment` as the name, and make sure `/my decision service/deployment` is the deployment folder.

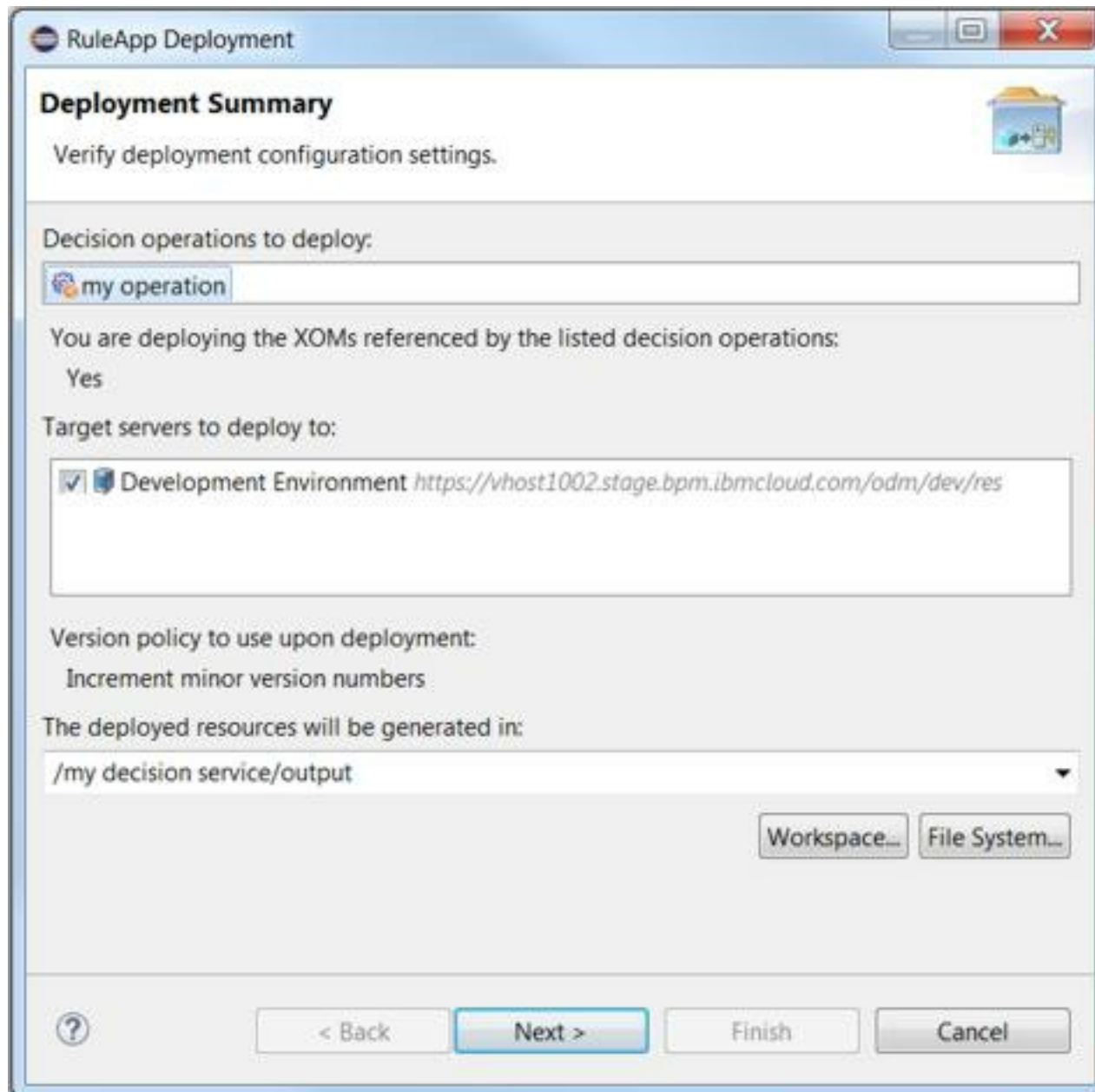
The name `my deployment` is used in this task. You can use a different name to avoid conflicts with other projects when you deploy the decision service. For example, you can prefix `my deployment` with your initials.
3. Click **Finish**. The deployment overview opens.
4. Make sure **Nonproduction** is selected for the type of deployment. You use **Nonproduction** because the decision service is destined for the cloud development environment. **Production** is used to deploy a decision service for use by client applications. From Rule Designer, you can deploy to only the cloud development environment, and not to the test or production environment.
5. Check that `my_deployment` is the RuleApp name, and that `1.0` is the version number. Rule Designer added the underscore to the RuleApp name.
6. Click **Include decision operations** to select a decision operation.
7. Click **Add**  in Configured Decision Operations.
8. Click **Select existing decision operations**, select `my decision service`, and then click **Finish**.
9. In the **Target Servers** tab, click **Add** , select **Development Environment**, and click **Finish**. Check that the URL for the Development Environment matches the address of the Rule Execution Server in your cloud development environment. The address starts with the URL that you use to connect to the cloud portal, and ends with `/odm/dev/res`, for example, `https://<vhostname>.bpm.ibmcloud.com/odm/dev/res`.
10. Save your work.

Step 2: Deploying the decision service

You deploy the RuleApp from Rule Designer to Rule Execution Server. The RuleApp contains a ruleset that holds the rules in your decision service.

Procedure

1. Right-click `my decision service` in the Rule Explorer.
2. Click **Rule Execution Server > Deploy**. The RuleApp Deployment wizard opens. It shows `my operation` as the decision operation, the `Development Environment` as the target server, and `/my decision operation/output` as the deployed resources.



3. Click **Next**. If you are not already authenticated, click **Connect** to authenticate your cloud connection. You must enter your cloud credentials when you authenticate your connection.
4. Click **Next**. Verify the RuleApp and ruleset versions, and then click **Finish**.

When the RuleApp is deployed, a deployment report is displayed in Rule Designer.



Step 3: Viewing the deployed RuleApp

You view the RuleApp in Rule Execution Server.

Procedure

1. Sign in to your instance of the Operational Decision Manager on Cloud portal.
2. Click **Launch** in the Rule Execution Server console section to open the component.
3. Click the **Explorer** tab.
4. In the Navigator, expand **RuleApps**, and then **/my_deployment/1.0**.

You see that Rule Execution Server contains version 1.0 of my_deployment, which contains version 1.0 of the my_operation ruleset:



5. Click **/my_operation/1.0** to view the details of the ruleset in the Ruleset View.

The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties. It also shows the version of the decision engine, and that the status of the ruleset is enabled,

indicating that it can be run.

 /my_deployment/1.0/my_operation/1.0	
Name	my_operation
Version	1.0
Creation Date	Mar 24, 2016 3:09:32 PM GMT+01:00
Display Name	my operation
Description	
Rule engine	Decision Engine - 1.40.0
Status	✓ enabled
Debug	⚙ disabled

Step 4: Testing the ruleset by using the REST API

Before you publish your decision service to Decision Center, you run it in Rule Execution Server to see whether it produces the expected results.

Procedure

1. Click **Retrieve HTDS Description File** in the Ruleset View. The file retrieval page opens.
2. Select **REST**, and click **Test**. The Hosted Transparent Decision Service opens.
3. Change the run request:
 - creditScore: 100
 - yearlyIncome: 70000
 - amount: 8000
 - duration: 12
 - yearlyInterestRate: 2.5
 - Remove lines 16-19

Your run request looks as follows:

```
1 <par:Request xmlns:par="http://<...>
2   <!--Optional:-->
3   <par:DecisionID>string</par:DecisionID>
4   <!--Optional:-->
5   <par:borrower>
6     <!--Optional:-->
7     <name>string</name>
8     <creditScore>100</creditScore>
9     <yearlyIncome>70000</yearlyIncome>
10  </par:borrower>
11  <!--Optional:-->
12  <par:loan>
13    <amount>8000</amount>
14    <duration>12</duration>
15    <yearlyInterestRate>2.5</yearlyInterestRate>
16  </par:loan>
17</par:Request>
```

4. Click **Execute Request**. You get the following server response:

```
<?xml version="1.0" encoding="UTF-8"?><par:Response xmlns:par="http://<...>
  <par:DecisionID>string</par:DecisionID>
  <par:loan>
    <amount>8000</amount>
    <duration>12</duration>
    <yearlyInterestRate>2.5</yearlyInterestRate>
    <yearlyRepayment>22301</yearlyRepayment>
    <approved>false</approved>
    <messages>Credit score below 200</messages>
    <messages>Too big Debt-To-Income ratio</messages>
    <messages>debt-to-income too high compared to credit score</messages>
  </par:loan>
</par:Response>
```

The rule application produces the expected results. The loan is rejected, and messages explain that the debt-to-income ratio is too high when compared to the credit score.

Step 5: Removing the RuleApp

After you test the RuleApp, you no longer need it in Rule Execution Server and remove it.

Procedure

1. Click **RuleApps** in the Navigator.
2. Select the check box for /my_deployment/1.0.
3. Click **Remove**. Rule Execution Server deletes the RuleApp.

What to do next

In the next task, you share your rules with other business users by publishing your project to Decision Center.

[< Previous](#) | [Next >](#)

[< Previous](#)

Task 6: Publishing to Decision Center

You publish your decision service to Decision Center in the Operational Decision Manager on Cloud portal, where you view your files in the Business console.

About this task

You now make your decision service available in Decision Center, a cloud environment where users view, create, and modify rules. You publish your decision service from Rule Designer to Decision Center, and can periodically synchronize the work of the business users with your Rule Designer copy.

This task is optional for learning how to make rules in Rule Designer. To do this task, you must have access to Decision Center in the cloud portal.

Important:

Before publishing your decision service, make sure the Decision Center Business console does not contain a decision service with the same name. If so, remove the other decision service from the console or change the name of your decision service in Rule Designer before you publish it.

To change the name of your decision service:

1. Right-click your decision service.
2. Click **Refactor** > **Rename**.
3. Enter a new name or modify the existing name.
4. Click OK. Rule Designer refactors the decision service.

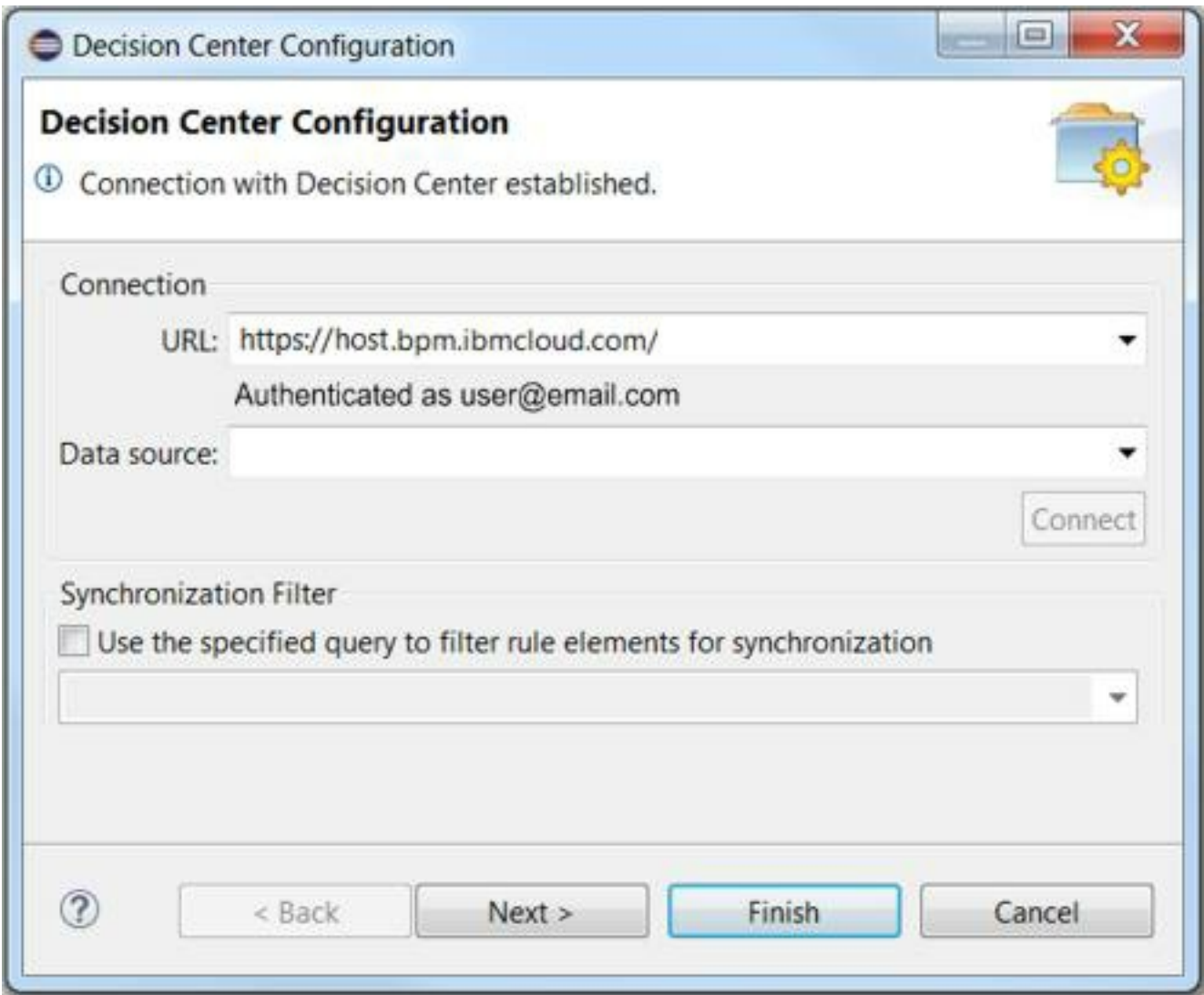
Step 1: Publishing the decision service to Decision Center

Ensure that you have user access to the cloud portal. You connect Rule Designer to Decision Center, and then publish your decision service to Decision Center.

Procedure

1. In Rule Designer, right-click your decision service (my decision service or the renamed version), and then click **Decision Center** > **Connect**.
2. In the connection wizard, enter the URL for your instance of the cloud portal if it is not already predefined. Use the following format: `https://<myodmcloud>.bpm.ibmcloud.com/`
3. Click **Connect**, and enter your user name and password for your cloud instance.

When the connection is established, the following message is displayed in the connection wizard: Connection with Decision Center established.



4. Click **Finish** to publish the decision service to Decision Center.

Tip: If the wizard shows that you are authenticated but you cannot click **Finish**, click **Connect** to

reestablish your connection.

5. Click **OK** to close the Synchronize Complete dialog.
6. Click **Yes** in the a dialog to change to the Team Synchronizing perspective.

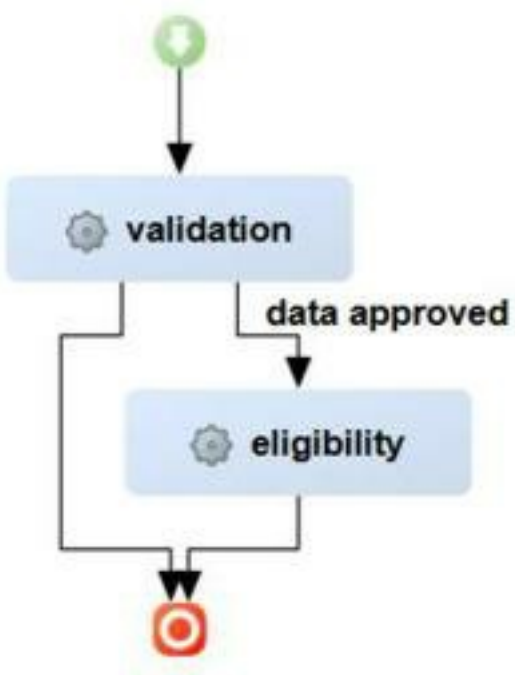
An empty Synchronize view opens. It shows that there are no changes in the project. Your rules are now published to Decision Center.

Step 2: Exploring the rule project in Decision Center

You open the Decision Center Business console to see your rules in the web-based editing environment.

Procedure

1. Sign in to the cloud portal, and open the Decision Center Business console.
2. Click **Library**. The library opens on the decision services by default.
3. Click your decision service (my decision service or the renamed version). The decision service shows tabs for releases and branches.
4. Click **Branches > main > Decision Artifacts** to view your rule files.
5. Click **All Project**, select **All Types**, and click **Apply** to display all the decision artifacts.
6. Click miniloan to see the ruleflow:



7. In the validation package, click maximum amount to see the contents of the rule:

```
if
  the amount of 'the loan' is more than 1000000
then
  add "The loan cannot exceed 1,000,000" to the messages of 'the loan' ;
  reject 'the loan' ;
```

8. In the eligibility package, click repayment and score to see the contents of the decision table:

	debt to income		credit score		message	loan approved
	min	max	min	max		
1	0 %	30 %	0	200	debt-to-income too high compared to credit score	<input type="checkbox"/>
2	0 %	30 %	200	800		<input checked="" type="checkbox"/>
3	30 %	45 %	0	400	debt-to-income too high compared to credit score	<input type="checkbox"/>
4	30 %	45 %	400	800		<input checked="" type="checkbox"/>
5	45 %	50 %	0	600	debt-to-income too high compared to credit score	<input type="checkbox"/>
6	45 %	50 %	600	800		<input checked="" type="checkbox"/>
7	≥ 50 %		0	800	debt-to-income too high compared to credit score	<input type="checkbox"/>

The warning icons indicate empty cells. These cells fill gaps in the table, and they are ignored at run time.

Results

You have completed the tutorial. Your business rules are now available in Decision Center, and can be maintained by business users.

Deleting your decision service

If you no longer need the decision service, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the](#)

[tutorial project.](#)

[**< Previous**](#)

[< Previous](#) | [Next >](#)

Debugging a decision service in Rule Designer

This tutorial provides a methodology for debugging a decision service in Rule Designer.

Learning objectives

You use debugging features to find and fix errors in a decision service. You do the following tasks:

- Use automatic exception handling.
- Set breakpoints in a ruleflow, action rule, and decision table.
- Run a debugging session.
- Identify and fix errors.

Best practices

This tutorial includes the following best practices for debugging decision services:

- Carefully read the error messages. They tell you where to look for errors. [Example...](#)
- Try automatic exception handling when an exception occurs in a rule condition. [Example...](#)
- Check the stack trace to get the names of ruleflows and rule tasks. [Example...](#)
- Switch to the debug mode to get more precise error information. [Example...](#)
- Use the search function to find rule artifacts. [Example...](#)
- Use queries to search for rules by their content. [Example...](#)
- Check the values in the Variables view at each breakpoint during debugging. [Example...](#)
- Check the Agenda view to see which rules are going to be run. [Example...](#)

Time required

40 minutes

[< Previous](#) | [Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work as a developer for a fictitious loan company. You are currently working on a decision service that validates loan requests. Called the Loan Validation Service, the decision service applies several criteria to determine the eligibility of potential borrowers, and contractual constraints for approved loans.

A business user is reviewing the results of the decision service. The user contacts you to tell you that the decision service rejects low-risk loans that actually qualify for approval. You must debug the decision service to determine why it rejects the loans.

In Rule Designer, you tag rule artifacts with breakpoints that stop the running of the decision service at key points. You can then examine the behavior of the rules in a step-by-step debugging process. You determine where the errors take place, and track them to specific rules. Then, you modify the rules to fix the errors.

Note: This tutorial is also available on GitHub. You can find it at [Tutorial: Debugging a decision service in Rule Designer](#).

Audience

This tutorial is for users who make decision services in Rule Designer.

Prerequisites

You need the following component, projects, and information:

- Rule Designer: Download this component from the Operational Decision Manager on Cloud portal and install it as directed.
- The following project files in the [Tutorial: Debugging a decision service in Rule Designer](#) GitHub repository:
 - answer: Contains the completed version of the decision service. You can run it to see the expected results.
 - start: Contains the faulty version of the decision service. You look for errors in it during the debugging process.

In the GitHub repository, click the **Clone or download** button to download the contents of the repository to a directory on your computer. Expand the downloaded file in the directory. In the tutorial, the directory is referred to as InstallDir.

- Knowledge of business rule programming principles, Java™, and the Eclipse environment

Lessons in this tutorial

[Running the completed decision service](#)

You run the completed version of the decision service to see the expected results.

[Task 1: Using automatic exception handling](#)

You import and run the faulty decision service. You find an exception error in a rule, and try to fix the problem by using automatic exception handling.

[Task 2: Setting breakpoints](#)

You set breakpoints to determine where a null pointer exception occurs.

[Task 3: Debugging a rule](#)

You set breakpoints in a rule and run a debug session. When you find an error, you make changes to correct it.

[Task 4: Debugging a decision table](#)

You add breakpoints to an action rule and a decision table, and then run a debugging session to find and fix an error.

[Task 5: Debugging a ruleflow](#)

You debug a ruleflow to determine why the decision table does not approve the loan. When you find the cause, you reorganize the ruleflow to achieve the correct results.

Running the completed decision service

You run the completed version of the decision service to see the expected results.

About this task

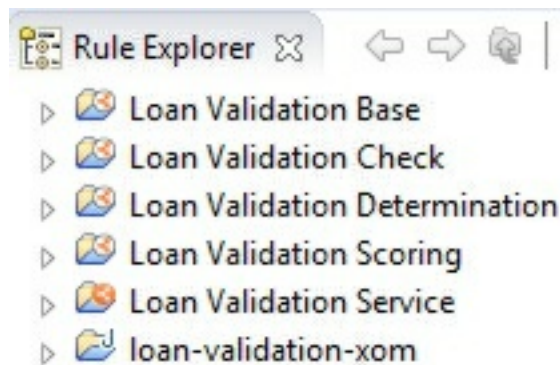
This version of the decision service contains the changes that you make in the tutorial. It produces the results that you can expect to achieve at the end of the tutorial.

Tip: If you want to refer to the completed decision service while you are going through the tutorial, use the answer and start projects in different Eclipse workspaces.

Procedure

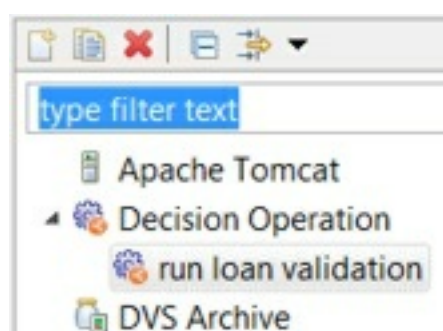
1. Start Rule Designer on your computer in the en_US (American English) locale (see [Installing Rule Designer](#)).
2. Close the Eclipse welcome page if it is open.
3. Open the Rule perspective if it is not open. Click **Window > Perspective > Open Perspective > Other > Rule** to open the perspective.
4. Import the answer project that you downloaded from GitHub:
 - a. In the Rule perspective, click **File > Import**.
 - b. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
 - c. Choose Select root directory, and browse to `<InstallDir>/answer`. InstallDir is the name of the directory to which you downloaded the answer project from GitHub.
 - d. Click **Select All**, and in Options, click **Copy projects into workspace**.
 - e. Click **Finish**.

The Rule Explorer displays six projects:



The Java™ project loan-validation-xom defines an execution object model (XOM) for rule execution. The main decision service, Loan Validation Service, references the other projects.

5. In the **Run** menu, select **Run Configurations**.
6. Expand **Decision Operation**, and click the run loan validation configuration:



7. Click **Run**. The Console view displays the following results, which show that the input data is valid and the loan is approved:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
  - Yearly income 20,000
  - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved true
  - Rate 0.055
  - Monthly repayment 570.83
  - Insurance required true Insurance rate 2%
  - Message Low risk loan
Congratulations! Your loan has been approved
```

What to do next

In the task, you run the faulty version of the decision service and see the errors in the output.

[< Previous](#) | [Next >](#)

Task 1: Using automatic exception handling

You import and run the faulty decision service. You find an exception error in a rule, and try to fix the problem by using automatic exception handling.

About this task

Before you make any changes to the faulty decision service, you want to see the results that it produces. You import the decision service into Rule Designer, and then run it. The results show a null pointer exception in a rule condition. Before going further, you activate the automatic exception handling feature to see whether it can correct the exception.

Step 1: Running the decision service

You import the start projects for the tutorial, and run the decision service.

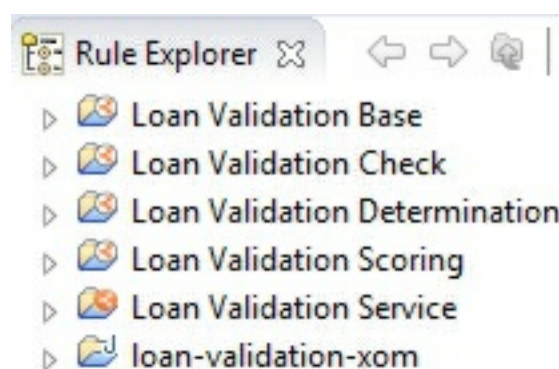
Before you begin

If you have imported the answer projects to run the completed decision service, delete them before importing the start projects into the same Eclipse workspace. Otherwise, use the start projects in a different Eclipse workspace.

Procedure

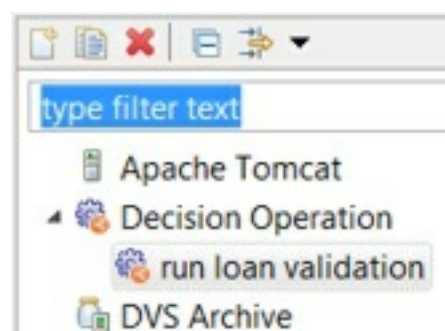
1. Start Rule Designer on your computer in the en_US (American English) locale.
2. Close the Eclipse welcome page if it open.
3. Open the Rule perspective if it is not open. Click **Window > Open Perspective > Other > Rule** to open the perspective.
4. Import the start project that you downloaded from GitHub:
 - a. In the Rule perspective, click **File > Import**.
 - b. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
 - c. Choose Select root directory, and browse to `<InstallDir>/start`. InstallDir is the name of the directory to which you downloaded the start project from GitHub.
 - d. Click **Select All**, and in Options, click **Copy projects into workspace**.
 - e. Click **Finish**.

The Rule Explorer displays six projects:



The Java™ project loan-validation-xom defines an execution object model (XOM) for rule execution. The main decision service, Loan Validation Service, references the other projects.

5. From the **Run** menu, select **Run Configurations**.
6. Expand **Decision Operation**, and select the run loan validation configuration:



7. Click **Run**.

The Console view shows the results, which include the following exception error:

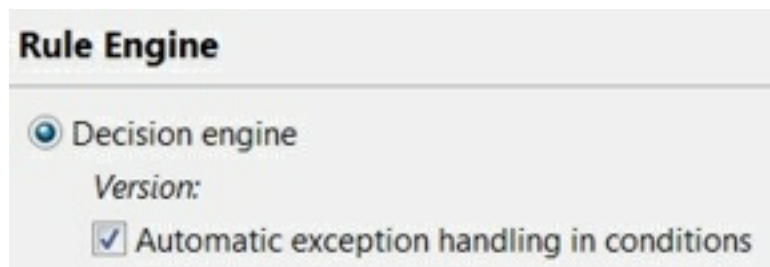
```
Error while executing ruleset: java.lang.NullPointerException
com.ibm.rules.engine.util.EngineExecutionException: java.lang.NullPointerException
    at com.ibm.rules.engine.fastpath.runtime.AbstractFastEngineServices.handleExceptionRaisedInCondition(AbstractFastEngineServices.java:69)
```

Step 2: Applying automatic exception handling

Because the exception occurs in a rule condition, you decide to try to fix it by using automatic exception handling.

Procedure

1. In the Rule Explorer, right-click Loan Validation Service.
2. In the pop-up menu, click **Properties**, and select **Rule Engine** in the left column of the **Properties** dialog.
3. Select **Automatic exception handling in conditions**, and then click **OK**:



4. Run the run configuration that is shown in step 1 of this task. The results no longer show an exception error. The decision engine automatically handles the exception. To determine where the exception takes place, you add more logging information to the results.

Step 3: Adding exception handing logging

You add additional logging information to determine where the decision engine automatically handles the exception error. First, you look at the Loan Validation Service/logging.properties file, and set it as the logging configuration file. Then, you set the Java VM parameter of the run configuration to declare the path to the logging configuration file.

Procedure

1. In the Rule Explorer, double-click Loan Validation Service/logging.properties. The file sets the Console as a logging handler:

```
handlers = java.util.logging.ConsoleHandler
```

It also sets the log level for the automatic exception handling to FINE.

2. Open **Run > Run Configurations**, and select the run loan validation configuration.
3. Open the **Parameters and Arguments** tab, and add the following VM argument:

```
-Djava.util.logging.config.file="${workspace_loc}/Loan Validation  
Service/logging.properties"
```

4. Click **Apply**, and then click **Run**. The Console shows the results, which include a log message for the automatic exception handling:

```
Jul 27, 2017 12:28:25 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger  
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

The results tell you that the exception takes place in a condition in the checkZipcode rule.

What to do next

In the next task, you add breakpoints to a ruleflow to determine where the exception handling occurs.

[< Previous](#) | [Next >](#)

Task 2: Setting breakpoints

You set breakpoints to determine where a null pointer exception occurs.

About this task

You look at the automatic exception handling log message:

```
Jul 27, 2017 12:28:25 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger  
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

It lists the loanvalidation ruleflow and its validation rule task as follows:

```
ruleflow.loanvalidation$003evaluation
```

You decide to track the processes in the task by placing a breakpoint on it. Then, you search for the rule that produces the error, add more tracing, and then run a debug configuration.

Step 1: Setting a breakpoint

You run the decision service, search for the source of an error, and add a breakpoint.

Procedure

1. Run the decision service as shown in [Task 1: Using automatic exception handling](#). The results contain the following error code:

```
ruleflow.loanvalidation$003evaluation
```

You understand the code as follows:


- Ruleflow name: loanvalidation
- Rule task: validation

You want to find the validation task in the loanvalidation ruleflow.

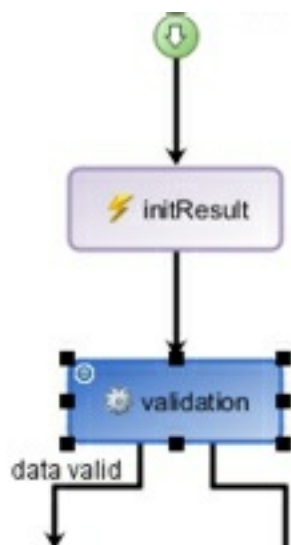
2. Click **Search** > **Search** in the toolbar.
3. Open the **Rule Search** tab.
4. Enter loanvalidation in the search field.

Tip: You can copy the name of the ruleflow in the error message, and paste it in the search field.

5. Select **Ruleflow**, and then click **Search**. The loanvalidation ruleflow is displayed in the Search view:

 loanvalidation - [Loan Validation Service] - /Loan Validation Service/rules

6. Double-click the ruleflow in the Search view to open it.
7. Click the validation task to select it.
8. Right-click the task and click **Toggle Breakpoint** at the bottom of the pop-up menu to tag the task with a breakpoint. The task now shows a dot to indicate the addition of the breakpoint:



Step 2: Preparing the debug session

Because the checkZipcode action rule shows an exception, you look for the rule and add trace information to it.

Before you begin

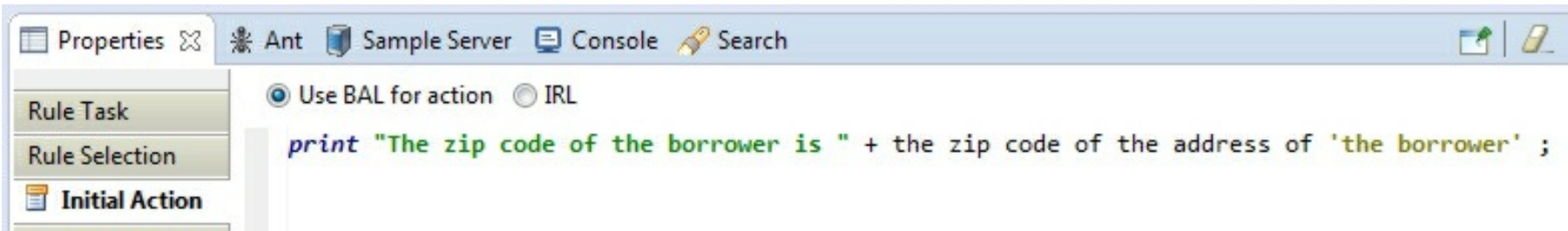
In this step, you navigate to the action rule through the **Rule Selection** tab of the properties of the ruleflow task. Optionally, you can do a rule search in **Search** to find the action rule by name.

Procedure

- 1. Look at the automatic exception handling message. You see that the null pointer exception occurs in the checkZipcode rule condition:
- 2. Double-click the validation task in the loanvalidation ruleflow.
- 3. In the Properties tab, click **Rule Selection** and expand the validation package.
- 4. Expand borrower, and double-click the checkZipcode action rule to open it in the rule editor. You see that the condition in the rule uses the length of the zip code. You want to know the value of the zip code, so you add trace information to print the value.
- 5. Go back to the loanvalidation ruleflow in the ruleflow editor.
- 6. Double-click the validation task.
- 7. In the Properties view, select **Initial action**. Keep **Use BAL for action** selected.
- 8. Add the following code to display the borrower zip code:

```
print "The zip code of the borrower is " + the zip code of the address of 'the borrower' ;
```

The code looks as follows:



- 9. Save your work.

Step 3: Running the debugger on a ruleflow

You run a debug configuration to check the values of the borrower zip code. You are looking for null values to determine where the null pointer exception occurs.

Procedure

- 1. Click **Run > Debug Configurations** in the toolbar.
- 2. Open the run loan validation configuration and click **Debug**.
- 3. If the Save and Launch dialog opens, click **OK** to save your changes before running the debugger.
- 4. Click **Yes** in the Confirm Perspective Switch dialog box to open the debug perspective. The debugger stops at the beginning of the validation rule task, at the breakpoint that you set in step 1 of this task.
- 5. Expand borrower in the **Variables** tab. Notice the null values for latestBankruptcy, spouse, and zipCode:

borrower	Borrower (id=1550)
birth	GregorianCalendar (id=1967)
creditScore	200
firstName	"Smith" (id=1403)
lastName	"John" (id=1790)
latestBankruptcy	null
spouse	null
SSN	Borrower\$SSN (id=1828)
yearlyIncome	20000
zipCode	null

- 6. Click the **Resume** button. The results in the Console view begin with the following message:

```
The zip code of the borrower is null
Jul 27, 2017 12:48:39 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

The first line is the trace that you added to the initial action of the validation task. You see that the automatic exception handling catches the null pointer exception because of the null zip code in the checkZipcode rule. The condition part has an unknown status, so the rule is ignored. You tell the business user that the borrower in the configuration has a null zip code, and you are told that it is not a problem.

The automatic exception handling correctly addresses the problem. You can remove the trace for the logging information that comes from the automatic exception handling. You keep the trace on the initial action of the validation task to get the zip code value.

Note: The Console shows the AEH_LOG message twice. Because the checkZipcode rule contains a then action and an else action, its condition part is evaluated twice, producing two AEH_LOG messages.

7. Switch to the Rule perspective.
8. Click **Run > Run Configurations**, and open the run loan validation configuration.
9. Delete the VM arguments from the **Parameters and Arguments** tab.
10. Click **Apply**, and then **Run**. The automatic exception handling trace information is no longer shown. However, the loan is still not approved, so there are more errors to fix:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
  - Yearly income 20,000
  - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved false
  - Rate 0.055
  - Monthly repayment ∞
  - Message Very risky loan
Too big Debt/Income ratio: ∞
We are sorry. Your loan has not been approved
```

What to do next

You see in the output that the monthly repayment value has an infinite value, which is not normal. In the next task, you look for the rule that generates the value.

[< Previous](#) | [Next >](#)

Task 3: Debugging a rule

You set breakpoints in a rule and run a debug session. When you find an error, you make changes to correct it.

About this task

When you run the decision service, it does not generate the correct monthly repayment rate. In the results, the rate is prefaced with the variable name `monthly repayment`. You create a query to find the rule that uses the variable in an action. Then, you place a breakpoint on the action in the rule, and another breakpoint on the code that computes the monthly repayment rate. You run a debugging session, which leads you to an error in the Java code, which you try to fix.

Step 1: Setting a breakpoint in a rule

You create a query to find the action rule that uses the variable `monthly repayment`, and then add a breakpoint to track the rule.

Procedure

1. Run the decision service as shown in [Task 1: Using automatic exception handling](#). In the results, you see that the report does not show the correct monthly repayment value. You want to find the source of this value to determine how to fix it.
2. In the Rule Explorer, expand the `Loan Validation Service` project.
3. Right-click the queries folder, and click **New > Query**.
4. Enter `Monthly` as the name in the New Query dialog, and click **Finish**.
5. Click **<enter a condition>** and add search parameters to form the following query:

```
Content
1 Find all business rules
2 such that the definition of each business rule contains "monthly"
```

Tip: The search function is case sensitive. Enter the name of the variable as you expect to find it in the action rule.

6. Click **Run query**, and click **Yes** in the Save Resource dialog. The Search view shows the repayment action rule because it contains the word `monthly`:

```
Monthly - 1 matches
🔍 repayment - [Loan Validation Scoring] - /Loan Validation Scoring/rules/computation
```

7. Double-click the action rule in the Search view to open it in the rule editor. You see that the first action, line 7, sets the value of the `monthly repayment` variable.
8. Open the **ARL** tab. It shows the Advanced Rule Language that is applied by the action rule.

The rule has three main parts: the definition of the variables, the condition beginning with `when`, and the action beginning with `then`. In the action part, you see that the rule calls the `loan.LoanUtil.getMonthlyRepayment` method to calculate the monthly repayment rate. You must step into this method.

9. Return to the rule editor, right-click line 7, and click **Toggle Breakpoint** in the pop-up menu. A breakpoint is displayed next to the line:


```
• 7 set the monthly repayment of 'the loan report' to
```


In the next step, you debug the action rule.

Step 2: Debugging the action rule

You run a debugging session. When it stops at breakpoints, you look for reasons for the `monthly repayment` error.

Procedure

1. Run the `run loan validation` configuration in the Debug Configurations.
2. Click **OK** in the Save and Launch dialog if it opens, and click **Yes** in the Confirm Perspective Switch dialog. The Debug perspective opens.
3. The debugger stops at the `validation` task in the `loanvalidation` ruleflow. Right-click the task, and click **Toggle Breakpoint** to remove the breakpoint. You do not need this breakpoint for this task.
4. Click the **Resume** button . The Content view now shows the repayment action rule.
5. Open the **Variables** tab and expand `report`. The `monthlyRepayment` variable shows `0.0`.

- Click the **Step Into** button . The debugger stops at `LoanUtil.java`. The Java code shows the following method at the breakpoint:

```
32 public static double getMonthlyRepayment(double amount, int numberOfMonth,
33     double yearlyRate) {
34     double i = yearlyRate / MonthsInYear;
35     double p = i * amount / Math.pow(1 + i, -numberOfMonth);
```

The function computes a monthly repayment rate by dividing `yearlyRate` by `MonthsInYear`. In the Variables view, you see that the value of `MonthsInYear` is 0. The error is that the function is dividing by 0, when it should divide by 12.

- Stop the debugging session by clicking the **Terminate** button .

Step 3: Fixing the error in the rule

You change the value of the `MonthsInYear` variable to 12, and run the decision service to check the results.

Procedure


- Switch to `LoanUtil.java` in the Rule perspective.
- Change the value of the `MonthsInYear` variable to 12:

```
20 public class LoanUtil implements Serializable {
21     private static final long serialVersionUID = 7764736178720624835L;
22     private static final short MonthsInYear= 12;
```

- Save your changes, and run the decision service normally by selecting `run loan validation` in the Run Configurations. The Console now shows the correct monthly repayment rate of 570.83:

```
Report: Valid data true Approved false
- Rate 0.055
- Monthly repayment 570.83
- Message Low risk loan
We are sorry. Your loan has not been approved
```

However, the decision service still does not approve the loan, so you have more work to do.

- Remove all the breakpoints because you no longer need them:
 - Open **Window > Show view > Other**.
 - In the filter field, type `breakpoints`, select **Breakpoints** in the list, and click **OK**.
 - Click the **Remove All Breakpoints** button .
- Save your work.

What to do next

You have corrected the monthly repayment error. In the next task, you debug a decision table to try to fix the approval error.

[< Previous](#) | [Next >](#)

Task 4: Debugging a decision table

You add breakpoints to an action rule and a decision table, and then run a debugging session to find and fix an error.

About this task

The decision service still doesn't approve the loan. You carefully read through the log in the Console, and it leads you to a rule that determines whether a loan is approved under certain conditions. You add a breakpoint to the rule, and look for the source of the decision in the condition. This search takes you to a decision table, where you insert breakpoints. Using the debugger, you trace the error to a value in the decision table, and then fix it.

Step 1: Setting breakpoints in an action rule

You run a query to find the rule that contains the message Your loan has not been approved. Then, you add breakpoints to the rule.

Procedure

1. Click **Run > Run Configurations**, open the run loan validation configuration, and click **Run**.

The results from running the decision service still show an error. The output indicates that the loan is low risk, but the decision service still does not approve the loan:

```
Report: Valid data true Approved false
- Rate 0.055
- Monthly repayment 570.83
- Message Low risk loan
We are sorry. Your loan has not been approved
```

You decide to search for the rule that makes the rejection message by running a query for the message.

2. In the Rule Explorer, expand the Loan Validation Service file.
3. Right-click the queries folder, and click **New > Query**. You create a query to find the rule that uses the following rejection message:

Your loan has not been approved.

Tip: If you do not see the **Query** command in the **New** menu, make sure that you are in the Rule perspective.

4. Enter Approval as the name in the New Query dialog, and click **Finish**.
5. Click **<enter a condition>** and add search parameters to create the following query to look for the rejection message:

```
1 Find all business rules
2 such that the definition of each business rule contains "Your loan has not been approved"
```

6. Click **Run query**, and click **Yes** in the Save Resource dialog. The Search view shows the approval action rule:

```
Approval - 1 matches
approval - [Loan Validation Determination] - /Loan Validation Determination/rules/eligibility
```

7. Double-click the action rule in the Search view to open it in the rule editor. The action depends on the grade that is given to the loan. If the grade does not equal A, B or C, the loan is rejected:

```
if
    'the grade' is one of { "A" , "B" , "C" }
then
    in 'the loan report', accept the loan with the message
    "Congratulations! Your loan has been approved" ;
else
    in 'the loan report', refuse the loan with the message "We are sorry.
    Your loan has not been approved" ;
```

8. Place breakpoints on both actions (lines 4 and 6) in the approval action rule:

```
3 then
4     in 'the loan report',
5 else
6     in 'the loan report',
```

Now you search for the rule that computes the grade condition for the approval rule.

Step 2: Setting breakpoints in a decision table

The approval action rule applies a decision that is based on the grade that is given to the loan. You decide to locate the rule that assigns grades to the loans. You create a query that finds a decision table, and then you add breakpoints to the decision table to observe it in a debugging session.

Procedure

1. Right-click the queries folder, and click **New > Query**.
2. Name the query gradeSet, and define the query by entering the following search parameters:

```
1 Find all business rules
2 such that each business rule may lead to a state where [ 'the grade' is not one of { "A", "B", "C" } ]
```

The query looks for a rule that can assign a grade that is not A, B or C, and therefore, causes the decision service to reject the loan.

3. Run the query. The Search view shows the grade decision table, and lists the rows that do not contain grade A, B or C:





4. Double-click the decision table or one of the rows in the list to open the table in the decision table editor.
5. Select the Grade column by clicking the column header cell. Right-click the A cell in row 1, and click **Toggle Breakpoint** in the pop-up menu. Every Grade cell now has a breakpoint:

Grade	
•	A
•	A
•	D
•	A
•	B
•	C
•	B
•	C
•	D
•	C
•	D
•	E

Step 3: Debugging the decision table

You use a debugging session to find an error in the decision table.

Procedure

1. Run the run loan validation configuration in the Debug Configurations. The debugger stops at row 3, which shows grade D and the message Low risk loan.
2. Click the **Resume** button . The debugger stops at the second breakpoint in the approval action rule. The loan is rejected because the grade is D. The error is coming from row 3 of the grade decision table.
3. Click the **Terminate** button  to stop the debugging session.

You contact the business user, who tells you to change the grade in row 3 to B. You update the decision table to correct the error.

4. Open the grade decision table in the Rule perspective, and change the grade in row 3 to B.
5. Save your changes, and run the decision service normally by selecting run loan validation in the Run Configurations. The results show that the loan is low risk, but the decision service still rejects the loan.

What to do next

You have corrected the error in the decision table, but you are still not getting the expected results. In the next task, you debug a ruleflow to fix the last error.

Task 5: Debugging a ruleflow

You debug a ruleflow to determine why the decision table does not approve the loan. When you find the cause, you reorganize the ruleflow to achieve the correct results.

About this task

The decision service still rejects the loan. The rules run correctly now, so you decide to test the flow of decisions among the rules. You start by running a debugging session that associates the error with two rules. You determine that the rules reside in the same rule task in a ruleflow. This arrangement prevents one rule from processing information from the other rule. You reorganize the ruleflow by adding a rule task to make sure that the condition part of one rule takes into account the values that are computed in the action part of the other rule.

Step 1: Debugging the ruleflow

You run a debugging session to determine the source of the error.

Procedure

1. Run the `run_loan_validation` configuration in the Debug Configurations. The debugger stops at row 3 of the grade decision table.
2. Open the **Agenda** tab in the Debug perspective. The tab shows two rules that are in the same eligibility rule task. The first entry points to the rule that is formed by row 3 in the grade decision table, and the second entry points to the approval action rule:

Name	Value
▶ ➡ eligibility.grade_3	AgendaRuleInfo (id=1971)
▶ ● eligibility.approval	AgendaRuleInfo (id=1039)

In the grade decision table, you see that an arrow points to the grade cell in row 3. The grade is B, and the message for the row is Low risk loan:

Grade	Message
➡ B	Low risk loan

3. Click the **Resume** button . The debugger stops on the `else` line in the approval action rule, which rejects the loan:

```
5 else
6 in 'the loan report', refuse the loan with the message "We are sorry. Your loan has not been approved" ;
```

You open the Variables view and look for the grade value. You see that it is B as set in the grade decision table, but the action rule still goes to the `else` condition, which is for any grade other than A, B or C. The grade value in the approval rule is not set to B. The grade value is set in the decision table, but it is not seen by the action rule. This disconnect between the rules indicates a ruleflow problem. Both rules are probably in the same rule task. You look to see which algorithm is used by the rule task.

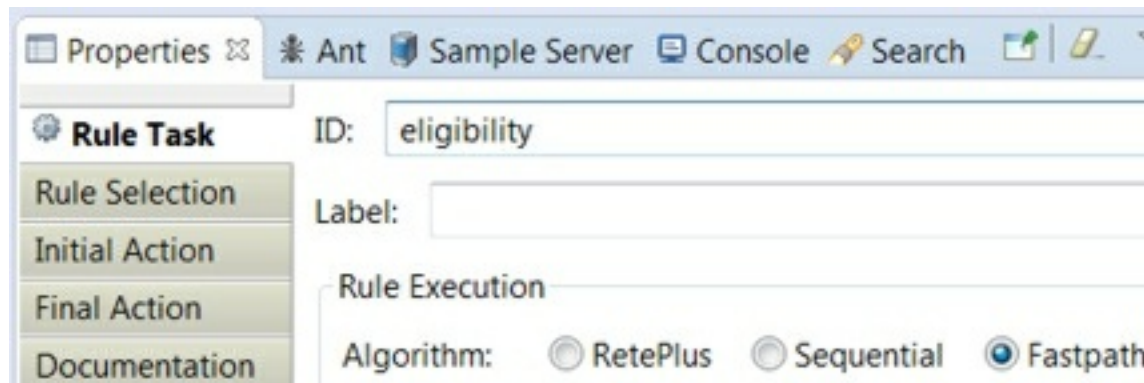
4. Click the **Terminate** button to stop the debugging session.

Step 2: Modifying the ruleflow

You open the eligibility rule task in the `loanvalidation` ruleflow. The task uses an algorithm that prevents the approval rule from seeing changes to data in another rule in the task. You decide to reorganize the ruleflow by creating a task, and moving the approval rule to the new task. Data can then flow from the old task to the new task, where the approval rule can then see the data changes.

Procedure

1. Open the `loanvalidation` ruleflow in the Rule perspective.
2. Select the eligibility rule task. The rules in the **Agenda** tab are prefixed with `eligibility`, which is the name of the rule task.
3. Open **Properties > Rule Selection**, and expand `eligibility`. You see that the task contains both the approval action rule and the grade decision table.
4. Open the **Rule Task** tab. You see that the Fastpath algorithm is selected:




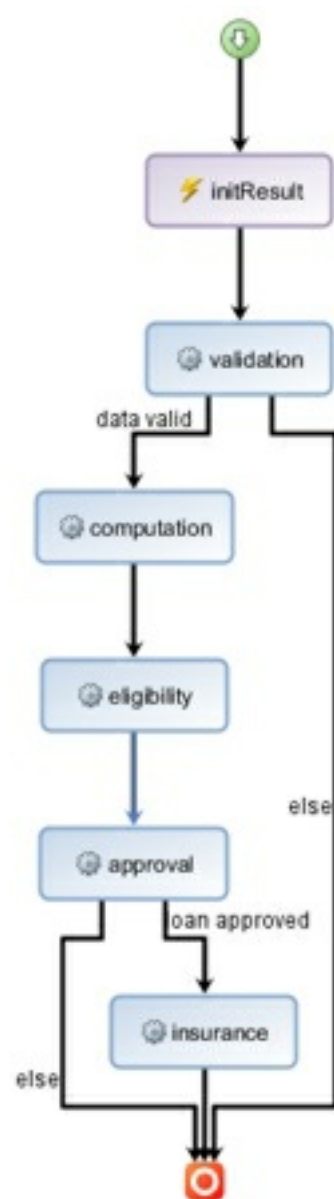
You know that the Fastpath algorithm does not support inference. The agenda is computed once in the rule task, but it is not changed if variables are modified by other rules in the task. The approval rule runs with an unset grade value, and the value is not changed by the grade decision table.

To fix the error, you decide to create a rule task, and move the approval action rule to the new task. Then, you organize the transitions so that data from the eligibility task flows to the new task. This arrangement allows the approval rule to see the data from the grade rule, which remains in the eligibility task.


5. Create a task node next to the insurance task in the ruleflow (see [Ruleflows](#)).

Enter the following parameters in the properties for the new task:

- Rule Task ID: approval
 - Rule Selection: eligibility.approval
6. Click the transition line from the eligibility task to the end node, and reset it to run from the approval task to the end node.
 7. Click the transition line from the eligibility task to the insurance task, and reset it to run from the approval task to the insurance task.
 8. Create a transition line between the eligibility task and the approval task.
 9. Click the **Layout All Nodes** button : The ruleflow now looks as follows:



10. Edit the eligibility task to remove the approval action rule from the **Rule Selection** tab. Remove the eligibility package, and then import all the eligibility rules except the approval rule:

 eligibility.checkCreditScore
 eligibility.checkIncome
 eligibility.grade

Now, the grade decision table runs in the eligibility task, and its grade value passes to the approval rule in the approval task.

11. Save your work.
12. Run the decision service by using the run loan validation configuration in the Run Configurations. The decision service runs correctly, and produces the expected report:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
  - Yearly income 20,000
  - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved true
  - Rate 0.055
  - Monthly repayment 570.83
  - Insurance required true Insurance rate 2%
  - Message Low risk loan
Congratulations! Your loan has been approved
```

13. Remove all the breakpoints as shown at the end of Task 3, and save your changes. Your decision service now works correctly. You can now make the new version of the decision service available to the business user for further review.

Tip: In this tutorial, you created a new rule task in a ruleflow, and changed the rule selection. This approach is not always possible. Alternatively, you can use inference by selecting the RetePlus algorithm in the Rule Task properties of the eligibility task (see [RetePlus algorithm](#)).

Results

You have completed the tutorial. It showed you how to add breakpoints, and use a debugging session to follow the breakpoints through a decision service to find problems in rules.

[< Previous](#)

[< Previous](#) | [Next >](#)

Getting started with decision modeling

This tutorial shows the basics of creating a decision model service. You do all the modeling and authoring of a decision service in the Decision Center Business console.

Learning objectives

You do the following tasks:

- Create a decision model service.
- Author the decision logic.
- Create data types.
- Edit existing data and decision nodes.
- Test and deploy a decision model service.

Time required

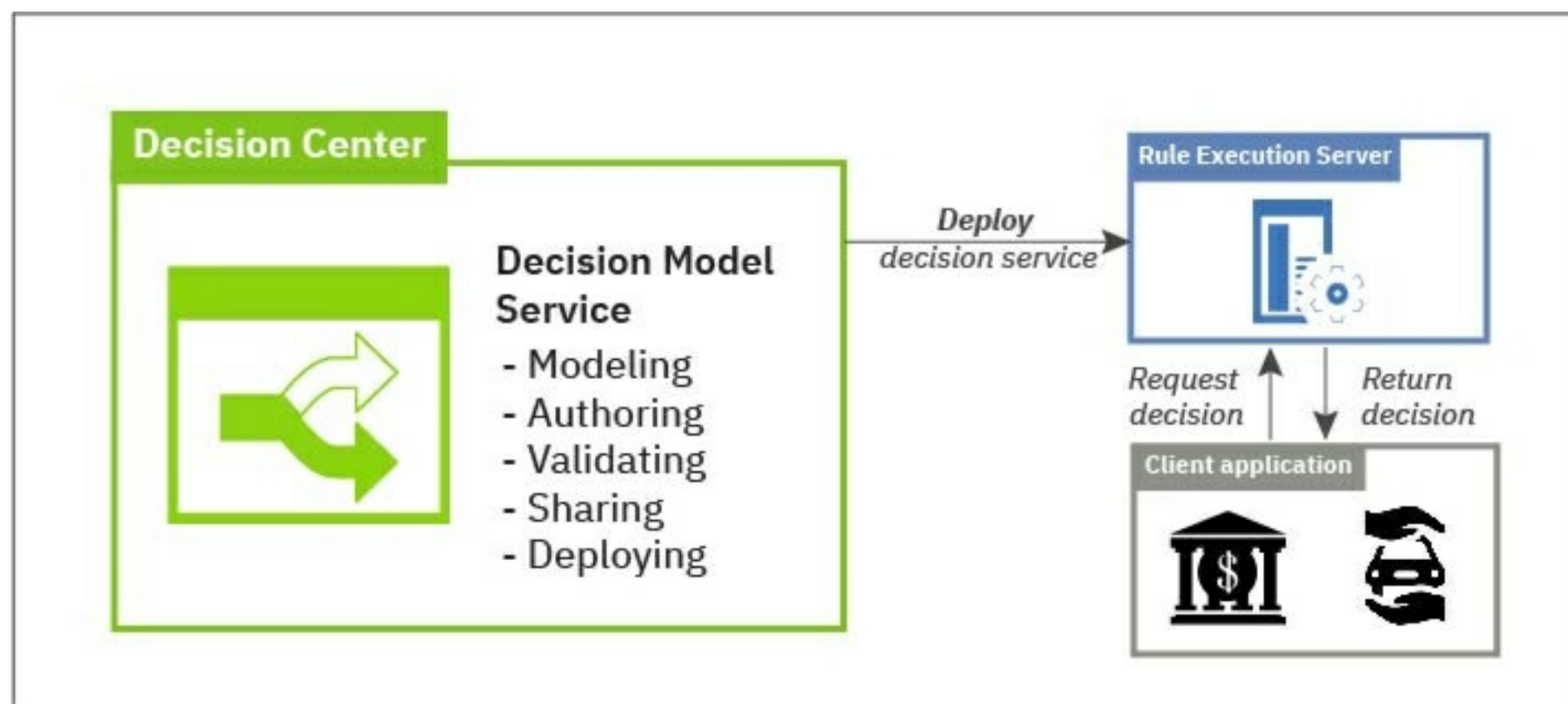
40 minutes

[< Previous](#) | [Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

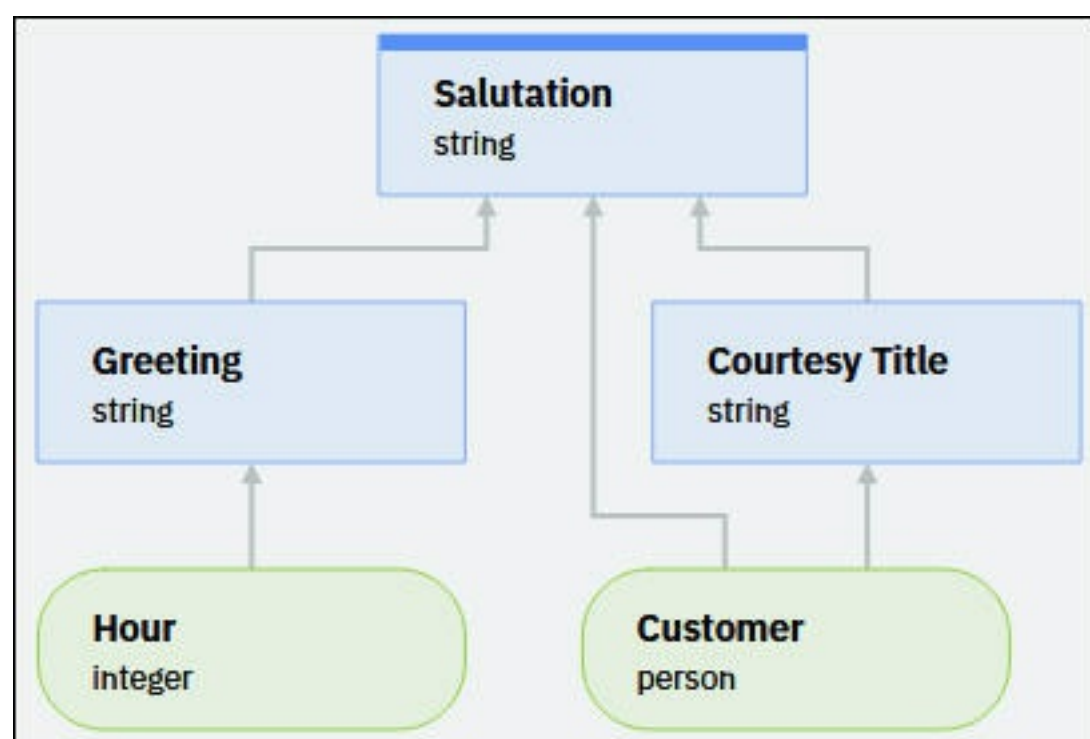
In the Decision Center Business console, you can create both the model and the logic of a decision service, and then deploy it to the execution environment, where it can be called by the client application.



The term *decision model service* identifies a decision service in which you create the model in the Business console. In a decision model service, you use a diagram to create data and decision nodes, and then author the decision logic of each decision node.

You create a *main* branch in this tutorial. The main branch serves as the foundation of a decision model service. After you complete a main branch, you should work in branches based on the main branch to avoid introducing errors to the original branch.

This tutorial guides you through creating, testing, and deploying a decision model service, and highlights differences with regular decision services. The decision model service provides a salutation to a customer. Initially, your salutation says Hello to the customer. Then, you add nodes to provide a salutation based on time of day, the gender of the customer, and any degree the customer might have. The final diagram looks as follows:



Audience

This tutorial is intended for business users who want to create decision model services in the Business console.

Prerequisites

The sample decision service and tutorial files are in English. You must use a supported browser (see [IBM® Operational Decision Manager on Cloud - Detailed System Requirements](#).)

More information

For additional information on decision modeling in Decision Center, see the following sections:

- [Modeling decisions in the Business console](#)

- [Introducing the Business console](#)

Lessons in this tutorial

[**Task 1: Creating the decision model service**](#)

In this task, you create the project, the first nodes in the diagram, and the initial decision logic for your decision model service.

[**Task 2: Adding a decision node**](#)

You add a decision node that affects the final decision.

[**Task 3: Creating a data type**](#)

You create a new type of data, and a decision node that uses the new type.

[**Task 4: Modifying an existing decision**](#)

You create an enumeration data type, add an attribute to the data type that you created in the previous task, and edit the model to take these changes into account.

[**Task 5: Testing and deploying**](#)

You prepare and run a test suite, and then deploy a RuleApp from your decision model service to the execution environment.

[< Previous](#) | [Next >](#)

Task 1: Creating the decision model service

In this task, you create the project, the first nodes in the diagram, and the initial decision logic for your decision model service.

About this task

In a decision model service, you use a diagram to model data and decision nodes, and then author the decision logic of each decision node. Data nodes represent the data that is received from the client application, and decision nodes contain the logic in the form of business rules.


In this task, the data node is for the customer name, and the decision node returns a salutation with two possible replies:

- Says Hello to the customer and adds the customer's name.
- Replies with a default statement if no name is provided.

Step 1: Creating the project

You first create the **Salutation2** project for your decision model service. The project contains all the information required to build and deploy a decision service. The project can also be exported as a .zip file.

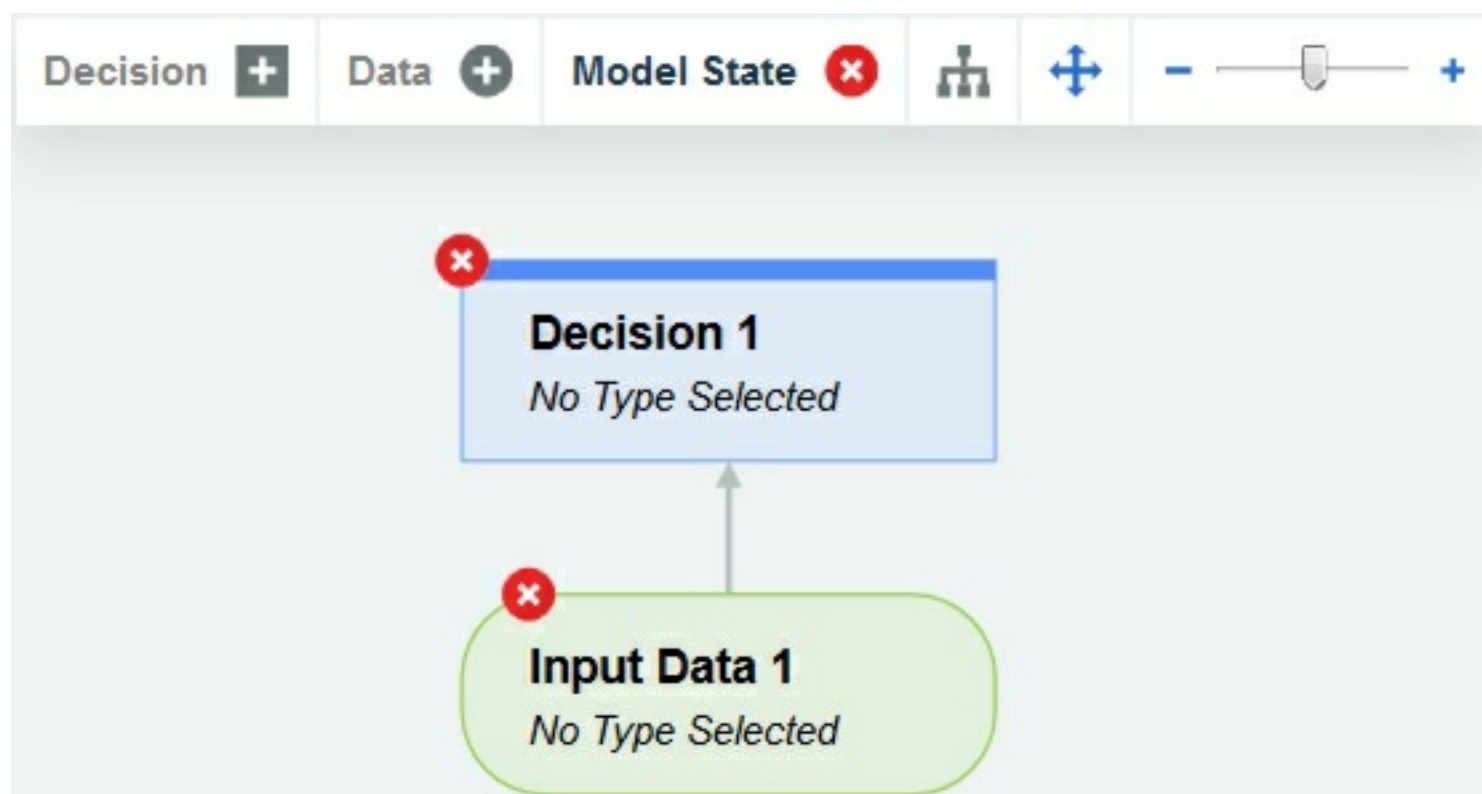
Procedure

1. Log in to your instance of Operational Decision Manager on Cloud, and open the Decision Center Business console.
2. In the **Library** tab, click the **Create New Decision Model Service** button .
3. Enter **Salutation2** as the project name for your decision model service. For the description, you can enter the following information:

Getting started with decision model tutorial. Decision model service composes greeting that includes the time, degree, and gender.

Note: This tutorial does not cover the decision governance framework, so you do not have to select it.

4. Click **Create**.
5. Make sure that the **Branches** tab is selected, and then click **main** to access the main branch. You can see the modeling diagram, which contains a decision node and a data node:




Decision Center assigns a new version number to the decision model every time you save it. The decision model works as one versionable element. Changes to individual rules or nodes can be consulted through the version of the decision model.

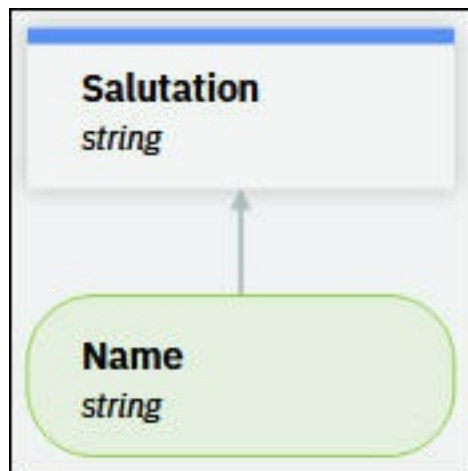
Note: The nodes are in an error state until you give them a type.

Step 2: Creating the initial nodes

You create and define the relationship between the data that is provided by the client application, and the decision that is returned by your decision model service. Initially, you want your decision model to accept a customer name as input data, and provide a salutation to the customer.

Procedure

1. Click the **Edit** button  below the tab names, just above the diagram. Version 1.0 of the decision model diagram is now editable.
2. In the diagram, click the default data node **Input Data 1**.
3. In the **Model** tab, change the **Output variable name** of the node to **Name**. You can do this in **Output variable name** field or the name field just above the **Details** field.
4. Under **Output type**, click **Make a selection**, and then click **string** to select it as the type to handle the customer name as text. The type determines what the node can receive as input and send as output. Data modeling requires you to carefully assign the correct type to each node.
5. In the diagram, click once on the **Decision 1** decision node.
6. Name your new decision node **Salutation**, and choose **string** as the type. Your decision model now has a data node feeding a top-level decision node:



Step 3: Creating the decision logic

You have the minimum node structure: one data node feeding one decision node. The decision node accepts input data from the nodes that point to it, and produces a decision that is based on the logic that you author.

About this task

Authoring the decision logic involves two aspects:

- Writing business rules that provide a decision that is usually based on conditions.
- Writing sufficient business rules to cover the possible cases that the client application encounter.

The nodes in the diagram are available in the business rules as variables of the form the 'node name'. For now, you have the **Name** data node available as 'the name', and the **Salutation** decision node as 'the salutation' in the rule and decision table editors.

Procedure

1. Select the **Salutation** node in the diagram to edit its content.
2. In the **Model** tab, under **Decision logic**, click **Add rule**. The wizard proposes a starting point for the rule that is based on conditions that you might want to apply to the data that it receives.
3. There are no conditions to consider for the name, so click **Create rule**.
4. In the middle panel, change the name from **rule-1** to **salutation rule**.
5. Click the `<a string>` placeholder, and select '**the name**' from the proposed options that appear. In this state, your decision model service simply returns the name provided as input. To add more value to the process, change the rule as follows:

```
set decision to "Hello " + 'the name' + "!" ;
```

Note: Add a space in "Hello " to insert a space before the name.

The keyword **decision** is available in each decision node to designate the output value of the node. Replacing with 'the salutation' is exactly equivalent.

6. Still under **Decision logic**, click **Set output default**, and replace the rule as follows:


```
set decision to "We don't have correct information to greet you!";
```

7. Click **Save and Continue** at the top of the window, and then **Create New Version** to save version 1.1 of your decision model service.

Step 4: Validating the decision model

The decision modeling tool enables you to validate your model immediately. You run your decision model service on data that you provide.

Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  beside **Data sets**.
- 3. Enter Jones in the **Name** field for your data set, and click **Run**. The validation report shows the output from running your decision model, and information that is useful when troubleshooting:


▼ **Decision outputs**

Salutation output	Salutation type
"Hello Jones!"	string

▼ **Messages**

No output messages to display.

▼ **Run history**

Node	Type	Rule interaction	Output	Rules
▼  Salutati	string	sequence	"Hello Jones!"	2

- 4. Click the three dots next to **Name**, and click **Delete item** to clear the name from the data set.

▼ **Name**

Jones

Delete item

- 5. Click **Run** again. The report shows the default message.

"We don't have correct information to greet you!"

- 6. Click **Cancel**, and then **Yes** because the next tasks use different data.

What to do next

In the next task, you add to the salutation by making the final decision dependent on another decision.

Task 2: Adding a decision node

You add a decision node that affects the final decision.

About this task




You want your salutation to reflect the time of day. Instead of just saying Hello, for example, you want it to say Good morning. To do this, you create a new decision node and link it to the existing decision node.

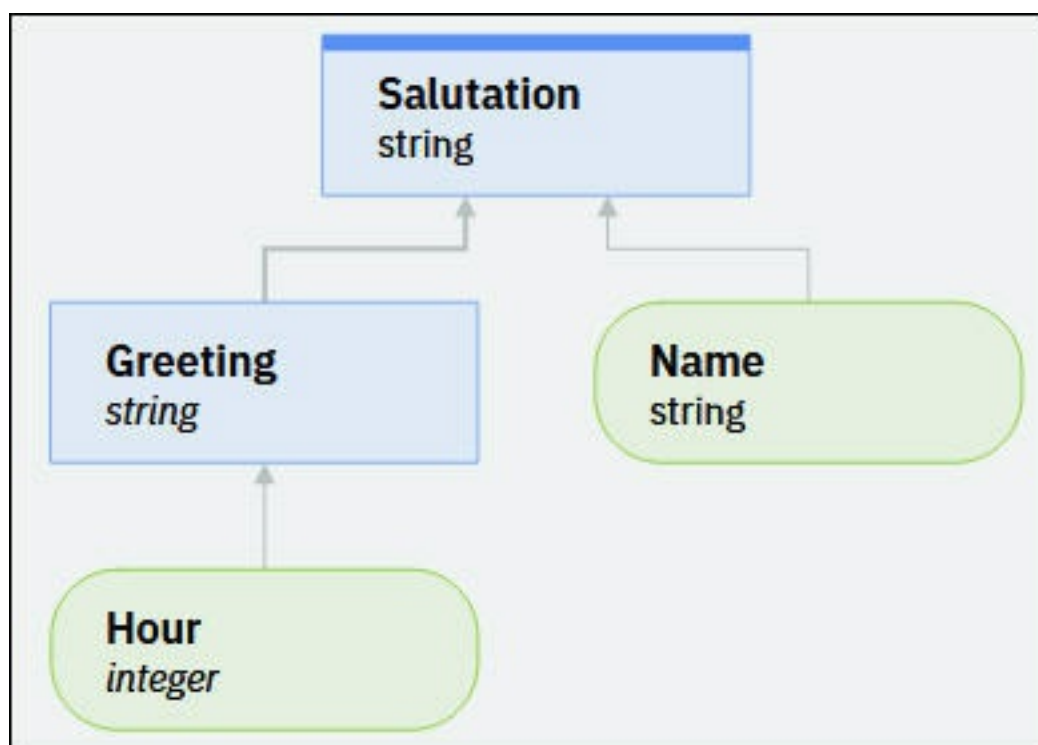
Each node in your diagram *feeds* the nodes to which it points. In other words, a node is *dependent* on the nodes that feed it. The new node feeds the existing decision node.

Step 1: Adding nodes to the diagram

You create a decision node, called **Greeting**, that uses **Hour** as input. Then, you link this decision node so that it feeds data to the **Salutation** node.

Procedure

1. Click the **Edit** button  to continue editing the decision model.
2. In the floating toolbar, click **Data** to add a data node.
3. Name the new node **Hour**, and set its type to integer.
4. Hover over the **Hour** data node, click the **Add** button , and then click **Decision** to create another decision node.
5. Name the new node **Greeting**, and set its type to string.
6. Hover your mouse over the **Greeting** decision node, click and hold the **Add** button , and drag and drop the node onto the **Salutation** node. **Salutation** now depends on both **Greeting** and **Name**. The diagram looks as follows:



Step 2: Adding the decision logic

You add the logic that decides which greeting is appropriate based on the provided time of day. Because there are many possibilities, you create a decision table, which is ideal for grouping business rules that have a common structure.

Procedure

1. In the diagram, click once on the **Greeting** node to select it.
2. In the **Model** tab, under **Decision logic**, click **Add table**.
3. In the displayed panel, select '**the hour**' as the data on which you place conditions, and click **Create table** in the lower right corner of the window. You now have an empty decision table that has one condition column and one decision column.
4. Rename your decision table **greeting table**, and then double-click the cells and enter the following values:

	Hour		Greeting
	min	max	
1	0	5	Good night
2	5	12	Good morning
3	12	18	Good afternoon
4	18	23	Good evening
5	23	24	Good night

- 5. Close the decision table panel.
- 6. Click the **Salutation** node in the diagram.
- 7. Under **Decision logic**, click **salutation rule** and change it as follows:

```
set decision to 'the greeting' + " " + 'the name' + "!" ;
```


Notice how the **Greeting** node is represented as 'the greeting', and the **Name** node as 'the name'.

- 8. Click **Save and Continue**, and then **Create New Version**.

Step 3: Validating the changes

After each modification, you can validate your decision model.

Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  beside **Data sets**.
- 3. Enter Jones in the **Name** field, and enter 20 in the **Hour** field.
- 4. Click **Run**. The validation report displays the Salutation output:

```
Good evening Jones!
```

- 5. Click **Cancel**, and then **Yes**.

What to do next

In the next task, you create a new type to make your decision model easier to read and maintain.

Task 3: Creating a data type

You create a new type of data, and a decision node that uses the new type.

About this task



Creating a custom type is important to creating a data model because it groups related data under one entity. By doing so, you produce a diagram that is easier to read and maintain. A second benefit is that you don't have to change the diagram when you add information to a custom type.

In this task, you modify your decision model so that it greets the customer with a courtesy title instead of just the customer's name. The courtesy title is either Mr . or Mrs . , depending on the gender of the customer.

Step 1: Creating a new data type

Your decision model already uses the name of the customer. Now, you want to add the gender of the customer. You start by creating a **person** data type that groups these attributes together.

Procedure

1. Click the **Edit** button  to continue editing your decision model.
2. In the **Types** tab, click the **Add** button to create a new data type.
3. Click **Add type**, and name your new data type **person**.
4. Click the **Add** button  twice to add the following attributes to the **person** data type:

Name

person



Attributes

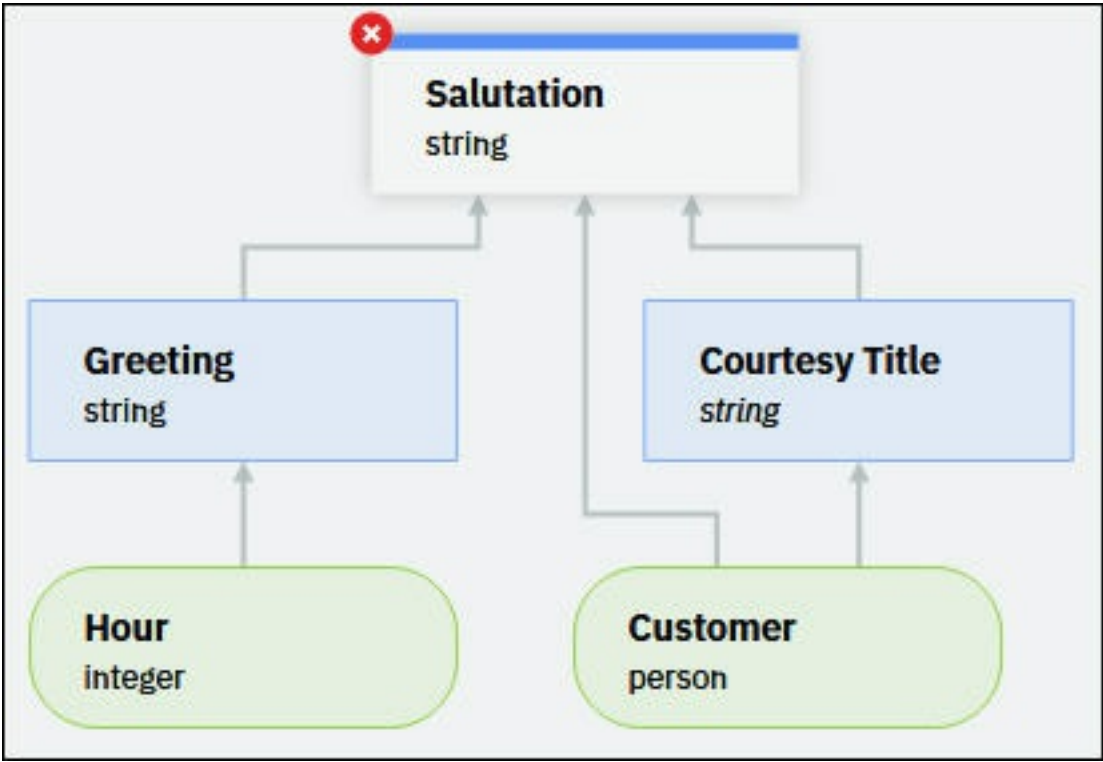
Name	Type
name	string
gender	string

Step 2: Creating a data node that uses the new type

You create a new decision node to greet the customer with a courtesy title instead of just the name. Your new decision node uses the new **person** data type.

Procedure

1. Click the **Model** tab.
2. Click the **Name** data node. Change its name to **Customer**, and its type to **person**. The **Salutation** node shows an error.
3. Hover over the **Customer** data node, click the **Add** button , and click **Decision** to create another decision node.
4. Name the new node **Courtesy Title**, and set its type to string.
5. Hover over the **Courtesy Title** decision node, click and hold the **Add** button , and drag and drop the new node onto the **Salutation** node. The node structure in the diagram is now in its final state, but it displays an error, which you fix in the next step. Along with the **Greeting** and **Courtesy Title** nodes, the **Salutation** node is now dependent on the **Customer** data node for the name of the customer. **Courtesy Title** is also dependent on the **Customer** data node for the gender of the customer:



Step 3: Adding and adjusting the decision logic

You create the logic of your new decision node and adjust the top-level node. You also see how these nodes handle the custom data type.

Procedure

- 1. In the diagram, click once on the **Courtesy Title** node to select it.
- 2. In the **Model** tab, under **Decision logic**, click **Add table**.
- 3. In the displayed panel, select the gender of 'the customer' as input to the node on which you place conditions, and then click **Create table**. You now have an empty decision table that contains a condition column and a decision column.
- 4. Change the name of your decision table to **courtesy table**, and enter the following values:

gender	Courtesy Title
Male	Mr.
Female	Mrs.

- 5. Close the decision table panel, and click the **Salutation** node in the diagram.
- 6. Click **salutation rule** and change it as follows:

```
set decision to 'the greeting' + " " + 'the courtesy title' + " " + the
name of 'the customer' + "!" ;
```

Notice how the name of the customer is usable in the rules as the name of 'the customer'.

- 7. Click **Save and Continue**, and then **Create New Version**.

Step 4: Validating the changes

You add data and validate your changes.

Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button (+) to create a new data set.
- 3. Enter the following data:
 - Name: Jones
 - Gender: Female
 - Hour: 20
- 4. Click **Run**. The validation report displays the Salutation output:

```
Good evening Mrs. Jones!
```

- 5. Click **Cancel**, and then **Yes**.

What to do next

In the next task, you add another type of data and explore some other features of a decision model service.

Task 4: Modifying an existing decision

You create an enumeration data type, add an attribute to the data type that you created in the previous task, and edit the model to take these changes into account.

About this task




An *enumeration* is a data type in which you establish all the possible values beforehand. You should use enumerations over text entries whenever possible. They facilitate data entry in the editors, the correctness of your rules is verified immediately, and gaps in decision tables are identified.

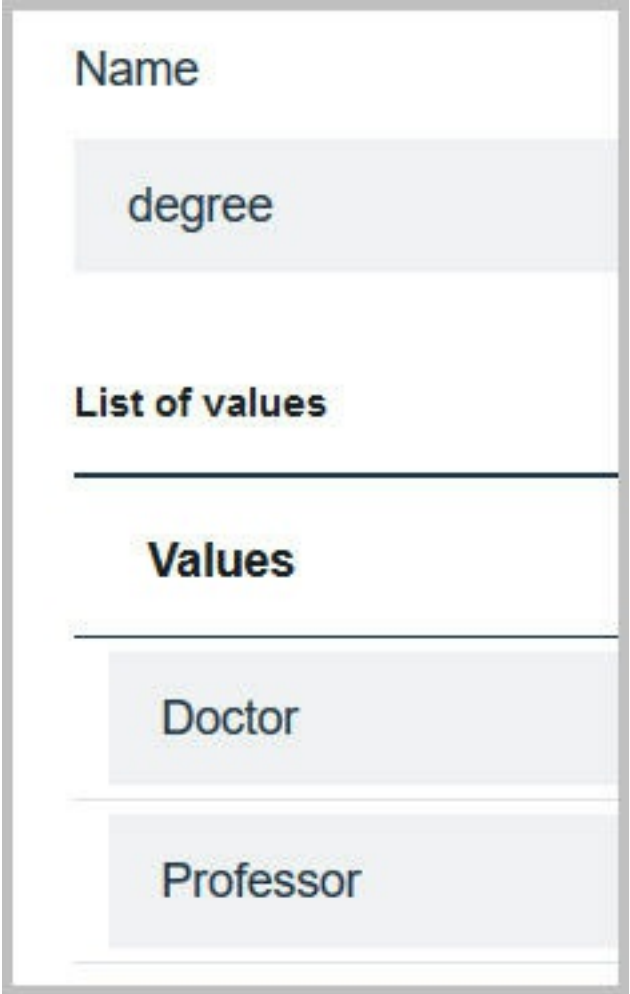
In this task, you create an enumeration that contains values that correspond to an academic degree, either Doctor or Professor. Then, you edit the decision table and rule to take this enumeration into account.

Step 1: Adding an enumeration

You add an enumeration to the decision model.

Procedure

1. Click the **Edit** button  to continue editing of your decision model.
2. In the **Types** tab, click the **Add** button  to create a new data type.
3. Click **Add enumeration type**, and name your new data type **degree**.
4. Click the **Add** button  twice to add the following values to your enumeration:



The screenshot shows a dialog box for adding an enumeration type. It has a 'Name' field at the top with the text 'degree'. Below it is a section titled 'List of values' which is expanded to show a list of values. The list has a header 'Values' and contains two entries: 'Doctor' and 'Professor', each in its own box.

5. Still in the **Types** tab, click the **person** data type. Add an attributed named **degree**, and give it the type **degree**.
6. Close the panel.

Step 2: Adjusting the logic

You define the logic for the degree condition.

Procedure

1. In the diagram, click the **Courtesy Title** node, and open **courtesy table**.
2. Right-click the header of the **gender** column, and select **Insert column > Condition before**.
3. In the new column, change the heading to **Degree**.
4. Right-click the heading of the **Degree** column, and click **Define column**.
5. In the editor, add the following condition:

```
the degree of 'the customer' is <an object>
```

6. Click **Define**.

- 7. In the table, right-click the number 1 of the first row and click **Insert row > Above**. Repeat to create a second empty row.
- 8. Complete the table with the following values. Notice that you can select **Doctor** and **Professor** from the drop-down list:



	Degree ▼	Gender ▼	Courtesy Title
1	Doctor		Dr.
2	Professor		Prof.
3		Male	Mr.
4		Female	Mrs.

- 9. Close the panel.
- 10. Click **Save and Continue**, and then **Create New Version**.

Step 3: Validating the changes

You validate your changes. An important aspect of decision modeling is to ensure that you cover situations where different rules apply. This is referred to as rule interaction. Validating as you go helps identify these situations.

Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  to create a data set.
- 3. Enter the following data:
 - Name: Jones
 - Gender: Female
 - Degree: Professor
 - Hour: 20
- 4. Save your data set. Click the **Edit** button  next to **New input**, and replace it with **Prof Jones**.
- 5. Click **Run**. The validation report displays the **Salutation** output:

Good evening Mrs. Jones!

In this situation, customer Jones corresponds to both Mrs. and Prof. You decide to give priority to the degree.

- 6. Click the **Model** tab.
- 7. In the diagram, click the **Courtesy Title** node.
- 8. In the **Decision logic** section, click **Rules are applied in sequence** and change it to **First rule applies**. You might have to scroll down to reach the setting.
- 9. Return to the **Validate** tab and run the same data set. This time the validation report displays the following message:


Good evening Prof. Jones!

- 10. Click **Save and Close**, and then **Create New Version**.

Task 4: Comparing versions

Each time you save, Decision Center creates a new version of the decision model. You can compare versions to see the progress, and restore previous versions when needed.

Procedure

- 1. Click the **Open the decision model view** button  next to the edit button.
- 2. Click **Compare**. The comparison window opens with the current version already selected.
- 3. Select version 1.1, and then click **Compare**. All the changes made since the initial version of the decision model are shown.
- 4. Click **main** in the breadcrumbs to return to the **Decision Model** tab.

What to do next

In the next step, you create a test suite, and then deploy your decision model service.

[< Previous](#) | [Next >](#)

Task 5: Testing and deploying

You prepare and run a test suite, and then deploy a RuleApp from your decision model service to the execution environment.


About this task

Now that your decision model service is complete, you create a test suite to do regression testing, and then deploy the service as a RuleApp to the execution environment.

Step 1: Generating the scenario file

The first step to creating a test suite is to generate an empty *scenario file*. Scenarios represent real or fictitious use cases that you use to validate the behavior of your rules. Each scenario contains all the information that is needed to run your rules properly.

Procedure

1. Click the **Tests** tab.
2. Click the **Generate Scenario File** button .
3. Name your scenario file **salutation**.
4. In the tests to include, select **Salutation**.
5. Click **Download**, and save the file to a location on your computer.

Step 2: Entering the scenarios and expected results

You create scenarios of representative cases for the client application. Each row in the first spreadsheet represents a scenario. Its columns indicate where to put the data for the scenarios. In the second spreadsheet, you enter the expected results of each scenario.

Procedure

1. Open the scenario file on your computer.
2. Enable editing if required, and enter the following scenarios in the **Scenarios** tab:

Scenario ID	description	customer			hour
		degree	gender	name	
Mr Jones			Male	Jones	10
Mrs Jones			Female	Jones	16
Dr Jones		Doctor	Female	Jones	21
Prof Jones		Professor	Male	Jones	3

3. At the bottom of the spreadsheet, click the **Expected Results** tab to open it.
4. Enter the following expected results for the scenarios:


Scenario ID	Salutation equals
Mr Jones	Good morning Mr. Jones!
Mrs Jones	Good afternoon Mrs. Jones!
Dr Jones	Good evening Dr. Jones!
Prof Jones	Good night Prof. Jones!

5. Save and close the file.

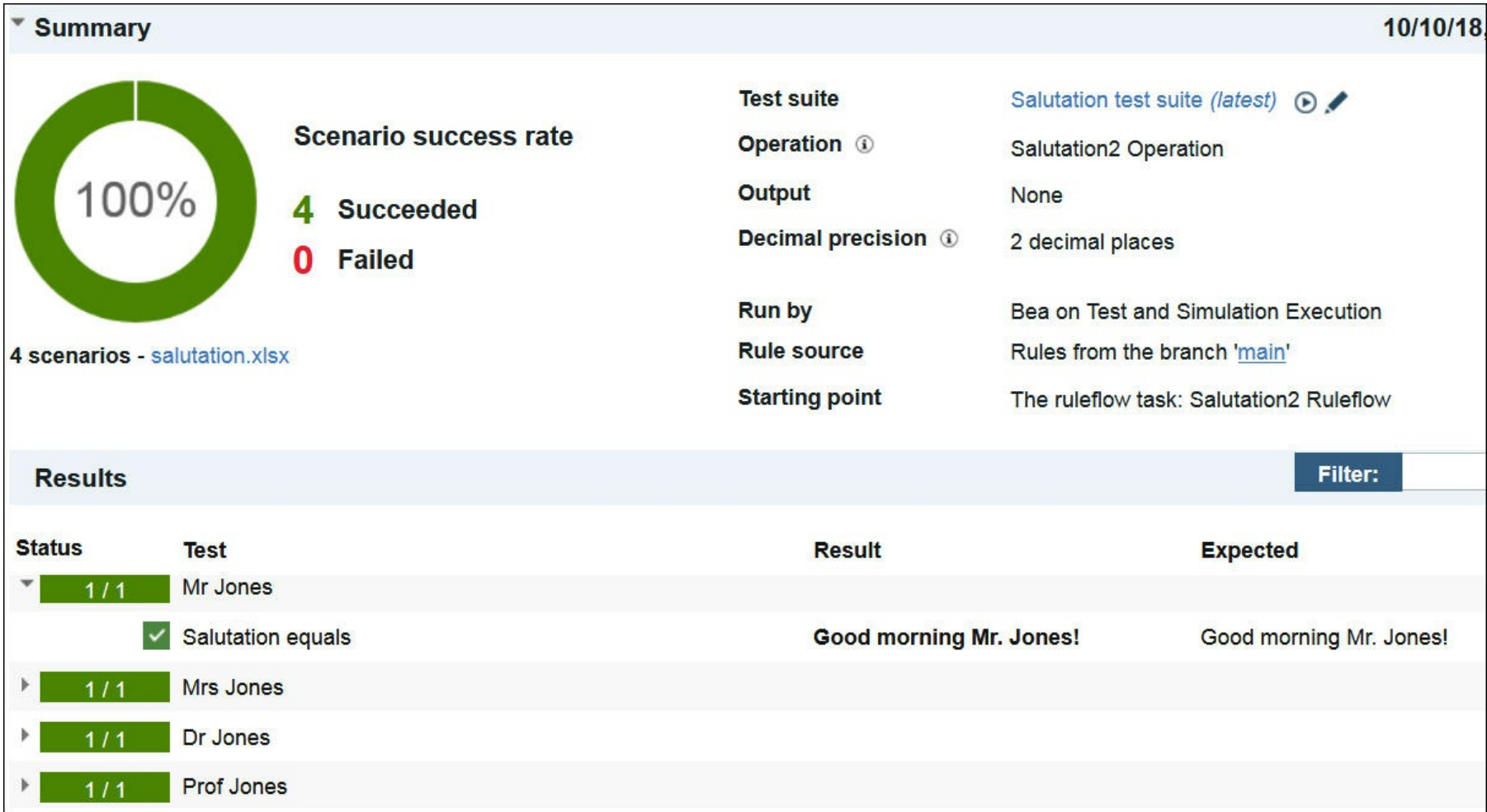
Step 3: Creating and running the test suite

You run the test suite to compare your expected results with the actual results.

Procedure

1. In the Business console, click **main** in the breadcrumbs.
2. On the **Tests** tab, click the **Add** button  to create a new test suite.
3. Name your test suite **Salutation test suite**.

- 4. In the **File to use** section, click **Choose** and upload the **salutation** scenario file from your computer.
- 5. Click **Save and Run > Create New Version**, and then **OK** to the message on the progress report.
- 6. When the status of the test is complete, click the report name. You see that the scenarios ran successfully. The expected results that you entered are the same as the returned results.



- 7. Close the report.

(Optional) Step 4: Deploying the decision model service

About this task

A *deployment configuration* contains all the information needed to deploy a *RuleApp* from your decision model service, including the target server and the decision operation. The *decision operation* defines the rules for testing or deployment, and the input and output parameters. In a decision model service, the decision operation is created automatically and updated every time you save.

The RuleApp contains the *ruleset* developed in the decision model service. The execution environment runs the ruleset for the client application (see [Deployment configurations](#)).

Restriction: You must be a release manager to do this step. You cannot do this step if you are a business user.

Procedure

- 1. On the **Library** tab, click the **Salutation2** decision service, and then click **main** on the **Branches** tab.
- 2. Click the **Deployments** tab. The **Salutation2 Deployment** deployment configuration is automatically generated when you save the project.
- 3. Click **Salutation2 Deployment**. Look at the different subtabs to understand what is contained in the deployment configuration.
- 4. In the top banner, click **Deploy**.
- 5. Select **Server Development Environment** as the target server, and then click **Deploy**.
- 6. Click **OK** to close the status report message.
- 7. When the status shows a check mark, click the report link to show the results of the deployment. The report shows that the deployment was successful.
- 8. Click **Close**, and then log out.

Results

You have completed this tutorial, and learned how to create, test, and deploy a decision model service.

[< Previous](#)

[< Previous](#) | [Next >](#)

Creating a ruleflow in the Business console

This tutorial shows you how to create a ruleflow in a decision service in the Decision Center Business console.

Learning objectives

You do the following tasks:

- Explore the contents of a ruleflow.
- Create a ruleflow.
- Modify an existing ruleflow to call a subflow.
- Specify an order for running rules.
- Create a decision operation that uses the new ruleflow.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Best practices

This tutorial includes the following best practices for creating a ruleflow in the Business console:

- Create a subfolder in a project folder to group similar rules. [Example...](#)
- Create a ruleflow within a ruleflow to sequence certain rule together. [Example...](#)
- Select properties that optimize the running of the rules. [Example...](#)

Time required

40 minutes

[< Previous](#) | [Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a ruleflow in implementing a new business policy. You work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans.

The decision service has two folders that contain business rules for different decisions:

Folder	Decision
validation	Determines whether the submitted data is valid.
eligibility	Determines whether the loan can be approved.

The decision service also contains a ruleflow that states the sequence for running the rules. If a loan request passes the rule in the validation folder, the ruleflow directs the loan data to the rules in the eligibility folder. Otherwise, the decision service rejects the loan.

The loan company wants you to add an eligibility policy that requires the borrower’s credit score to be at least 30 times bigger than the requested loan duration. To implement the policy, you create an action rule. Because of the growing number of eligibility rules, you decide to implement separate testing at this decision point. To do so, you create a separate ruleflow for eligibility, and a decision operation that uses the ruleflow.

Audience

The tutorial introduces is for users who want to create ruleflows in the Decision Center Business console.

Prerequisites

You work on a branch that you create in the decision service that is provided in the Miniloan Service sample project. The project is available in the Rule Designer section of the cloud portal.

To publish the decision service and create your branch, see [Preparing and removing the tutorial project](#).

Important:

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

More information

If you are not familiar with the Decision Center Business console or the ruleflows, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Editing ruleflows](#)

Lessons in this tutorial

[Task 1: Exploring a ruleflow](#)

You explore a ruleflow to determine how it works, and create an action rule for a business policy.

[Task 2: Creating a ruleflow](#)

You create a ruleflow, and modify another ruleflow.

[Task 3: Setting rule task properties](#)

You change the properties that define activities in a task.

[Task 4: Applying a ruleflow in a decision operation](#)

You create a decision operation that uses the new ruleflow, and test the ruleflow.

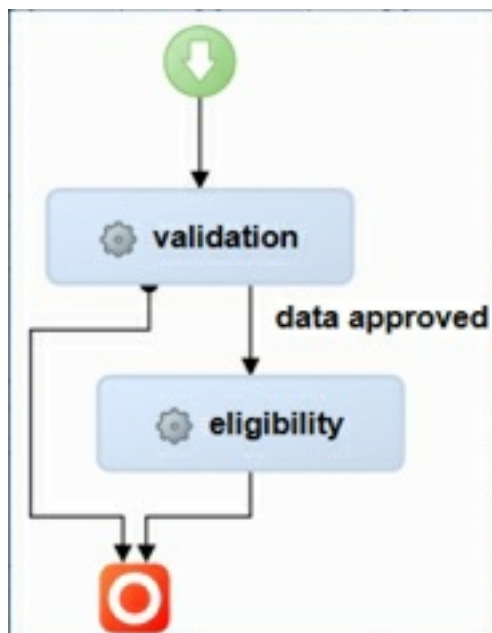
Task 1: Exploring a ruleflow

You explore a ruleflow to determine how it works, and create an action rule for a business policy.

About this task

A decision service keeps its decision points in projects and folders. The Miniloan Service decision service contains one project, and two folders for different decision points. The rule in the validation folder determines whether incoming loan data is valid, and the rules in the eligibility folder apply various conditions for eligibility.

The decision service also contains a ruleflow that states the sequence for applying the rules:



The ruleflow starts by running the validation rule. If the data from a loan request passes the validation rule, the ruleflow directs the data to the eligibility rules. However, if the validation decision point does not approve the data, the ruleflow skips the eligibility decision point, and the decision service stops processing the loan request.

You start by opening the branch that you created for this tutorial (see the software prerequisites in [Before you start](#)). Then, you create the action rule that applies the new policy.

Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Step 1: Opening your branch

You open the branch that you created for this project in the Miniloan Service decision service.

Procedure

1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Hover over the Miniloan Service box, and click anywhere except the name.
4. Click **My Tutorial** to open your branch. If you gave your branch a different name, click that name to open your branch.
5. Open the **Decision Artifacts** tab. It contains the following components:



Tip: To see all the components, click **All Types** at the top of the tab, and select **All Types**.

Step 2: Taking a snapshot

Before you continue, you take a snapshot to capture the initial state of the main branch of the decision service. You cannot edit the contents of the snapshot, but you can compare it to other versions of the branch. You can also use the snapshot to restore the branch to its initial state.

Procedure

1. Return to the list of artifacts, and click **Take Snapshot**.
2. Name your snapshot something like My snapshot. Do not reuse the name of an existing snapshot. For example, you can personalize your snapshot by using the initials of your name.
3. Click **Create**

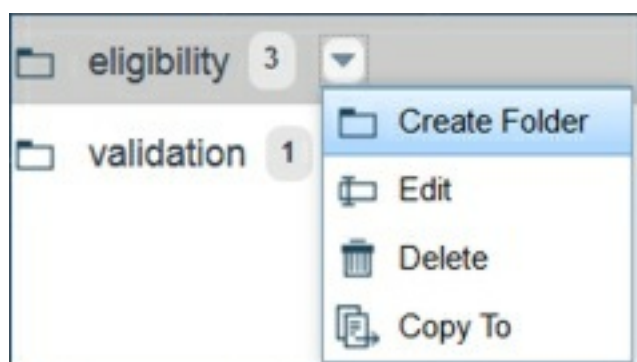
You can find the new snapshot in the **Snapshots** tab.


Step 3: Adding an eligibility rule

The new policy is an eligibility constraint. To apply it, you create a folder and an action rule in the eligibility folder.

Procedure

1. In the **Decision Artifacts** tab, hover over the eligibility folder, click the down arrow, and select **Create Folder** in the drop-down menu.



2. Enter duration as the name of the folder, and then click **Create**. The eligibility folder now contains the new folder.
3. Open the duration folder, and click the **Create** button  and select **New Rule**.
4. Enter max duration and score as the name of the rule, and then click **Create**.
5. Enter the following rule:

```
if
  the duration of 'the loan' / 12 is more than the credit score of 'the
  borrower' / 30
then
  reject 'the loan' ;
  add "Sorry, your credit score must be at least 30 times bigger than
  the loan duration" to the messages of 'the loan' ;
```

Tip:

You can copy and paste the rule into the editor, or you can press Ctrl+Space to use the completion menu. (For information on the completion menu, see [Building rules using the Intellirule editor](#). The [Rule editor flash demo](#) on the IBM® Customer Support site also provides an English-only tour of the rule-editing features.)

6. Click **Save**, enter the following comment, and then click **Create New Version** to save the rule.

Created an action rule for the new eligibility policy.

You can find the new rule in the duration folder.

What to do next

In the next task, you create a separate ruleflow for the eligibility rules.

[< Previous](#) | [Next >](#)

Task 2: Creating a ruleflow

You create a ruleflow, and modify another ruleflow.

About this task


A ruleflow links rules into a series of decisions. It states when, how, and why rules are run, and serves as a key component of a decision operation. When you deploy a decision service as a ruleset, the decision operation includes the ruleflow. The deployed ruleset contains all the rules in the decision service, but it only runs the rules that are referenced by the ruleflow.

The miniloan ruleflow in Miniloan Service uses all the rules in the decision service. You create a ruleflow that orders only the eligibility rules, and then you update the miniloan ruleflow to call the new ruleflow.

Step 1: Creating a ruleflow

You create a ruleflow for the eligibility rules.

Procedure

1. Click **Miniloan Service** to display the decision artifacts.
2. Click the **Create** button , and select **New Ruleflow**.
3. Enter **eligibilityflow** as the name of the new ruleflow, and then click **Create**. The ruleflow editor opens. It displays instructions for creating tasks and transitions, and a ruleflow that contains a start task and an end task:





4. Click **Save**. Type the following comment, and then click **Create New Version**.

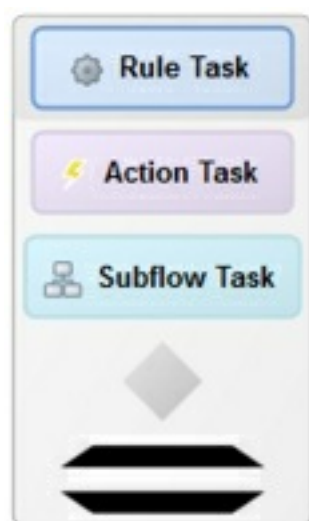
Created a ruleflow for the eligibility rules.

Step 2: Adding and defining a rule task

You now add the tasks that call the rules, and the transitions between the tasks.

Procedure

1. Hover over **eligibilityflow** in the list of decision artifacts, and click the **Edit** button  to open the file in the ruleflow editor.
2. Hover over the transition line between the start and end tasks.
3. Click the **Creation** button . The button opens a menu that contains different types of tasks:

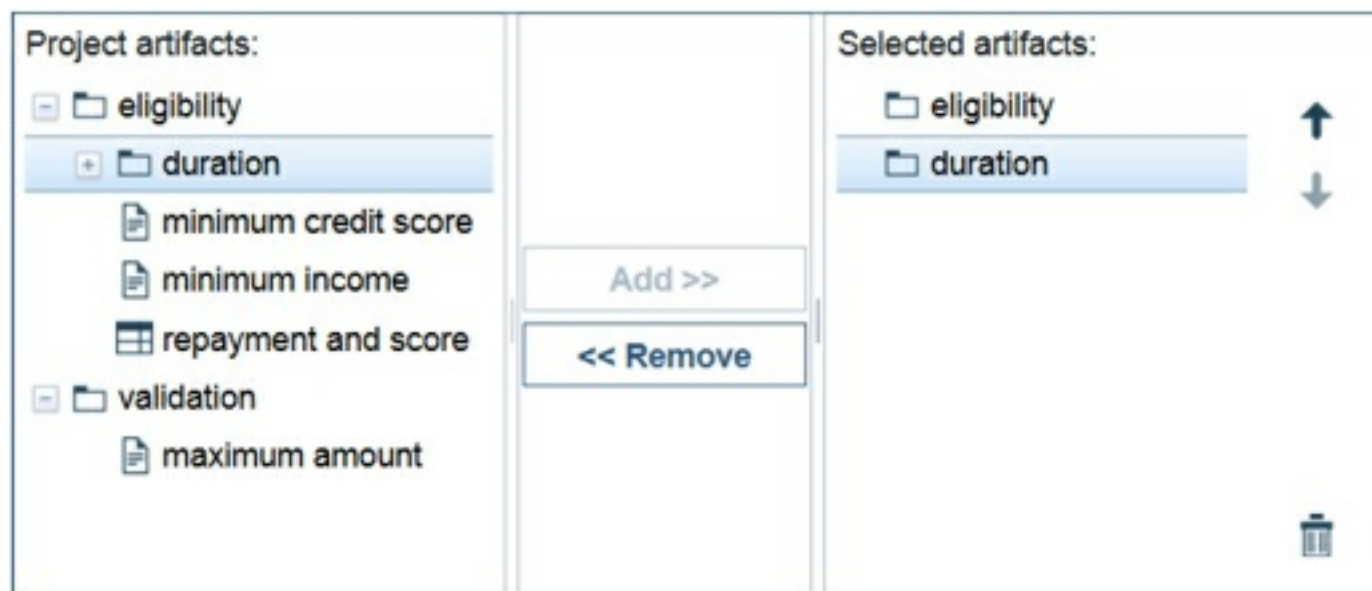


Each task provides a different function:

- **Rule Task**: Contains one or more rules to run at a specific point in the ruleflow.
- **Action Task**: Contains one or more rule statements to run at a specific point in the ruleflow.
- **Subflow Task**: Calls another ruleflow.
- **Branch Node**: Organizes conditional transitions.
- **Fork Node**: Splits the run flow into parallel paths.
- **Join Node**: Combines parallel paths back into a run flow.

- Click **Rule Task** to add a task.
- Click the new task in the ruleflow. In the task editor, enter `eligibility task` as the label, and following description in the Documentation field:

`Runs the eligibility rules.`
- Click **Edit** next to **Uses**.
- In the Rule Selection Editor, select the `eligibility` folder, and then click **Add** to add the folder to the rule task. The rule task runs all the rules in the folder. You can add more rules to the task by placing them in the folder.
- Add the `duration` folder to the list of selected artifacts.



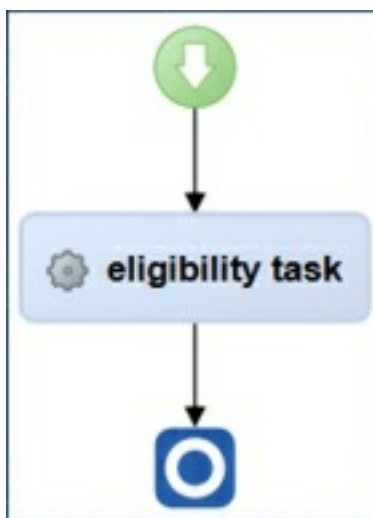
You must explicitly add the `duration` folder even though it is a subfolder in the `eligibility` folder. When you run the ruleflow, it applies the rules in the two folders.

- Click **OK**, and close the task editor.
- Click the End node, and then click the **Edit** button.
- Enter the following instructions, and then click **OK**:

```
print "Eligibility rules have been evaluated : result is :" + the
approval status of 'the loan' ;
```

The instructions give you some visibility into the running of the ruleflow. You can apply the same instructions in an action task after a rule task.

- Click the **Arrange** button  to optimize the layout. The ruleflow looks as follows:





- Click **Save**, and then click **Create New Version**.

Step 3: Modifying the miniloan ruleflow




Now that you have a ruleflow that is dedicated to the eligibility rules, you must replace the eligibility task in the miniloan ruleflow with a call to the eligibilityflow ruleflow. You create a subflow task so that you do not have to update the miniloan ruleflow every time you change the eligibilityflow ruleflow.

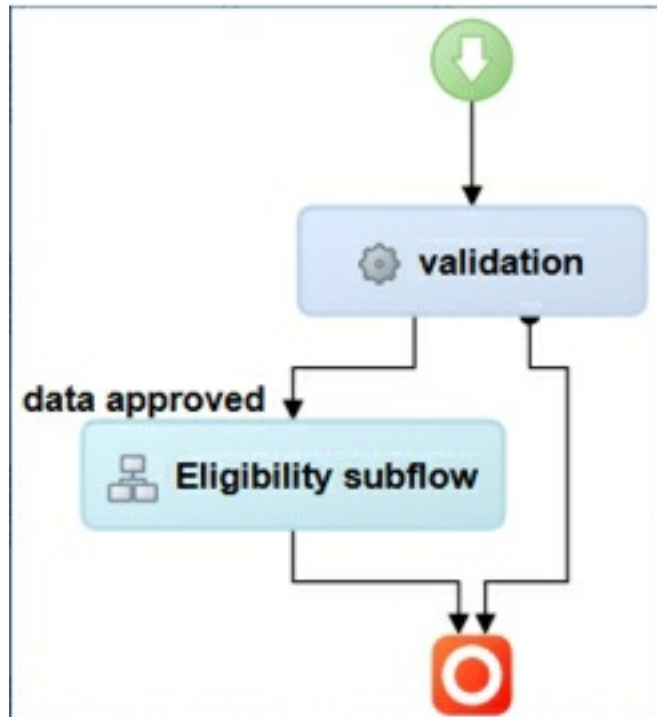
Procedure

- Click the miniloan ruleflow, and click the **Edit** button  in the display area.
- Click the data approved transition link between the validation and eligibility tasks anywhere except the **Create** button . You see the following condition:

```
'the loan' is approved
```

When the condition is met, the ruleflow moves to the next task. Otherwise, ruleflow goes to the end task.

3. Close the condition editor.
4. Hover over the transition link, click the **Create** button , and click **Subflow Task**.
5. Hover over the subflow task, click the **Create** button  at the bottom, and drag it to the end task.
6. Click the eligibility task, and press Delete or Backspace to delete the task.
7. Click the **Arrange** button  to optimize the layout.
8. Click the subflow task and change the label to Eligibility subflow.
9. In the Subflow field, select eligibilityflow, and then close the task editor. The ruleflow looks as follows:



10. Click **Save**, and then click **Create New Version**.

What to do next

In the next task, you look at setting the order for running rules in a rule task.

[< Previous](#) | [Next >](#)

Task 3: Setting rule task properties

You change the properties that define activities in a task.

About this task

The task treats the list of selected rules as pool of rules. You must set the ordering of the rules to literal to make the task run the rule artifacts in the order they are listed.

You set the ordering of the rule artifacts in the task properties. You can also set other properties for the task, including the engine algorithm and the number of rules the engine should run. For more information on these properties, see [Execution Properties for rule tasks](#).

Procedure

1. Open eligibilityflow in the ruleflow editor.
2. Click the eligibility task to open the task editor.
3. Click the arrow at the bottom of the task editor to expand the window.

Label:

eligibility task

Documentation:

Runs the eligibility rules.

Uses:

Edit...

eligibility

duration

Selects rules where:

Edit...

no rule selection code

Rule selection:

Static

Algorithm:

Fastpath

Ordering:

Default

Exit criteria:

None

Initial actions

Edit...

no initial actions

Final actions

Edit...

no final actions

Task ID:

rule_0

4. In the **Ordering** field, select **Literal** from the drop-down menu.

Ordering:

Exit criteria:

Initial actions

Default

Default

Literal

Priority

The ruleflow now runs the rules in the eligibility folder before it runs the rules in the duration folder.

5. Click **Save**, and then click **Create New Version**.

What to do next

In the next task, you create a decision operation that uses the new ruleflow, and run a test to check the ruleflow.

Task 4: Applying a ruleflow in a decision operation

You create a decision operation that uses the new ruleflow, and test the ruleflow.



About this task

In this task, you create a decision operation that uses the new ruleflow. Then to check the results of ruleflow, you run a test that uses the decision service.

Step 1: Creating a decision operation

You create a decision operation that uses the new ruleflow.

Procedure

1. In the **Decision Artifacts** tab, click the Operations folder.
2. Select Miniloan Service Operation, and click the **Copy** button .
3. Select the Operations folder as the location for the new decision operation, change the name to Miniloan Eligibility Operation, and then click **OK**.
4. Hover over Miniloan Eligibility Operation and click the **Edit** button. .
5. For **Main Ruleflow**, select the eligibilityflow ruleflow in the drop-down menu.
6. Click **Save**, and then click **Create New Version**.

Step 2: Creating a test scenario file



You now run a test to check the results of the ruleflow.

Procedure

First, you must create a scenario file that contains loan data and expected results. When the test runs the data, it compares the actual results to the expected results, and creates a report. (see [Testing sets of rules in the Business console](#)).

The following steps show you how to make a scenario file for testing rules. A ready-made scenario file is provided in the sample project that is downloaded from the Operational Decision Manager on Cloud portal (see the software prerequisites in [Before you start](#)). If you downloaded the sample project, you can use the miniloan-test.xlsx scenario file at `<Install Dir>/Miniloan Service/scenarios/`.

Attention: To do this step, you need Open Office, or Excel 2007 or later.

1. Open the **Tests** tab.
2. Click the **Generate Scenario File** button .
3. Select the Miniloan Eligibility Operation decision operation, and click **OK**.
4. Enter eligibility as the file name.
5. In Tests, expand the loan.
6. Select **approved** and **messages**.
7. In the Operator menu for **messages**, select **contains**.
8. Click the **Download** button , and save the generated Excel file to a directory on your computer, for example, `<InstallDir>/Miniloan Service/scenarios`.
9. Open the scenario file in Excel or Open Office.
10. In the **Scenario** tab, enter the following values. Leave the description cells empty. You can copy and paste the values into your scenario file. After you enter the values for the first scenario, duplicate the row to create the second scenario.

Tip: Instead of completing the table, if you downloaded the sample project, you can use the included eligibility.xlsx scenario file to complete the tutorial.

Scenario ID	name	credit score	yearly income	amount	duration	yearly interest rate
Bad credit score	Joe	250	80000	80000	240	0.05
Debt to income	Joe	600	80000	25000	5	0.05

Your **Scenario** table should look as follows:

		the borrower			the loan		
Scenario ID	description	name	credit score	yearly income	amount	duration	yearly interest rate
Bad credit score		Joe	250	80000	80000	240	0.05
Debt to income		Joe	600	80000	25000	5	0.05

11. Open the **Expected Results** tab, and enter the following values.

Scenario ID	the loan is approved equals	the messages of the loan contains
Bad credit score	FALSE	Sorry, your credit score must be at least 30 times bigger than the loan duration
Debt to income	FALSE	Too big Debt-To-Income ratio

Your **Expected Results** table should look as follows:




Scenario ID	the loan is approved equals	the messages of the loan contains
Bad credit score	FALSE	Sorry, your credit score must be at least 30 times bigger than the loan duration
Debt to income	FALSE	Too big Debt-To-Income ratio

12. Save your changes, and close the file.

Step 3: Running a test suite

You now run a test to see the results that are produced by the new ruleflow. If the ruleflow works correctly, the test shows successful results for both test scenarios.

Procedure

1. Return to the **Tests** tab.
2. Click the **Create** button .
3. Select the Miniloan Eligibility Operation decision operation, and then click **OK**.
4. Enter Eligibility Test Suite as the name, and Eligibility Test Report as the report name.
5. Under **Scenarios**, in **File to use**, click **Choose** and navigate to `<InstallDir>/Miniloan Service/`. Select `miniloan-simu.xlsx`, and click **Open**.
6. Under **Report**, select the following run options:
 - The list of rules fired
 - The list of ruleflow tasks
7. Click **Save**, and then click **Create New Version**.
8. Hover over the **Eligibility Test Suite** row and click the **Run** button .
9. Click **OK** in the Run Test Suite window. The test run begins.
10. Wait until the status for **Eligibility Test Report** shows a check mark , and then click the name of the report to open the report. The report shows that 100% of the scenarios ran successfully.



11. Under Results, expand the Debt to income test, and its run details. The run details show that the test ran the rules and tasks in the eligibilityflow ruleflow.

▼	2 / 2	Debt to income	Joe
▼	i	Execution Details	
		The list of rules fired	Details...
		The list of all tasks	eligibilityFlow>rule_0 miniloan miniloan>validation miniloan>subflow_0 eligibilityFlow
✓		the loan is approved equals	false false
✓		the messages of the loan contains	Details...

12. Click the **Close** button  to close the report.

Results

You have completed the tutorial. It showed you how to create a ruleflow in a decision service.

Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the decision service, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

[< Previous](#) | [Next >](#)

Creating a simulation in the Business console

This tutorial shows you how to create a simulation in a decision service in the Decision Center Business console.

Learning objectives

You do the following tasks:

- Create metrics and KPIs.
- Create and upload a scenario data file.
- Create a template for a report format.
- Create and run a simulation configuration.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Best practice

This tutorial includes the following best practice for creating a simulation in the Business console:

- Use the completion editor to define metrics and KPIs. [Example...](#)

Time required

40 minutes

[< Previous](#) | [Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a simulation to review the output of a decision service when you run the service on representative business data.

You work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans. When you run the decision service, it produces measurable information such as the amounts, durations, and credit scores of loans. You create a simulation that gathers this information as key performance indicators (KPIs), and displays the KPIs in a report.

Audience

The tutorial introduces you to the simulation feature in the Decision Center Business console. It is for Business console users who want to run simulations in decision services.

Prerequisites

You work on a branch that you create in the decision service that is provided in the Miniloan Service sample project. The project is available in the Rule Designer section of the cloud portal. You must have Rule Designer installed on your computer to publish the decision service to Decision Center.

To publish the decision service and create your branch for the tutorial, see [Preparing and removing the tutorial project](#).

Important:

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

More information

If you are not familiar with the Decision Center Business console or the simulation feature, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Overview: Simulations in the Business console](#)

Lessons in this tutorial

[Task 1: Creating metrics and KPIs](#)

You create metrics and use them to define KPIs.

[Task 2: Creating and uploading a scenario file](#)

You generate a scenario file, and add data to it offline. Then, you upload the file for use in simulations.

[Task 3: Formatting a report](#)

You create a template for the simulation report.

[Task 4: Running a simulation](#)

You configure and run a simulation in a decision service.

Task 1: Creating metrics and KPIs

You create metrics and use them to define KPIs.

About this task

A metric is a measurement of data. In the Miniloan Service decision services, the metrics include the amount, duration, and credit score of a loan. The decision service produces these metrics for each loan request. You gather these metrics for comparison in key performance indicators (KPIs). For example, a KPI can be the average amount of a group of loans, or it can be the amount of loans by credit score.

You start by opening the branch that you created in the decision service (see the software prerequisites in [Before you start](#)), and making a snapshot of the branch. Then, you create the metrics and KPIs that you use in the tutorial.

Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Step 1: Opening your branch

You open the branch that you created for this project, and create a snapshot. You can use the snapshot to compare versions of the branch, or to reinitialize the branch.

Procedure


1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Hover over the Miniloan Service box, and click anywhere except the name.
4. Click **My Tutorial** to open your branch. If you gave your branch a different name, click that name to open your branch.
5. Click **Take Snapshot**.
6. Name your snapshot something like My snapshot. Do not reuse the name of an existing snapshot. For example, you can personalize your snapshot by using the initials of your name.
7. Click **Create**

You can find the new snapshot in the **Snapshots** tab.

Step 2: Creating metrics

The Business console uses the contents of the decision service to determine the types of metrics that you can create for the simulation.

Procedure

1. Open the **Simulations** tab, and then open the **Metrics** subtab.
2. Click the **Create** button  to make a metric.
3. Enter Amount of the loan as the name of the metric, and leave **Numeric** as the type. The type determines the expression that you can declare for the metric. You must set the type before you compose the expression.
4. In the metric expression field, enter the following text:

```
the amount of 'the loan'
```

Tip:

You can copy the expression into the editor, or you can press Ctrl+Space to use the completion menu. (For information on the completion menu, see [Building rules using the Intellirule editor](#).)

5. Click **Save**, and then **Create New Version**.
6. Repeat steps 2 - 5 to make the following metrics:


Name	Type	Metric expression
Credit score	Numeric	the credit score of 'the borrower'

Approved loan	Boolean	'the loan' is approved
Duration of the loan	Numeric	the duration of 'the loan'
Yearly income	Numeric	the yearly income of 'the borrower'

Step 3: Creating KPIs

A KPI compiles similar metrics into a representative value. In a simulation report, you can present a KPI as a string or a comparative graph.

Procedure

- 1. Open the **KPIs** subtab.
- 2. Click the **Create** button  to make a KPI.
- 3. Enter Average credit score as the name of the KPI.
- 4. In the KPI expression field, enter the following text:

average 'Credit score'

Tip:
You can copy the expression into the editor, or you can press Ctrl+Space to use the completion menu.

- 5. Click **Save**, and then **Create New Version**.
- 6. Repeat steps 2 - 5 to create the following KPIs:

Name	KPI expression
Average income	average 'Yearly income'
Average loan amount	average 'Amount of the loan'
Income and duration	sum of 'Duration of the loan' grouped by 'Yearly income' with a 6 interval
Loan by credit score	average 'Amount of the loan' grouped by 'Credit score' with a 10 interval
Number of approved loans	number of 'Approved loan'

What to do next

In the next task, you create a scenario data file, and import it into the Business console.

[< Previous](#) | [Next >](#)

Task 2: Creating and uploading a scenario file

You generate a scenario file, and add data to it offline. Then, you upload the file for use in simulations.

About this task


You create a scenario file that holds loan data for the simulation that you run on Miniloan Service. You generate a file, download it to your computer, and add data in Excel or OpenOffice. When you complete the scenario file, you upload it back to the Business console. You can use the file in more than one simulation configuration, and in different instances of the Business console.

Attention: To work on the scenario file, you must have OpenOffice, or Excel 2007 or later.

Step 1: Creating a scenario file

The following steps show you how to make the miniloan-simu scenario file. Alternatively, you can skip this step and use the miniloan-simu.xlsx scenario file in the sample project downloaded from your instance of Operational Decision Manager on Cloud.

Procedure

1. Open the **Data** subtab in the **Simulations** tab.
2. Click the **Generate Scenario File** button .
3. Provide the following information in the dialog box:
 - File name: miniloan-simu
 - Operation: Miniloan Service Operation
 - Scenario file format: Excel Workbook (.xlsx)
 - Locale: English (United States)
4. Save the file to a directory on your computer, for example, `<Install Dir>/Miniloan Service/`.
5. Open the miniloan-simu.xlsx file in Excel or OpenOffice. The expected values of the rules in the decision service determine the contents of the scenario file:

		the borrower			the loan		
Scenario ID	description	name	credit score	yearly income	amount	duration	yearly interest rate
Scenario 1							

Each scenario row represents a loan application from a customer.

6. Enter the following values into the file. After you enter the values for the first scenario, duplicate the row to create the following scenarios.

Tip: Instead of completing the table, if you downloaded the sample project, you can use the included miniloan-simu.xlsx scenario file to complete the tutorial.


		the borrower			the loan		
Scenario ID	description	name	credit score	yearly income	amount	duration	yearly interest rate
Scenario 1		Joe	600	8000	140000	240	0.05
Scenario 2		Joe	500	80000	240000	240	0.05
Scenario 3		Joe	600	80000	130000	240	0.05
Scenario 4		Joe	600	90000	300000	300	0.05
Scenario 5		Joe	600	80000	560000	240	0.05
Scenario 6		Joe	600	80000	240000	240	0.05
Scenario 7		Joe	600	80000	130000	120	0.05
Scenario 8		Joe	600	80000	700000	240	0.05
Scenario 9		Joe	600	80000	140000	240	0.05
Scenario 10		Joe	1000	80000	240000	120	0.05
Scenario 11		Joe	600	80000	130000	240	0.05
Scenario 12		Joe	600	90000	300000	300	0.05
Scenario 13		Joe	600	80000	560000	240	0.05
Scenario 14		Joe	600	80000	240000	240	0.05
Scenario 15		Joe	600	80000	130000	60	0.05
Scenario 16		Joe	400	100000	600000	340	0.05

7. Save your changes and close the scenario file.

Step 2: Uploading a scenario file

You upload the miniloan-simu.xlsx file to use it in your simulation.

Procedure

1. Return to the **Data** subtab, and click the **Create** button .
2. Enter Miniloandata as the name, and select Miniloan Service Operation as the operation.
3. Click **Choose**, navigate to the miniloan-simu.xlsx file, and then click **Open**.

Tip: If you have skipped step 1, you can find the premade scenario file at the following location on your computer: <Install Dir>/Miniloan Service/miniloan-simu.xlsx.

4. Click **Create** to add the data file to the **Data** tab.

What to do next

In the next task, you format a report for your simulation.

[< Previous](#) | [Next >](#)

Task 3: Formatting a report

You create a template for the simulation report.

About this task

You add KPIs to a report template, and format their presentation. When you do so, you also tell the Business console which KPIs to process in the simulation.



You can present KPIs as strings or graphs. When a simulation produces a single value for a KPI, the value is presented as a line of text in the simulation report. However, if the KPI compares two metrics, you can present the information in a graph.

You format the presentation of the KPIs individually. You use a text editor to format strings, and a graph editor to format graphs. The graph editor gives you a choice of display types: bar, area, line, and pie. You can also add labels, and set the colors for different elements.

Step 1: Creating a report template

You create a report template and add KPIs.

Procedure

1. Open the **Report Formats** subtab in the **Simulations** tab.
2. Click the **Create** button  to open the report editor.
3. Enter Miniloan Simulation as the name.
4. Click the **Create** button  twice to add two sections.
5. In the first section, click **New Section**, and type Results as the new name. This section holds the strings.
6. In the second section, click **New Section**, and type Graphs as the new name. This section holds the graphs.
7. In the column of Key Performance Indicators, drag the Average credit score KPI to the Results section.
8. Repeat step 7 for the following KPIs:
 - Average income
 - Average loan amount
 - Number of approved loans

Note: Your browser might affect the order in which you list the KPIs.



9. Drag the following KPIs to the Graphs section:
 - Income and duration
 - Loan by credit score

10. Click **Save**, and then click **Create New Version**.

Step 2: Formatting the KPIs

You edit the appearance of the KPIs.

Procedure

1. Hover over Miniloan Simulation, and click the **Edit** button .
2. In the Results section, hover over the Average credit score KPI and click the **Configuration** button .
3. Open the **Format** menu, select **1,200**, and then click **OK**:



4. Format the other Results KPIs as follows:

- Average loan amount: \$1,200.46
- Average income: \$1,200.46
- Number of approved loans: 1,200

5. Format the KPIs in the Graphs section as follows:

Income and duration KPI

- Display Type: Bar
- Chart Title: Income to duration
- X-Axis Title: Income
- X-Label Format: 6
- Y-Axis Title: Duration
- Y-Label Format: 35
- Color: Blue (default)
- Outline Width: Default
- Style: Neon

Loan by credit score KPI

- Display Type: Pie
- Chart title: Average amount of loans by credit score
- Legend Format: 10
- Data Label: Value
- Outline Width: Default
- Style: Neon

6. Click **Save**, and then **Create New Version**.

What to do next

In the next task, you create and run a simulation configuration.

[< Previous](#) | [Next >](#)

Task 4: Running a simulation

You configure and run a simulation in a decision service.


About this task

You create and run a simulation configuration. When you run the simulation, the configuration brings your report template and scenario file together with the rule artifacts in the decision service. The simulation produces a report that shows the results of the simulation as defined in your report template.

Step 1: Creating a simulation configuration

You create a simulation configuration.

Procedure

1. Open the **Simulations** subtab in the **Simulation** tab.
2. Click the **Create** button .
3. Enter `Miniloan Service Simulation` as the name of the configuration, and enter `Miniloan Service Simulation Report` as the title of the simulation report.
4. Make sure that the following parameters are set as shown:
 - Server: `Testing and Simulation Runtime`
 - Report format: `Miniloan Simulation`
 - Input data: `Miniloandata`



The configuration editor automatically selects available parameters.

5. Check the sample report view to make sure that it shows your KPIs as you formatted them.
6. Click **Save**, and then click **Create New Version**.

Step 2: Running a simulation

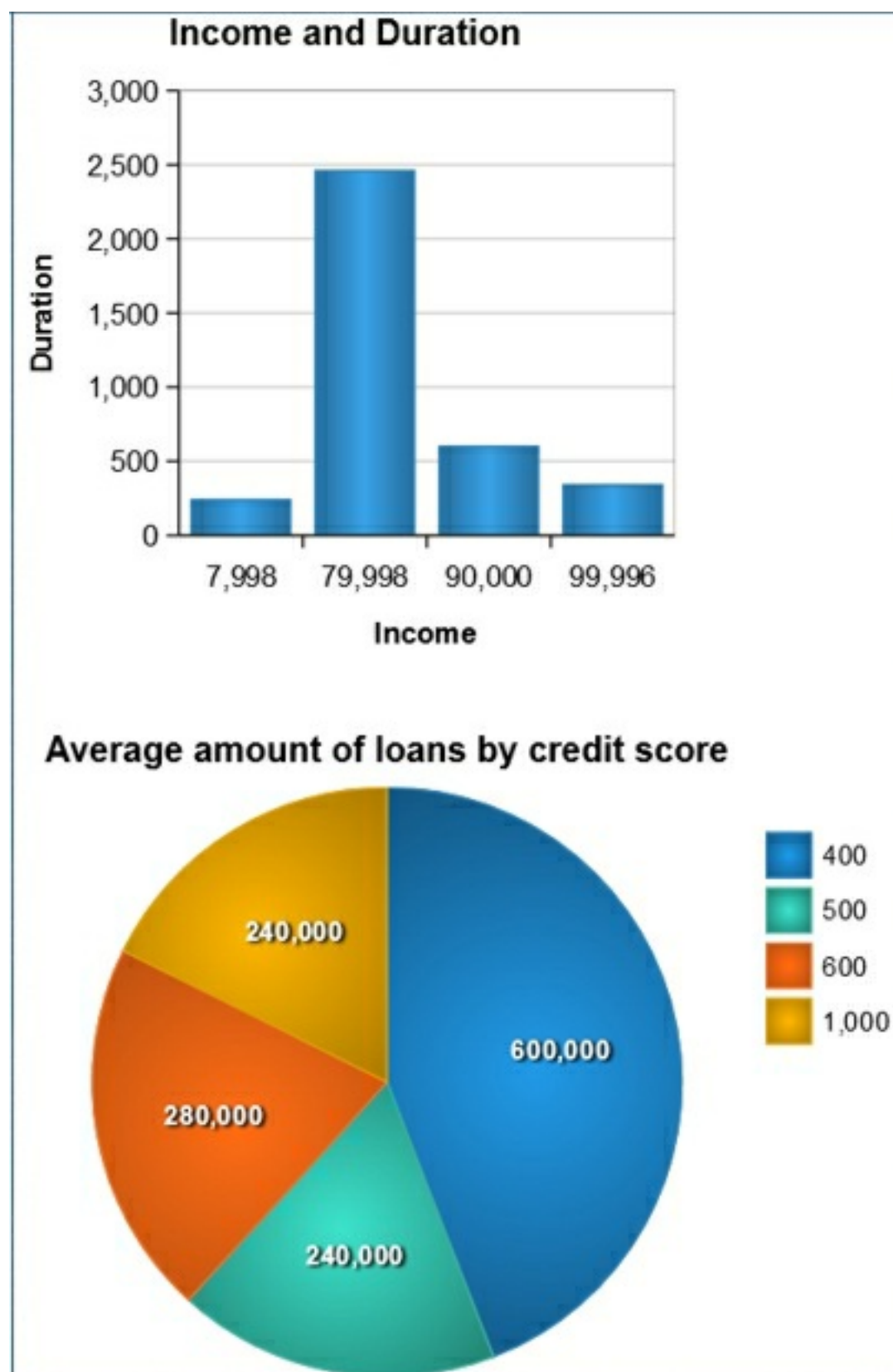
You run a simulation from your simulation configuration.

Procedure

1. Hover over the `Miniloan Service Simulation` configuration, and click the **Run** button .
2. Click **OK** to start the simulation. The **Reports** subtab opens to display the progress of your simulation.
3. When the status of the simulation changes to a check mark , click `Miniloan Service Simulation Report` to open the report. The report displays the results of the simulation:

Results
Average credit score: 606
Average loan amount: \$ 298,750.00
Average income: \$ 78,000.00
Number of approved loans: 9

The report also contains graphs for comparative purposes:



The property window lists source, performance, and status information for the simulation, as well as links to the simulation configuration and the rules in the decision service:

Simulation	Miniloan Service Simulation (latest)
Operation	Miniloan ServiceOperation
Input data	Miniloandata
Run date	September 28, 2016 3:52 PM
Run duration	5 seconds
Processed scenarios	16
Unsuccessful scenarios	0
Status	
View the rules used in this simulation	

4. Close the report.

Results

You have completed the tutorial. You created the elements for a simulation, and ran the simulation. The simulation produced a report that showed your KPIs. In developing a decision service, you can make changes to the decision service, run a simulation repeated, and then compare the reports from the simulation runs. In doing so, you can determine how to improve the decision service.

Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the Miniloan Service decision service in the Business console, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

[< Previous](#) | [Next >](#)

Merging branches in the Business console

This tutorial shows you how to merge changes between branches in a decision service in the Decision Center Business console.

Learning objectives

You do the following tasks:

- Compare branches before merging changes.
- Select changes and update options.
- Merge changes between branches.
- Verify your changes by using a snapshot.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Best practices

This tutorial includes the following best practices for merging branches:

- Keep **Lock branches before merge** selected to prevent other users from making changes to the branches while you merge them. [Example...](#)
- Check the merge results by comparing snapshots. [Example...](#)

Time required

30 minutes

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans. The decision service processes data that includes loan durations, credit scores, and loan amounts.

You make your changes in the Decision Center Business console, which is a shared rule-editing environment. When you work on changes that might be used in a decision service, you can create a branch that isolates your work. The changes that you make in your branch do not affect the other branches in the decision service until you share them.

To add your changes to another branch, you can use the merge feature in the Business console. It lets you select the branches, changes, and update options, and it provides progress information on your updates.

In this tutorial, you create a branch in a decision service. You change three rules in the branch, and merge changes into another branch. Then, you check the second branch to ensure that it contains your changes.

Audience

The tutorial is for users who want to merge branches in the Business console.

Prerequisites

You work in the Miniloan Service decision service, the sample project that is available in the Rule Designer section of the cloud portal. You must have Rule Designer installed on your computer to publish the decision service to Decision Center.

To publish the decision service and create one of the branches for the tutorial, see [Preparing and removing the tutorial project](#).

Important:

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

More information

If you are not familiar with the Decision Center Business console or the merge function, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Merging branches](#)

Lessons in this tutorial

[Task 1: Creating a branch and modifying rules](#)

You duplicate a branch in the decision service, and then modify three rules in the new branch.

[Task 2: Merging the branches](#)

You merge two branches to synchronize their rules, and then you check your changes by using a snapshot.

[< Previous](#) | [Next >](#)

Task 1: Creating a branch and modifying rules

You duplicate a branch in the decision service, and then modify three rules in the new branch.

About this task

You open the Business console and locate the branches in the Miniloan Service decision service (see [Before you start](#)). You make a branch, and then update three rules in the branch.


Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

Step 1: Creating a branch

You open the Miniloan Service decision service in the Business console, and create a branch.

Procedure

1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Click **Miniloan Service** to open the decision service.
4. Open the **Branches** tab, and expand main. You see the My Tutorial branch that you created when you published the decision service.
5. Click the **Add** button .
6. Enter Increase Minimum as the name, and select **main** as the parent branch.

Note: To distinguish your branch from other branches in the decision service, add a personal identifier to the name of the branch, for example, the initials of your name or your job title.

7. Enter the following goal, and then click **Create**:


Test to increase the minimum score.

The Business console creates the Increase Minimum branch, which opens in the console. It contains the same contents as the main branch.

Step 2: Modifying rules

You modify two action rules and a decision table in the Increase Minimum branch.

Procedure



1. Open the **Decision Artifacts** tab in the Increase Minimum branch.
2. Click **All Projects**, select **Rules and Decision Tables**, and click **Apply** to see the folders that contain business rules.
3. Click the eligibility folder to see its contents in the preview window.
4. Hover over the minimum credit score action rule in the preview window, and click the **Edit** button  to open the rule in the rule editor.
5. Change 200 to 300 in the condition and the action statements:

```
if
  the credit score of 'the borrower' is less than 300
then
  add "Credit score below 300" to the messages of 'the loan' ;
  reject 'the loan' ;
```

6. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the minimum credit score to 300.


The Decision Artifacts tab opens, and shows the contents of the eligibility folder.

7. Hover over the repayment and score decision table, and click the **Edit** button . Click **OK** in the dialog box to keep the default settings. The table opens in the decision table editor.
8. Change 200 to 300 in rows 1 and 2, and then click the **Optimize** button .

9. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the minimum credit score to 300.

The Decision Artifacts tab opens, and shows the contents of the eligibility folder.

10. Click the validation folder to see its contents in the preview window.
11. Hover over the maximum amount action rule in the preview window, and click the **Edit** button  to open the rule in the rule editor.
12. Change 1,000,000 to 100,000 in the condition and the action statements.
13. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the maximum amount to 100,000.

What to do next

In the next task, you merge your changes into the My Tutorial branch.

[< Previous](#) | [Next >](#)

[< Previous](#)

Task 2: Merging the branches

You merge two branches to synchronize their rules, and then you check your changes by using a snapshot.

About this task

You use the merge feature to add changes from the Increase Minimum branch to the My Tutorial branch.

Step 1: Selecting the branches

You select the branches to be merged.

Procedure

1. In the **Increase Minimum** branch, click the **Merge Branches** button , and then expand main in the dialog box to see the following list:



2. Click **My Tutorial** to select the branch. Leave **Lock branches before merge** selected to prevent other users from making changes to the branches while you merge them.
3. Click **Merge**. The merge page opens. It shows the selected branches in a table, and options for merging the branches:







Step 2: Merging the rules

You compare the rules, select the options for updating the branches, and merge the branches.

Procedure

1. Click **Expand all** to see all the artifacts in the table:

Name	Increase Minimum	My Tutorial	Action
▼  Miniloan Service			
▼  eligibility			
 minimum credit score	modified	-	update in My Tutorial
 repayment and score	modified	-	update in My Tutorial
▼  validation			
 maximum amount	modified	-	update in My Tutorial

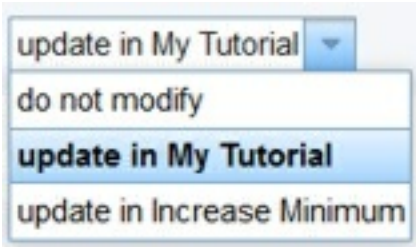
2. Hover over the row for the minimum credit score rule, and click the **Compare** button . The comparison page shows the rule in the two branches, and highlights the differences:


200 was changed to 300
200' was changed to 300'

version 1.0 (current) in My Tutorial Created by paulreleasemanager@gmail.com on Feb 22, 2017	version 4.0 (current) in Increase Minimum Created by paulreleasemanager@gmail.com on Feb 22, 2017
<pre> if the credit score of 'the borrower' is less than 200 then add "Credit score below 200" to the messages of 'the loan' reject 'the loan' ; </pre>	<pre> if the credit score of 'the borrower' is less than 300 then add "Credit score below 300" to the messages of 'the l reject 'the loan' ; </pre>

3. Close the comparison page, and do the same comparison operation for repayment and score and maximum amount to see the differences between the branches.
4. After you compare the rules, return to the minimum credit score row, and click **update in My Tutorial**.

A menu opens with three update options:



5. Set the merge options for the rules as follows:
 - minimum credit score: **update in My Tutorial** (The changes in the Increase Minimum rule are made in the same rule in My Tutorial.)
 - repayment and score: **update in Increase Minimum** (The changes in the Increase Minimum table are replaced with variables in the same table in My Tutorial.)
 - maximum amount: **do not modify** (No changes are made between the branches.)
6. Click the **Apply Merge** button  at the top of the table.
7. Enter the following comment, and select **Create snapshots of the branches before merging**:

Merged changes between branches.

8. Click **Apply Merge**. When the operation finishes, the merge table shows the following message:

Merge completed successfully

9. Click **Expand All**. The table shows the following information:


Name	Increase Minimum	My Tutorial
Miniloan Service		
eligibility		
minimum credit score	-	updated
repayment and score	updated	-

The minimum credit score rule was updated in the My Tutorial branch, and the repayment and score decision table was updated in the Increase Minimum branch. The table does not show the maximum amount rule because no update was made to the rule in either branch.

Step 3: Checking the merge results

You compare the My Tutorial branch to its snapshot to see what was changed by the merge operation.

Procedure

1. Click **Miniloan Service** in the breadcrumbs to return to the **Branches** tab.
2. Expand main, and click My Tutorial to open the branch.
3. Open the **Snapshots** tab, and click Before merge from 'Increase Minimum' to open the snapshot.
4. Click the **Compare** button . The following list opens:



Current State of the Project

✓



Before merge from 'Increase Minimum'


Created by paulreleasemanager@gmail.com on Feb 22, 2017 at 4:35 PM


State of this branch before the merge with the branch 'Increase Minimum'.

✓

5. Click **Compare**. A page opens with the results of the comparison. It contains a table that shows the modified rule:

1 artifact has been updated in *current state* since *Before merge from 'Increase Minimum'* snapshot

	Before merge from 'Increase Minimum'	current state
	Created by paulreleasemanager@gmail.com Feb 22, 2017	Created by paulreleasemanager@gmail.com Feb 22, 2017
 minimum credit score In eligibility	v1.0 Created by paulreleasemanager@gmail.com Feb 22, 2017	v4.0 Last updated by paulreleasemanager@gmail.com Feb 22, 2017

6. Hover over the `minimum credit score` row, and click the **Compare** button . The comparison window opens, and shows the same results that are in step 2.2:

★ 200 was changed to 300

★ 200' was changed to 300'

version 1.0	version 4.0 (current)
Created by paulreleasemanager@gmail.com on Feb 22, 2017	Created by paulreleasemanager@gmail.com on Feb 22, 2017
<div><div>if</div><div>the credit score of 'the borrower' is less than 200</div><div>then</div><div>add "Credit score below 200" to the messages of 'the loan' ;</div><div>reject 'the loan' ;</div></div>	<div><div>if</div><div>the credit score of 'the borrower' is less than 300</div><div>then</div><div>add "Credit score below 300" to the messages of 'the loan' ;</div><div>reject 'the loan' ;</div></div>

Results

You have completed the tutorial. You created a branch, updated the new branch, and merged your changes into another branch. Then, you compared the second branch to a snapshot to check the merged changes.

Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the Miniloan Service decision service in the Business console, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

Preparing and removing the tutorial project

The Miniloan Service decision service is used in most of the tutorials. This topic provides information for setting up the project for a tutorial and removing it when it is no longer needed.

About this task

The sections in this topic cover different activities for preparing and removing the Miniloan Service project. You do not have to follow all the instructions in this topic with every tutorial. Follow the instructions pertinent to your tutorial.

Downloading the decision service

The files for the project are kept in a GitHub repository. You download a compressed file, and extract its files to your computer.

Procedure

1. Sign in to the Operational Decision Manager on Cloud portal.
2. Click **Download** in the Decision Server Rule Designer section of the development environment.
3. Click **Miniloan Service project and Miniloan Server web application**. A download dialog opens.
4. Select **Save File**, and click **OK**.
5. Select a directory for the download file, and click **Save**. A compressed file is downloaded to the directory: `<InstallDir>\odm-cloud-getting-started-master.zip`. InstallDir is your directory for the GitHub files.
6. Click **Close** in the Rule Designer download dialog.
7. Open the compressed file, `<InstallDir>\odm-cloud-getting-started-master.zip`.
8. Extract the contents of the compressed file to the same directory. If you keep the original name, the new folder is `odm-cloud-getting-started-master`.

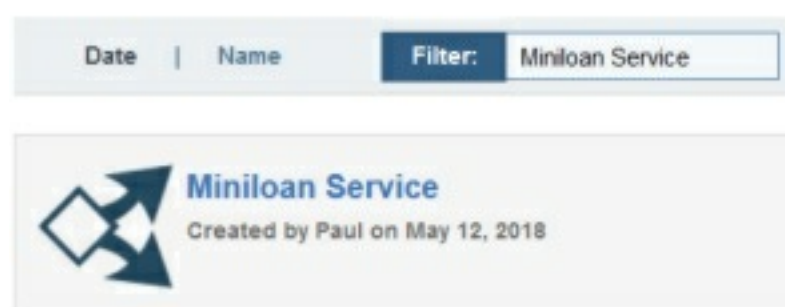
Checking the availability of the decision service

You determine whether the Miniloan Service decision service is already in the cloud portal.

Procedure

1. Sign in to the cloud portal.
2. Launch the Decision Center Business console in the development environment. The console opens to its home page.
3. Click **LIBRARY** to open the list of decision services that are currently in Decision Center.
4. Enter Miniloan Service in the filter to look for the decision service.

Decision Services




If the Miniloan Service decision service is in the library, skip step 3, and do step 4.

Importing the decision service into Decision Center

You import the decision service into the Business console.

Procedure

1. Open the folder that you created in the previous step, `odm-cloud-getting-started-master`.
2. Compress the Miniloan Service and miniloan-xom files into a .zip file named, for example, `miniloan.zip`.
3. Open the **LIBRARY** tab in the Business console, and click the **Import Decision Service** button .
4. Click **Choose**, and navigate to the compressed file that you created, `miniloan.zip`.
5. Select your compressed file, and click **Open**.
6. Click **Import**. The decision service is added to the library.


Creating a branch

You create a branch to isolate your changes to the decision service.

About this task

The main branch in the decision service contains the rule artifacts in their original form. To update the decision service, you create a branch based on the main branch, and work in the new branch.

Procedure

1. Click **Miniloan Service** to open the decision service.
2. Open the **Branches** tab, and expand the main branch. Look at the names of the existing branches. When you name your branch, do not reuse the name of an existing branch.
3. Click the **New Branch** button .
4. Enter a name for your branch, for example, **My Tutorial**. Remember not to reuse the name of an existing branch. For instance, you can personalize your branch by using the initials of your name.
5. Select **main** as the parent branch, and then click **Create**. The Business console duplicates the artifacts of the main branch in a new branch that you can modify.

Results

You can now work on the decision service in the tutorials.

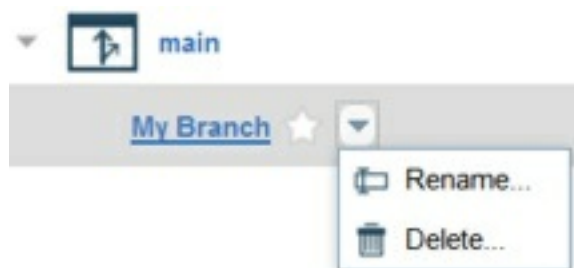
Deleting your branch from Decision Center

About this task

When you no longer need your branch in the Miniloan Service decision service, you can delete it from Decision Center.

Procedure

1. Open **LIBRARY** in the Business console
2. Open the Miniloan Service decision service, and open the Branches tab.
3. Expand the main branch to see your branch.
4. Hover over the name of your branch, click the down arrow to open the command menu, and click **Delete**:



5. Click **Yes** in the Delete Branch dialog.

Results

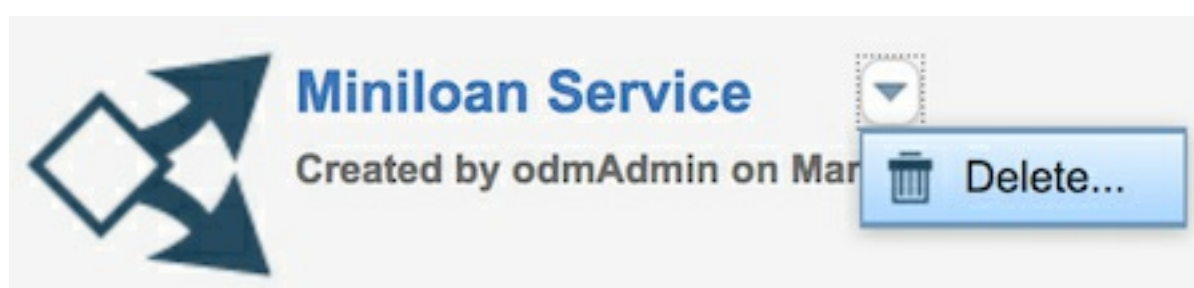
Your branch is deleted from Decision Center, and is no longer displayed in the Business console.

Deleting the decision service from Decision Center

You remove the Miniloan Service decision service from the Decision Center database.

Procedure

1. Open the **Library** tab in the Business console.
2. Hover over the Miniloan Service box, open the drop-down menu and click **Delete**:



A warning message opens. It shows the decision service that you selected.

3. Click **Delete**. The decision service is removed from Decision Center.

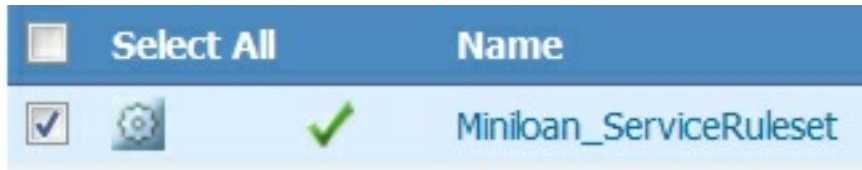
Removing a ruleset from Rule Execution Server

About this task

You can remove a ruleset that you deployed to Rule Execution Server.

Procedure

- 1. Open the Operational Decision Manager on Cloud portal.
- 2. Launch the Rule Execution Server console.
- 3. Click **Explorer** to open the tab.
- 4. Open the RuleApp for your decision service, for example, RuleApps/Miniloan/1.0.
- 5. Select the ruleset for your decision service in the RuleApp view, for example, Miniloan_ServiceRuleset:



- 6. Click **Remove** to delete the ruleset. A warning message opens. It shows the ruleset that you selected.
- 7. Click **Confirm** in the warning message. The console deletes your ruleset. The RuleApp view opens, and it no longer shows your ruleset.

Results

You removed your ruleset from the Rule Execution Server console.

Creating decision services

Your rule applications start in Rule Designer, where you create decision services for collaborative development.

Developing rulesets in Rule Designer

A ruleset is a set of business rules that serve as an executable decision unit. You develop the contents of rulesets in decision services and rule projects, and call the rulesets from client applications.

Designing projects for rule authoring

You work in projects to define a data model for rules and a vocabulary for business users. You also author different types of business rules.

Building and running rules

You can execute business rules on different platforms, optimize their execution, and automate certain tasks.

Publishing decision services to Decision Center

To enable developers and business users to collaborate on projects, you publish decision services from Rule Designer to Decision Center.

Developing rulesets in Rule Designer

The set of business rules that are put together as one executable decision unit is called a ruleset. You develop decision services and rule projects to define the contents of the ruleset that is called by the client application.

Setting up rule projects

The first step in developing a ruleset is to create a decision service with its rule projects, either from scratch or by using a template. You can then set up the rule project structure and add rule packages, to make your application more modular.

Orchestrating ruleset execution

You combine rules and rule artifacts into a ruleset for subsequent execution. You can pass data and share code in a ruleset, and manage the flow of rule execution.

Defining the ruleset content and signature

You define the content of a ruleset and its signature.

Setting up rule projects

The first step in developing a ruleset is to create a decision service with its rule projects, either from scratch or by using a template. You can then set up the rule project structure and add rule packages, to make your application more modular.

[Setting up a project hierarchy](#)

You can organize your content as a hierarchy of rule projects.

[Defining rule project references](#)

You use project references to make your business rule application more modular.

[Defining a folder structure for rule project items](#)

When you create a rule project, a folder is automatically created for each type of rule project item.

[Rule project item naming conventions](#)

When you create a new rule project item, follow the naming conventions.

[Improving performance on large rule projects](#)

In large rule projects, you can improve the performance of Rule Designer.

Parent topic: [Developing rulesets in Rule Designer](#)

Setting up a project hierarchy

You can organize your content as a hierarchy of rule projects.

You create a hierarchy by establishing dependencies between projects. Use rule project hierarchies to separate and share the rules and the BOM among different projects, especially as your application becomes more complex. Using different rule projects allows for easier maintainability, lets you test different parts of the logic independently, and organize for various decision points. You can then set up the projects to refer to one another.

The project organization that you set up in Rule Designer also impacts how you manage these projects in Decision Center.

Table of contents

- [Organizing the rule project structure](#)
- [Organizing the project structure for multiple rulesets](#)

Organizing the rule project structure

Decision services are designed to facilitate the creation of the project structure. When you create a decision service, you create a main rule project that serves as the top-level project of the rule project hierarchy. If the project contains many rules, you create other types of projects to group these rules by domain and to store the BOM. Then, you set dependencies between the projects. All projects that are directly or indirectly referenced by the main project become part of the decision service. For example, a main decision project might reference two standard rule projects, which in turn both reference a project that contains the BOM. All four projects form the decision service.

You can create the different types of projects in the **Decision Service Rule Project** section of the New Rule Project wizard as follows:

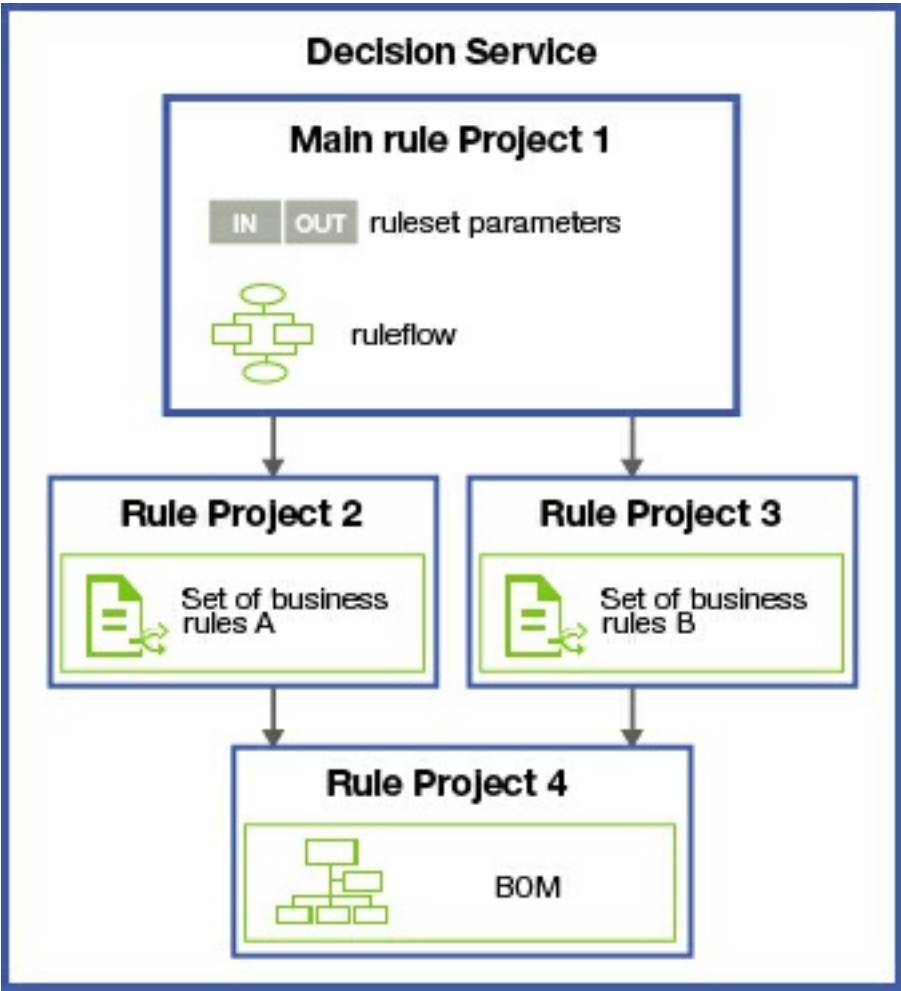
- Select **Main Rule Project** to create a project that is the entry point of the decision service and that might reference other projects.
- Select **Standard Rule Project** to create one or more projects to store your rules.
- Select **Rule Project with a BOM** to create a project to store the BOM.

Note:

It is possible to change the type of any project to transform it into a main project or vice versa, in **Properties > Decision Service**. You can change a rule project to a main rule project when the rule project is not referenced by another project. Changing a project can require changes to the way projects reference each other, and can affect the deployment and synchronization of the project.

In Rule Designer, many operations take place at the decision service level, such as build, queries, refactoring, and ruleset extraction. As a consequence, grouping rule projects in separate decision services limits the scope of these operations.

The following diagram shows a possible rule project organization in a decision service. The main project contains a ruleflow that uses rules from projects in the hierarchy, and the projects share a BOM that is kept in one of the projects. All four projects are part of the same decision service because rule projects 2, 3, and 4 are directly or indirectly referenced by the main rule project.

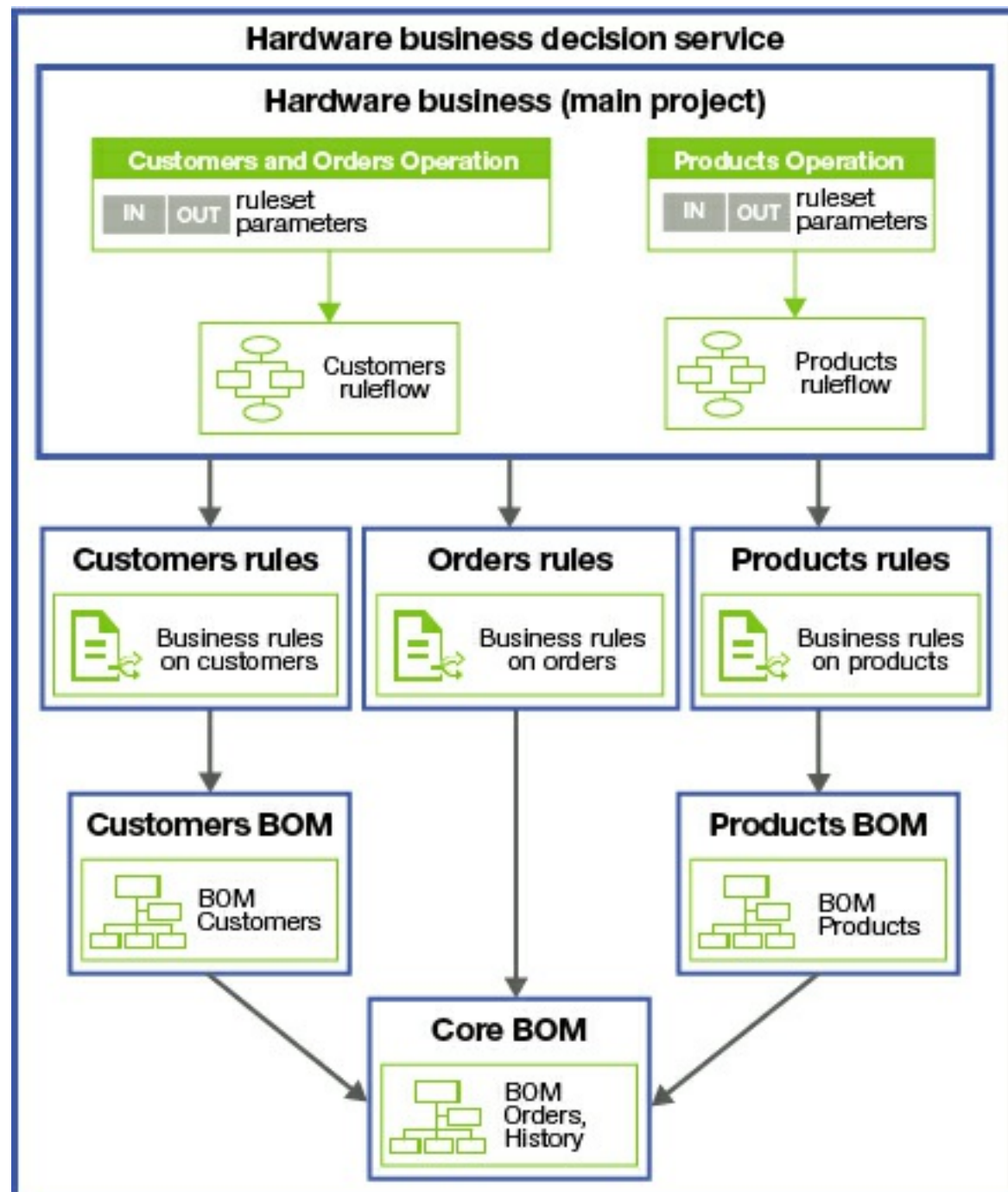


Organizing the project structure for multiple rulesets

If the client application calls different decision points, you should organize your project structure for multiple rulesets.

If an operation and its rules require only part of a BOM, you can keep those rules and that part of the BOM in a separate rule project hierarchy that specifies the vocabulary that is available to those rules. The size of the BOM can result in usability and performance issues. To reduce the number of completion entries in the rule editors, and to make them more relevant to your rules, use categories to organize your vocabulary into subsets.

You can package all decision operations, ruleflows, and rules in the same decision service. The following diagram shows a decision service that is designed to distribute the decision points in separate rule projects. It also shows a strategy for splitting the BOM for efficiency.



Parent topic: [Setting up rule projects](#)

Related concepts:

[Defining the ruleset content and signature](#)
[Executing rulesets in the decision engine](#)

Related tasks:

[Running and debugging decision operations](#)

Related information:

[Categories](#)
[Improving performance on large rule projects](#)
[Business object model \(BOM\)](#)

Defining rule project references

You use project references to make your business rule application more modular.

About this task

A rule project can reference other rule projects. All items in the referenced rule project are then available to you for use in your current rule project.

Rule project references are a way to make your business rule application modular, by reusing rule project items such as business object models or templates in several rule projects. The decision service approach to creating your rule project hierarchy is perfectly suited to handle the dependencies of the rule projects it contains.

You can either define rule project references when you create a decision service, or add them later using the rule project Properties dialog.

Procedure

To add project references:

1. In the Rule Explorer view, select the rule project, and on the **Project** menu click **Properties**.
2. Click **Project References** to display a list of rule projects that you can reference.
3. Select each rule project that you want to reference and then click **OK**.

Results

Your rule project references are now defined.

Parent topic: [Setting up rule projects](#)

Defining a folder structure for rule project items

When you create a rule project, a folder is automatically created for each type of rule project item.

About this task

When you create a rule project, you automatically create folders to store the different types of rule project items:

- BOM entries: bom
- Queries: queries
- Rule artifacts: rules
- Resources: resources
- Templates: templates
- Deployment: deployment

Procedure

1. In the Rule Explorer view, select the rule project and then on the Project menu click **Properties**.
2. In the pane of the Rule Project Properties dialog, click **Rule Project Folders** to display the list of rule project folders.
3. Optional: If you want to modify the Resource folder:
 - a. In the Rule project folders area, select the Resource Folder path, and then click **Edit**.
 - b. In the Select Folder dialog, either select an existing folder, or create a new one. To create a new folder, select the rule project, and click **Create New Folder**, specify the new folder name and click **OK**.
 - c. Click **OK** to close the Select Folder dialog.
4. Optional: If you want to modify the Output folder:

The Properties dialog displays the new Output folder path.

- a. Click **Edit** next to the Output folder field.
 - b. Either select the folder you want to use as the output folder, or create a new one.
5. Click **OK** to close the Properties dialog.

Results

Your rule project folders are now redefined.

Parent topic: [Setting up rule projects](#)

Rule project item naming conventions

When you create a new rule project item, follow the naming conventions.

Make sure that you use the Java™ naming conventions when you create a name for your rule project items. You can use spaces because they are processed into valid characters when you execute, but do not use spaces at the beginning of a name.

The name of a rule project item must not:

- Be empty
- Be longer than 255 characters
- Start or end with white space characters
- Contain the characters ", ., :, *, /, <, >, ?, \, or |
- Contain a character that has a ASCII or Unicode code below 0x1f, included

Restriction: Be careful with the use of Japanese, Korean, and Chinese (simplified and GB18030 encoding) characters in your rule project names.

Parent topic: [Setting up rule projects](#)

Related information:

[Overview. Ways to express business rules](#)

Improving performance on large rule projects

In large rule projects, you can improve the performance of Rule Designer.

When your rule projects grow larger, you can usually improve Rule Designer performance by using some Rule Designer build configurations that are better suited to large projects, limiting the BOM footprint, and reducing the size of business rule artifacts.

Disabling the automatic build

By default in Eclipse, a build process is started as soon as a resource changes in the workspace. As a consequence, as soon as you save a file in Rule Designer, a build runs. The build is usually incremental and fast, so there is no particular issue with the fact that it runs regularly. However, if you make changes that affect the rule project globally, such as modifications on the business object model (BOM), the build might have to recheck many elements. The build can be slow if your project is large.

Deselect **Project > Build Automatically** to disable the automatic build.

Note:

If you disable the Eclipse automatic build, rule refactoring might not always be activated. In this case, you must explicitly ask for a build to refresh the errors in the Problems view.

Setting up preferences for a faster build

When you have a large rule project, particularly one with many decision tables and ruleflows, the build can become slower. In Rule Designer you can define preferences for the rule project build.

1. To access the build preferences page, select **Windows > Preferences**. (On Mac, click **Eclipse > Preferences**.)
2. Select **Rule Designer > Build**

When you save a rule project item, Rule Designer automatically generates IRL code and runs some checks on this code, including rule analysis. You can choose to disable the rule analysis checks only, or to disable all checks by clearing the **Perform IRL checks during build** option.

Note:

If you disable these options, you do not receive warning and error messages when saving your rules, but the rule project build is faster. However, more time is required when you extract a ruleset archive from the rule project, because the IRL code is generated and checked at that point. You can also disable IRL code generation to make the build even faster.

Limiting the number of business rules in rule packages

Rule Designer is more efficient when you work with several small or medium-sized packages, as opposed to a few large packages, for the following reasons:

- Rule Designer loads only the required packages. If all your rule artifacts are stored in a single package, Rule Designer must load all the rule artifacts when you start it. If your rules are organized into packages, Rule Designer loads only the rule artifacts from the required packages.
- To resolve the fully qualified name of a rule, Rule Designer navigates through packages to find the rule. If the tree of packages is well distributed, Rule Designer accesses the rule faster.

Note:

The fully qualified name of a rule is the short name of the rule prefixed with its package name.

However, make sure that you do not have too many small packages, because looking for rule model elements slows down Rule Designer.

Reducing the business object model footprint

When your rule project contains a large business object model (BOM), the features related to business rule language are slower. Parsing, checking, Content Assist, and navigation in business rules and technical rules is slower with large business object models.

To reduce the BOM footprint, you can apply the following techniques:

BOM size

- Reduce the BOM size by removing unused verbalizations, BOM members and classes: When you define a BOM entry from a XOM, the BOM entry might include numerous technical classes and methods that are not actually used in your rules. Because the size of the BOM affects Rule Designer performance, it is better to remove all unused business elements from the BOM.
- Split the BOM into several smaller BOMs that can be spread among your rule projects. See [Setting up a project hierarchy](#).

BOM tree

Collapse the BOM tree in the Rule Explorer View before saving any changes to the BOM and starting a build. A collapsed BOM tree significantly reduces the time taken to build the project.

Vocabulary parsing

A large BOM with many verbalizations might cause noticeable pauses in Rule Designer. To improve performance, you can reduce the time taken by BRL parser generation by disabling this option: **Window > Preferences > Rule Designer > Business Rule Editing > Enable Quick Fix**. As a consequence, the user is no longer prompted with Quick Fix suggestions when editing rules with the IntelliRule editor. The guided editor behavior is not affected because it does not support the Quick Fix feature.

Limiting the size of decision tables and decision trees

Decision tables that contain more than 3,000 rows, given a reasonable number of columns, significantly increase editing and build time. Tables with many columns should have fewer rows, because the main scalability factor is the number of cells.

In Rule Designer, you can significantly improve the usability and performance of large decision tables and trees:

- If possible, break up your large decision tables into smaller tables to improve their response time and make it easier to maintain their contents.
- Disable overlap and gap checking.
- Set up the build to disable IRL generation and checking. See [Setting up preferences for a faster build](#).

Limiting the size of RuleApps

When a RuleApp archive is generated, the number and size of its rule artifacts account for most of the resource consumption.

To reduce the resource usage cost of a business rule application, you can apply the following techniques:

- Transform each large rule project into several smaller ones.
- Keep each rule project as it is, and generate multiple smaller rulesets out of it. You can use ruleset extractors, one per ruleset, to extract part of the rule project. In this case, you might have to change the application that executes the rulesets.

Upgrading to 64-bit Java SDK

If your system configuration supports it, run Rule Designer with 64-bit Java™ SDK to help building large projects if memory runs scarce.

Parent topic: [Setting up rule projects](#)

Orchestrating ruleset execution

You combine rules and rule artifacts into a ruleset for subsequent execution. You can pass data and share code in a ruleset, and manage the flow of rule execution.

Rules apply decisions, but they do not have a defined sequence or succession. A ruleflow organizes rules into a sequence of decisions by assembling the rules into a group of rule tasks that uses a set execution pattern.

Each rule task is evaluated to produce a result, or decision. All these results and decisions are combined to produce a single business decision, which is represented by a ruleflow. Ruleflows also specify the transitions between rule tasks. The transitions determine how, when, and under what conditions to use each rule task.

The following diagram shows the levels of refinement for selecting the rules.



The evaluation during ruleset execution selects rules in the following manner:

1. Ruleflow scope selection

Each rule task in a ruleflow has a scope that is defined by a list of rule packages and individual rules. At run time, ruleflow scope selection generates a list of the rule packages and rules that you defined for each task. When a task is run, the rule engine considers only the rules within this scope.

2. Runtime rule selection

The runtime rule selection process further determines which rules the rule engine must consider. Filtering is typically based on the value of rule properties and execution parameters.

3. Rule overriding

After the other selection mechanisms are executed, certain rules that you defined beforehand override other rules. The overridden rules are filtered out of the selection.

The rule engine evaluates the conditions of the selected rules, and runs their rule actions on the matching objects.

[Working with ruleflows](#)

In Rule Designer, you create ruleflows and add different types of tasks to control the execution of rules.

[Configuring rule execution](#)

You can notify the rule engine of changes in object states. You can also specify how rules in a ruleset are executed at run time by setting up the ordering, priorities, or overriding of rules.

Parent topic: [Developing rulesets in Rule Designer](#)

Working with ruleflows

In Rule Designer, you create ruleflows and add different types of tasks to control the execution of rules.

Overview: Ruleflows

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

Creating a ruleflow

You can create a ruleflow in Rule Designer, and add elements to your ruleflow in the Ruleflow Editor. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

Configuring rule task execution

You define how rules are selected at run time and specify the general order in which the rule tasks associated with these rules are executed.

Configuring ruleflow properties

You configure the properties of ruleflow elements in the **Properties** view of each element.

Parent topic: [Orchestrating ruleset execution](#)

Overview: Ruleflows

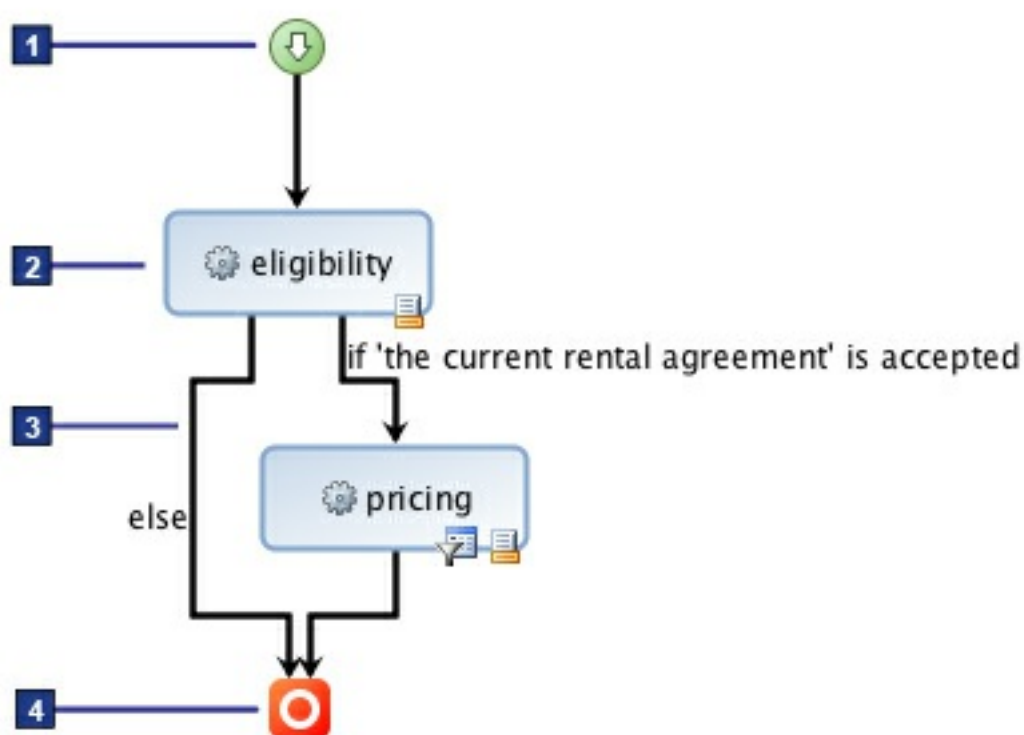
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



Note: The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter


statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

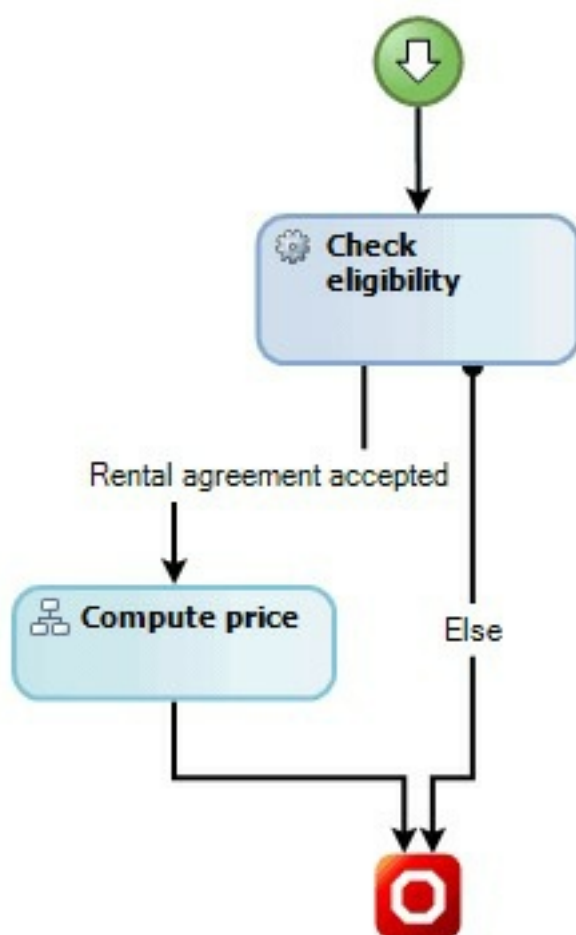
You can specify initial and final actions on subflow tasks.

Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.




Note: In transition conditions, variables cannot reference objects in the working memory. They can only reference ruleset parameters.

A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.


An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.

Branches

A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch. Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

Note: Like action tasks, initial and final actions cannot call methods on objects in the working memory. They can only call the methods of ruleset parameters.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

Parent topic: [Working with ruleflows](#)

Creating a ruleflow

You can create a ruleflow in Rule Designer, and add elements to your ruleflow in the Ruleflow Editor. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

About this task

You can add a ruleflow to a project, or any of its packages. You can have multiple ruleflows in a project, but if you have more than one ruleflow in a project, you must define one of them as the main ruleflow. After adding a ruleflow to a rule project or package, you define the structure by adding the elements that you need in the ruleflow diagram. To understand the use of each element, see [Ruleflows](#). Then, you connect these elements by adding transitions between them.

Procedure

1. In the Rule Explorer, select the name of the package or project you want to add a ruleflow to, and click



New Ruleflow on the New Rule Project Item toolbar.

In the **New Ruleflow** dialog that opens, enter the relevant information for your ruleflow. If you are adding the ruleflow to a package, make sure the name of this package is added in the **Package** field. No entry is needed for a ruleflow created under the top folder.

2. Use the buttons in the Ruleflow Editor palette to add elements in the ruleflow diagram as required. You must create one start node for your ruleflow, and at least one end node.
3. Create rule tasks, and add rules to be executed at this point in the ruleflow.
 - a. Add a rule task element in the diagram.
 - b. Set the properties that you need in the **Properties** view. For more information about rule tasks properties, see [Configuring ruleflow properties](#).
 - c. In the **Rule Selection** tab of the **Properties** view, click **Edit** to open the **Select Rules** dialog, and add rules to a rule task. You can use the **Up** and **Down** buttons to order the rules and packages. Depending on the rule execution properties of the task, this order might impact the output of ruleflow execution.

Tip: You can also drag an existing rule package or rule from the Rule Explorer into the ruleflow diagram. The rule task has the name of the element and already contains the rules and packages.

4. Optional: If you need to execute rule action statements, add action tasks in your ruleflow, and set the action statement, initial action, and final action in the **Properties** view.
5. Add transitions between the tasks to define the flow of your ruleflow.
 - a. To specify a condition for the transition, in the **Properties** view for the transition, click **Condition**.
 - b. Provide a name for the condition in the **Label** field.
 - c. Select **Use BAL for transition condition** and type a condition statement. For example:

'the current rental agreement' is accepted

You can also write the condition using IRL. In this case, make sure the text fields contain a valid Boolean expression. If you do not use BAL, in the ruleflow diagram the transition arrow displays the label and the expression.

In transition conditions, the variable scope is restricted to ruleset parameters and variables. It does not include access to the working memory.

6. Optional: You can create multiple, parallel paths in your ruleflow, if you need to execute rules simultaneously. For example, if you are checking the eligibility of a customer for a loan, you might want to check whether the customer meets the criteria for the loan, and also if the amount requested is valid. To do so, you use forks and joins in your ruleflow.
 - a. Add a fork node where you want your ruleflow to execute several rules in parallel. You can then add your rule tasks in the ruleflow.
 - b. Add a join where you want to combine the transitions created from the fork. Make sure to create all transitions between the different elements.


The transitions from a fork node to a join node must not have conditions, because the ruleflow follows all paths in parallel between the fork and the join.

7. Optional: You can add branches to the ruleflow to organize conditional transitions, in the same way that you could start several conditional transitions from a task.

Important:

When multiple transitions originating from a branch or a task define overlapping conditions, the path taken to execute the ruleflow is unpredictable. Make sure the conditions you define for multiple

transitions do not overlap.

- a. Add a branch node where you want your ruleflow to organize the different conditions.
 - b. To name the branch, in the ruleflow diagram click the branch icon and in the **Properties** view, click **Branch Node** and enter the name in the field provided.
 - c. Add transitions to and from the branch.
 - d. Add transition conditions for each transition from the branch. One of the transitions must be an Else transition.
8. Optional: If you want to execute another ruleflow at some point in your main ruleflow, add a subflow task in the diagram:
- a. In the **Properties** view for the subflow task, click **Subflow Task**.
 - b. Click **Select** to select a ruleflow to include in the subflow task.
9. To align the ruleflow automatically, click  **Layout All Nodes** in the Ruleflow Editor toolbar. You can also align items manually by selecting them, right-click the ruleflow, and select **Align**. Select the alignment options in the pop-up menu that opens.
10. Save the ruleflow (Ctrl+S).

Parent topic: [Working with ruleflows](#)

Related concepts:

[Overview: Ruleflows](#)

Related information:

[Execution properties for rule tasks](#)

Configuring rule task execution

You define how rules are selected at run time and specify the general order in which the rule tasks associated with these rules are executed.

Runtime rule selection

You can define a selection filter on a rule task to specify dynamically what rules of the rule task must execute.

Execution properties for rule tasks

You can define rule task execution properties to select the rule engine algorithm that is used to execute the rules, and refine the number and order of rules that the rule engine executes.

Parent topic: [Working with ruleflows](#)

Runtime rule selection

You can define a selection filter on a rule task to specify dynamically what rules of the rule task must execute.

You specify the rules selected at run time in the **Rule Selection** tab of the **Properties** view. You can select individual rules, or packages containing several rules, that are considered for execution in this rule task, and filter out some of them. You can specify this filter with dynamic or static BAL constructs, or IRL. The filter is applied at run time, and the rule engine evaluates only the rules that pass through the filter.

Note:

An empty list of rules and packages means that all the rules from the project are selected for execution at run time.

Selecting rules with BAL

Typically, the filtering process is based on the values of rule properties and execution parameters. The filter tests each rule in the ruleflow task to determine whether the rule should be selected for execution. For example, the following code filters on the name of the rule and is called for every rule in the task.

```
the name of 'the rule' contains "Age"
```

There is no difference between dynamic BAL and static BAL. The filter is evaluated each time the rule task is invoked. Candidate rules can be selected based on the ruleset parameter state. For example, you can specify that the expiry date of the rule is after the date of the loan.

Selecting rules with IRL

Static body

The list of rules is specified simply by including the names inside the rule task body.

```
body = { R1, R2 }
```

Static body using the select keyword

Here is an example of selecting rules using the select keyword:

```
body = select(?rule) { <boolean returning code> }
```

This first example shows that a body of the rule task can be an IRL predicate that selects the relevant rules among all the rules of the ruleset.

In this second example, the specification selects all the rules available in the ruleset:

```
body = select(?rule) { return true; }
```

Important:

There is no difference between `dynamicselect` and `select`. `select` behaves the same way as `dynamicselect` where the statement is called each time the task is executed.

Dynamic body

The list of rules is selected by a function-like expression that takes an [IlrRule](#) object as its argument and evaluates to a Boolean expression. The pseudo-variable `?rule` represents the `IlrRule` argument of the expression. This filter applies to each rule in the ruleset each time the rule task is activated. Hence, this expression might well depend on environment variables such as ruleset parameters or fields whose values can change between two executions of the rule task.

```
body = dynamicselect(?rule) {  
    return myRuleSelection(?rule);  
}
```

Dynamic body in a domain

The domain is an expression that evaluates to a Java™ array or collection of [IlrRule](#) objects. The dynamic body is also an expression. You can use expressions to select the subset of the rules of the domain for the next rule task execution. The dynamic body is evaluated each time the rule task is activated.

It accepts two different signatures.

- **Dynamic body that returns a collection of rules:**

The dynamic body is an expression that evaluates to a Java array or collection of an `IlrRule` object.

```
body = dynamicselect() {  
    return myRuleDomainSelection();  
}  
in myInitialRuleDomain();
```

Attention:

There is no check at run time that the collection of rules returned by the dynamic body is really a subset of the initial rule domain. You must enforce such verification.

- **Dynamic body that returns a Boolean value:**

The dynamic body is a function-like expression that takes an [IlrRule](#) object as its argument and evaluates to a Boolean expression. The pseudo-variable `?rule` represents the single `IlrRule` argument of the expression. The engine passes to this rule filter only the rules that belong to the initial rule domain.

```
body = dynamicselect(?rule) {  
    return myRuleSelection(?rule);  
}  
in myInitialRuleDomain();
```

See [Memory consumption reduction](#) for more information.

Scope

The scope defines in a symbolic way what rules compose the body of a rule task. This provides a simpler way of describing the selection than using IRL expressions such as an `in` expression. The scope makes use of the package organization.

Note:

Because the IRL `in` expression and the scope provide alternate selection methods, you cannot use them together.

Parent topic: [Configuring rule task execution](#)

Related information:

[Working with ruleflows](#)

[Setting simple priorities among rules](#)

[Ordering rules in a ruleset](#)

Execution properties for rule tasks

You can define rule task execution properties to select the rule engine algorithm that is used to execute the rules, and refine the number and order of rules that the rule engine executes.

You specify the rule task execution properties in the **Rule Task** tab of the **Properties** view to control rule execution in different execution modes. You can choose a rule engine algorithm to execute the rules in a particular rule task, and use execution control properties to further refine the number and order of rules the rule engine should execute.

In general, the order of the rule artifacts in the list of selected rules is not guaranteed, and the list of selected rules is considered as a pool of rules to be executed. However, with **Ordering** set to **Literal**, the order of the rule artifacts selected becomes important.

Table 1. Control settings for rule tasks

Action	Location	Description
Rule selection	Rule Selection tab in Properties view for rule task.	Click Edit to add and change the order of rules and rule packages by using the Select Rules dialog.
Runtime rule selection	Dynamic BAL , Static BAL and IRL in the Rule Selection tab of the Properties view for rule task.	Control which rules are evaluated in a rule task.
Execution mode	Rule Task tab in Properties view for rule task.	Specifies an execution mode for the rule task. See Changing the execution mode for details.
Rule task ordering	Rule Selection tab in Properties view for rule task.	<p>The default ordering of a rule task:</p> <ul style="list-style-type: none">• Default the ordering of rules depends on the execution mode.• Literal follows the rule task ordering.• Priority sorts rules according to the execution mode in operation. <p>See Deciding on an execution mode and Rule execution order for details.</p>
Rule task exit criteria	Rule Task tab in Properties view for rule task.	Choose exit criteria specifies how rules are executed before the task terminates. See Exit criteria property for details.

Control properties for rule tasks in execution modes

You can set specific properties to order rule tasks. These settings operate differently in each execution mode: RetePlus, Sequential, and Fastpath. See [Engine execution modes](#) for a description of each execution mode.

The following table outlines how ruleflow control properties operate in RetePlus execution mode.

Table 2. Task ordering properties for RetePlus

Ordering property	Exit Criteria property	Advanced Properties only
<p>Default</p> <p>A RetePlus network with full agenda management is created.</p>	<p>None</p> <p>All the rules in the task body are activated. Rule instances are executed until the agenda is empty.</p> <p>Equivalent to:</p> <p>firing=allrules,</p>	<p>firinglimit > 0: instances are executed until either the given number is reached or the agenda is empty.</p>

	firinglimit=0	
	Rule Only the highest priority rule of the task body is activated. Rule instances are executed until the agenda is empty. Equivalent to: firing=rule, firinglimit=0	
	RuleInstance Only the highest priority rule of the task body is activated. Just the first instance of the rule is executed, or none if the agenda is empty. Equivalent to: firing=rule, firinglimit=1	
Literal A RetePlus network is created but there is no agenda. Rules are activated one by one according to the order provided by the body, and the instances are executed without reevaluation. Or: Priority A RetePlus network is created but there is no agenda. Rules are first sorted in decreasing order of priority. They are then activated one by one and the instances executed without reevaluation.	None A loop is made on each of the rules provided in the task body, their instances computed and executed. Instances are executed until the end of the loop. Equivalent to: firing=allrules, firinglimit=0	firinglimit > 0: rule instances are executed until either the given number or the end of the loop is reached.
	Rule A loop is made on each rule provided in the task body. As soon as a rule can be instantiated (that is, at least one rule instance is created), all those instances are executed and the loop ends. Equivalent to: firing=rule, firinglimit=0	
	RuleInstance A loop is made on each rule provided in the task body. As soon as a rule can be instantiated (that is, at least one rule instance is created) that instance is executed and the loop ends. Equivalent to: firing=rule, firinglimit=1	

The following table outlines how ruleflow properties operate in sequential execution mode.

Table 3. Task ordering properties for Sequential

Ordering property	Exit Criteria property	Advanced Properties only
Default In the sequential mode, the Default property acts the same way as the Priority property.		

<p>Literal</p> <p>Rules are compiled according to the order provided by the task body. Rules are evaluated and executed sequentially against the incoming tuple objects.</p> <p>Or:</p> <p>Priority</p> <p>Rules are sorted according to their priorities. They are then compiled according to the sorted order. The rules are evaluated and executed sequentially against the incoming tuple objects.</p>	<p>None</p> <p>All the rules provided in the task body are evaluated. All the rules evaluated as <code>true</code> are executed until there are no more rules to select. The order is the one determined by the ordering property.</p> <p>Equivalent to:</p> <p><code>firing=allrules,</code> <code>firinglimit=0</code></p>	<p><code>firinglimit > 0</code>: rules evaluated as <code>true</code> are executed until the given number of rules is reached, or there are no more rules to select.</p>
	<p>Rule</p> <p>The rules provided in the task body are evaluated sequentially. As soon as one rule evaluates to <code>true</code>, it is executed and the loop ends.</p> <p>Equivalent to:</p> <p><code>firing=rule</code></p> <p>This property forces the <code>firinglimit</code> to 1.</p>	<p><code>firinglimit</code></p> <p>Not applicable.</p>
	<p>RuleInstance</p> <p>The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), that instance is executed and the loop ends.</p> <p>Equivalent to:</p> <p><code>firing=rule, firinglimit=1</code></p>	

The following table outlines how ruleflow properties operate in Fastpath execution mode.

Table 4. Task ordering properties for Fastpath

Ordering property	Exit Criteria property	Advanced Properties only
<p>Default</p> <p>In the Fastpath mode, the Default property acts the same way as the Priority property.</p>		
<p>Literal</p> <p>Rules are compiled according to the order provided by the task body. Rules are evaluated and executed sequentially against the incoming tuple objects.</p> <p>Or:</p> <p>Priority</p> <p>Rules are sorted according to their priority. They are then compiled according to the sort order. The rules are evaluated and executed sequentially against the incoming tuple objects.</p>	<p>None</p> <p>All the rules provided in the task body are evaluated. All the rules evaluated as <code>true</code> are executed until there are no more rules to select. The order is determined by the ordering property.</p> <p>Equivalent to:</p> <p><code>firing=allrules,</code> <code>firinglimit=0</code></p>	<p><code>firinglimit > 0</code>: rules evaluated as <code>true</code> are executed until the given number of rules is reached, or there are no more rules to select.</p>
	<p>Rule</p> <p>The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), those instances are</p>	

	executed and the loop ends. Equivalent to: firing=rule, firinglimit=0	
	RuleInstance The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), that instance is executed and the loop ends. Equivalent to: firing=rule, firinglimit=1	

Rule execution order

When the rule artifacts for a rule task are selected at compile time or run time, the packages in the list of selected rules are expanded. The expansion starts at the top of the list. If a business rule is listed separately above its package, it retains its execution position in the list. However, if it is listed separately under its package, the business rule executes from the package. Under the package, the business rule is redundant because it is already in the package.

For example, consider a package that contains rules R1, R2 and R3:

- If you add R2 and then the package, the expanded list is R2, R1 and R3.
- If you add the package and then R2, the expanded list is R1, R2 and R3.
- To force R2 to be last in the list, you must list the rules separately: R1, R3, and then R2.

Exit criteria property

The Exit Criteria property specifies whether all the rules execute or just the first instance of the first rule executes. You can set the property to one of the following values:

- None: All the instances of all the applicable rules execute for a tuple before moving to the next tuple.
- Rule: All the rule instances of the first applicable rule execute for a tuple before moving to the next tuple.
- Rule Instance: Only the first rule instance of the first applicable rule executes for a tuple before moving to the next tuple.

Parent topic: [Configuring rule task execution](#)

Related reference:

[firing](#)
[firinglimit](#)

Related information:

[Choosing an execution mode](#)
[Working with ruleflows](#)
[Setting simple priorities among rules](#)

Configuring ruleflow properties

You configure the properties of ruleflow elements in the **Properties** view of each element.

To display the **Properties** view below the editor, you must select a ruleflow element in the diagram. The **Properties** view opens, with a list of tabs where you set the properties for your ruleflow elements. When you add a node or a task to your ruleflow, you must set their properties.

To set the properties for the whole ruleflow, select an empty area in the diagram to open the **Properties** view.

Some properties, or tabs, are common to all ruleflow elements, and are described in the section [Common properties for ruleflow elements](#). Specific properties for each ruleflow element are detailed in the following sections:

- [Ruleflow properties](#)
- [Rule tasks properties](#)
- [Action tasks properties](#)
- [Subflow tasks properties](#)
- [Transitions properties](#)

Common properties for ruleflow elements

Some properties, or tabs, can be found in the **Properties** view for ruleflows, start nodes, end nodes, rule tasks, action tasks, subflows, branches, fork nodes, join nodes, and transitions.

Attention: A start node does not contain the **Final action** tab, and an end node does not contain the **Initial action** tab.

Table 1. Common properties in the tab corresponding to the ruleflow element name

Property	Purpose
ID	<p>An ID is the technical name for a ruleflow element. All ruleflow elements have an ID that identifies them in an XML schema.</p> <p>In Rule Designer Java™ code, the ID of a ruleflow element generates the IRL code that executes at run time. To view this code, click the IRL tab under the Ruleflow Editor.</p> <p>If you duplicate a ruleflow element in a ruleflow, it has a unique ID.</p> <p>IDs are not locale-dependent. When you specify an ID, it is good practice to use only characters (a-z, A-Z) and numbers (0-9).</p>
Label	<p>You can specify a label for the ruleflow element that is shown in the Diagram page. A label has no impact on the IRL that is generated.</p> <p>Labels are locale-dependent. You can specify different labels in a new locale and recover the labels in the original locale for elements that do not change. For example, if you change the ID of a ruleflow element, you lose the labels defined for this element in all other locales except the original.</p> <p>When you specify a label, it is good practice to use only characters (a-z, A-Z) and numbers (0-9).</p>

Table 2. Initial action tab

Property	Purpose
Initial action	<p>The action that launches before the task starts.</p> <p>Select Use BAL for action to define the action in BAL, or enter IRL code.</p>

Table 3. Final action tab

Property	Purpose
Final action	<p>The action that launches after the task ends.</p> <p>Select Use BAL for action to define the action in BAL, or enter IRL code.</p>

Table 4. Documentation tab

Property	Purpose
Documentat ion	<p>This property displays optional information about the start node. Use this property to provide notes, comments, and other useful information that open as a tooltip.</p>

--	--

Ruleflow properties

Note: To show the properties, ensure that you click in an empty area of the Diagram page and do not select a ruleflow node.

Table 5. Properties tab


Property	Purpose
locale	This property indicates the language setting of the ruleflow. The default setting is en_US. You cannot edit this field.
main flow task	Use this property to indicate if the ruleflow is the main task. Double-click the value to set it to true or false.
name	The name of the ruleflow.
tags	<div> Use this property to view and edit the values used for the import tags. Click a field and  to open the Values for Tags dialog. </div> <div> Important: Tags are supported for compatibility purposes. Leave the tags empty. </div>

Table 6. Category Filter tab

Property	Purpose
Category Filter	Use this property to edit the rule category filter for the selected ruleflow. Click Edit to open the Category Filter dialog.

Table 7. Imports tab

Property	Purpose
View the imports used for BOM classes	<div> This property displays the imports for BOM classes. You edit this list to manually add or delete imports. </div> <div> The imported BOM or Java classes required by the IRL code in a ruleflow can be viewed and edited in the ruleflow Properties view. The list is populated automatically when you type in IRL code with Content Assist. Edit the list to remove unused imports, that is, the imports that were not removed automatically, or manually add classes if you have entered IRL code without Content Assist. </div> <div> If a ruleflow in a rule package contains IRL code that references variables of the default package, you must import these variables with the use <code><variable></code> declaration. </div>

Rule task properties

Table 8. Rule Task tab

Property	Purpose
Algorithm	<div> Use this section to specify the processing algorithm for the rule task: <ul style="list-style-type: none"> RetePlus Sequential Fastpath </div> <div> See Engine execution modes for more information. </div>
Exit Criteria	<div> Use this section to specify if all rules, one rule, or one rule instance execute: <ul style="list-style-type: none"> None: The default setting that all rules are executed until conditions terminate execution. The rules are executed in a particular order determined by the selected ordering. Rule: Execution terminates after the chosen rule is executed, according to the selected algorithm. </div>

	<ul style="list-style-type: none">• RuleInstance: A single instance of one rule is executed. This rule is determined by the selected ordering.				
Ordering	Use this section to specify the order of rule execution in a rule task as follows: <ul style="list-style-type: none">• Default: The algorithm determines the execution order.• Literal: The order of the rules in the rule task determines the order of rule execution• Priority: the rules are executed following a static priority in decreasing order.				
Advanced properties	<p>The following table summarizes some features you can set using advanced properties.</p> <p><i>Table 8. Some features set by advanced properties</i></p> <table><tr><th>Advanced property</th><th>Description</th></tr><tr><td>firinglimit = n;</td><td>This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to none and in Advanced Properties add firing = allrules rule and firinglimit = n, n>0. For n = 0 use the Exit Criteria options.</td></tr></table>	Advanced property	Description	firinglimit = n;	This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to none and in Advanced Properties add firing = allrules rule and firinglimit = n, n>0. For n = 0 use the Exit Criteria options.
Advanced property	Description				
firinglimit = n;	This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to none and in Advanced Properties add firing = allrules rule and firinglimit = n, n>0. For n = 0 use the Exit Criteria options.				

Table 10. Rule Selection tab

Property	Purpose
List the rules and rule packages in the rule task	Use this property to list the rules and rule packages that are considered when the rule task is executed. To edit this list and reorder it, use the Up and Down buttons.
Dynamic BAL	Use this property to specify runtime rule selection using a dynamic statement BAL. The property runs each time you call the task.
Static BAL	Use this property to specify runtime rule selection using static statement in BAL. The property runs the first time that you call the task.
IRL	Use this property to specify runtime rule selection using IRL. Define a rule filter in IRL with "body ="

Action task properties

Table 11. Action Task tab

Property	Purpose
Rule Execution	You can select the algorithm, exit criteria, and ordering for running the rule task.

Table 12. Rule Selection tab

Property	Purpose
Edit	You list the rules and rule package that are used in the rule task.
Choose runtime rule selection	You select the runtime language for the rule: Dynamic BAL, Static BAL, or IRL.

Subflow properties

Table 13. Subflow Task tab

--	--

Property	Purpose
Select a ruleflow	Click Edit to open the Select a ruleflow dialog.

Transition properties

Table 14. Condition tab

Property	Purpose
Conditions	<p>In the Conditions tab, you must first select the rule language you use to write the transition: BAL or IRL. In the rule language you chose, you write an expression whose return value must be true or false, for example:</p> <ul style="list-style-type: none">• BAL: <div>'the loan' is approved</div> <ul style="list-style-type: none">• IRL: <div>(bicycle.speed > 10) && (bicycle.speed <= 30)</div> <p>If you have several transitions originating from a task, there must be one (and only one) transition with no conditions, which is considered as the default transition, also called <code>else</code> transition. By default, if you do not specify a label, it is displayed as "else".</p> <p>With several transitions, you must make sure that transitions do not overlap, which means that no more than one transition condition must return "true". Typically, you might want to use two transitions, to choose between one ruleflow path with a specific condition and one default <code>else</code> transition for the remaining cases.</p> <p>If there is only one transition that links two tasks, you must not specify any conditions.</p>

Parent topic: [Working with ruleflows](#)

Configuring rule execution

You can notify the rule engine of changes in object states. You can also specify how rules in a ruleset are executed at run time by setting up the ordering, priorities, or overriding of rules.

Updating object states in the rule engine

When rules are executed with the RetePlus execution mode, you can control whether the rule engine is notified of changes in the state of objects in the working memory that result from rule execution. Notifying the rule engine of an object state change causes the rules to be matched against the new state, and can result in new rule instances being added to the agenda.

Ordering rules in a ruleset

The order in which rules are presented in the ruleset affect the order in which they are placed in the ruleset archive during the final packaging of the ruleset for execution. You can sort rules alphabetically, or manually.

Setting simple priorities among rules

Rules have priorities, which can be static or dynamic. Use a constant to define a static priority, or an expression containing a ruleset variable to define a dynamic priority.

Rule overriding

Rule overriding is the last selection mechanism after ordering and priorities. Overridden rules are not selected for execution. You can combine rule overriding with the hierarchical property.

Parent topic: [Orchestrating ruleset execution](#)

Updating object states in the rule engine

When rules are executed with the RetePlus execution mode, you can control whether the rule engine is notified of changes in the state of objects in the working memory that result from rule execution. Notifying the rule engine of an object state change causes the rules to be matched against the new state, and can result in new rule instances being added to the agenda.

Object state update

In RetePlus execution mode, objects are stored in working memory. You set the "object state update" option to notify the rule engine of a change, and match the rules against the object new state.

Notifying the rule engine of object updates

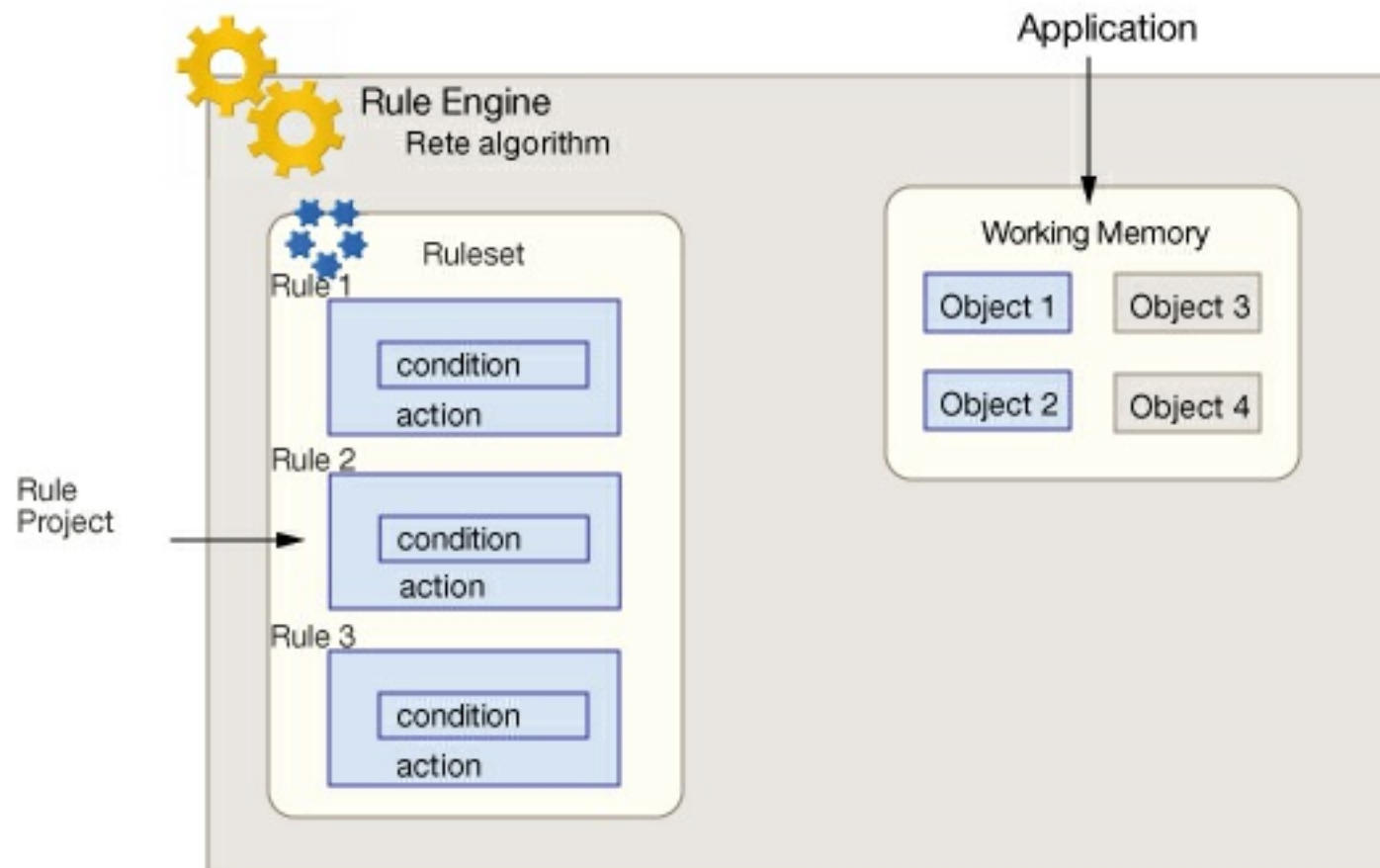
When operating in RetePlus execution mode, you can specify which methods to use to notify the rule engine of object updates.

Parent topic: [Configuring rule execution](#)

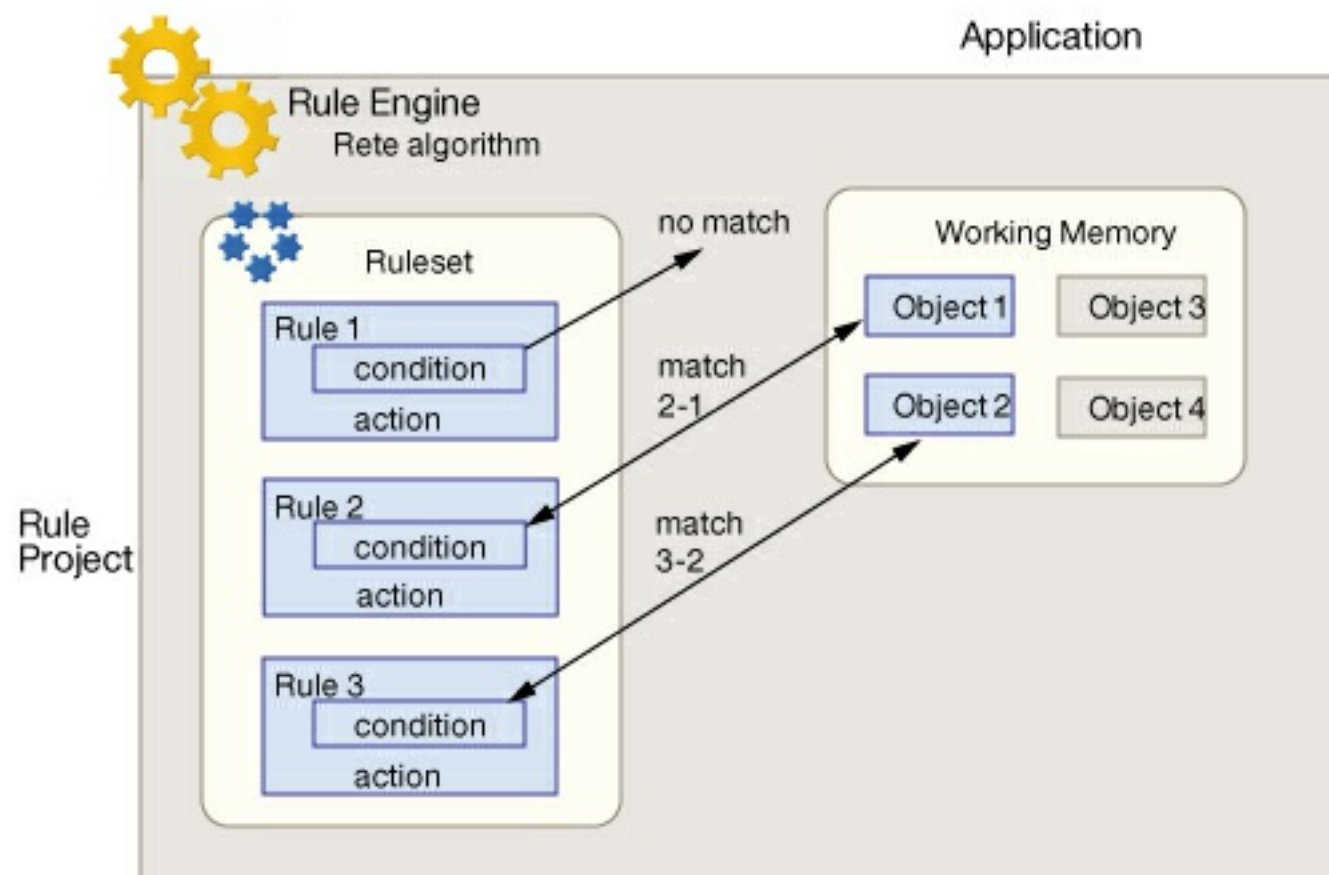
Object state update

In RetePlus execution mode, objects are stored in working memory. You set the "object state update" option to notify the rule engine of a change, and match the rules against the object new state.

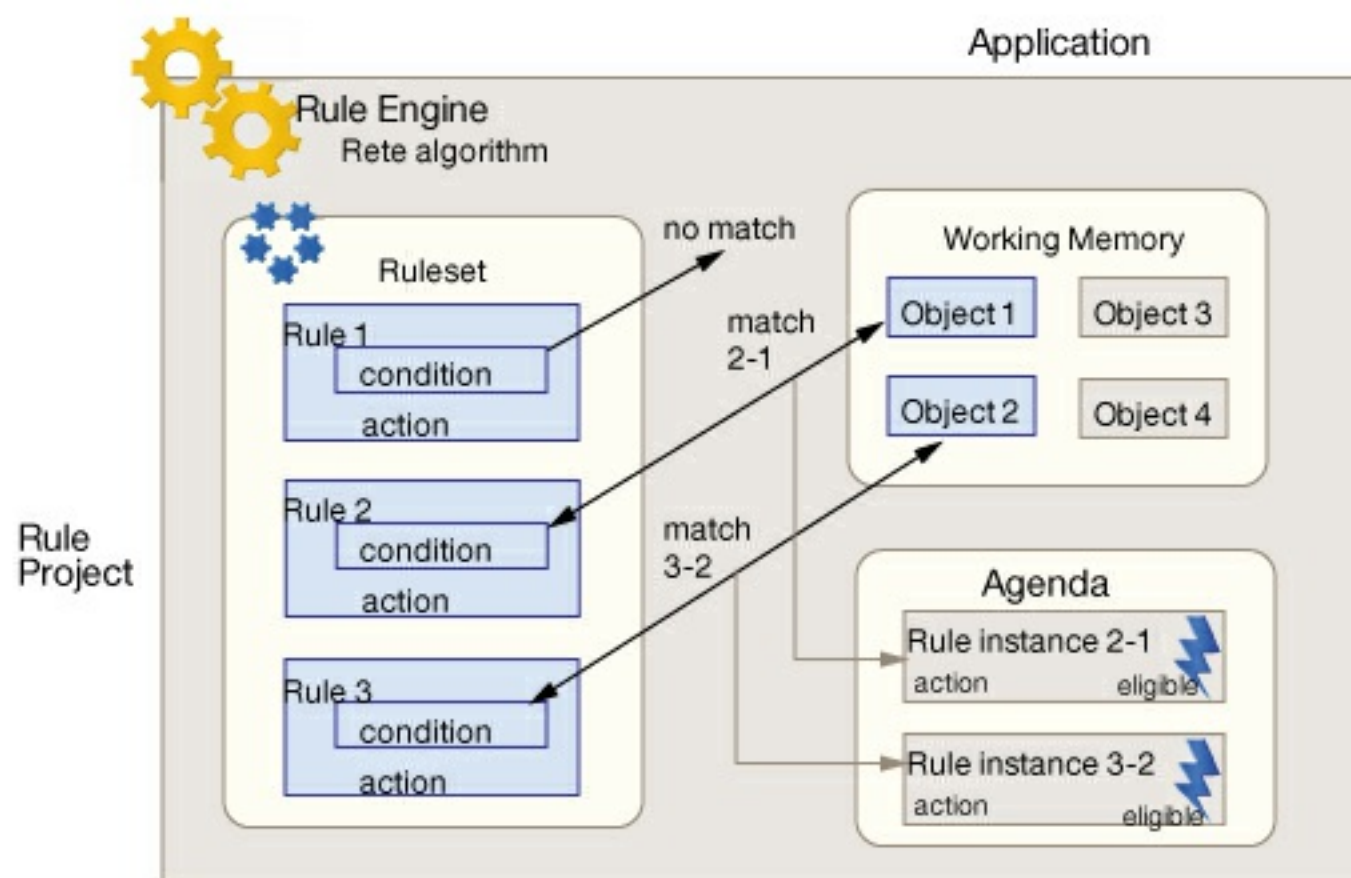
When you execute rules using the RetePlus execution mode, the rules are compiled into a ruleset and sent to the rule engine. In the meantime, application objects are loaded into the rule engine in what is called the working memory.



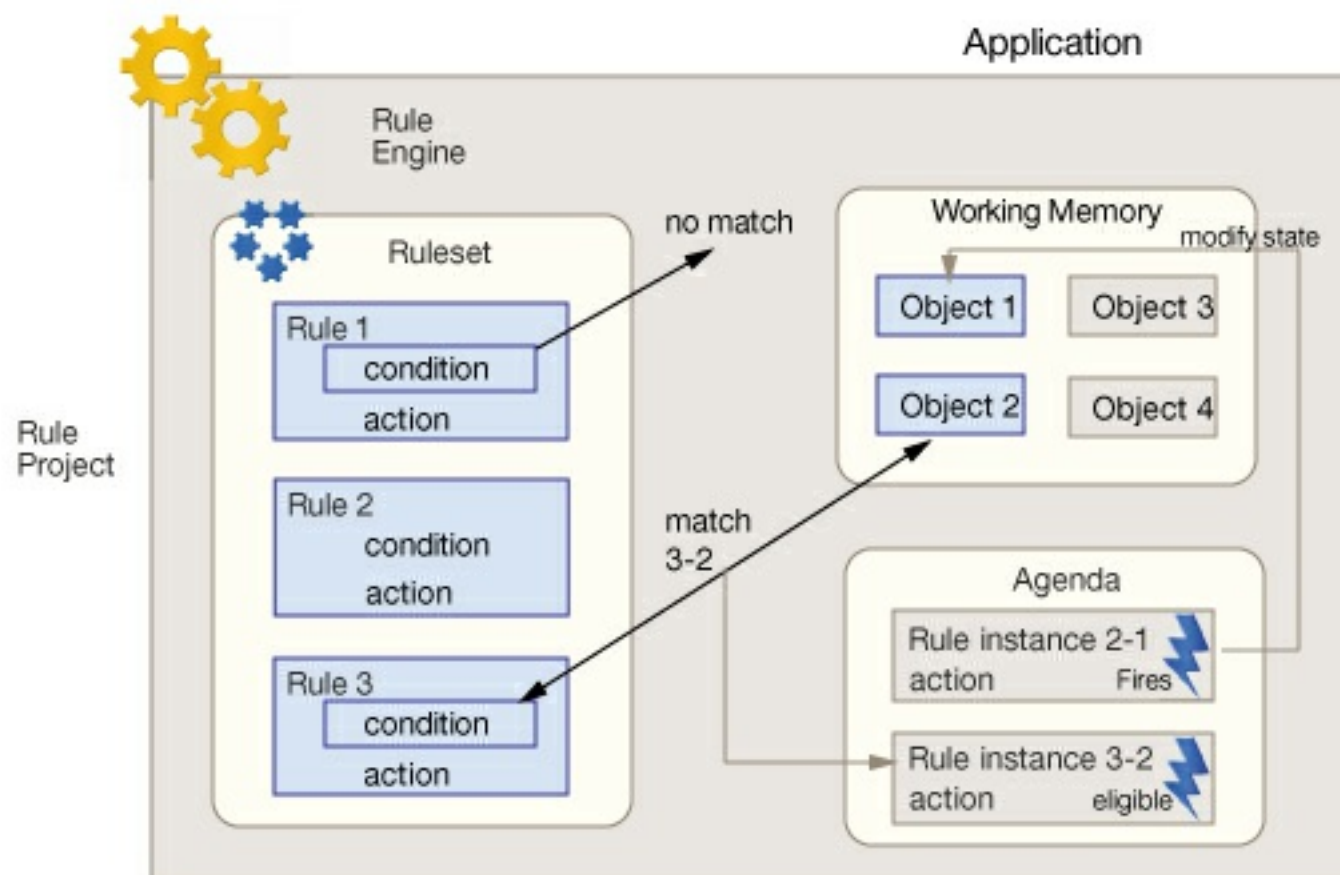
The condition part of the rules is then matched against the objects in working memory.



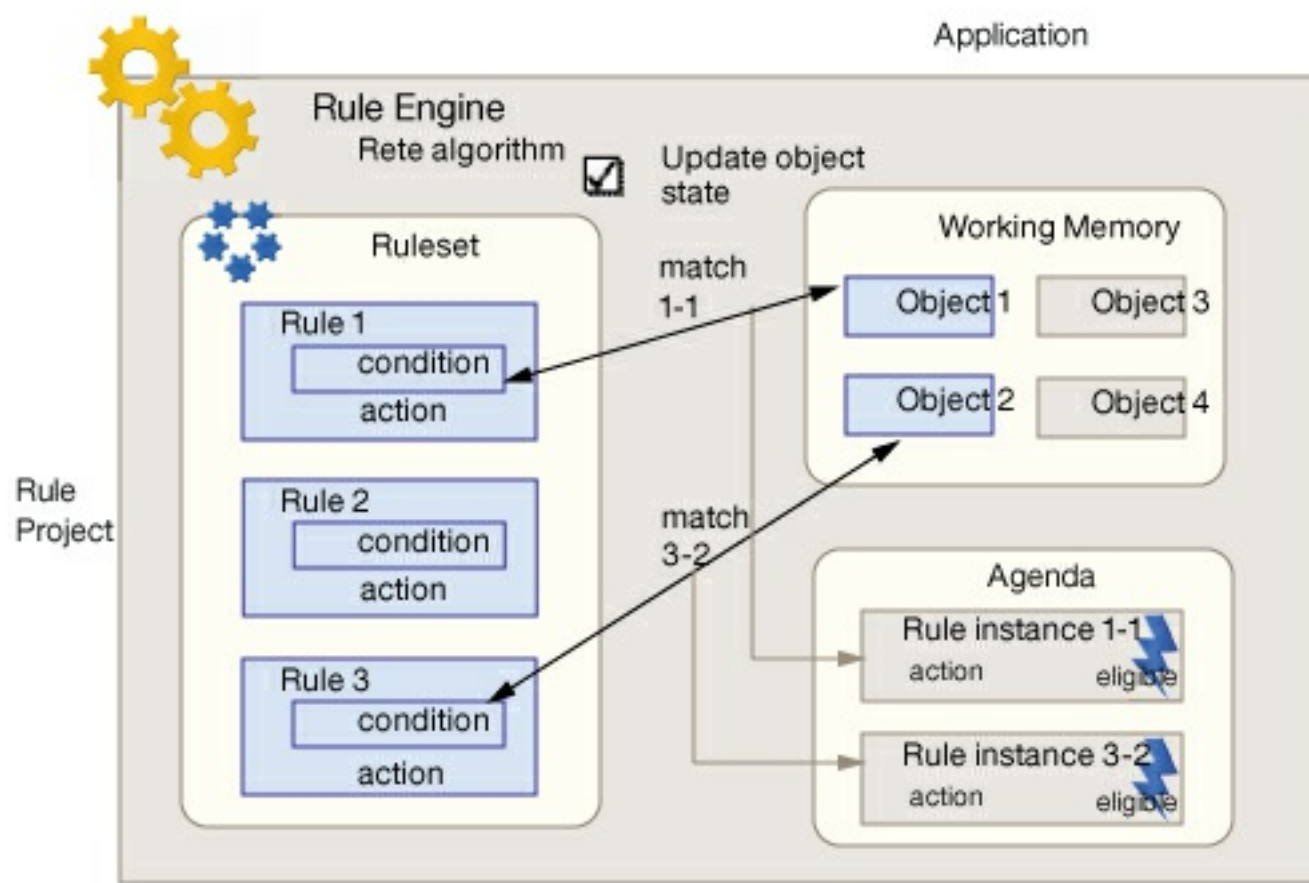
For each match between the conditions of the rules and an object, a rule instance is put into the rule execution agenda.



The first rule in the agenda is then executed.



The action of this rule sometimes modifies the state of an object in working memory. When the **Update object state** box is not selected in the BOM Editor for the method that is executed in the rule actions, the rule engine is not notified of this change in object state. If you select the **Update object state** box, the rule engine is notified of the change and rules are matched against the new state of the object, which might result in new rule instances being added to the agenda.



Parent topic: [Updating object states in the rule engine](#)

Notifying the rule engine of object updates

When operating in RetePlus execution mode, you can specify which methods to use to notify the rule engine of object updates.

About this task

If you want the rule engine to be notified when a method changes the state of an object, ruleset parameter, or ruleset variable when rules are executed with the RetePlus execution mode, you can specify which methods to use to activate a notification.

Procedure

To specify the methods that can activate updates:

1. In the Outline view, click the method you want to specify.
2. On the Member page of the BOM Editor, in the General Information section, select the **Update object state** check box.
3. Save the BOM entry.

Now when you execute rules with the RetePlus execution mode, the rule engine is notified of changes made by this method on ruleset parameters, ruleset variables, or objects.

Parent topic: [Updating object states in the rule engine](#)


Related information:

[Object state update](#)

Ordering rules in a ruleset

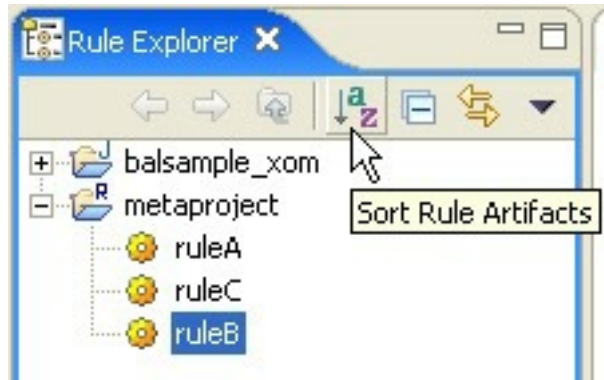
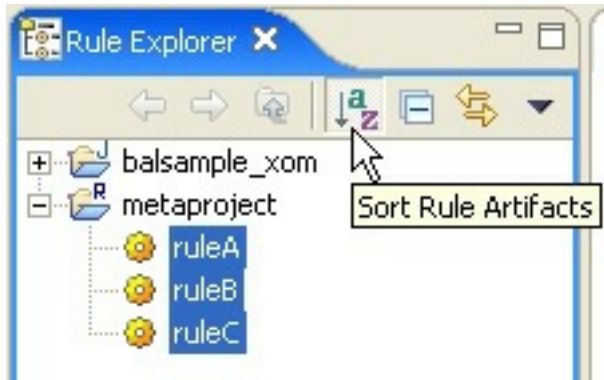
The order in which rules are presented in the ruleset affect the order in which they are placed in the ruleset archive during the final packaging of the ruleset for execution. You can sort rules alphabetically, or manually.

Order rules alphabetically

In the rule project, you can sort rule artifacts alphabetically. Select your rule project in the Rule Explorer, and click  **Sort Rule Artifacts**.

You can display the rules in a project alphabetically without changing the order in which the rules run.

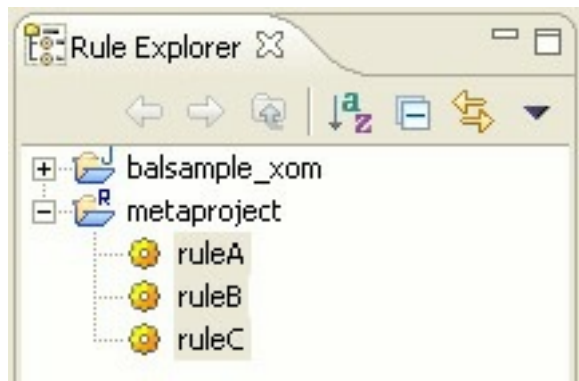
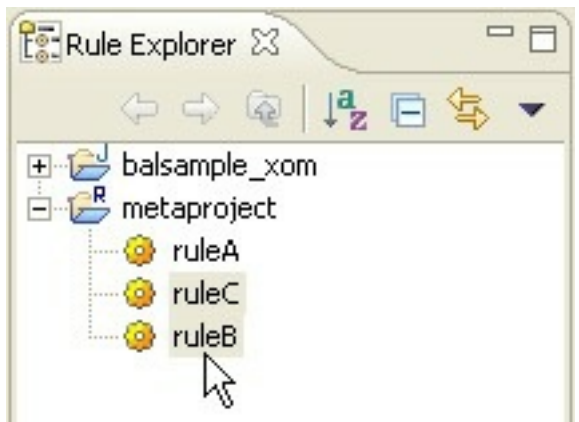
Note: The button remains active until you click it again.

BEFORE: Rules in random order	AFTER: Rules in alphabetic order
	

Order rules manually

One of the ways of determining the initial rule order is to order them manually in the Rule Explorer. Drag each rule you want to move, and drop them over the new location.

Note: The **Sort Rule Artifacts** icon must be deactivated before you can use the drag-and-drop method. If it is activated, click it again to deactivate it.


BEFORE: Rules not in required order	AFTER: ruleB moved to required position
	

Parent topic: [Configuring rule execution](#)

Setting simple priorities among rules

Rules have priorities, which can be static or dynamic. Use a constant to define a static priority, or an expression containing a ruleset variable to define a dynamic priority.

To define the order in which rules are executed, you set the priority of a rule in the **Properties** view of the relevant rule artifact, with the **priority** property.

Note: To be able to edit the property, make sure that the “hand” icon  is visible next to the **Value** column header in the **Properties** view.

Static priority

Use a static priority to change the sequence of rule execution among rules. Static priorities are integers, the relative values of which determine the priorities among the rules. You can also use a static priority to change the order of execution between several instances of the same rule when they are eligible for execution at the same time.

You define static priorities using a constant. In the **Value** field of the **priority** property, you type a number that represents the priority. The number can be any Java™ integer value between -10^9 and $+10^9$. The larger the number, the higher the execution priority of the rule.

Dynamic priority

A dynamic priority is an expression whose value depends on ruleset variables bound in the condition part of a rule.

Note: An error message is displayed when a ruleset archive includes a ruletask in sequential mode that selects rules with a dynamic rule priority.

In the **Value** field of the **priority** property, you enter an expression that uses a ruleset variable. The expression can use any variables defined in the condition part of the rule provided that its scope is for the entire ruleset. If the expression returns a number that is not an integer, it is converted to an integer following the Java language specification.

Here we have two examples:

- `prior1 + ?p`
- `-?a`

Parent topic: [Configuring rule execution](#)

Rule overriding

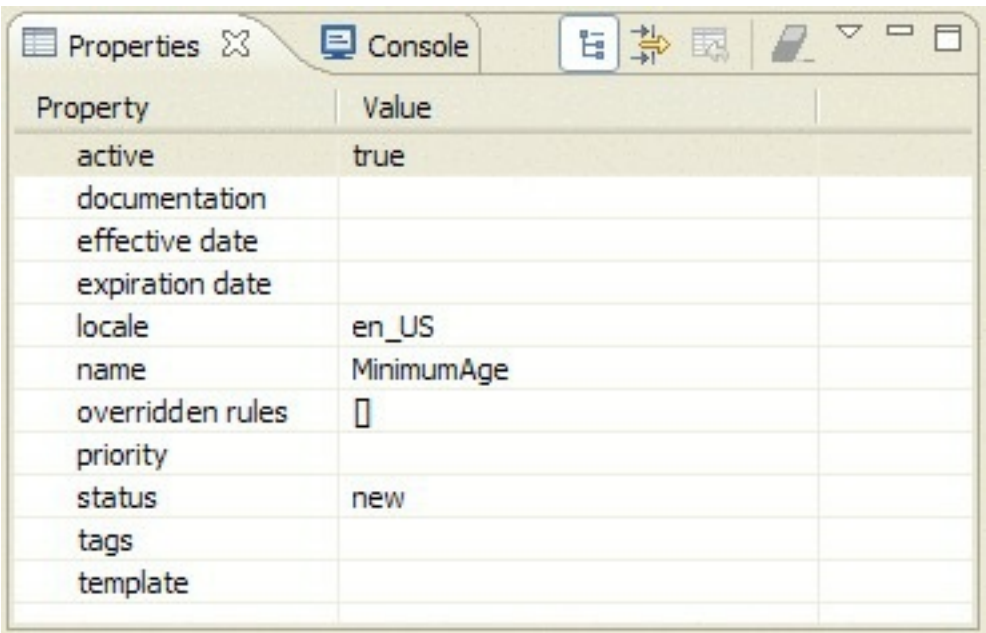
Rule overriding is the last selection mechanism after ordering and priorities. Overridden rules are not selected for execution. You can combine rule overriding with the hierarchical property.


Rule overriding is the final stage of rule selection. After all other selection mechanisms, you can specify that certain rules override other rules. Overridden rules are filtered out and not selected for execution.

A rule can override one or more other rules if these rules are selected in the same rule task at run time. Rule overriding occurs at the individual rule, decision table, or decision tree level. You can set an entire business decision table to override another decision table or tree, for example.

You create a rule override in the **Properties** view of the relevant artifact, with the **overridden rules** property. Click the . . . button next to the property name, and click **Add** to add rules to be overridden by the current artifact. In the **Values for overridden rules** dialog, enter one of the following attributes:

- The name of the rule that you want this artifact to override
- The name of each rule that you want this artifact to override, separated by commas; or
- Select from a list by clicking the . . . button.

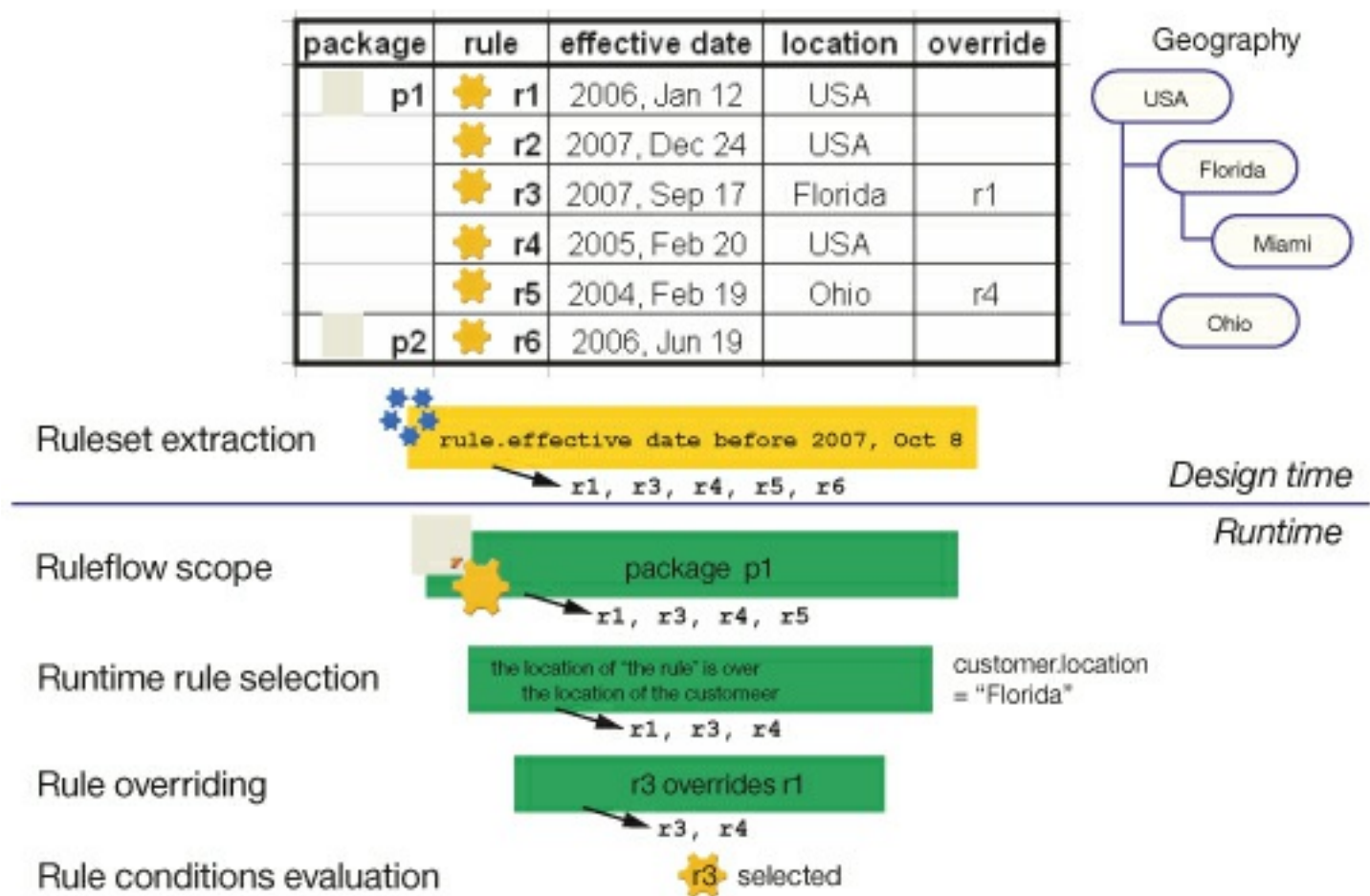


Note: To be able to edit the property, make sure that the “hand” icon  is visible next to the **Value** column header in the Properties view.

Combine rule overriding with the hierarchical property

You can also combine rule overriding with the hierarchical property, enabling specific rules to override more general rules. You specify overriding in Rule Designer by setting the overriddenRules rule property. Using the overriddenRules property, you can select which rule artifacts are to be overridden and specify the overriding relationship between a rule and other rules.

The following diagram shows how rule overriding works in a hierarchical rule structure.



Selection operates as follows:

1. The first selection is based on the effective date property. This drops rule r2 from the selection, leaving r1, r3, r4, r5, and r6.
2. At run time, only the first package is selected (for example, p1 has been added to the rule task, but p2 has not). As a result, r6 is filtered out of the selection.
3. Runtime rule selection is set to the location of the rule **is over** 'Florida', so only USA and Florida rules are considered for execution (see the Geography hierarchy tree in the diagram). As a result, r5 is filtered out of the selection.
4. Finally, r3 overrides r1. Therefore, only the conditions of r3 and r4 are left to be evaluated. The rules whose actions match the required conditions are executed.

Parent topic: [Configuring rule execution](#)

Related reference:

[firing](#)
[firinglimit](#)

Related information:

[Working with ruleflows](#)
[Setting simple priorities among rules](#)
[Ordering rules in a ruleset](#)

Defining the ruleset content and signature

You define the content of a ruleset and its signature.

A ruleset is an executable package that includes rule artifacts and other elements. It contains a set of rules that can be executed by the rule engine. You must define the ruleset content and the parameters that allow the client application to exchange information with the ruleset.

In a decision service, the decision operation includes all the settings needed to define the contents of the ruleset and its parameters.

Ruleset content

The ruleset content is defined by specifying the following information:

- The source rule project. All the rules and variables contained in this project and any of its dependent projects become eligible to be included in the ruleset.
- You can specify any ruleflow to include in the ruleset.
- Any queries or validators to filter the rules, so that only a subset of the rules of the source rule project and its dependent projects are included in the ruleset. Typically, this is used to include rules from specific packages of a project or according to a rule property such as status.

Ruleset signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation for a decision service. The ruleset signature is stored in a ruleset archive as an XML file.

The parameters of a ruleset can have three directions:

- IN: The parameter value is provided as input to the ruleset on execution.
- OUT: The parameter value is set by the execution of the ruleset and provided as output from the ruleset at execution completion.
- IN_OUT: The parameter value is provided as input to the ruleset on execution and its value can be modified by the ruleset and provided as output at execution completion.

In a decision service, you create the parameters from any of the ruleset variables that are available in the rule projects that are part of the scope of the decision operation. Ruleset variables are internal to a ruleset and provide a way to exchange data between rules, functions, and tasks.

Note:

Only ruleset variables that are defined at the top-level package are eligible to be used as ruleset parameters.

Parent topic: [Developing rulesets in Rule Designer](#)

Related concepts:

[Setting up a project hierarchy](#)

[Executing rulesets in the decision engine](#)

Related tasks:

[Running and debugging decision operations](#)

Designing projects for rule authoring

You use projects to design the data model for rules, and define a vocabulary fitted for business users. You can also author different types of business rules.

[Designing business object models](#)

The way you design a business object model (BOM) depends on the nature of the data you use as its basis. You can design a BOM for a Java™ model or for an XML model.

[Configuring the BOM for rule authoring](#)

After designing a BOM for an underlying object model, you configure it to maximize the efficiency of rule authoring activities.

[Authoring business rules](#)

You use Rule Designer to create and edit different types of rules. You can use automatic variables, ruleset variables, templates, and categories to simplify the rule creation process.

[Reviewing a rule project](#)

Manage rule project items and run queries and reports using the features provided in Rule Designer.

Designing business object models

The way you design a business object model (BOM) depends on the nature of the data you use as its basis. You can design a BOM for a Java™ model or for an XML model.

Overview: BOM and execution object model (XOM)

The XOM is the model used to execute the rules.

Creating BOM entries

BOM entries are files that describe a part of a Business Object Model (BOM).

Designing a BOM for a Java model

You design a BOM for a Java model by building a XOM from compiled Java classes or from an XML schema, mapping BOM elements to XOM elements, and configuring BOM updates.

Designing a BOM for an XML model

You can use data other than Java as the basis for your XOM.

Updating the BOM when the XOM changes

How you manage XOM changes in the BOM depends on whether your rule project references a Java project as a XOM.

Parent topic: [Designing projects for rule authoring](#)

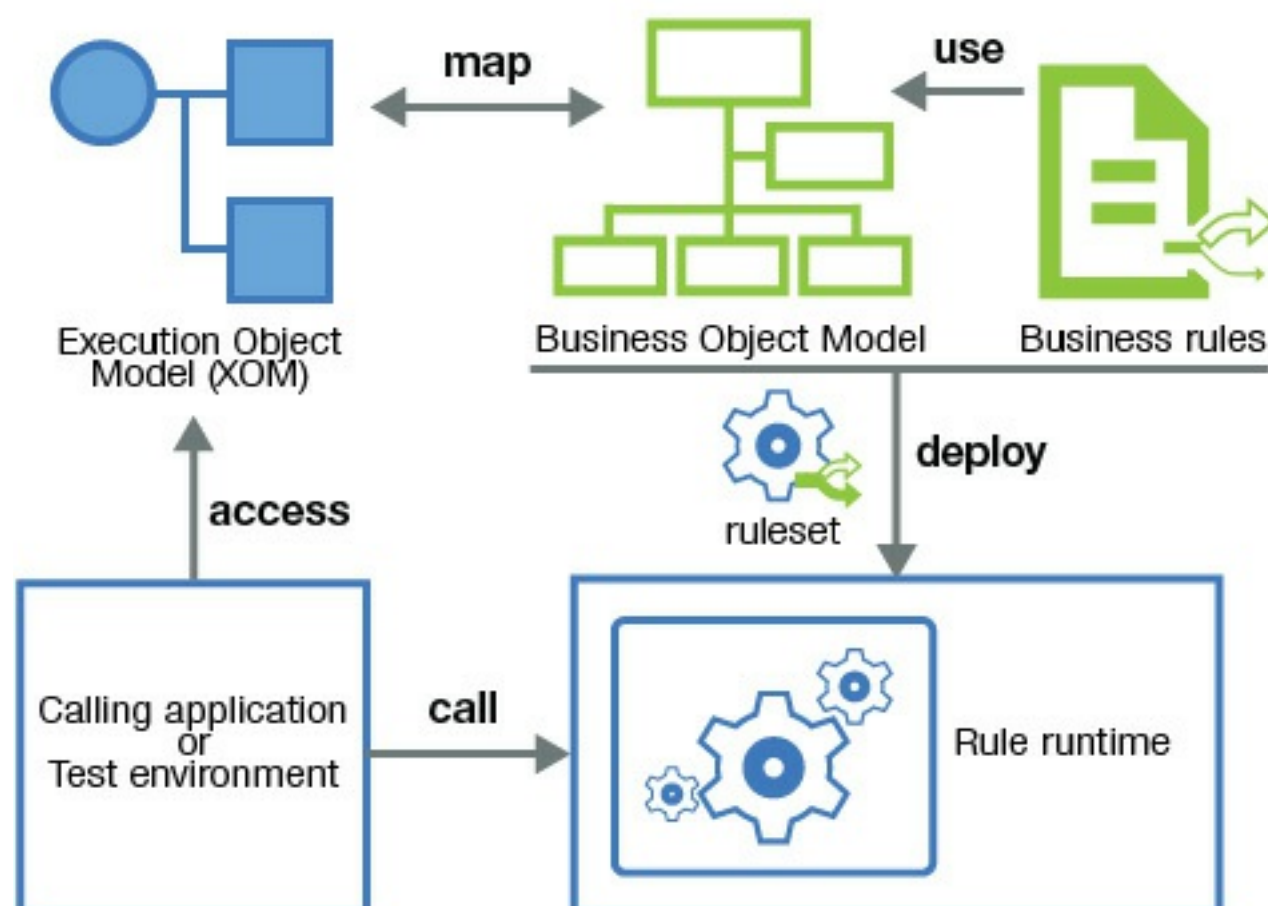
Overview: BOM and execution object model (XOM)

The XOM is the model used to execute the rules.

You can build the XOM from different data sources.

The execution object model (XOM) is the model against which you run rules. It references the application objects and data, and is the base implementation of the business object model (BOM). Rule projects reference the XOM.

Through the XOM, the rule engine can access application objects and methods, which can be Java™ objects, XML data, or data from other sources. At run time, rules that were written against the BOM are run against the XOM.



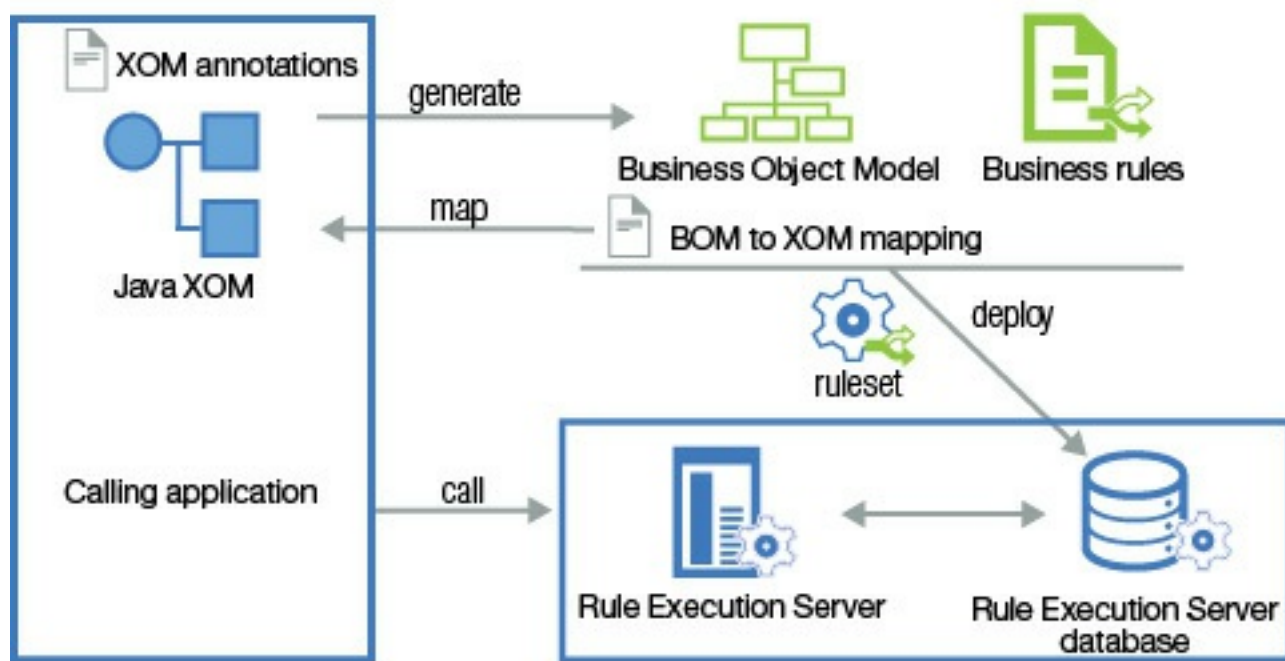
Every BOM element (business element) must have a corresponding XOM element (execution element). The correspondence between execution elements and business elements does not need to be one-to-one. If a business element originates from an execution element, you do not need to specify an explicit mapping. If a business element does not originate from an execution element, you must specify a BOM-to-XOM mapping.

Some internal rules exist for the implicit BOM-to-XOM mapping when you generate a BOM from a XOM. For example, a BOM class is implicitly mapped to a XOM class of the same name.

Designing a BOM for a Java model

When your data model is in Java, you can generate a BOM directly from this Java Execution Object Model (XOM), see [Mapping of a BOM created from a Java XOM](#). You can use XOM annotations to configure BOM generation, see [XOM annotations](#). If you want to extend the BOM with business classes and methods, you can then map these business elements to XOM elements using BOM-to-XOM mapping in the BOM Editor. You have two means of specifying BOM-to-XOM mapping: extender mapping or IRL mapping, see [Rule language and extender mapping](#). Business users can then author business rules without having to worry about the underlying data types and platform.

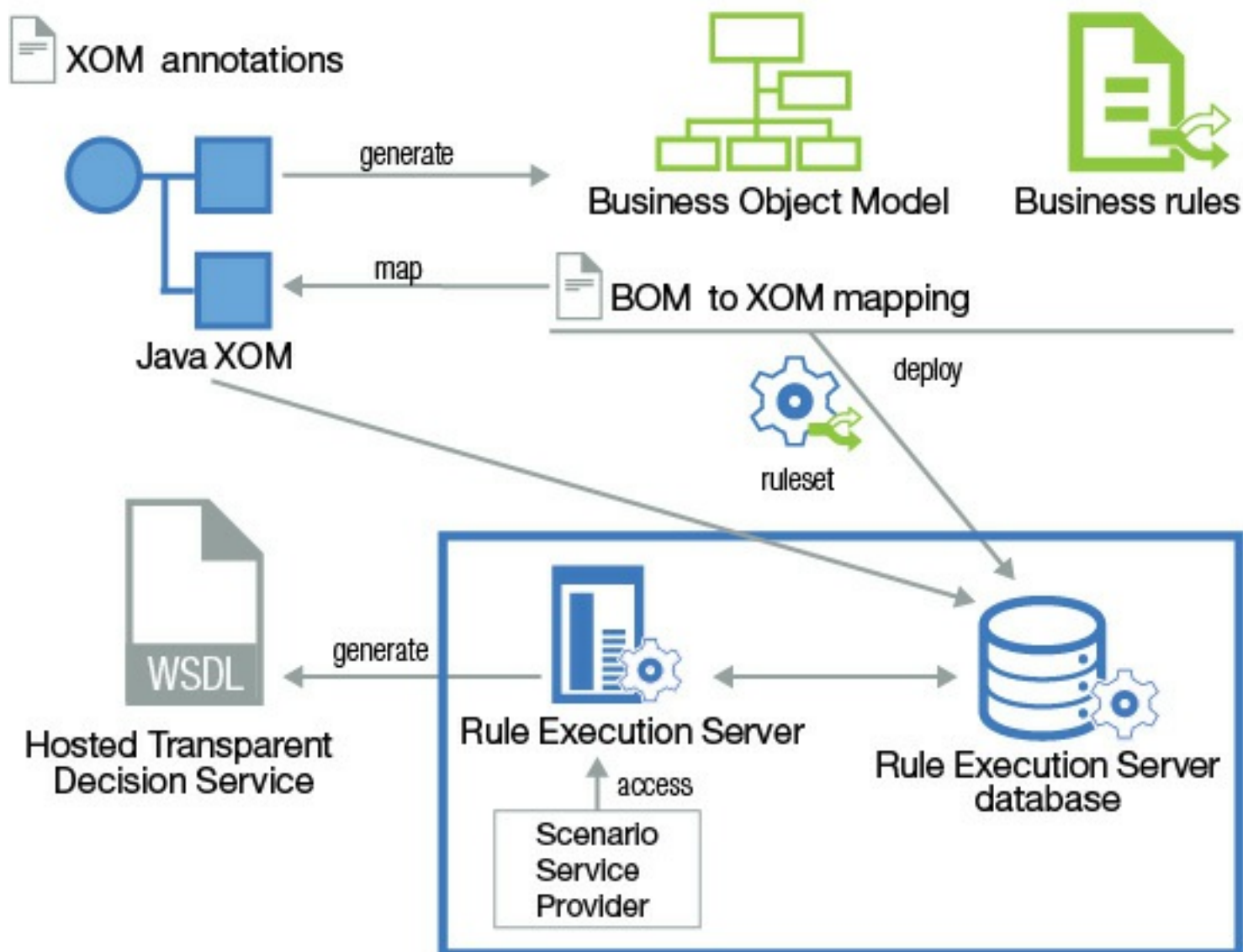
At deployment time, if your application is a Java or web application, then the Java XOM is packaged into the application and there are no specific deployment steps you need to take for the XOM. You package the rules and BOM-to-XOM mapping definition into a ruleset archive, which is then available to your calling application through the rule runtime, Rule Execution Server.



If you do not have a Java or web calling application, either because you want to deploy to a testing server for testing and simulation, or because you want to expose your rules as a Hosted Transparent Decision Service (HTDS), then you must deploy the Java XOM to the Rule Execution Server database.

When you test and simulate rule execution, the Scenario Service Provider (SSP) accesses Rule Execution Server for execution, and Rule Execution Server retrieves the XOM from the database.

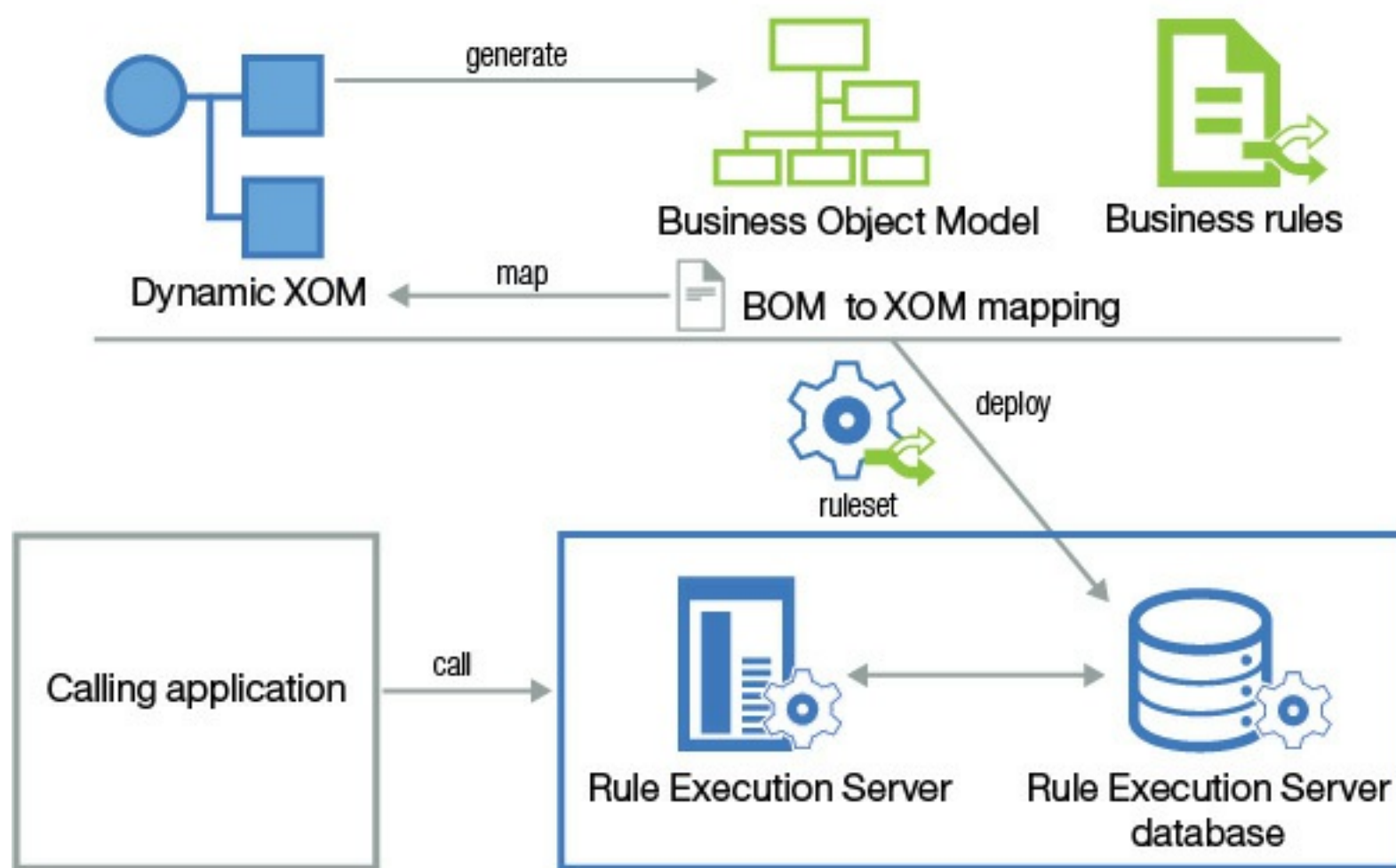
When you generate a HTDS, Rule Execution Server retrieves the XOM from the database in the same way (see [Downloading a HTDS WSDL file](#)).



Designing a BOM for an XML model

When your data model is in XML, you can generate a BOM directly from this Dynamic XOM, see [Overview: Dynamic execution object model \(XOM\)](#) and [Built-in simple types](#). If you want to extend the BOM with business classes and methods, you can then map these business elements to XOM elements using BOM-to-XOM mapping in the BOM Editor. You specify this mapping with IRL mapping, see [Rule language and extender mapping](#). Business users can then author business rules without having to worry about the underlying data types and platform.

At deployment time, the Dynamic XOM and BOM-to-XOM mapping files are packaged with the rules in the ruleset archive, which is then available to your calling application through the rule runtime, Rule Execution Server.



Parent topic: [Designing business object models](#)

Related information:

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

[Defining BOM-to-XOM mapping](#)

Creating BOM entries

BOM entries are files that describe a part of a Business Object Model (BOM).

[Creating a BOM without a XOM](#)

You can work with BOM entries to create business elements and map them to execution elements in the XOM, or reference BOM entry files in other rule projects.

[Creating a BOM entry from a XOM](#)

If you create a BOM entry from a XOM, you can subsequently extend the BOM without modifying the XOM.

[Specifying the order of BOM entries](#)

If you have an element of the same name in more than one BOM entry, you can control which element to use.

Parent topic: [Designing business object models](#)

Creating a BOM without a XOM

You can work with BOM entries to create business elements and map them to execution elements in the XOM, or reference BOM entry files in other rule projects.

[Creating an empty BOM entry](#)

If you want to create business elements without having to consider how the execution elements are implemented, you can create an empty BOM entry that you then map to the XOM.

[Disabling BOM to XOM mapping checks](#)

To stop the display of build warnings when defining initial BOM entries, you disable BOM to XOM mapping checks.

[Associating an empty BOM entry with a XOM](#)

You associate an empty BOM entry with a XOM by setting a reference from the BOM entry to the XOM.

Parent topic: [Creating BOM entries](#)

Creating an empty BOM entry

If you want to create business elements without having to consider how the execution elements are implemented, you can create an empty BOM entry that you then map to the XOM.

About this task

You can create business elements without considering how the execution elements are implemented. In this case, you create an empty BOM entry that you later map to the Execution Object Model (XOM). You can also extend the BOM without modifying the original XOM.

Procedure

To create an empty BOM entry:

1. In the Rule Explorer view, select the bom folder and then, on the **File** menu, click **New > BOM Entry**.

The New BOM Entry wizard opens.

2. In the **Name** field, type a name for the BOM entry and then select the option **Create an empty BOM entry**.
3. Click **Finish**.

In the Explorer view, the bom folder displays a new BOM entry.

Results

You can now add business elements to this BOM entry. Later, you can map them to an execution object model (XOM).

Parent topic: [Creating a BOM without a XOM](#)

Related tasks:

[Creating a BOM entry from a XOM](#)

[Disabling BOM to XOM mapping checks](#)

[Associating an empty BOM entry with a XOM](#)

[Specifying the order of BOM entries](#)

Related information:

[Business object model \(BOM\)](#)

Disabling BOM to XOM mapping checks

To stop the display of build warnings when defining initial BOM entries, you disable BOM to XOM mapping checks.

About this task

When you work on an empty BOM entry, the business elements you create are not yet mapped to the XOM and so warnings are generated when you build. To suppress these warnings, you can disable checks on the BOM to XOM mapping.

Procedure

To disable BOM to XOM mapping checks:

1. On the **Window** menu, click **Preferences**. (On Mac, click **Preferences** on the Eclipse menu.)
2. In the Preferences dialog, expand **Rule Designer**, and then click **Build**.
3. On the Build page, clear the check box **Perform BOM to XOM checks during build**.
4. Click **Apply**, and then click **Apply and Close**.

BOM to XOM warnings are now disabled at build time.

Parent topic: [Creating a BOM without a XOM](#)

Associating an empty BOM entry with a XOM

You associate an empty BOM entry with a XOM by setting a reference from the BOM entry to the XOM.

About this task

If you define an empty BOM entry and then want to associate it with a XOM, you set the origin of the BOM entry as the path to the XOM.

The BOM origin is a reference from a BOM entry to a XOM. Rule Designer uses this reference to update the BOM entry to:

- Import some additional classes from the XOM into the BOM entry
- Update some BOM classes from a modified version of the XOM

The BOM origin can refer to a JAR, a Java™ project, an XSD, a class folder, or a rule project. If it refers to a rule project, it refers to the XOM of the current rule project.

Procedure

To associate an empty BOM entry with a XOM:

1. Right-click your rule project and then click **Properties**.
2. In the Properties dialog, click **Business Object Model**.
3. In the **Required BOM** entries field, expand the BOM entry and then double-click the item **Origin:none**.
4. In the BOM Entry Origin dialog, in the **Origin URL** field, type the path to the XOM.

Use this format: `xom:/myRuleProject/myXOM`

5. Click **OK**.

The empty BOM entry is now associated with a XOM.

Parent topic: [Creating a BOM without a XOM](#)

Creating a BOM entry from a XOM

If you create a BOM entry from a XOM, you can subsequently extend the BOM without modifying the XOM.

About this task

You can create BOM entries from an existing XOM. You can later extend the BOM without modifying the original XOM.

Procedure

To create a BOM entry from a XOM:

1. In the Rule Explorer view, select the bom folder and then, on the **File** menu, click **New > BOM Entry**.
2. In the New BOM Entry wizard, in the Name field, type a name for the BOM entry, and select the option **Create a BOM entry from a XOM**.
3. Click **Next**.

In the BOM Entry step, you select the XOM entries that you have defined in your rule project to be used as a basis for the BOM entry. These XOM entries can be in your current rule project or in a referenced rule project.

Important:

You can also select the Java Runtime Environment (JRE) or Java Development Kit (JDK) system libraries as a XOM. However, it can make the BOM entry creation fail because some classes cannot be loaded directly. You can click **Browse XOM** again to reload the XOM types. The second time, the failing classes are ignored.

4. In the **Choose a XOM entry** field, type a path with the format xom:/myRuleProject/myXOM, or click **Browse XOM** to select a XOM entry.

The classes in the XOM entry you selected are now shown in the **Select classes** field.

5. In the **Select classes** field, select the classes that you want to have in the BOM entry.

Any classes that you do not select are not available from any of the rule project items. When you select a package or a class, all its members are automatically included in the selection. You can redefine the list of selected classes later on.

Important: BOM types that are derived from XML simple types are filtered out because they do not have a binding at run time. To prevent runtime errors that are caused by a BOM created in a previous release, do any of the following:

- Manually delete BOM types that are derived from XML simple types.
- Regenerate your BOM to ensure that the XML simple types are no longer generated into the BOM.

6. Click **Next**.

In the BOM Verbalization step, you define which BOM elements you want to be verbalized. BOM elements that you do not verbalize are not available in business rule artifacts, but you can still use them in technical rule artifacts. You can redefine the elements to be verbalized later on.

7. Click **Finish**.

In the Rule Explorer view, the bom folder displays a new BOM entry.

Results

You can now edit the verbalization of business elements in the BOM Editor, and use them in your rule artifacts.

Parent topic: [Creating BOM entries](#)

Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

Related tasks:

[Disabling BOM to XOM mapping checks](#)

[Associating an empty BOM entry with a XOM](#)

[Specifying the order of BOM entries](#)

[Creating an empty BOM entry](#)

Related information:

[Defining BOM-to-XOM mapping](#)

[Business object model \(BOM\)](#)

Specifying the order of BOM entries

If you have an element of the same name in more than one BOM entry, you can control which element to use.

About this task

The business object model (BOM) defines a path to a set of BOM entries. These BOM entries can be in the current rule project or in a referenced rule project.

If you have several BOM entries, an element in the first BOM entry in the path overrides any other element with the same name in the other BOM entries. You can modify the order of the BOM entries in the path to control which element is selected at run time.

Procedure

To modify the order of BOM entries:

1. In the Rule Explorer view, select the rule project and, on the **Project** menu, click **Properties**.
2. In the side pane of the Rule Project Properties dialog, click **Business Object Model** to display the list of BOM entries in the business object model.
3. Click the **Order** tab.
4. On the Order page, select a BOM entry and then use the **Up** and **Down** buttons to change the order in which they are listed.
5. Click **Apply and Close**.

The order in which the BOM entries are found in the business object model path is now changed.

Parent topic: [Creating BOM entries](#)

Designing a BOM for a Java model

You design a BOM for a Java™ model by building a XOM from compiled Java classes or from an XML schema, mapping BOM elements to XOM elements, and configuring BOM updates.

Defining a Java XOM for a rule project

You can define a Java XOM for a rule project, add resources and references, and order Java XOM entries. You can also deploy a Java XOM using the Java Build Path.

Defining BOM-to-XOM mapping

You can map different business elements to the XOM. These elements include business classes, attributes, method calls, and constructor calls. You can also implement synthetic objects, and access ruleset variables, parameters, and functions from rule language mapping code.

Mapping of a BOM created from a Java XOM

When you create a BOM from the XOM, Rule Designer processes the Java compiled classes into business elements.

Adding annotations to the XOM

You can add annotations to the XOM to customize the way that the BOM is created from Java classes.

Parent topic: [Designing business object models](#)

Defining a Java XOM for a rule project

You can define a Java™ XOM for a rule project, add resources and references, and order Java XOM entries. You can also deploy a Java XOM using the Java Build Path.

Defining a Java XOM

You can specify the Java XOM when creating the rule project, or later using the rule project Properties dialog.

Adding a JAR file or class folder to a Java XOM

You can add a JAR file or a class folder to a Java XOM.

Setting the source location of a library

You must specify the location of the source files for a library.

Defining the order of Java XOM entries

You can specify the order in which Java XOM entries are listed.

Parent topic: [Designing a BOM for a Java model](#)

Defining a Java XOM

You can specify the Java™ XOM when creating the rule project, or later using the rule project Properties dialog.

Procedure

To define a Java XOM using the Properties dialog:

1. In the Rule Explorer view, select the rule project and then on the **Project** menu, click **Properties**.
You can also right-click the project name and select **Properties** from the contextual menu.
2. Click **Java Execution Object Model** in the list of properties.
3. From the list **Required Java projects**, select each Java project that you want for the XOM by clicking the check box next to it.
Click the **Select All** button if all the Java projects are necessary.
4. Click **Apply and Close**.

Results

The Java XOM is now defined. Using the other tabs of the same dialog, you can also add a JAR file or class folder, and define the order of Java XOM entries.

Parent topic: [Defining a Java XOM for a rule project](#)

Related tasks:

[Adding a JAR file or class folder to a Java XOM](#)

[Setting the source location of a library](#)

[Defining the order of Java XOM entries](#)

Adding a JAR file or class folder to a Java XOM

You can add a JAR file or a class folder to a Java™ XOM.

Procedure

To add a JAR file or a class folder:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.

You can also right-click the project name and select **Properties** from the contextual menu.

2. Click **Java Execution Object Model** in the list of properties.
3. Click the **Libraries** tab to display the Libraries page so that you can select the required JAR files and class folders.
 - To add JAR files that have been imported in the Eclipse workspace, click **Add JARs**. A dialog opens for you to browse JAR files in the workspace.
 - To add JAR files that are not in the workspace, click **Add External JARs**. A dialog opens for you to browse the JAR files in your file system.
 - To add class folders, click **Add Class Folder**. A dialog opens for you to browse directories in the workspace. In this same dialog, you can add external class folders by creating a folder in the workspace that links to an external folder in the file system.

Note:

By default, the Java Runtime Environment (JRE) library is already in the list of libraries. This library is the same as the one used to run Eclipse. You cannot remove it. You can browse the JRE library to see the JAR files used by this library.

4. Click **Apply and Close**.

Results

You must now specify the location of the source to be used to reference a library.

Parent topic: [Defining a Java XOM for a rule project](#)

Related tasks:

[Defining a Java XOM](#)

[Setting the source location of a library](#)

[Defining the order of Java XOM entries](#)

Setting the source location of a library

You must specify the location of the source files for a library.

Procedure

To set a source location for a library:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.
You can also right-click the project name and select **Properties** from the contextual menu.
2. Click **Java Execution Object Model** in the list of properties.
3. On the Libraries tab, expand the item that you want to edit and then select the **Source** item.
4. Click **Edit**.

A dialog opens for you to select a .zip file or a source folder from the workspace or the file system.

Parent topic: [Defining a Java XOM for a rule project](#)

Related tasks:

[Defining a Java XOM](#)

[Adding a JAR file or class folder to a Java XOM](#)

[Defining the order of Java XOM entries](#)

Defining the order of Java XOM entries

You can specify the order in which Java™ XOM entries are listed.

Procedure

To order Java XOM entries:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.

You can also right-click the project name and select **Properties** from the contextual menu.

2. Click **Java Execution Object Model** in the list of properties.
3. To display a list of all selected Java XOM entries, click the **Order and Export** tab.

Using this tab, you can specify what libraries you want to export when the XOM is deployed to Rule Execution Server.

Note:

All items are checked by default except the JDK.

4. Select one or more entries and then use the **Up** and **Down** buttons to move them relative to each other.

The classes are searched for in the order in which they are shown in this list,

5. Click **Apply and Close**.

Results

If you want to deploy a Java XOM that has Java class dependencies, use the export function of the Java project used to build the BOM.

Parent topic: [Defining a Java XOM for a rule project](#)

Related tasks:

[Defining a Java XOM](#)

[Adding a JAR file or class folder to a Java XOM](#)

Defining BOM-to-XOM mapping

You can map different business elements to the XOM. These elements include business classes, attributes, method calls, and constructor calls. You can also implement synthetic objects, and access ruleset variables, parameters, and functions from rule language mapping code.

[Rule language and extender mapping](#)

Rule artifacts that are created from the BOM must be mapped to the XOM to run at run time. You can use the default mapping, or mappings based on rule language functions or Java extender classes.

[Execution class mapping](#)

You can map a business class to an execution class. By default, the elements of the business class are mapped to the elements of the execution class. Conversely, the BOM-to-XOM mapping also helps to deduce the business class from a particular class instance, such as for testing and simulation.

[Rule language mapping](#)

You can use the rule language mapping code to define the BOM-to-XOM mapping.

[Extender mapping](#)

Extender mapping is designed to use Java rather than rule language. In a Java extender class, you create static elements that have the same name as business class elements.

Parent topic: [Designing a BOM for a Java model](#)

Rule language and extender mapping

Rule artifacts that are created from the BOM must be mapped to the XOM to run at run time. You can use the default mapping, or mappings based on rule language functions or Java™ extender classes.

The BOM-to-XOM mapping mechanism can translate BOM-based rule artifacts into XOM-based rule artifacts at run time. The mechanism provides a default mapping, and supports two kinds of explicit mapping:

- Rule language mapping: You associate a business element with rule language code that is based on the XOM. You can call functions, ruleset parameters and variables, and rule instances.
- Extender mapping: In a Java extender class, you create static elements that have the same name as business class members. You can use extender mapping only on a Java XOM. If you have a dynamic XOM, use rule language mapping.

Important: For rule language mapping, write the mapping in ARL, which is stored in a .b2xa file.

The BOM-to-XOM mapping is defined in a BOM-to-XOM mapping XML schema.

At run time, the rule engine uses the following order to find the right mapping:

1. Explicit mapping in rule language.
2. Explicit mapping with extender class.
3. Implicit mapping, by default when nothing is specified.

If the rule engine does not find a mapping by default, it gives you an error.

Unexpected results for overriding methods

The BAL object operator `is one of` does not use BOM-to-XOM redefined equals. If you apply BOM-to-XOM mapping to a method that overrides another method, you might receive an unexpected result. If there is a direct call to the overriding method, the BOM-to-XOM mapping is applied. However, in an indirect call, such as a call to the method in a superclass or interface, the BOM-to-XOM mapping is not applied to the overriding method.

For example, if you override the method `Object.equals(Object)` in a BOM class and provide a BOM-to-XOM implementation, the BOM-to-XOM implementation is called when an instance of such a class is inside a collection, such as when `HashSet.add(Object)` or `ArrayList.contains(Object)` is called.

Parent topic: [Defining BOM-to-XOM mapping](#)

Related concepts:

[Consistency checking](#)

[Overview: BOM and execution object model \(XOM\)](#)

Related information:

[Business object model \(BOM\)](#)

Execution class mapping

You can map a business class to an execution class. By default, the elements of the business class are mapped to the elements of the execution class. Conversely, the BOM-to-XOM mapping also helps to deduce the business class from a particular class instance, such as for testing and simulation.

To map a business class to an execution class, you must specify the name of the execution class in the BOM Editor.

You define the execution class in the `Execution` name field of the **BOM to XOM Mapping** section of the BOM Editor. For complex mapping, you can use a tester to test the instances of the execution class by using rule language mapping.

When you run the rules, the execution class is used instead of the business class whenever needed.

For best results, apply the following rules to the mapping:

- When the business class is a utility class, map it to void.
- When the business class is an enumerated class, do not provide a tester.
- When you do provide a tester, write it carefully so that it filters out all non-matching class instances, by returning false.

For example, you can map the business class `RichCustomer` to an execution class `Customer`: In the BOM Editor, in the BOM to XOM Mapping section, type a rule language statement that returns a Boolean value in the Tester field. Use this to represent the current object. For example:

```
return this.money > 100000;
```

If members of the business class are not explicitly mapped, the BOM-to-XOM mechanism assumes that they are the same as the members of the execution class. The following table describes how the BOM-to-XOM mechanism uses these members:

Table 1. Implicit mapping to members of an execution class

Member of business class BusinessClass	Case	Mapping to execution member of class ExecutionClass
Constructor BusinessClass(MyBizClass B, MyBizClassC)	Call by new	Constructor ExecutionClass(MyClass B, MyClassC)
Attribute MyBizClassA attr	Assignment	Attribute attr (not read-only)
	Access	Attribute attr (not write-only)
Method MyBizClassA myBizMethod(MyBizClassB, MyBizClassC)	Invocation	Method MyClassA myMethod(MyClassB, MyClassC)
BusinessClass is used	Use of operator InstanceOf, or cast, or classification in conditions	ExecutionClass

Parent topic: [Defining BOM-to-XOM mapping](#)

Rule language mapping

You can use the rule language mapping code to define the BOM-to-XOM mapping.

You define the rule language mapping by selecting an item in the Outline view and entering the mapping in the **BOM to XOM Mapping** section of the BOM Editor.

This table shows members of the business class. The table assumes that an execution class name ExecutionClass is used for the business class BusinessClass.

Table 1. IRL and ARL mapping possibilities

Member of business class BusinessClass	Case	S c o p e	A function body in IRL or ARL where...
Constructor BusinessClass(MyBizClassB, MyBizClassC)	Call by new	-	this cannot be used, returning an instance of ExecutionClass.
Attribute MyBizClassA attr	Assignment	s t a t i c	value can be used and this cannot be used.
		i n s t a n c e	value and this can both be used.
	Access	s t a t i c	this cannot be used, returning an instance of MyClassA.
		i n s t a n c e	this can be used, returning an instance of MyClassA.
Method MyBizClassA myBizMethod(MyBizClassB, MyBizClassC)	Invocation	s t a t i c	this cannot be used.
		i n s t a n c e	this can be used.
Tester	Use of operator InstanceOf, or cast, or classification in conditions	-	this can be used, returning a Boolean.

Class tester mapping

You can use a tester to test the instances of the execution class.

For example, you have a business class that is named RichCustomer in your BOM, and it corresponds to an execution class Customer, which has the attribute money greater than 100000:

- 1. Set the execution name to Customer.
- 2. Enter the following code in the Tester field:

```
return this.money > 100000;
```

Attribute mapping

You can map a BOM attribute to an expression to return an attribute value at run time.

For example, you have a class that is named ShoppingCart in the XOM, and the class contains two methods that return the number of items in the shopping cart and the total value of the items:

```
// Execution class
public class ShoppingCart {
    public double getValue() ...
    public int getNumberOfItems() ...
}
```

The BOM contains an attribute that represents the average item price, which is computed by using the following expression:

```
// BOM class
public class ShoppingCart {
    public double averageItemPrice;
    ...
}
```

In the Getter field of the BOM Editor, enter the expression that computes the value of the attribute:

```
if (this.getNumberOfItems() == 0 )
    return 0;
return this.getValue()/this.getNumberOfItems();
```

Method call mapping

You can map a method call to any expression.

For example, you have a class Customer in the XOM, and the class contains a method that returns the age of the customer:

```
// Execution class
public class Customer {
    public int getAge() ...
}
```

In the BOM, you have a predicate that checks whether the customer is older than a certain age, and the age is passed to the method as a parameter:

```
public class Customer {
    public boolean isOlderThan(int age);
}
```

In the Body field of the BOM Editor, you express the BOM-to-XOM mapping as follows:

```
return this.getAge() > age;
```

Constructor call mapping

You can map a constructor call to a specified expression.

For example, you have a class Customer in the XOM, and the class contains an age attribute and a constructor that takes the name of the customer parameter:

```
// Execution class
public class Customer {
    public Customer(String name);
    public int age;
}
```

The BOM has a different constructor that takes the name and the age of the customer:

```
public class Customer {
    public Customer(String name, int age);
}
```


In the Body field of the BOM Editor, you express the BOM-to-XOM mapping as follows:

```
Customer result = new Customer(name);
result.age = age;
return result;
```

Synthetic object mapping

Instead of regular Java objects, you can use synthetic objects as application data. You can then use BOM-to-XOM mapping to implement a complete business object model by using only the synthetic objects.

In the following example, you use the `java.util.Map` interface to implement synthetic objects, where attributes are stored as pairs of key values. Therefore, only the regular `java.util.Map` interface exists in the XOM.

In the BOM, you have a business class `Customer` defined as follows:

```
Class Customer {
    Customer(String name);
    readonly String name;
    int money;
}
```

The execution class for `Customer` is `java.util.Map`. To facilitate the writing of the member mappings, add the following import:

```
import java.util.*;
import java.lang.*;
```

For the name attribute, the getter part of the BOM-to-XOM mapping is expressed as follows:

```
return (String)this.get("name");
```

For the money attribute, by using an entry `money` that contains an `Integer` in the `Map`, the getter part of the BOM-to-XOM mapping is expressed as follows:

```
return ((Integer)this.get("money"));
```

The setter part is expressed as follows:

```
this.put("money",value);
```

Because you can use the same execution class, which is `java.util.Map` in this example, for more than one business class, you must distinguish which instances belong to which business class. This information must be stored in the `Map` when objects are constructed, and is used by the tester. You express the BOM-to-XOM mapping body for the constructor as follows:

```
Map result = new HashMap(3);
result.put("name", name);
result.put("money",0);
// as assumed, it always contains an Integer
result.put("className", "Customer");
return result;
```

The tester part of the BOM-to-XOM mapping for the class `Customer` uses the `className` entry to differentiate instances of `java.util.Map`:

```
return "Customer".equals(this.get("className"));
```

Parent topic: [Defining BOM-to-XOM mapping](#)

Extender mapping

Extender mapping is designed to use Java™ rather than rule language. In a Java extender class, you create static elements that have the same name as business class elements.

You define an extender class name for a class in the **Extender name** field of the **BOM to XOM Mapping** section of the BOM Editor. The BOM-to-XOM mapping mechanism then looks up extender elements that have the same name as your business elements in the extender class. Therefore, when you create the extender class, make sure that you create elements that can be found from the business element name.

The following table shows extender mapping options. It assumes that an extender class name ExtenderClass is used for a business class named BusinessClass.

Table 1. Extender mapping options

Member of business class BusinessClass	Case	S c o p e	Extender member of class ExtenderClass
Constructor BusinessClass(MyBizClass B, MyBizClassC)	Call by new	-	static ExtenderClass create(MyClassB, MyClassC)
Attribute MyBizClassA attr	Assignment	st at ic	static MyClassA attr —OR— static void setAttr(MyClassA)
		in st a n c e	static void setAttr(ExtenderClass, MyClassA)
	Access	st at ic	static MyClassA —OR— static MyClassA getAttr()
		in st a n c e	static MyClassA getAttr(ExtenderClass)
Method MyBizClassA myBizMethod(MyBizClassB, MyBizClassC)	Invocation	st at ic	static MyClassA myMethod(MyClassB, MyClassC)
		in st a n c e	static MyClassA myMethod(ExtenderClass, MyClassB, MyClassC)
InstanceOf operator (Tester)	Use of operator classification in conditions.	-	static boolean tester(ExtenderClass)

Example

The following code shows an example of an extender class for a BOM class that is named EventDispatcher.

```
public class DispatcherExtender {
    /**
     * This extender method implements
     * EventDispatcher.exception(FinancialEvent, String)
     */
    public static void exception(GenericObject dispatcher,
                               GenericObject event,
```

```

        String message) {
    System.out.println("###> Processing Exception : event "
        + event + message + "<###");
}

/**
 * This extender method implements
 * EventDispatcher.send(FinancialEvent, String)
 */
public static void send(GenericObject dispatcher,
    GenericObject event,
    String target) {
    System.out.println("===> Sending " + event + " to " + target + " <===");
}
}

```

Class tester mapping

The mapping from a business class to an execution class can become complex. For example, you might have several business classes that are mapped to one execution class. To make mapping easier, you can specify a tester to test the instances of the execution class.

For example, you have a RichCustomer business class that corresponds to an execution class Customer:

- Set **Execution name** to Customer.
- Set **Extender name** to RichCustomerExt. This class has no naming conventions.
- Enter the following code in the Tester field:

```

public static boolean tester(Customer customer) {
    return customer.money > 100000;
}

```

The code specifies that the tester method is a Java static method of the extender class that is named tester. It returns a boolean and takes the execution class Customer as its parameter.

Attribute mapping to an expression to return a value

You can use extender mapping to map a BOM attribute to an expression and return an attribute value at run time.

In this example, the method that is used when the attribute averageItemPrice is accessed is provided as a Java static method that is named getAverageItemPrice:

```

public class CartExtender {
    public static double getAverageItemPrice(ShoppingCart cart) {
        if (cart.getNumberOfItems() == 0)
            return 0;
        return cart.getValue()/ cart.getNumberOfItems();
    }
}

```

Attribute mapping to an expression to set a value

You can map a BOM attribute to an expression that sets the value of the attribute at run time.

In this example, the method that is used when the attribute averageItemPrice is accessed is provided as a Java static method that is named setAverageItemPrice:

```

public class CartExtender {
    public static void setAverageItemPrice(ShoppingCart cart, double v) {
        cart.setValue(cart.getNumberOfItems()*v);
    }
}

```

Method call mapping

You can map a method call to a specified expression.

In this example, the method that is used when the method `isOlderThan` is called is provided as a Java static method with the same name in the extender class:

```
public class ExtCustomer {
    public static boolean isOlderThan(Customer customer, int age) {
        return customer.getAge() > age;
    }
}
```

Constructor call mapping

You can map a constructor call to any expression.

In this example, the method that is used when the constructor is called is provided as a Java static method that is called `create`:

```
public class ExtCustomer {
    public static Customer create(String name, int age) {
        Customer c = new Customer(name);
        c.setAge(age);
        return c;
    }
}
```

Synthetic object mapping

You can use a synthetic object to simulate the state of real objects. Like a real object, a synthetic object can store attributes. Service Data Objects (SDO) is an example of a synthetic object framework. You can use synthetic objects instead of regular Java objects as application data. You can then use BOM-to-XOM mapping to implement a complete business object model that uses only synthetic objects.

Specify the name of your extender class in the **Extender name** field, and specify the name of the execution class in the **Execution name** field.

In this example, the extender class defines all the necessary mappings to the interface of the execution class `java.util.Map`:

```
import java.util.*;
public class CustomerExtender {
    public static String getName(Map customer) {
        return (String) customer.get("name");
    }

    public static int getMoney(Map customer) {
        return ((Integer) customer.get("money")).intValue();
    }

    public static void setMoney(Map customer, int value) {
        customer.put("money", new Integer(value));
    }

    public static Map create(String name) {
        Map result = new HashMap(3);
        result.put("name", name);
        result.put("money", new Integer(0));
        // as assumed it always contains an Integer
        result.put("className", "Customer");
        return result;
    }

    public static boolean tester(Map customer) {
        return "Customer".equals(customer.get("className"));
    }
}
```

Parent topic: [Defining BOM-to-XOM mapping](#)

Mapping of a BOM created from a Java XOM

When you create a BOM from the XOM, Rule Designer processes the Java™ compiled classes into business elements.

The following table describes the default mapping for business elements originating from a Java XOM when you keep the option **Load getters and setters as attributes** selected in the New BOM Entry wizard.

Table 1. Business elements originating from a Java XOM

Java XOM element	Becomes element in the BOM...
Nongeneric public class.	Class with the same name.
<div>Class that implements the java.util.Collection interface.</div> <div>For example:</div> <div>MyCollection implements java.util.Collection</div>	<div>Class with a collection domain to be recognized as a collection when verbalized.</div> <div>For example:</div> <div>public class MyCollection implements java.util.Collection { domain 0, * ; }</div>
Public constructor.	Constructor with the same parameters.
Public attribute.	Attribute with the same name and return type.
Final attribute.	Read-only attribute with the same name and return type.
<div>Public static final attributes whose types are the current class.</div> <div>For example:</div> <div>public class Color { private Color(String name) {...} public static final Color red = new Color("red"); public static final Color blue = new Color("blue"); public static final Color green = new Color("green"); }</div>	<div>Enumerated domain of static references of the class.</div> <div>For example:</div> <div>public class Color { domain { static red, static blue, static green } public static final readonly Color red; public static final readonly Color blue; public static final readonly Color green; }</div>
<div>Public method that does not follow the JavaBeans convention for property accessors (void setFoo(PropertyType value) and PropertyType getFoo()).</div> <div>For example:</div> <div>public interface Customer { public boolean register(java.sql.Connection db); }</div>	<div>Method with equivalent parameters.</div> <div>For example:</div> <div>public interface Customer { public abstract boolean register(java.sql.Connection arg); }</div>
Public method that follows the JavaBeans convention, with a get method and no set method.	Read-only attribute.

<p>For example:</p> <pre>public interface Customer { public int getAge(); }</pre>	<p>For example:</p> <pre>public interface Customer { public readonly int age; }</pre>
<p>Public method that follows the JavaBeans convention, with only a set method.</p> <p>For example:</p> <pre>public interface Customer { public void setBirthDate(Date date); }</pre>	<p>Write-only attribute.</p> <p>For example:</p> <pre>public interface Customer { public writeonly java.util.Date birthDate; }</pre>
<p>Public method that follows the JavaBeans convention, with both a get method and a set method.</p> <p>For example:</p> <pre>public interface Customer { public String getName(); public void setName(String name); }</pre>	<p>Attribute.</p> <p>For example:</p> <pre>public interface Customer { public java.lang.String name; }</pre>

Business elements that originate from a dynamic XOM are processed differently. For information about elements that originate from a dynamic XOM, refer to [Designing a BOM for an XML model](#).

Parent topic: [Designing a BOM for a Java model](#)

Related concepts:
[Overview: BOM and execution object model \(XOM\)](#)

Related tasks:
[Creating a BOM entry from a XOM](#)

Related information:
[Business object model \(BOM\)](#)
[Updating the BOM when the XOM changes](#)

Adding annotations to the XOM

You can add annotations to the XOM to customize the way that the BOM is created from Java™ classes.

XOM annotations

You can use annotations in your Java source code. Annotations are a type of metadata that you can add to classes, members and parameters.

@NotBusiness

You use the @NotBusiness annotation to filter out classes and members that you do not want to import.

@BusinessName

You use the @BusinessName annotation to provide a name for a class, a member, or a parameter in the BOM.

@BusinessType

You use the @BusinessType annotation to change the type of a member or a parameter in the BOM.

@BoundedIntDomain

You use the @BoundedIntDomain to specify an interval between two bounding values.

@CollectionDomain

You use the @CollectionDomain to specify the cardinality and the type of collection elements.

@PatternDomain

You use the @PatternDomain annotation to define a pattern domain on a member or parameter.

@CustomProperty

You use the @CustomProperty annotation to set a property to a class, a member, or a parameter.

@CustomProperties

You use the @CustomProperties annotation to set several properties to a class, a member, or a parameter.

@java.beans.ConstructorProperties

You can use the @java.beans.ConstructorProperties annotation to provide a name for a constructor.

Parent topic: [Designing a BOM for a Java model](#)

XOM annotations

You can use annotations in your Java™ source code. Annotations are a type of metadata that you can add to classes, members and parameters.

You can add annotations to the execution object model (XOM) to customize the way that the business object model (BOM) is created from Java classes. You use the annotations to apply changes to classes, members, or parameters in the BOM.

For example, you can add an annotation to a class to define a business name for the class. In the BOM, the class takes the name defined by the annotation.

When you create the BOM from the XOM, the BOM Verbalization page of the New BOM Entry wizard displays the business name of the class.

Important:

To use the annotations from the [ilog.rules.bom.annotations](#) package, you must add `<InstallDir>/studio/lib/jrules-engine.jar` to the classpath of your XOM. However, if you just want to use the `@java.beans.ConstructorProperties` annotation, you do not need to add the `jrules-engine.jar` to the classpath.

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[Mapping of a BOM created from a Java XOM](#)

@NotBusiness

You use the @NotBusiness annotation to filter out classes and members that you do not want to import.

Purpose

Filters out classes and members that you do not want to import in the BOM.

Syntax

```
@NotBusiness
```

Description

The classes and members with the @NotBusiness annotation are not imported in the BOM. For example, if you encounter problems when loading a class, you can filter out the member that uses this class by setting the @NotBusiness annotation to this member.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    public Customer(String name) {...}  
    public String getName() {...}  
  
    @NotBusiness  
    public Object readResolve() throws ObjectStreamException {...}  
}
```

The result in the BOM is the following code:

```
class Customer {  
    Customer(string arg);  
    readonly string name;  
}
```

Parent topic: [Adding annotations to the XOM](#)

@BusinessName

You use the @BusinessName annotation to provide a name for a class, a member, or a parameter in the BOM.

Purpose

Provides a name for a class, a member, or a parameter in the BOM.

Syntax

```
@BusinessName(<a string>)
```

Description

In Java™, the parameter names are not stored in the class. You can set the @BusinessName annotation to give a business name to a parameter. The parameter names are stored in the BOM, and they are used in the BOM to XOM mapping and in the testing and simulation constructor for testing.

A discrepancy between a BOM name and a XOM name can cause an error. To resolve the error, you must edit the BOM to XOM mapping.

For information on annotation for changing the name of a parameter in the BOM, see [Annotation Type BusinessName](#).

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    public Customer(@BusinessName("name") String name) {...}  
    public String getName() {...}  
}
```

The result in the BOM is the following code:

```
class Customer {  
    Customer(string name);  
    readonly string name;  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related reference:

[@java.beans.ConstructorProperties](#)

@BusinessType

You use the @BusinessType annotation to change the type of a member or a parameter in the BOM.

Purpose

Changes the type of a member or a parameter in the BOM.

Syntax

```
@BusinessType(<a string>)
```

Description

The @BusinessType annotation changes the type of a member or a parameter in the BOM. For example, if you want to change a type int to an enumeration in the BOM to verbalize it. An enumeration is a class with a domain set as an enumeration of static references.

When you apply the @BusinessType annotation to a method, the return type of the method is modified.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    public Customer(@BusinessType("Category") int category) {...}  
}
```

The result in the BOM is the following code:

```
class Customer {  
    Customer(Category category);  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[Domains](#)

@BoundedIntDomain

You use the @BoundedIntDomain to specify an interval between two bounding values.

Purpose

Specifies an interval between two bounding values on a member or a parameter of type int.

Syntax

```
@BoundedIntDomain(min = <an int>, max = <an int>)
```

Description

The @BoundedIntDomain provides a bounded domain to specify an interval between two bounding values on a member or a parameter of type int.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    @BoundedIntDomain(min = 0, max = 120)  
    public int age;  
}
```

The result in the BOM is the following code:

```
class Customer {  
    int age domain [0,120];  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[Domains](#)

@CollectionDomain

You use the @CollectionDomain to specify the cardinality and the type of collection elements.

Purpose

Specifies the cardinality and the type of collection elements on a member or a parameter where a collection type is used.

Syntax

```
@CollectionDomain(<see the example>)
```

Description

The @CollectionDomain provides a collection domain to specify the type of collection elements and the cardinality.

The collection domain and the element type are used by the Business Action Language (BAL).

Example

In the XOM, the annotation can be used as follows:

```
public class Cart {  
  @CollectionDomain(elementType = "Item")  
  public List items;  
  
  @CollectionDomain(min = 1, max = 12)  
  public Person[] passengers;  
}
```

The result in the BOM is the following code:

```
class Cart{  
  List items domain 0,* class Item;  
  Person[] passengers domain 1,12 class Person;  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[Domains](#)

@PatternDomain

You use the @PatternDomain annotation to define a pattern domain on a member or parameter.

Purpose

Defines a pattern domain for a member or a parameter.

Syntax

```
@PatternDomain(<a string>)
```

Description

The @PatternDomain annotation enables you to specify a pattern domain for a member or a parameter.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    @PatternDomain("[A-Za-z]")  
    public String getName() {...}  
}
```

The result in the BOM is the following code:

```
class Customer {  
    readonly string name domain "[A-Za-z]";  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:
[Domains](#)

@CustomProperty

You use the @CustomProperty annotation to set a property to a class, a member, or a parameter.

Purpose

Sets a property on a class, a member, or a parameter.

Syntax

```
@CustomProperty(name=<a string>, value=<a string>)
```

Description

The @CustomProperty annotation sets a property on a class, a member, or a parameter.

You can use any BOM property.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
  
    @CustomProperty(name = "dataio.default", value = "true")  
    public Customer(@BusinessName("name") String name);  
    public String getName() {...}  
}
```

The result in the BOM is the following code:

```
class Customer {  
    Customer(string name)  
        property dataio.default "true";  
    readonly string name;  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[BOM properties](#)

@CustomProperties

You use the @CustomProperties annotation to set several properties to a class, a member, or a parameter.

Purpose

Sets several properties on a class, a member, or a parameter.

Syntax

```
@CustomProperties(names={<strings>},values={<strings>})
```

Description

The @CustomProperties annotation sets several properties on a class, a member, or a parameter.

You can use any BOM property.

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
    @CustomProperties(names = {"java_length", "other" }, values = { "9",
"true"})
    public String getName() {...}
}
```

The result in the BOM is the following code:

```
class Customer {
    Customer(string name)
        property java_length "9"
        property other "true";
    readonly string name;
}
```

Parent topic: [Adding annotations to the XOM](#)

Related concepts:

[BOM properties](#)

@java.beans.ConstructorProperties

You can use the `@java.beans.ConstructorProperties` annotation to provide a name for a constructor.

Purpose

Provides a name for a constructor.

Syntax

```
@java.beans.ConstructorProperties({<strings>})
```

Description

You can use the `@java.beans.ConstructorProperties` annotation to provide a name for a constructor. You can also use the `@BusinessName` on the parameters to give a name to each argument in the BOM.

For more information about the `@java.beans.ConstructorProperties` annotation, see [ConstructorProperties](#).

Example

In the XOM, the annotation can be used as follows:

```
public class Customer {  
    @ConstructorProperties("name", "age")  
    public Customer(String name, int age);  
}
```

The result in the BOM is the following code:

```
class Customer {  
    Customer(string name, int age);  
}
```

Parent topic: [Adding annotations to the XOM](#)

Related reference:

[@BusinessName](#)

Designing a BOM for an XML model

You can use data other than Java™ as the basis for your XOM.

Overview: Dynamic execution object model (XOM)

The dynamic execution object model (XOM) can be generated from native Java classes or from XML data. The rule engine accesses XML directly.

Defining a dynamic XOM for a rule project

Define the dynamic XOM when creating a rule project, or later in the properties of the rule project.

Mapping between XML schema and dynamic classes

XML binding maps XML schema types to XOM dynamic classes and objects. It supports the map function, schema types defined from other types, and XML declarations, subject to limitations.

Parent topic: [Designing business object models](#)

Overview: Dynamic execution object model (XOM)

The dynamic execution object model (XOM) can be generated from native Java™ classes or from XML data. The rule engine accesses XML directly.

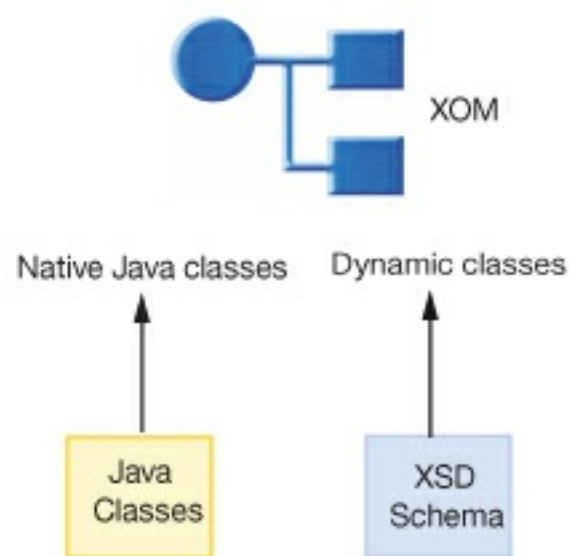
The execution object model (XOM) is an object model against which rules are executed. Business rules are written against the Business Object Model (BOM), and then translated into the ILOG® Rule Language (IRL) and run against the XOM. The rule engine evaluates the rules against the application objects and executes them when appropriate.

In Decision Server, a XOM can be generated from native Java classes or from dynamic classes, or derived from XML schema definitions (XSDs). The word *dynamic* means that no native Java object is constructed or generated. Instead, dynamic XML objects are created to represent the instances of the dynamic classes.

- A XOM generated from native Java classes is known as a **Java XOM**.
- A XOM generated from dynamic classes is known as a **Dynamic XOM**.

When the rule engine looks for XOM classes, it looks first in the dynamic XOM, then in the Java XOM.

The following diagram shows the mapping between a XOM and Java classes and dynamic classes.



Parent topic: [Designing a BOM for an XML model](#)

Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

Related information:

[Mapping between XML schema and dynamic classes](#)

Defining a dynamic XOM for a rule project

Define the dynamic XOM when creating a rule project, or later in the properties of the rule project.

About this task

You can define a dynamic XOM either in the New Rule Project wizard when you create the project, or later, using the rule project Properties dialog. The procedure is the same in each case.

Procedure

To define a dynamic XOM using the Properties dialog:

1. In the Rule Explorer view, select the rule project and then on the **Project** menu click **Properties**.
2. In the pane of the Properties dialog, click **Dynamic Execution Object Model**.
3. In the Dynamic Execution Object Model wizard, on the Dynamic Bindings tab, select **Add XSD** or **Add External XSD**:
 - To add XSDs that have been imported into the Eclipse workspace, click **Add XSD**. Select each of the required XSD files from the **XSD Files** dialog and then click **OK**.
 - To add XSDs that are held in your file system, click **Add External XSD**. Select each of the required XSD files from the **External XSD Files** dialog and then click **Open**.

For each schema namespace, a package name is defined in which the XOM classes are stored. You can expand each list entry to see the XML namespaces and the package names.

At the bottom of the Dynamic Bindings tab, a hidden section lists the XSD files that are included in the ruleset archive. To see this list, click the black arrow next to **Schemas to be included in Ruleset archive for enabling the Transparent Decision Service**.

This list of files might be important when you use the hosted transparent decision service.

4. The default package name is derived from the namespace. If you want to rename a package:
 - a. Select the package and then click **Edit**.
 - b. Type the new package name in the **Package Name Configuration** dialog and then click **OK**.
5. If you want to change the order in which the selected dynamic binding files are listed, click the **Order** tab and then use the up or down arrows to move the required files.

You might want to change the order of the files if the same element exists in more than one file and you want to dictate which schema or file to use first to identify it.

6. When you have selected all your dynamic binding files, click **Apply and Close** to close the **Dynamic Execution Object Model** dialog.

The dynamic XOM is now defined. The new packages are shown in the BOM Entry and behave like any normal package.

7. To remove dynamic binding files from the XOM, select each of the files you want to remove and then click **Remove**.

Parent topic: [Designing a BOM for an XML model](#)

Related concepts:
[Transparent decision services](#)

Mapping between XML schema and dynamic classes

XML binding maps XML schema types to XOM dynamic classes and objects. It supports the map function, schema types defined from other types, and XML declarations, subject to limitations.

Map function

The map function is used to design a dynamic type when its related schema type is known.

Schema types

You use schemas to define types and refine your definition by deriving previously defined types.

XML declarations

XML elements, attributes, groups, and appinfo structures map to dynamic classes.

Schema-related markup in XML documents

Schema-related markup in XML documents includes `xsi:nil`, `xsi:type`, `xsi:schemaLocation`, and `xsi:noNamespaceSchemaLocation`.

Schema mapping limitations

Some features or keywords are not mapped by the current XML binding and the value types of some schema types are lost.

Parent topic: [Designing a BOM for an XML model](#)

Map function

The map function is used to design a dynamic type when its related schema type is known.

The map function is used to design a dynamic type when its related schema type is known.

- If T is a *complex* type, $map(T)$ represents the dynamic class related to T .
- If T is an *atomic* simple type, $map(T)$ represents its mapped dynamic type by following these rules:
 1. If T is a built-in type, the dynamic type is given by XXX.
 2. $map(T) = map(base(T))$ where $base(T)$ is the direct base simple type of T .

Example of a map function

With a simple type MyT defined as:

```
<simpleType name="MyT">
  <restriction base="string">
</simpleType>
```

You have, $map(MyT) = java.lang.String$.

The following expression:

```
class C
{
  [map(MyT)] myField;
}
```

Stands for:

```
class C
{
  java.lang.String myField;
}
```

Parent topic: [Mapping between XML schema and dynamic classes](#)

Related information:

[Schema types](#)

Schema types

You use schemas to define types and refine your definition by deriving previously defined types.

Built-in simple types

Built-in schema types map to XOM dynamic class types.

Simple type mapping

Simple types map schemas to dynamic models.

Simple types derived from other simple types

You can derive simple types from other simple types and incorporate restrictions to the more general simple type using facets.

List types and union types

You can use list and union types in a schema and XML document.

Local simple types mapped to inner classes

Embedded local simple types map to inner classes.

Complex types

You can use mixed content in a schema and an XML document.

Extension of simple type content in complex types

You can extend simple type content in a schema and an XML document.

Complex type restriction

The XOM takes complex type restrictions into account in the XOM.

Complex type extension

The XOM takes complex type extensions into account.

Local complex types mapped to inner classes

Local complex types map to inner classes.

Default constructor dynamic methods

To map default constructors, a method is generated for each dynamic class.

Type identifier mapping

Simple type names map to XOM class names according to a number of rules.

Parent topic: [Mapping between XML schema and dynamic classes](#)

Built-in simple types

Built-in schema types map to XOM dynamic class types.

The following table summarizes how built-in schema types are mapped to dynamic class types.

Table 1. Built-in type mapping between XML schemas and XOM dynamic classes

XML schema type	XOM dynamic class type
boolean	boolean
decimal	java.math.BigDecimal
double	double
integer	java.math.BigInteger
int	int
ID	java.lang.String
nonPositiveInteger	java.math.BigInteger
negativeInteger	java.math.BigInteger
nonNegativeInteger	java.math.BigInteger
positiveInteger	java.math.BigInteger
unsignedLong	java.math.BigInteger
unsignedInt	long
unsignedShort	int
unsignedByte	short
normalizedString	java.lang.String
token	java.lang.String
language	java.lang.String
name	java.lang.String
ENTITY	java.lang.String
IDREFS	java.util.Vector
ENTITIES	java.util.Vector
float	float
duration	ilog.rules.xml.types.IlrDuration
dateTime	ilog.rules.xml.types.IlrDateTime
time	ilog.rules.xml.types.IlrTime
gYearMonth	ilog.rules.xml.types.IlrGYearMonth
gMonthDay	ilog.rules.xml.types.IlrGMonthDay
gDay	ilog.rules.xml.types.IlrGDay
gMonth	ilog.rules.xml.types.IlrGMonth
date	ilog.rules.xml.types.IlrDate
gYear	ilog.rules.xml.types.IlrGYear
NMTOKENS	java.util.Vector
anyURI	java.lang.String
QNAME	java.lang.String
NOTATION	java.lang.String
hexBinary	byte[]
base64Binary	byte[]
IDREF	java.lang.String
string	java.lang.String
NCNAME	java.lang.String
NMTOKEN	java.lang.String
long	long
short	short

Note:

IDRef is mapped to a String. Management of reference and ID is currently not available.

Parent topic: [Schema types](#)

Related information:

- [Complex types](#)
- [Local simple types mapped to inner classes](#)
- [Simple type mapping](#)

Simple type mapping

Simple types map schemas to dynamic models.

The following table shows some simple type mappings between a schema and a dynamic model.

Table 1. Standard simple type mapping

Case	Schema	Dynamic model
Global simple type derived by restriction	<pre><simpleType name="st1" > <restriction base="st0"> ... </restriction> </simpleType> <element name="e1" type="st1"></pre>	<pre>type s1 extends st0 { // [map(st0)] javatype; [map(st0)] getMinInclusive(); [map(st0)] getMaxInclusive(); [map(st0)] getMinExclusive(); [map(st0)] getMaxExclusive(); String getPattern(); int getLength(); int getMaxLength(); int getMinLength(); [map(st0)][] getEnumerations(); int getTotalDigits(); int getFractionDigits() } [map(st0)] e1;</pre>
Local simple type derived by restriction	<pre><complexType name="ct1"> <sequence> <element name="e1" > <simpleType name="st1" > <restriction base="st0"> ... </restriction> </simpleType> </element> </sequence> </complexType></pre>	<pre>class Ct1 { type s1 extends st0 { [map(st0)] javatype; } [map(st0)] e1; }</pre>
Simple type derived by list	<pre><simpleType name="st1" > <list itemType="st0"/> </simpleType> <element name="e1" type="st1"></pre>	<pre>type s1 { java.util.Vector javaType; } java.util.Vector e1;</pre>

Simple type derived by union	<pre><simpleType name="stUnion" > <union memberTypes="st0 st1"/> </simpleType> <element name="e1" type="st1"></pre>	<pre>type stUnion { java.lang.Object javaType; } java.util.Object e1;</pre>

Parent topic: [Schema types](#)

Related information:

[Built-in simple types](#)

[Simple types derived from other simple types](#)

[Map function](#)

[Complex types](#)

Simple types derived from other simple types

You can derive simple types from other simple types and incorporate restrictions to the more general simple type using facets.

The decision to derive a simple type from another simple type implies incorporating restrictions. To incorporate such restrictions, you use facets. Facets are sets of predefined constraints that help define new simple types by restricted built-in types.

Example of a facet: a zip code

An example of a facet is a pattern where a zip code is defined as containing only five digits.

The following schema shows a simple type definition:

```
<simpleType name="myType">
  <restriction base="string">
    <pattern value="*A"/>
  </restriction>
</simpleType>
<attribute name = "typeValue" type = "myType"/>
```

In this example, the simple type `myType` is defined as a subtype of `string`. The `typeValue` attribute is mapped to a dynamic class field of type `java.lang.String`.

When simple type definitions are mapped to dynamic class types, the type hierarchy tree is searched. Starting from the most restrictive type, the hierarchy tree is searched upwards until a built-in type is found that maintains the mapping type as the simple type definition mapping.

The following zip code example shows how user schema simple types are represented in the XOM and which methods are generated to manipulate them. A set of static methods are declared. They can be called from a rule. Two rules named `checkFloatInInterval` and `findZipCodePattern` are used to test the `ZipCode` objects.

Here is the schema:

```
<simpleType name="zip-code">
  <restriction base="string">
    <pattern value="A[0-9]{5}"/>
  </restriction>
</simpleType>
```

```
<simpleType name="Interval">
  <restriction base="float">
    <minInclusive value="1.1"/>
    <maxInclusive value="3.1"/>
  </restriction>
</simpleType>
```

Here is the XOM representation:

```
class ZipCode extends IlrXmlObject
{
    static String getPattern();
    static int getMinLength();
    static int getMaxLength();
    static getLength();
    static String[] getEnumerations();
}
```

```
class Interval
{
    static String getPattern();
    static int getMinLength();
    static int getMaxLength();
    static getLength();
    static float[] getEnumerations();
    static float getMinInclusive ();
}
```

```
static float getMaxInclusive ();
static float getMinExclusive ();
static float getMaxExclusive ();
static int getTotalDigits();
static int getFractionDigits();
}
```

And here are the rules:

```
rule checkFloatInInterval
{
  when {
    f: Float ( i: floatValue(); Interval.getMinInclusive() <= i;
              i <= Internal.getMaxInclusive() );
  }
  then {
    out.println ( f + " in Interval" );
  }
}
```

```
rule findZipCodePattern
{
  when {
    s: String ( equals ZipCode.getPattern() );
  }
  then {
    out.println ( s + " equals zip code pattern" );
  }
}
```

The simple type dynamic method provides information on type facets values.

The fact that the method returns the last facet definition in the type hierarchy has the following consequences:

- If a facet is defined twice in the type hierarchy, the nearest definition is returned.
- If a facet is not defined, a special value (usually null) is returned depending on the facet type.

Facets can be of the following types:

- Range facets: `getMinInclusive`, `getMaxInclusive`, `getMinExclusive`, `getMaxExclusive`
- Lexical space facets: `getPattern`, `getMinLength`, `getMaxLength`, `getLength`
- Enumeration facets: `getEnumerations`
- Numeric facets: `getTotalDigits`, `getFractionDigits`

The following table shows the returned values when the facet is not defined.

Table 1. Returned values with undefined facet

Types	getMinInclusive getMinExclusive	GetMaxInclusive getMaxExclusive	getMinLength getMaxLength getLength getTotalDigits getFractionDigits	getEnumerations
float	- Float.MAX_VALUE	Float.MAX_VALUE	IlrXmlObject.UNKNOWN_POSITIVE_VALUE	float[0]
double	- Double.MAX_VALUE	Double.MAX_VALUE		double[0]
byte	Byte.MIN_VALUE	Byte.MAX_VALUE		byte[0]
int	Integer.MIN	Integer.MAX		int[0]

	VALUE	VALUE	
long	Long.MIN_VALUE	Long.MAX_VALUE	long[0]
Object	null	null	Object[0]

Parent topic: [Schema types](#)

Related information:

[Built-in simple types](#)

[Simple type mapping](#)

List types and union types

You can use list and union types in a schema and XML document.

This example demonstrates the use of list types and union types in a schema and an XML document. The rules `displayExtendedIntMeasure` and `displayStandardInt` are used to test the `MeasureSet` objects.

Here is the schema:

```
<simpleType ExtendedInt>
  <union memberTypes="int string"/>
</simpleType>
```

```
<simpleType IntMeasures>
  <list itemType="ExtendedInt"/>
</simpleType>
```

```
<element name="measureSet" >
  <complexType>
    <sequence>
      <element name="intMeasures" type="IntMeasures" />
      <element name="extendedInt" type="ExtendedInt" />
    </sequence>
  </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<measureSet>
  <intMeasures>1 2 3 4 ten 5 unknown</intMeasures>
  <extendedInt>3</extendedInt>
</measureSet>
```

Here is the XOM representation:

```
class MeasureSet extends IlrXmlObject
{
  Vector intMeasures; // list of (Integer| String) instances
  Object extendedInt; // either Integer or String instances
  ...
}
```

And here are the rules:

```
rule displayExtendedIntMeasure
{
  when {
    ms: MeasureSet ();
    m: String () in ms.intMeasures;
  }
  then {
    out.println ( m + " is an extended int.");
  }
}
```

```
rule displayStandardInt
{
  when {
    ms: MeasureSet ();
    m: Integer () from ms.extendedInt;
  }
  then {
    out.println ( m + " is a standard int.");
  }
}
```

```
}
```

Parent topic: [Schema types](#)

Related information:

[Built-in simple types](#)

[Complex types](#)

Local simple types mapped to inner classes

Embedded local simple types map to inner classes.

Note:

The inner class policy is activated by default.

Mapping embedded local simple types to inner classes

This example shows how to map embedded local simple types to inner classes.

The example includes an element named `ident` that contains a simple type restriction. The simple type is represented in the XOM by the dynamic class `BorrowedBook.Ident`.

Here is the schema:

```
<complexType name="borrowed-book">
  <sequence>
    <element name="ident"/>
    <simpleType>
      <restriction base="string">
        <pattern value="A[0-9]*"/>
      </restriction>
    </simpleType>
  </sequence>
</complexType>
```

Here is the excerpt from an XML document:

```
<borrowed-book>
  <ident>12345</ident>
</borrowed-book>
```

Here is the XOM representation:

```
class BorrowedBook extends IlrXmlObject
{
  class Ident
  {
    ...
  };
}
```

And here is the rule:

```
rule identifyBorrowedBook
{
  when {
    b: BorrowedBook ( );
    BorrowedBook.Ident equals "12345";
  }
  then {
    ...
  }
}
```

Parent topic: [Schema types](#)

Related information:

[Built-in simple types](#)

[Simple type mapping](#)

[Map function](#)

Complex types

You can use mixed content in a schema and an XML document.

Mixed content in a schema and XML document

This example demonstrates the use of mixed content in a schema and an XML document. The schema contains an element of type `int` and an element of type `string`.

Here is the schema:

```
<element name="observation">
  <complexType mixed="true">
    <sequence>
      <element name="temperature" type="int">
      <element name="city" type="string">
    </sequence>
    ...
  </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<observation>
It is <temperature>10</temperature>degrees in <city>London</city>.
</observation>
```

And here is the XOM representation:

```
class Observation extends IlrXmlObject
{
  int temperature;
  String city;
  String content;
}
```

The content of the observation element is translated to a single `String` that concatenates all the strings and put in a special attribute named `content`.

Complex type mapping

The following table shows some complex type mappings between a schema and dynamic model.

Table 1. Standard complex type mapping

Case	Schema	Dynamic model
Element with local complex type of unary component	<pre><element name="e1"> <complexType> <sequence> <element name="e2" type="t2"/> <element name="e3" type="t3"/> </sequence> <attribute name="a1" type="t4"/> </complexType> </element></pre>	<pre>class E1 extends IlrXmlObject { [map(t2)] e2; [map(t3)] e3; [map(t4)] a1; } E1 e1;</pre>
Element with local complex type of unary component	<pre><element name="e1"> <complexType> <sequence></pre>	<pre>class E1 extends IlrXmlObject {</pre>

	<pre><element name="e2" type="t2" maxOccurs="unbounded"/> <element name="e3" type="t3" minOccurs="2" /> </sequence> </complexType> </element></pre>	<pre>// collection of [map(t2)] Vector e2List; // collection of [map(t3)] Vector e3List; } E1 e1;</pre>
Typed element	<pre><complexType name="t1"> <sequence> <element name="e2" type="t2"/> <element name="e3" type="t3"/> </sequence> </complexType> <element name="e1" type="t1" /></pre>	<pre>class T1 extends IlrXmlObject { [map(t2)] e2; [map(t3)] e3; } T1 e1;</pre>
Typed element with collection group	<pre><complexType name="t1"> <sequence maxOccurs="1001"> <element name="e2" type="t2"/> <element name="e3" type="t3"/> </sequence> </complexType> <element name="e1" type="t1" /></pre>	<pre>class T1 extends IlrXmlObject { // collection of [map(t2)] Vector e2List; // collection of [map(t3)] Vector e3List; } T1 e1;</pre>
Local complex type: when inner classes are permitted	<pre><complexType name="t1"> <sequence> <element name="e1"> <complexType> ... </complexType> </complexType> </complexType> </element></pre>	<pre>class T1 extends IlrXmlObject { class E1 extends IlrXmlObject { ... } E1 e1; }</pre>
Local complex type: when inner classes are not permitted	<pre><complexType name="t1"> <sequence> <element</pre>	<pre>Class E1 extends IlrXmlObject { }</pre>

	<pre> name="e1"> <complexType> ... </complexType> </complexType> </element> </pre>	<pre> class T1 extends IlrXmlObject { E1 e1; } </pre>
--	---	---

Simple content mapping

The following table shows some simple content mappings between a schema and dynamic model.

Table 2. Simple content mapping

Case	Schema	Dynamic model
Complex type with simple content	<pre> <simpleType name="st0" > ... </simpleType> <complexType name="ct0"> <simpleContent> <extension base="st0"> <attribute name="a1" type="st1" /> </extension> <simpleContent> </complexType> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st0)] content; [map(st1)] a1; } </pre>
Complex type inheriting by extension from complex type with simple content	<pre> <complexType name="ct0" > <simpleContent> ... <simpleContent> </complexType> <complexType name="ct1"> <simpleContent> <extension base="ct0"> <attribute name="a1" type="st1" /> </extension> </simpleContent> </complexType> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st0)] content; ... } class Ct1 extends Ct0 { [map(st0)] content; [map(st1)] a1; ... } </pre>
Complex type inheriting by restriction from complex type with simple content	<pre> <complexType name="ct0" > <simpleContent> ... <simpleContent> </complexType> <complexType name="ct1"> <simpleContent> <restriction base="ct0"> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st0)] content; ... } class Ct1 extends Ct0 { </pre>

	<pre> <attribute name="a1" type="st1" /> </restriction> <simpleContent> </complexType> </pre>	
--	---	--

Complex content mapping

The following table shows some complex content mappings between a schema and dynamic model.

Table 3. Complex content mapping

Case	Schema	Dynamic model
Complex type inheriting by extension	<pre> <complexType name="ct0"> <attribute name="a1" type="st1" /> </complexType> <complexType name="ct0"> <complexContent> <extension base="ct0"> <attribute name="a2" type="st2" /> </extension> <complexContent> </complexType> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st1)] a1; } class Ct1 extends Ct0 { [map(st2)] a2; } </pre>
Complex type inheriting by restriction	<pre> <complexType name="ct0"> <attribute name="a1" type="st1" /> </complexType> <complexType name="ct0"> <complexContent> <restriction base="ct0"> <attribute name="a1" type="st2" /> </restriction> <complexContent> </complexType> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st1)] a1; } class Ct1 extends Ct0 { } </pre>
Complex type with mixed content, inheriting by extension	<pre> <complexType name="ct0" > <simpleContent> ... </simpleContent> </complexType> <complexType name="ct0"> <complexContent mixed="true"> </pre>	<pre> class Ct0 extends IlrXmlObject { [map(st0)] content; } class Ct1 extends Ct0 { [map(st1)] a1; } </pre>

	<pre><extension base="st0"> <attribute name="a1" type="st1" /> </extension> <simpleContent> </complexType></pre>	<pre>}</pre>
--	--	--------------

Parent topic: [Schema types](#)

Related information:
[Complex type extension](#)
[Complex type restriction](#)

Extension of simple type content in complex types

You can extend simple type content in a schema and an XML document.

The example extends temperature of base type `int` to include a `unit` attribute of type `string`.

Here is the schema:

```
<complexType temperature>
  <simpleContent>
    <extension base="int">
      <attribute name="unit" type="string"/>
    </extension>
  </simpleContent>
</complexType>
```

Here is the excerpt from an XML document:

```
<temperature unit= "celsius">15</temperature>
```

Here is the XOM representation:

```
Class Temperature extends IlrXmlObject
{
  int content ;
  String unit ;
  ...
}
```

Parent topic: [Schema types](#)

Related information:

[Complex types](#)

[Built-in simple types](#)

Complex type restriction

The XOM takes complex type restrictions into account in the XOM.

In the following example, the complex type `book` represents a standard book and the complex type `old-book` is a restriction because a lexical pattern constraint (`pattern value = "[0-9]{2} - [A-Z]{5} - [0-9]"`) has been added. The example includes a rule named `processOldBook` that matches “Hamlet” with the title of an `OldBook` object.

Note:

The `OldBook` dynamic class does not redefine nor restrict the `Book` class attributes, even if the `ident` field is more restricted.

Here is the schema:

```
<complexType name= "book">
  <sequence>
    <element name="ident" type="string"/>
    <element name="title" type="string" />
  </sequence>
</complexType>

<complexType name="old-book">
  <complexContent>
    <restriction base="book">
      <sequence>
        <element name="ident">
          <simpleType>
            <restriction base="string">
              <pattern value="[0-9]{2} - [A-Z]{5} - [0-9]">
            </restriction>
          </simpleType>
        </element>
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
    ...
    String ident;
    String title;
    ...
}
```

```
class OldBook extends Book
{
    ...
}
```

And here is the rule:

```
rule processOldBook
{
  when {
    OldBook ( title equals "Hamlet" );
  }
  then {
    ...
  }
}
```

Parent topic: [Schema types](#)

Related information:
[Complex type extension](#)
[Complex types](#)

Complex type extension

The XOM takes complex type extensions into account.

Extending a complex type by adding an attribute

In this example, the complex type book has been extended to computer-book by adding a new attribute named language. The example includes a rule named processComputerBook that matches “Java” with the title of a ComputerBook object.

Here is the schema:

```
<complexType name= "book" >
  <sequence>
    <element name="ident" type="string"/>
    <element name="title" type="string" />
  </sequence>
</complexType>
```

```
<complexType name="computer-book">
  <complexContent>
    <extension base="book">
      <sequence>
        <element name="language" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Here is an excerpt from an XML document:

```
<computer-book>
  <ident>12345</ident>
  <title>Decision Server Reference Manual</title>
  <language>Java IRL</language>
</computer-book>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
    ...
    String ident;
    String title;
    ...
}
```

```
class ComputerBook extends Book
{
    String language;
    ...
}
```

And here is the rule:

```
rule processComputerBook
{
    when {
        ComputerBook ( language equals "Java" );
    }
    then {
        ...
    }
}
```


Parent topic: [Schema types](#)

Related information:

[Complex type restriction](#)

[Complex types](#)

[Extension of simple type content in complex types](#)

Local complex types mapped to inner classes

Local complex types map to inner classes.

Note:

The inner class policy is activated by default.

This example shows how to map embedded local complex types onto inner classes.

The example includes a rule named `processBorrowedBook` that matches “Smith” with the attribute `surname` of an instance of the inner class `borrower`.

Here is the schema:

```
<complexType name="borrowed-book">
  <sequence>
    <element name="ident" />
    <element name="borrower">
      <complexType>
        <sequence>
          <element name="name">
            <element name="surname">
              </sequence>
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>
```

Here is the excerpt from an XML document:

```
<borrowed-book>
  <ident>12345</ident>
  <borrower>
    <name>John</name>
    <surname>Smith</surname>
  </borrower>
</borrowed-book>
```

Here is the XOM representation:

```
class BorrowedBook extends IlrXmlObject
{
  class Borrower extends IlrXmlObject
  {
    String name;
    String surname;
  };
  String ident;
  Borrower borrower;
}
```

And here is the rule:

```
rule processBorrowedBook
{
  when {
    b: BorrowedBook ( );
    BorrowedBook.Borrower ( surname equals "Smith" ) from b.borrower;
  }
  then {
    ...
  }
}
```

Parent topic: [Schema types](#)

Related information:
[Complex types](#)
[Complex type extension](#)
[Map function](#)

Default constructor dynamic methods

To map default constructors, a method is generated for each dynamic class.

Mapping of default constructors

This example shows how to map a default constructor. A method is generated for each dynamic class. This method sets the attributes with their default values, if they exist. The following example includes a function that adds an instance of a dynamic class Book, with the default attribute title equal to None.

Here is the schema:

```
<complexType name="book">
  <sequence>
    <element name="ident"/>
    <element name="title" minOccurs="0" default="None" />
  </sequence>
</complexType>
```

Here is the XOM representation:

```
Class Book extends IlrXmlObject
{
  String ident ;
  String title ;
  Book (); // default constructor
}
```

And here is a function in IRL:

```
function void insertBook() {

  // Add a new book, default value: ( ident = null, title="None" )
  insert ( new Book() );
}
```

Parent topic: [Schema types](#)

Related concepts:
[Functions](#)

Related tasks:
[Creating functions](#)

Related reference:
[function](#)

Type identifier mapping

Simple type names map to XOM class names according to a number of rules.

Simple type name map to XOM class names according to the following rules:

- The first type letter is translated into lowercase.
- All characters that could be contained in a Java™ identifier are translated without change.
- The other characters are discarded and the next letter is translated to uppercase.
- If the resulting class name is already used as a class or simple type identifier, a numeric suffix is appended to the name. For example, name becomes name_0.

Following these rules, zip-code becomes zipCode.

Parent topic: [Schema types](#)

Related information:

[Simple type mapping](#)

XML declarations

XML elements, attributes, groups, and appinfo structures map to dynamic classes.

Groups

Groups can be choice groups, composite groups, or collection groups. They are mapped using complex types and specific dynamic methods.

appinfo

appinfo structures are mapped to XOM properties.

Translation of schema member names to dynamic class field names

Schema member names translate to dynamic class field names according to a number of rules.

Parent topic: [Mapping between XML schema and dynamic classes](#)

Groups

Groups can be choice groups, composite groups, or collection groups. They are mapped using complex types and specific dynamic methods.

Group mapping is presented in three subsections:

- [Choice groups](#)
- [Composite groups](#)
- [Collection groups](#)

In each sections, an example is used to demonstrate the various types of mapping. The examples include a schema, an XML document, and the XOM representation. The examples of choice group mapping and composite group mapping also include the rules to help clarify the explanation.

Choice groups

A choice group is an XSD group of elements. Only one of them is shown in the XML document related to the group schema.

Additional dynamic methods are generated to help handle the choice group. For example, the dynamic method `getChoice` returns the dynamic attribute value selected in a group, and the dynamic method `getChosenAttribute` returns the dynamic attribute name selected in a group. The `String` parameter helps to locate the group when there are multiple groups in a complex type. The selected group is identified by the XOM name of a field of one of its children. When the method is used with no argument, the first choice is selected by deep search.

Example of choice group mapping

The following example uses the complex type `observation` which contains three elements: `temperature`, `pressure`, and `velocity`.

Here is the schema:

```
<complexType name="observation">
  <choice>
    <element name="temperature" type="double"/>
    <element name="pressure" type="double"/>
    <element name="velocity" type="double"/>
  </choice>
</complexType>
```

Here is the excerpt from an XML document:

```
<observation>
  <pressure>10.0</pressure>
</observation>
```

Here is the XOM representation:

```
class Observation extends IlrXmlObject
{
  double temperature ;
  double pressure ;
  double velocity ;
  ...
  // choice dynamic methods
  Object getChoice ( String attrName ) ;
  String getChosenAttribute ( String attrName );
  Object getChoice();
  String getChosenAttribute();
}
```

And here are the rules:

```
rule findTemperatureObservation
{
  when {
```

```

        Observation ( temperature isknown );
    }
    then {
        ...
    }
}

```

```

rule processObservationValue
{
    when {
        obs: Observation ( );
        v: Double ( ) from obs.getChoice ( );
        attr: String ( ) from obs.getChosenAttribute ( );
    }
    then {
        ...
    }
}

```

Composite groups

A composite group is a combination of elements and groups.

Note:

The W3C specification does not define the concept of a composite complex type. This concept has been introduced here to describe a new mapping state and differentiate it from the sequence or choice groups.

Example of composite group mapping

The following example reuses the same complex type observation already used in [Choice groups](#). For the purpose of mapping a composite group, one more choice group has been added to the observation type. This additional choice group has two options: error and comment.

Here is the schema:

```

<complexType name="observation">
  <sequence>
    <choice>
      <element name="temperature" type="double"/>
      <element name="pressure" type="double"/>
      <element name="velocity" type="double"/>
    </choice>
    <choice>
      <element name="error" type="double"/>
      <element name="comment" type="string" />
    </choice>
  </sequence>
</complexType>

```

Here is the excerpt from an XML document:

```

<observation>
  <pressure>10</pressure>
  <error>0.2</error>
</observation>

```

Here is the XOM representation:

```

class Observation extends IlrXmlObject
{
    double temperature ;
    double pressure ;
    double velocity ;
    double error;
    String comment;
}

```

```

...
// choice dynamic methods
Object getChoice ( String attrName ) ;
String getChosenAttribute ( String attrName );
}

```

And here are the rules:

```

rule findTemperatureWithErrorObservation
{
    when {
        Observation ( temperature isknown; error isknown );
    }
    then {
        ...
    }
}

```

```

rule processObservationValue
{
    when {
        obs: Observation ( );
        v: Double ( ) from obs.getChoice ( "velocity" );
        attr: String ( ) from obs.getChosenAttribute ( "velocity" );
    }
    then {
        ...
    }
}
rule displayErrorOrComment
{
    when {
        obs: Observation ( );
        c: Object ( ) from obs.getChoice ( "error" );
        attr: String ( ) from obs.getChosenAttribute ( "error" );
    }
    then {
        out.println ( attr + '=' + c.toString() );
    }
}

```

Collection groups

The following example demonstrates how to map a collection of choice groups. This example reuses again the complex type observation already used in the [Choice groups](#) and [Composite groups](#) sections. The collection of the choice group is defined in the complex type with maxOccurs="unbounded".

Here is the schema:

```

<complexType name="observations">
  <choice maxOccurs="unbounded">
    <element name="temperature" type="double"/>
    <element name="pressure" type="double"/>
    <element name="velocity" type="double"/>
  </choice>
</complexType>

```

Here is the excerpt from an XML document:

```

<observations>
  <velocity>12.3</velocity>
  <temperature>11.5</temperature>
  <velocity>9.2</velocity>
</observations>

```

And here is the XOM representation:

```
class Observations extends IlrXmlObject
{
  Vector temperatureList;
  Vector pressureList;
  Vector velocityList;
  ...
}
```

Parent topic: [XML declarations](#)

appinfo

appinfo structures are mapped to XOM properties.

XML schema appinfo structures are automatically mapped to XOM properties when the name is equal to "http://www.ilog.com/rules/xml".

Example of appinfo mapping

The following example shows how a simple appinfo structure is translated in the XOM.

Here is the schema:

```
<appinfo xmlns:irl="http://www.ilog.com/rules/xml">
  <irl:property name="property1">property1Value</irl:property>
</appinfo>
```

Here is the XOM representation:

```
property property1 "property1Value"
```

appinfo as a class property

The following example demonstrates how to create class properties using appinfo structures. This example maps the complex type book, which has the bookRenderer and position properties, to a class named Book.

Example of mapping appinfo as a class property

Here is the schema:

```
<complexType name="book">
  <annotation>
    <appinfo xmlns:irl="http://www.ilog.com/rules/xml">
      <irl:property name="bookRenderer">Renderer1</irl:property>
      <irl:property name="position">
        <irl:property name="x">10</irl:property>
        <irl:property name="y">20</irl:property>
      </irl:property>
    </appinfo>
  </annotation>
  <sequence>
    <element name="ident"/>
    <element name="title"/>
  </sequence>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
  property bookRenderer "Renderer1"
  property position {
    x "10",
    y "20"
  }
  String ident;
  String title;
}
```

appinfo as a field property

The following example demonstrates how to create field properties using appinfo structures. This example maps the complex type book, which has the bookRenderer and position properties, to a field named ident.

Here is the schema:

```
<complexType name="book">
  <sequence>
```

```
<element name="ident">
  <annotation>
    <appinfo xmlns:irl="http://www.ilog.com/rules/xml">
      <irl:property name="bookRenderer">Renderer1</irl:property>
      <irl:property name="position">
        <irl:property name="x">10</irl:property>
        <irl:property name="y">20</irl:property>
      </irl:property>
    </appinfo>
  </annotation>
</element>
<element name="title"/>
</sequence>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
  String ident
    property bookRenderer "Renderer1"
    property position {
      x "10",
      y "20"
    };
  String title;
}
```

Parent topic: [XML declarations](#)

Translation of schema member names to dynamic class field names

Schema member names translate to dynamic class field names according to a number of rules.

A schema member name is translated into a dynamic class field name according to the following rules:

- The first type letter is translated into lowercase.
- All characters that could be contained in a Java™ identifier are translated without change.
- The other characters are discarded and the next letter is translated into uppercase.
- If the resulting member name is already used as a member identifier, a numeric suffix is appended to that name. For example, field becomes field_0.

Following these rules, my-address becomes myAddress.

Parent topic: [XML declarations](#)

Schema-related markup in XML documents

Schema-related markup in XML documents includes `xsi:nil`, `xsi:type`, `xsi:schemaLocation`, and `xsi:noNamespaceSchemaLocation`.

The schema-related markup in XML documents includes the following elements:

- [xsi:nil](#)
- [xsi:type](#)
- [xsi:schemaLocation and xsi:noNamespaceSchemaLocation](#)

xsi:nil

When an XML element is set to `xsi:nil` in the XML document instance, its related dynamic class field is set to null. The `isunknown` operator applied on this field returns `true`. In other words, when an XML object is serialized via the `driver.writeObject` method, the following information is added to this object:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Example of mapping xsi:nil in an XML document

The following example demonstrates the mapping of `xsi:nil` in an XML document. The rule `detectPersonWithMissingAddress` uses the `isunknown` keyword to find a person whose address is not known.

In this example:

- `xsi` is the namespace name
- `http://www.w3.org/2001/XMLSchema` instance is the URL
- `nil` is one element of the namespace

Here is the schema:

```
<element name="person">
  <complexType>
    <sequence>
      <element name="surname"/>
      <element name="address" nillable="true"/>
    </sequence>
  </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <surname>Smith</surname>
  <address xsi:nil="true"/>
</person>
```

And here is the rule:

```
rule detectPersonWithMissingAddress
{
  when {
    Person ( address isunknown );
  }
  then {
    ...
  }
}
```

XML namespaces are designed to provide universally unique names for elements and attributes. Developers can use them to carry out the following actions:

- Combine fragments from different documents without any naming conflicts.

- Write reusable code modules that can be called for specific elements and attributes. Universally unique names guarantee that such modules are called only for the correct elements and attributes.
- Define elements and attributes that can be reused in other schemas without any risk of name conflicts.

xsi:type

To refine the type of an element in the XML document, you can use xsi polymorphism. This feature substitutes a subtype for the standard type of the element.

Example of mapping xsi:type in an XML document

The following example demonstrates how to use xsi polymorphism. The example uses a schema and an excerpt from an XML document.

Here is the schema:

```
<element name="person" type="person"/>
<complexType name=" person " >
  <sequence>
    <element name="name">
    <element name="surname">
  </sequence>
</complexType>
```

```
<complexType name=" inhabitant ">
  <complexContent>
    <extension base="person">
      <sequence>
        <element name="address"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

And here is an excerpt from an XML document:

```
<person xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:type="inhabitant">
  <name>John</name>
  <surname>Smith</surname>
  <address>123 Downing Street, London</address>
</person>
```

xsi:schemaLocation and xsi:noNamespaceSchemaLocation

These functions are supported by the parser. However, the resulting information is not used in XML binding.

Parent topic: [Mapping between XML schema and dynamic classes](#)

Schema mapping limitations

Some features or keywords are not mapped by the current XML binding and the value types of some schema types are lost.

Unmapped schema features and keywords

Some schema features or keywords are not mapped by the current XML binding. These keywords are correctly parsed but they are not processed and not translated into XOM dynamic classes or XML objects:

- key, keyref, unique
- abstract, block
- blockDefault, finalDefault
- any, anyAttribute
- skip, lax, strict
- processContents

Lost value types

The following schema types are mapped without errors to `java.lang.Object` by the current XML binding but the original value type is lost at runtime:

- anySimpleType
- anyType

If you want to process such elements with their original value types (for example, by assignation), you need to cast them to your required value.

Parent topic: [Mapping between XML schema and dynamic classes](#)

Updating the BOM when the XOM changes

How you manage XOM changes in the BOM depends on whether your rule project references a Java™ project as a XOM.

Managing changes to the XOM from a Java project

If your rule project references a Java project as a XOM, you can update the BOM from the Java project.

Managing changes to the XOM from a rule project

If your rule project does not reference a Java Project as execution object model (XOM), you can update the BOM from the rule project.

Configuring BOM updates to ignore differences

You can configure the BOM update feature to selectively ignore certain differences between the BOM and the XOM.

Parent topic: [Designing business object models](#)

Managing changes to the XOM from a Java project

If your rule project references a Java™ project as a XOM, you can update the BOM from the Java project.

About this task

Update the BOM directly from the Java project. The rule project must reference a Java project as Execution Object Model (XOM).

When you rename an element in a Java XOM, the Rename Compilation Unit dialog displays the impact on the related BOM.

Procedure

Accept the changes in the Rename Compilation Unit dialog. The BOM and its associated vocabulary are automatically updated. Any reference to a XOM class in the BOM to XOM mapping section of a business element is also updated.

Parent topic: [Updating the BOM when the XOM changes](#)

Managing changes to the XOM from a rule project

If your rule project does not reference a Java™ Project as execution object model (XOM), you can update the BOM from the rule project.

About this task

Update the BOM from the rule project.

Procedure

To update the BOM:

1. In the Rule Explorer, in the bom folder, right-click the BOM entry you want to update and select **BOM Update**.
2. In the **BOM Update** view, do the suggested actions required to update your BOM.

Results

Changes to the XOM are taken into account in the BOM. Any elements you added to the BOM when extending it remain as they are. If they have a BOM to XOM mapping, the mapping is updated. Members that have been modified in the XOM, through a change of name or a change to their arguments, are added as new business elements in the BOM.

Parent topic: [Updating the BOM when the XOM changes](#)

Related information:

[Configuring BOM updates to ignore differences](#)

Configuring BOM updates to ignore differences

You can configure the BOM update feature to selectively ignore certain differences between the BOM and the XOM.

If you add a `.cfg` file with new properties for the BOM file being updated, you can filter the differences that are detected when running a BOM update. That is, the BOM update feature filters out the differences that you do not want to display. However, ignoring the differences between the BOM and the XOM does not necessarily prevent them from being modified. The BOM methods and attributes can still be updated in action. For example, if the attribute that you ignore is the only difference between the BOM and the XOM class, then the BOM update feature does not propose an action. However, if you change another attribute in the same BOM class, the BOM update action updates the entire class, including the attributes that you choose to ignore.

The `.cfg` file should be placed in the same directory as the `.bom` file to which it applies. The name of the file must be identical to the `.bom` file, but with a `.cfg` file extension.

Values are comma-separated globs. Each glob must be preceded by either a plus "+" sign (inclusion) or a minus "-" sign (exclusion). The order in which the globs are shown is important. Globs are converted to REGEX by replacing "*" and "?" with JRE REGEX equivalents, and "." and "\" are escaped. All other characters are not converted, and are shown in the REGEX as written. The REGEX is prefixed with a "^" and is appended with a "\$".

The following example provides an annotated `.cfg` file:

```
/** This property ignores the following missing BOM elements:
 * + the 'ignore' package and all sub-packages
 * - except for the ignore.exception package and all sub-packages
 * + the XOM class 'bank.Loan'
 * + all classes in the 'branch' package as well as all sub-packages
 * + the method 'bank.Customer.doSomething' taking a java.lang.String
 * + all classes with 'test' in their package name
 * + the attribute bank.Customer.age
 * + classes starting with the name 'Test' followed by any 3 characters
 */
update.ignoreMissingBomElement = +ignore.,\
-ignore.exception.*,\\
+bank.Loan,\\
+branch.*,\\
+bank.Customer.doSomething([Ljava.lang.String[]),\\
+*.test.*,\\
+bank.Customer.age,\\
+*.Test???
```

```
/** This property ignores the following missing XOM elements:
 * + the BOM class 'new.BomClass'
 */
update.ignoreMissingXomElement = +new.BomClass
```

```
/** This property ignores the following differences between BOM and XOM
classes:
 * + all attributes on 'bank.Customer'
 * - except for the bank.Customer.name attribute
 */
update.ignoreDifferences = +bank.Customer.,-bank.Customer.name
```

Parent topic: [Updating the BOM when the XOM changes](#)

Related tasks:

[Managing changes to the XOM from a rule project](#)

Configuring the BOM for rule authoring

After designing a BOM for an underlying object model, you configure it to maximize the efficiency of rule authoring activities.

Overview: A data model for rule authoring

You use a vocabulary to write rules. You define the vocabulary in the business object model (BOM).

Business object model (BOM)

The BOM is the object model for business rules. You edit BOM members in the BOM editor.

Creating BOM elements

You can extend the business object model (BOM) with new business elements, and manage values in the editor with specific classes.

Defining a vocabulary

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model.

Working with categories

A category is an identifier that you can assign to business rules and certain business elements to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. The rule editor drop-down list shows only the business elements that a rule can see.

Working with domains

You can use domains to extend the business object model (BOM).

Translating an existing business vocabulary

To translate an existing vocabulary that is defined on the BOM, you can copy the vocabulary file or edit the verbalizations.

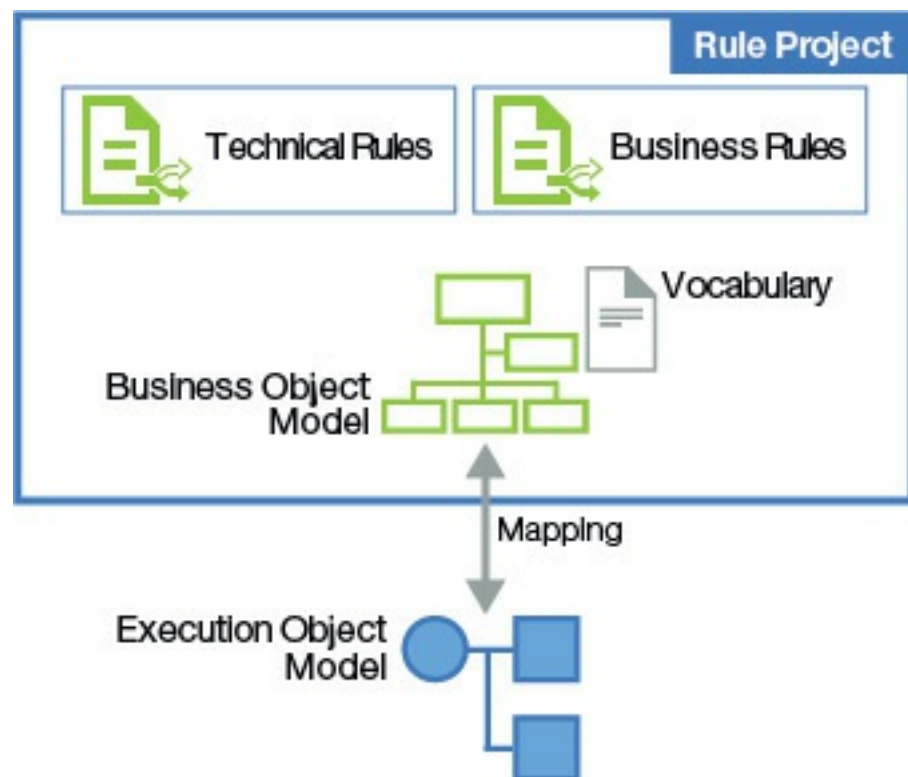
Parent topic: [Designing projects for rule authoring](#)

Overview: A data model for rule authoring

You use a vocabulary to write rules. You define the vocabulary in the business object model (BOM).

The vocabulary defines what policy managers can write business rules about. The elements of the vocabulary are defined in a Business Object Model (BOM).

You can define the BOM from scratch or create it from an Execution Object Model (XOM) that references compiled Java™ classes and other data sources. You can extend the BOM without modifying the XOM. The XOM describes quite extensively the elements that you need for rule editing



You can define a BOM in one of two ways:

- **Bottom-up:** In this case, you create a BOM entry from the XOM. If you find that you require additional elements in the BOM to facilitate rule editing for policy managers, you can extend the BOM by creating business elements and mapping them to the XOM.
- **Top-down:** In this case you create business elements without necessarily considering the way that the execution elements are going to be implemented. At a later date, when you implement the execution elements, you can specify a mapping for all your business elements.

The rule project contains:

- Business rules, which express a business policy. Business rules have a list of conditions to meet before doing a list of actions. You write business rules using the Business Action Language (BAL).
- Technical rules, which comprise a *condition* part and an *action* part. The condition part binds variables to objects and attribute values and specifies tests on attribute values. The action part specifies the actions to be carried out if the rule is executed. You write technical rules in Rule Designer.

Parent topic: [Configuring the BOM for rule authoring](#)

Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

[Vocabulary](#)

[Action rules](#)

Related information:

[Business object model \(BOM\)](#)

[Technical rules](#)

Business object model (BOM)

The BOM is the object model for business rules. You edit BOM members in the BOM editor.

Introducing the business object model (BOM)

You use the BOM to make business rule editing user-friendly by providing tools to set up a natural language vocabulary. With this vocabulary, policy managers can describe their business logic in a business rule language.

Collections

In the BOM, arrays and collections represent a set of objects. In the BAL, you must set a collection domain on a collection of type `java.util.Collection`.

Parent topic: [Configuring the BOM for rule authoring](#)

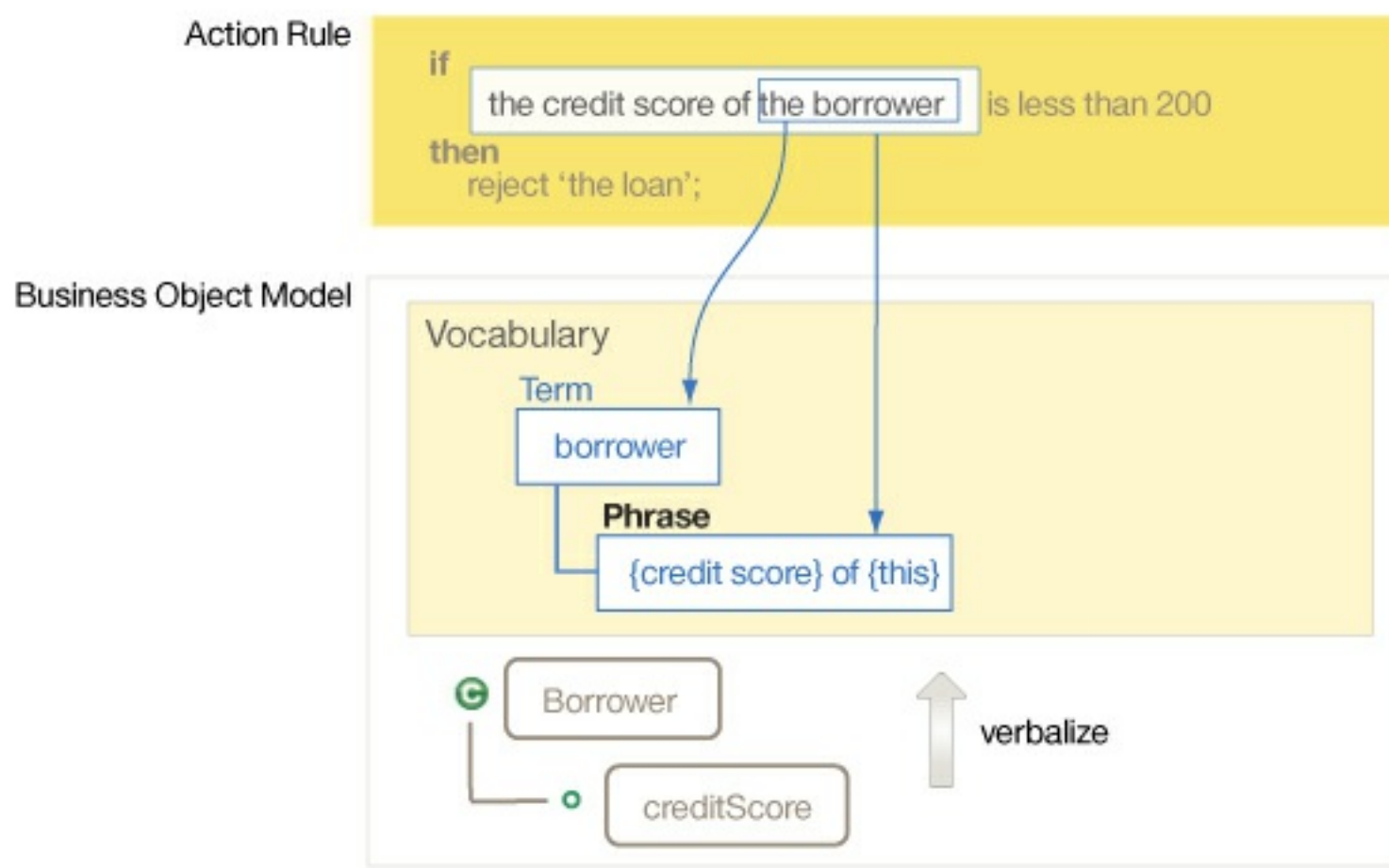
Introducing the business object model (BOM)

You use the BOM to make business rule editing user-friendly by providing tools to set up a natural language vocabulary. With this vocabulary, policy managers can describe their business logic in a business rule language.

The BOM is the basis for the vocabulary used in business rules. It is an object model similar to a Java™ object model, and contains elements that map to those of the XOM.

A BOM contains the classes and methods that rule artifacts act on. As an object model, the BOM is very similar to a Java object model. It consists of classes grouped into packages. Each class has a set of attributes, methods and, possibly, other nested classes.

BOM-to-XOM mapping defines the correspondence between the BOM and the execution object model (XOM) used at runtime.



System BOM

By default, the BOM always includes classes that map to specific JDK classes, and basic date and time-related classes. This set of classes is called the System BOM. For example, to compare the parts of a date, the System BOM contains the following classes, which map to the parts of a `java.util.Date` and have associated value editors:

- `ilog.rules.brl.SimpleDate`
- `ilog.rules.brl.Time`
- `ilog.rules.brl.DayOfWeek`
- `ilog.rules.brl.Month`
- `ilog.rules.brl.Year`

If you have a BOM member of type `java.util.Date`, you can change that type to one of the System BOM date types. The mapping is carried out automatically.

BOM entries

A business object model comprises one or more BOM entries. A BOM entry defines a set of business elements in the business object model.

You can order BOM entries so that if you have two business elements with the same name in two BOM entries, the one in the first BOM entry in the path overrides the other.

A BOM entry comprises:

- A BOM file, which describes the structure of the BOM
- A VOC file, which is locale-specific and describes the vocabulary associated to the BOM
- A B2X file, which describes the mapping between the BOM and the XOM

Parent topic: [Business object model \(BOM\)](#)

Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

[BOM properties](#)

[Collections](#)

[Vocabulary](#)

Related tasks:

[Defining and assigning categories](#)

[Setting up automatic variables](#)

Related information:

[Overview: A data model for rule authoring](#)

[Object state update](#)

[Creating BOM entries](#)

[Creating BOM elements](#)

[Defining a vocabulary](#)

Collections

In the BOM, arrays and collections represent a set of objects. In the BAL, you must set a collection domain on a collection of type `java.util.Collection`.

In the BOM, collections are members that are semantically linked to a BOM class with a one-to-many relationship.

Collections are shown in business rules with BAL operators and constructs such as:

- `<an object>` is one of `<objects>`
- the number of `<objects>` in `<objects>`

In the following example, the action rule uses collections:

```
definitions
set 'company1' to a company;
set 'customer1' to a customer in the employees of 'company1';
if
  the number of books in the items of the shopping cart of 'customer1' is more
  than 5
...
```

You can represent a set of objects in the BOM in two ways:

- Array (`java.lang.Object[]`)
- Collection (`java.util.Collection`)

Both are mapped to the concept of "collection of objects" in the BAL. From a BAL perspective, a collection is an element with multiple-cardinality. From a BOM perspective, it is an array, or a BOM member with a collection domain.

A collection is treated as a "collection of objects" in the BAL only if it has a collection domain. A collection domain specifies the cardinality and the type of collection elements, for example, `0,* class Customer`.

Note:

Ruleset parameters or variables of type or subtype `java.util.Collection` are treated as "collection of objects" in the BAL.

When you edit a business rule using the Content Assist box in designer, you can access the constructs relative to collections if the member is of type:

- array
- `java.util.Collection` and has a collection domain.

Collections of array type

Members (attributes, methods, constructors) of array type are automatically treated as collections.

For example, in the following BOM class, the method `getEmployees` is automatically treated in business rules as a collection of `Employee` objects.

```
public class Company {
    Employee[] getEmployees();
}
```

Collections of `java.util.Collection` type

In the BAL, a member of type `java.util.Collection` is not treated as a collection until you set a collection domain.

For example, in the following BOM class, a collection domain is set and defined.

```
public class Company {
    java.util.Collection getEmployees() domain 0, * class Employee;
}
```

Use the BOM editor to set a domain on a member of type `java.util.Collection`. For information about

defining domains, refer to [Defining domains](#).

Parent topic: [Business object model \(BOM\)](#)

Related concepts:
[Domains](#)

Related tasks:
[Creating packages and classes](#)

Creating BOM elements

You can extend the business object model (BOM) with new business elements, and manage values in the editor with specific classes.

[Creating packages and classes](#)

Extend the BOM with new business elements or properties.

[Adding nested classes](#)

You can enclose a class in an existing class.

[Adding members to a class](#)

In the BOM editor, you can add members to a class.

Parent topic: [Configuring the BOM for rule authoring](#)

Creating packages and classes

Extend the BOM with new business elements or properties.

About this task

You can extend the business object model (BOM) with new business elements or new properties on BOM elements. You can also manage values in the BOM editor by developing specific classes and referencing them in BOM custom properties.

You start by adding a package to a BOM entry, and then adding a class to the BOM package.

Procedure

To add a package to a BOM entry and add a class to the package:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in your BOM entry.

2. Click **New Package**.
3. In the New BOM Package wizard, in the **Name** field, specify the fully qualified name of the package you want to create.
4. Click **Finish**.

The BOM entry now contains the new package. You must now add a class to the BOM package.

5. To add a class, in the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in the BOM entry.

6. Click **New Class**.
7. In the New BOM Class wizard, in the **Package** field, specify the fully qualified name of the package in which you want to create the class.
8. In the **Name** field, specify the name of the class you want to create.
9. Click **Finish**.

Results

You now have a new BOM package, with a new class added to it.

Note:

When using the Intellirule editor to write rules, you can create business elements on the fly. If you use a new element in a rule, the Intellirule editor detects that it must be added to the BOM and displays a Quick Fix message so that you can add it automatically. See [Correcting errors by using Quick Fix](#).

Parent topic: [Creating BOM elements](#)

Related tasks:

[Adding nested classes](#)

[Adding members to a class](#)

Related information:

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

[Creating a BOM without a XOM](#)

Adding nested classes

You can enclose a class in an existing class.

About this task

You can add a new class in an existing class to create a nested class.

Procedure

To add a nested class:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in your BOM entry.

2. Click **New Class**.
3. In the New BOM Class wizard, select the **Enclosing class** box and then click **Browse** to select the enclosing class.
4. In the **Name** field, specify the name of the class you want to create.
5. Click **Finish**.

Results

The new class is added as a nested class.

Parent topic: [Creating BOM elements](#)

Related tasks:

[Creating packages and classes](#)

[Adding members to a class](#)

Adding members to a class

In the BOM editor, you can add members to a class.

About this task

Add members such as attributes, constructors, or methods to a class.

Procedure

To add a member to a class:

1. In the Outline view, expand the BOM entry and click the class to which you want to add members.
2. In the Class tab of the BOM Editor, in the Members section, click **New**.
3. In the New Member wizard, select whether the new member is an attribute, a constructor, or a method.
4. Specify its name, type, and arguments if any.
5. Click **Finish**.

Results

The Members section displays the new member.

Parent topic: [Creating BOM elements](#)

Related tasks:

[Creating packages and classes](#)

[Adding nested classes](#)

Related information:

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

Defining a vocabulary

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model.

Vocabulary

The vocabulary comprises terms and phrases that are attached to the elements of the BOM, and terms that are defined for ruleset variables and ruleset parameters. You use the vocabulary to create business rules.

Defining the vocabulary for a BOM entry

You can define a vocabulary for a complete BOM entry.

Editing terms

You edit terms to change, for example, the singular or plural form of a term.

Documenting terms

You document business terms to provide guidance to policy managers.

Editing and creating phrases

You can create a new phrase, edit a phrase, including the subject of a phrase, and use a reduced verbalization for simpler rules.

Editing the verbalization of constants

Constants correspond to static references in the BOM. You can change the verbalization of constants.

Editing the verbalization of ruleset variables

The default verbalization of a ruleset variable is its name. You can edit this verbalization.

Editing the verbalization of ruleset parameters

The default verbalization of a ruleset parameter is its name. You can edit this verbalization.

Parent topic: [Configuring the BOM for rule authoring](#)

Vocabulary

The vocabulary comprises terms and phrases that are attached to the elements of the BOM, and terms that are defined for ruleset variables and ruleset parameters. You use the vocabulary to create business rules.

Rule Designer verbalizes BOM elements to make them visible in business rules. Rule Designer creates a default vocabulary that can be translated.

When you create a BOM entry, you can choose to verbalize the business elements to create a vocabulary. A default verbalization is then applied to all attributes, getters, setters, and static references in the BOM entry. You can create a code-like verbalization of all other methods by selecting the option **All Methods** in the New BOM Entry or Verbalize BOM wizards.

In addition to vocabulary elements created from the BOM, you create vocabulary terms by defining a verbalization for ruleset parameters and ruleset variables.

Only business rules use the vocabulary. Technical rules and functions do not use the vocabulary. If you do not verbalize a business element, that element is not part of the vocabulary, and therefore not visible from business rules.

You can translate the vocabulary, and use several vocabularies in different languages on top of the same business object model.

Vocabulary elements

A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.

Phrase templates

A phrase template is a pattern for the verbalization of phrases. The pattern changes for navigation, predicate, or action phrases.

Default verbalization

Rule Designer applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization in the BOM Editor.

Vocabulary errors and warnings

If you have vocabulary ambiguities, missing placeholders, or problems on business elements, Rule Designer raises errors and warnings on terms and phrases at editing or build time.

Parent topic: [Defining a vocabulary](#)

Related concepts:

[Categories](#)

[Vocabulary elements](#)

[Phrase templates](#)

Related tasks:

[Defining and assigning categories](#)

Related information:

[Vocabulary errors and warnings](#)

[Overview: Ways to express business rules](#)

[Business object model \(BOM\)](#)

[Defining a vocabulary](#)

Vocabulary elements

A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.

Whether a vocabulary element is a business term, a phrase, or a constant depends on the nature of its corresponding business element.

The drop-down lists of business rule editors show vocabulary elements.

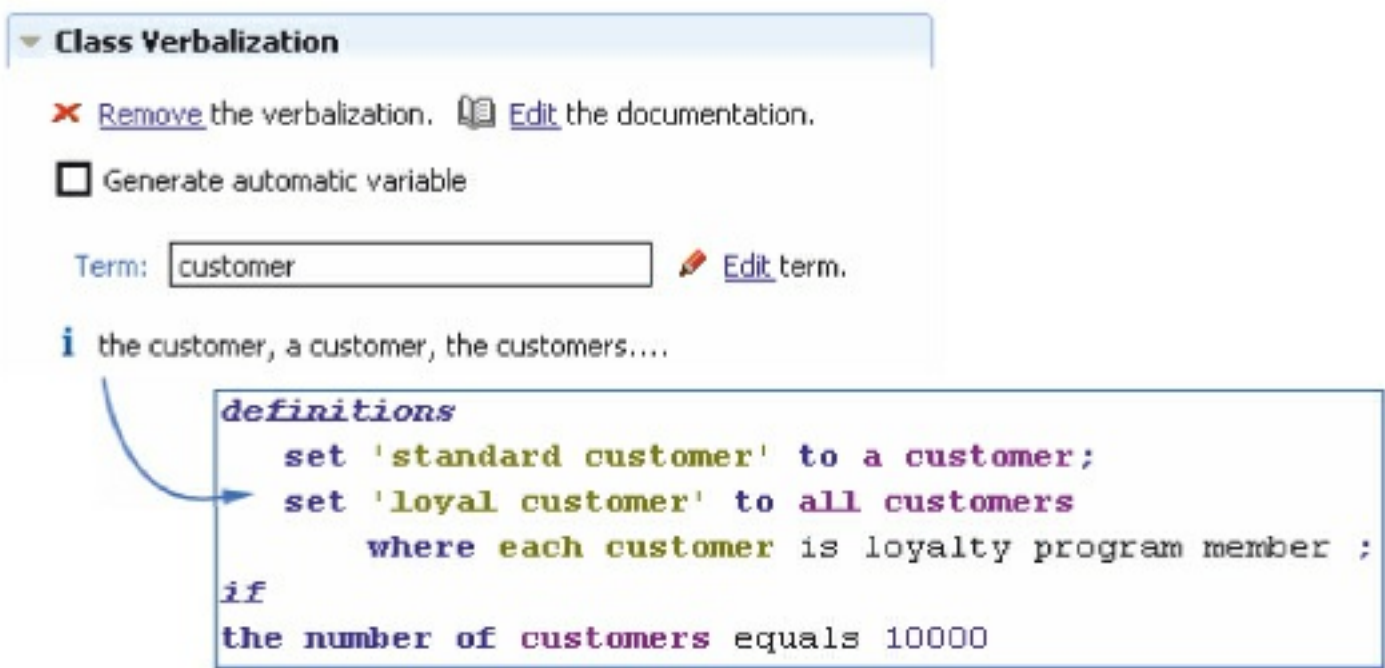
Important:

If you change the default verbalization for a vocabulary element, avoid using parentheses (). These characters are considered as delimiters and can cause false difference, overlap, or gap warnings in decision tables.

Business term

A business term is a key concept handled in business rules. It is the verbalization of a class or nested class in the BOM.

For example, customer is a business term.



Attention:

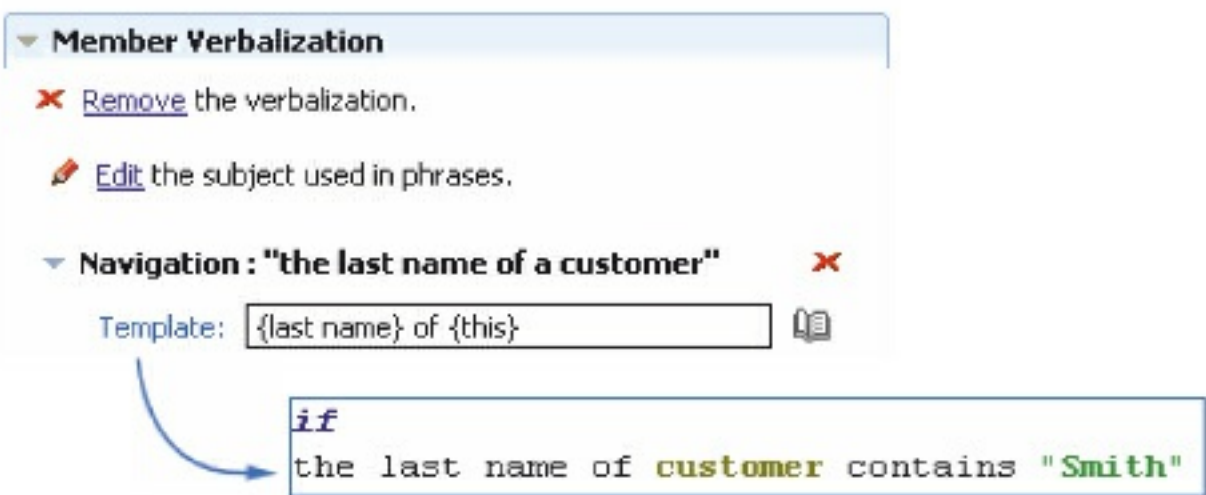
Do not use the characters “ and > in term labels. They cause warnings and ambiguity problems when used in rules.

Navigation phrase

A navigation phrase is a phrase that associates two business elements. It can be the verbalization of:

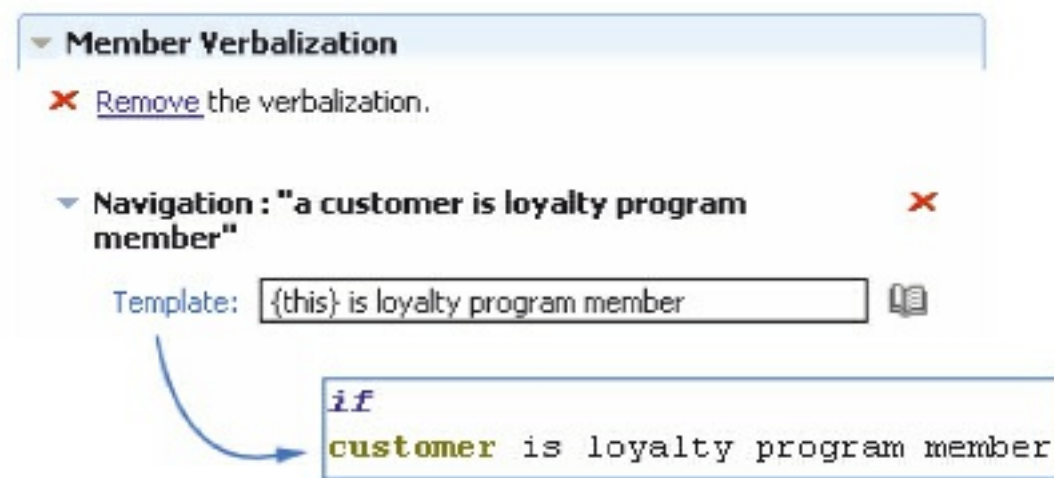
- A method that has a non void return type
- The getter part of an attribute

For example, {last name} of {this} is a navigation phrase for the attribute Customer.lastName.



You can also use predicate phrases in rules. A predicate phrase is a specific type of navigation phrase that verbalizes a non void method that returns a boolean or java.lang.Boolean. A predicate phrase has an implicit subject. For example, {this} is loyalty program member is a predicate phrase for the attribute

Customer.loyaltyProgramMember.



Action phrase

An action phrase applies an action to an object. It can be the verbalization of:

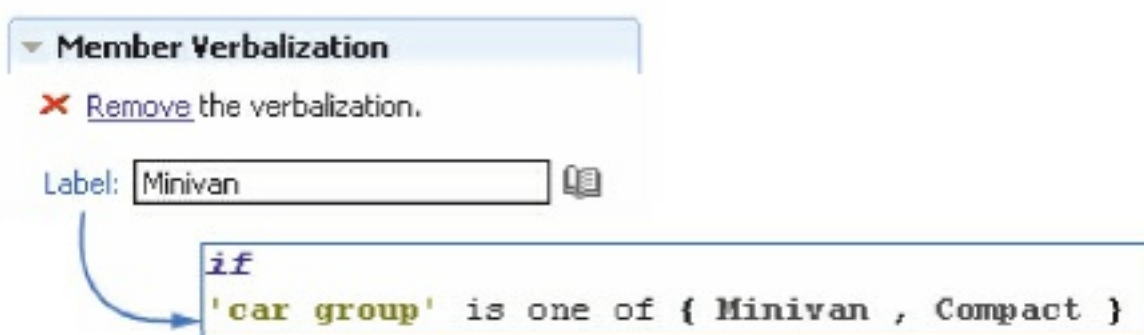
- A method that has a void return type
- The setter part of an attribute

For example, in {this}, display the message {0} is an action phrase for the method `Session.sendMessage(String)`.



Constant

A constant is the verbalization of the public static final attribute of a class with the same type as this class. For example, Minivan is a constant for the public static final attribute `CarGroup.Minivan`.



Parent topic: [Vocabulary](#)

Related concepts:

[Phrase templates](#)

[Vocabulary](#)

Phrase templates

A phrase template is a pattern for the verbalization of phrases. The pattern changes for navigation, predicate, or action phrases.

Phrase templates combine placeholders and text. Text binds placeholders to compose a phrase. For example, in the phrase template {last name} of {this}, {last name} and {this} are placeholders, and of is text.

Attention:

Do not use the characters “ and { and } in phrase templates. They cause warnings and ambiguity problems when used in rules.

Placeholders

Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type.

Simplified phrase templates

To avoid wordiness and make phrases shorter, you can remove the {this} placeholder if there is no ambiguity between BOM members.

Parent topic: [Vocabulary](#)

Related concepts:

[Vocabulary elements](#)

[Placeholders](#)

[Vocabulary](#)

Related tasks:

[Creating a BOM entry from a XOM](#)

Related information:

[Default verbalization](#)

[Simplified phrase templates](#)

Placeholders

Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type.

Vocabulary phrase templates contain placeholders. Placeholders represent gaps in phrases that can be filled in either automatically or manually when editing rules. Placeholders are identified by curly brackets {}.

There are three types of placeholder:

- Subject placeholders, which are completed automatically
- {this} placeholders, which you complete manually when editing rules
- Argument placeholders, which represent the arguments of a method, and which you complete manually when editing rules

To set a different text for an argument placeholder, use the syntax {x, "my text"} in the phrase template.

Subject placeholder

Subject placeholders are shown in navigation phrases. The subject is a business term that you can edit in the BOM Editor. The subject corresponds to the return type of a member.

The subject placeholder is optional. When used in rules, the subject placeholder automatically takes the appropriate number and article for the context. In the placeholder text, the subject needs to be specified using the singular form, without article.

{this} placeholder

The {this} placeholder represents the declaring class of a member. You cannot use this placeholder in the verbalization of static members.

For example, for an attribute `lastName` of type `String` in class `Customer`, you can specify the following phrase template:

```
{last name} of {this}
```

If there is no ambiguity, you can write simplified phrase templates by not using the {this} placeholder. See [Simplified phrase templates](#).

Argument placeholder

Argument placeholders represent the arguments of methods. They are represented by the index of the argument.

For example, the action phrase in {this}, display the message {0} is the verbalization of the method `Session.sendMessage(String)`. {0} is the argument placeholder for a `String`.

▼ Member Verbalization

✖ Remove the verbalization.

▼ Action : "in a session, display the message : a string" ✖

Template:

then

in 'the current session', display the message : <a string>

Parent topic: [Phrase templates](#)

Related concepts:
[Phrase templates](#)

Simplified phrase templates

To avoid wordiness and make phrases shorter, you can remove the `{this}` placeholder if there is no ambiguity between BOM members.

If there is no ambiguity between members of the BOM in different classes, you can remove the `{this}` placeholder for the declaring class from the verbalization. This produces shorter phrases and makes your business rules less verbose.

Removing the `{this}` placeholder causes a vocabulary warning by default, but as long as you know that the phrase is not ambiguous, you can set a preference to ignore this warning.

For example, if you change the verbalization of `Customer.name` from `{name} of {this}` to `{name}`, everywhere in your business rules you can use the phrase `the name` instead of `the name of the customer`. However, if you have `Company.name` verbalized as `{name}` in the same BOM, you must modify one of the phrase templates to avoid ambiguity. For example, you could verbalize `Customer.name` to `{customer name}`.

When using phrases that contain no `{this}` placeholder in their verbalizations, a binding on the working memory is generated in the IRL translation of the rule. In order to execute this kind of rule, an object of the right type must be inserted into the working memory.

Parent topic: [Phrase templates](#)

Related concepts:

[Placeholders](#)

[Phrase templates](#)

Related tasks:

[Using a reduced verbalization for simpler rules](#)

Related information:

[Vocabulary errors and warnings](#)

Default verbalization

Rule Designer applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization in the BOM Editor.

When you verbalize a BOM, a default verbalization is applied to all the members you select in the New BOM Entry or Verbalize BOM wizards. You might need to edit the default verbalization afterwards to add new phrases, or to make phrases more readable.

Table 1. Default verbalization

Business element	Default verbalization
Class For example: Customer	Class name in lowercase For example: customer
Constructor	None
Getters (non write-only attributes and methods of type getXXX without arguments) For example: getAge	{XXX} of {this} For example: {age} of {this}
Boolean getters For example: isRich	{this} is XXX For example: {this} is rich
All other methods For example: displayMessage(String)	Code-like verbalization, that is, the method signature, without subjects. For example: {this}.displayMessage({0})
Setters (non read-only attributes and void methods of type setXXX with one argument) For example: setDiscount	set the XXX of {this} to {0} For example: set the discount of {this} to {discount}
Boolean setters For example: loyaltyProgramMember	make it {0} that {this} is XXX For example: make it {loyalty program member} that {this} is loyalty program member
Static references (public static final attribute of a class with same type as this class) For example: Minivan	Name of the static reference For example: Minivan

Parent topic: [Vocabulary](#)

Related concepts:

[Phrase templates](#)

Related information:

[Vocabulary errors and warnings](#)

[Defining a vocabulary](#)

Vocabulary errors and warnings

If you have vocabulary ambiguities, missing placeholders, or problems on business elements, Rule Designer raises errors and warnings on terms and phrases at editing or build time.

Vocabulary elements, such as terms, phrases, articles and constants, might be made up of one or more words, called lexical units. These lexical units must comply with specific lexical rules. Vocabulary elements must be non ambiguous. You cannot verbalize two classes with the same term, and you cannot verbalize two members with the same phrase.

Terms

Terms must not conflict with the elements of the business rule language and the System BOM. For example, if a word in a term can be interpreted as a number, you get an error.

Ambiguity errors are reported locally when editing a vocabulary in the same vocabulary file, and reported globally at build time if the ambiguity comes from duplicate terms in two separate files.

Phrases

In phrases, you can use each type of placeholder only once.

You get a warning in the following circumstances:

- If the subject placeholder is not used in a navigation phrase
- If the {this} placeholder is missing

You can set a preference to ignore these warnings.

Errors in phrases can also come from their related business element:

- An action phrase for a read-only or final attribute
- A navigation phrase for a write-only attribute

Table 1. Examples of vocabulary errors

Message	Description
Character > is forbidden.	A vocabulary element uses a forbidden character. This error is reported when editing the vocabulary.
Character > in term for class balsample.Book is forbidden.	A vocabulary element uses a forbidden character. This error is reported at build time.
Term book is duplicated.	The term is already used for another class in the current vocabulary file.
Lexical conflict in production l-target(balsample.Book,SINGLE,INDEFINITE_ARTICLE,_,_,_) -> 'a' "book" "12" expecting construct "12" and having construct "<ilog.rules.brl.Number>"	Lexical conflict: the lexical unit 12 in the term associated to the class Book is interpreted as a number.
Placeholder "author" is missing in "balsample.Book: author"	The subject placeholder (author) is missing in the phrase associated to the member balsample.Book.author.
Duplicate verbalization: [balsample.Book: author] and [balsample.Book: title]	The members balsample.Book.author and balsample.Book.title have the same verbalization.
Action phrase not allowed for "balsample.Book: author" (member is read-only)	The attribute is read-only and must not be verbalized as an action phrase.
Navigation phrase not allowed for "balsample.Book: author" (member is write-only)	The attribute is write-only and must not be verbalized as a navigation phrase.

Delimiters in numbers

Commas and spaces are used to separate parts of rules and expressions. In certain locales, they also serve as delimiters in numbers:

- A comma can be used to group three digits, as in 10,000.
- A space can also be used to group three digits, as in 10 000.
- A comma can be used as a decimal mark, as in 3,14159265.

Follow these instructions to avoid problems when expressing numbers:

- When possible, do not use digits in verbalizations.
- When using digits, avoid space and comma separators between digits, especially if those characters are valid group delimiters or decimal markers in the locale of the vocabulary.
- If a verbalization must contain digits separated with spaces or commas, surround the numbers with double quotation marks, for example, "3,14159265".

Parent topic: [Vocabulary](#)

Related concepts:

[Vocabulary](#)

[Phrase templates](#)

Defining the vocabulary for a BOM entry

You can define a vocabulary for a complete BOM entry.

About this task

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model. If an element is not verbalized, it is not available in business rules. You can define a vocabulary for a complete BOM entry using the Verbalize BOM wizard.

Procedure

To define the vocabulary for a BOM entry:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.
2. In the BOM Editor Package page, in the Tasks section, click **Verbalize the elements of this BOM entry**.
3. In the Verbalize BOM wizard, choose whether you want to verbalize only the Getters part of methods and attributes, only the Setters part, all methods, or only static references. Select or clear the relevant check boxes to include or remove vocabulary elements from the vocabulary.
4. Click **Finish**.

Results

The BOM entry is now verbalized. You can edit the default verbalization in the BOM Editor to align the terms and phrases with your business model.

Attention:

If you change the default verbalization for terms, phrases, or constants, avoid using parentheses (). These characters are considered as delimiters and can cause false difference, overlap, or gap warnings in decision tables.

Parent topic: [Defining a vocabulary](#)

Editing terms

You edit terms to change, for example, the singular or plural form of a term.

About this task

You can edit terms in the BOM Editor to change, for example, the singular or plural form of a term.

Procedure

To edit a term:

1. In the Outline view, click the class corresponding to the term you want to edit.
2. In the Class Verbalization section of the BOM Editor, click **Edit term**.

The Edit Term dialog opens.

3. Change the term, as required:
 - Change the singular or plural form of the term
 - Change its definite and indefinite articles

To edit fields in this dialog, clear the check box next to the field.

4. When you have finished, click **OK** to close the Edit Term dialog.

Parent topic: [Defining a vocabulary](#)

Documenting terms

You document business terms to provide guidance to policy managers.

About this task

To make business terms reusable and provide guidance to policy managers, it is important to document them. The documentation you add displays in the information box next to the Content Assist box when you edit business rules.

Procedure

To document a term:

1. In the Outline view, click the class corresponding to the term you want to document.
2. In the Class Verbalization section of the BOM Editor, click **Edit the documentation**.
3. In the Business Term Documentation dialog, enter the required description or comment.
4. Click **OK**.

Results

The documentation text is shown at the bottom of the Class Verbalization section.

Parent topic: [Defining a vocabulary](#)

Related tasks:

[Documenting phrases](#)

Editing and creating phrases

You can create a new phrase, edit a phrase, including the subject of a phrase, and use a reduced verbalization for simpler rules.

Editing a phrase

Phrases correspond to methods in the BOM. You can edit the navigation and action phrases using the BOM Editor.

Editing the subject of a phrase

You can edit the subject of a phrase using the Edit Term dialog.

Creating new phrases

You can define several navigation and action phrases for a BOM element.

Using a reduced verbalization for simpler rules

You can reduce the verbalization to simplify your business rules.

Documenting phrases

You document phrases to provide guidance to policy managers.

Parent topic: [Defining a vocabulary](#)

Editing a phrase

Phrases correspond to methods in the BOM. You can edit the navigation and action phrases using the BOM Editor.

About this task

Phrases correspond to methods in the BOM. You can edit the verbalization of phrases.

Procedure

To edit a phrase:

1. In the Outline view, click the element corresponding to the phrase you want to edit.
2. In the Member Verbalization section of the BOM Editor, edit the navigation and action phrases in the **Template** field, using text and Content Assist.

Note:

To edit the subject of a phrase, click **Edit the subject used in phrases** and change the term using the Edit Term dialog.

3. Save the BOM.

The new phrase verbalization is now visible in business rules.

Parent topic: [Editing and creating phrases](#)

Editing the subject of a phrase

You can edit the subject of a phrase using the Edit Term dialog.

About this task

In the Edit Term dialog, you can edit a subject phrase through the subject placeholder.

Procedure

To edit the subject of a phrase:

1. In the Member Verbalization section of the BOM Editor, click **Edit the subject used in phrases**.

The Edit Term dialog opens.

2. Change the phrases, as required. You can make the following changes:

- Change the singular or plural form of the term
- Change its definite and indefinite articles

3. When you have finished, click **OK** to close the Edit Term dialog.

Parent topic: [Editing and creating phrases](#)

Creating new phrases

You can define several navigation and action phrases for a BOM element.

About this task

When you define several navigation or action phrases for a BOM element, rule authors can use different wordings while using the same method or attribute.

Procedure

To add a phrase to a method:

1. In the Member Verbalization section of the BOM Editor, click either **Create a navigation phrase** or **Create an action phrase**.

The phrase is shown in the list of phrases. An error might be display because the new phrase is defined with the default verbalization. If you already have the default verbalization for an existing phrase, there is a conflict because you should not define the same verbalization twice in the vocabulary.

2. Click the phrase to expand it, and then in the **Template** field edit the verbalization.
3. Save the BOM.

Results

You can now use the phrase in your business rules.

Parent topic: [Editing and creating phrases](#)

Using a reduced verbalization for simpler rules

You can reduce the verbalization to simplify your business rules.

About this task

If there is no ambiguity between members of the BOM in different classes, you can remove the {this} placeholder for the declaring class from the verbalization. This produces shorter phrases and makes your business rules less verbose.

Procedure

To simplify the verbalization:

1. In the Outline view, click the member for which you want a simpler verbalization.
2. In the Member Verbalization section of the BOM Editor, expand the phrase to show the Template field.
3. Remove the of {this} part of the verbalization.

You can ignore the following warning for the time being:

Placeholder “this” is missing

4. On the **Window** menu, click **Preferences**. (On Mac, **Preferences** is in the **Eclipse** menu.)
5. In the side pane of the Preferences dialog, click **Rule Designer** > **Vocabulary**.
6. On the Vocabulary page, clear the box **Check {this} placeholders in phrases**.
7. Click **Apply**, and then **Apply and Close**.
8. Save the BOM. Click **Yes** if a message opens.

The warning is no longer shown and you can start using the simplified verbalization in your business rules.

Parent topic: [Editing and creating phrases](#)

Documenting phrases


You document phrases to provide guidance to policy managers.

About this task

You can provide guidance to policy managers by adding documentation to phrases. Then, when you edit business rules, the description or comment you added is displayed in the information box next to the Content Assist box.

Procedure

To document a phrase:

1. In the Outline view, click the member you want to document.
2. In the Member Verbalization section of the BOM Editor, next to the **Template** field, click the  **Edit Phrase Documentation** button.
3. In the Phrase Documentation dialog, enter the required description or comment.
4. Click **OK**.

To view the documentation text again, click the  **Edit Phrase Documentation** button.

Parent topic: [Editing and creating phrases](#)

Related tasks:

[Documenting terms](#)

Editing the verbalization of constants

Constants correspond to static references in the BOM. You can change the verbalization of constants.

About this task

Constants correspond to static references in the BOM. A static reference can be a static final attribute of a type, for example, Borrower. To change the verbalization of a constant, you edit the corresponding static reference.

Procedure

To edit a constant:

1. In the Outline view, click the static reference you want to edit.
2. In the Member Verbalization section of the BOM Editor, edit the verbalization in the **Label** field.

You cannot use placeholders and special characters such as % or \$.

3. Save the BOM.

The new constant verbalization is now visible in the business rules.

Parent topic: [Defining a vocabulary](#)

Editing the verbalization of ruleset variables

The default verbalization of a ruleset variable is its name. You can edit this verbalization.

About this task

By default, a ruleset variable is verbalized as its name. You can change the verbalization using the Variable Set Editor.

Procedure

To edit the verbalization of a ruleset variable:

1. In the Rule Explorer view, double-click the variable set that contains the ruleset variable you want to verbalize.

The Variable Set Editor opens.

2. In the Variable Set Editor, in the Verbalization column, click the verbalization you want to change and then click **Refactor**.
3. In the Refactor Variable dialog, type a new verbalization for the ruleset variable.

You cannot use placeholders and special characters such as % or \$.

4. Click **Preview** to assess the impact of this change on the business rules.
5. Click **OK**.
6. Save the variable set.

The verbalization of the ruleset variable is now changed.

Parent topic: [Defining a vocabulary](#)

Editing the verbalization of ruleset parameters

The default verbalization of a ruleset parameter is its name. You can edit this verbalization.

About this task

By default, a ruleset parameter is verbalized as its name. You can change the verbalization using the Rule Project Properties dialog.

Note: You use ruleset parameters with classic rule projects only. For decision services, you use ruleset variables to define parameters for the decision operation.

Procedure

To edit the verbalization of a ruleset parameter:

1. In the Rule Explorer view, right-click the rule project and then click **Properties**.
2. In the Rule Project Properties dialog, click **Ruleset Parameters** in the side pane.

The Ruleset Parameters page opens.

3. In the Verbalization column, click the verbalization you want to change and then click **Refactor**.
4. In the Refactor Parameter dialog, type a new verbalization for the ruleset parameter.
5. Click **Preview** to assess the impact of this change on the business rules, and then click **OK**.
6. In the Rule Project Properties dialog, click **OK**.

The ruleset parameter verbalization is now changed.

Parent topic: [Defining a vocabulary](#)

Working with categories

A category is an identifier that you can assign to business rules and certain business elements to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. The rule editor drop-down list shows only the business elements that a rule can see.

Categories

A category is a filter that is applied to business rules and to business elements such as classes and members.

Defining and assigning categories

You define a category at the rule project level and then assign it to a business element.

Parent topic: [Configuring the BOM for rule authoring](#)

Categories

A category is a filter that is applied to business rules and to business elements such as classes and members.

A category is an identifier that you can assign to business rules and certain business elements (classes and members) in order to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. Only business elements that a rule can see are shown in the rule editor drop-down list. You can assign one or more categories to business object model (BOM) classes and members, or to a business rule category filter.

By default, all business rules, classes, and members belong to the predefined category any. All rules can see the any category, whatever the category filter in use. Therefore, the default is that all classes and members can be used in all business rules.

Note:

System BOM elements have no category, so are hidden from any business rule regardless of which rule category filter the rule uses.

Before you can use a category, you must define it at the rule project level. If you define a category in a project that is referenced by another project, the category is usable in both projects.

There is no inheritance between classes and members, but you can assign the same category to all members of the same class.

For more information about using categories, see:

- [Defining and assigning categories](#)
- [Applying a category filter](#)

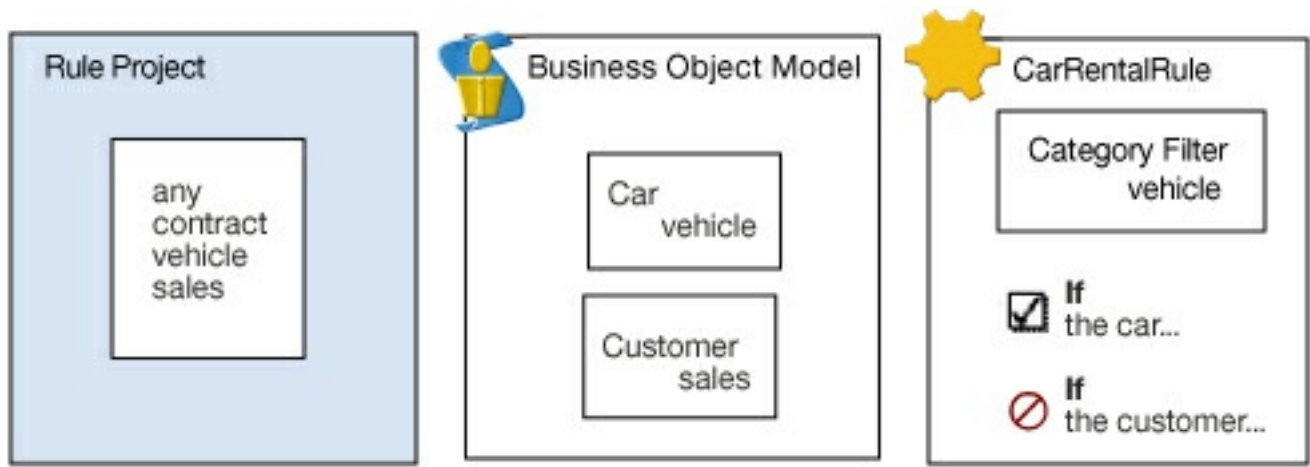
Table 1. Category usage summary

If a business element has the category...	And the business rule category filter is...	In that business rule, the business element is...
any	any	visible
any	another category	visible
another category	any	visible
category1 and category2	category1	visible
category1 and category2	category2	visible
category1 and category2	category1 and category2	visible
category1 and category2	category3 and category4	hidden

Example

In your business object model, you have a Car class with a category vehicle. You also have a rule project that contains a business rule CarRentalRule, which has vehicle in its category filter. From CarRentalRule, you can see all the classes that have the category vehicle, as well as all classes that have the any category. CarRentalRule can therefore see the Car class.

If you then define a category sales and assign it to a class Customer, you do not see the Customer class in the rule editor drop-down list for the rule CarRentalRule. If you still decide to use the Customer class in the business rule, it is reported as a warning.



Similarly, if the Car class has a member with a category other than vehicle, you cannot use this member in the conditions and actions for CarRentalRule, even though you can define a variable of the Car type in the definitions part of the rule.

Parent topic: [Working with categories](#)

Defining and assigning categories

You define a category at the rule project level and then assign it to a business element.

About this task

You define categories at the rule project level. When you have defined a new category, you can assign it to business elements and use it in business rule category filters.

Procedure

To define a new category and assign it to a business element:

1. In the Rule Explorer view, right-click the rule project and then click **Properties**.
2. In the side pane of the Properties dialog, click **Categories**.
3. On the Categories page, click **Add**.
4. In the New Category dialog, specify a name for the category and then click **OK**.

The Categories page displays the new category in the category list.

5. Click **Apply and Close** to close the Properties dialog.

The rule project now has a new category, which you can assign to a business element.

6. To assign a category to a business element, go to the Outline view and then click the business element to which you want to assign the category.
7. In the Categories section of the BOM Editor, click **Edit the categories**.
8. In the Categories dialog, select a category in the **All categories** field and then click > > to move it across to the **Selected categories** field.

You can also double-click the category to move it from one field to the other.

9. Repeat the previous step for each category you want to assign.
10. Click **OK** to close the Categories dialog.

Results

The rule project now has a new category and the Categories section of the BOM Editor lists the categories that you defined for the business element.

Note:

When you delete a category from the rule project, it is not deleted from the business elements and rule category filters that use it. To delete it fully, you must also delete it using the BOM Editor. You do not get a warning message to prompt you to delete using the BOM Editor.

Parent topic: [Working with categories](#)

Working with domains

You can use domains to extend the business object model (BOM).

Domains

Make the BOM more specific by setting domains on members. Domains can be static, dynamic, enumerated, or complex.

Defining domains

Use domains to extend the business object model (BOM).

Excel domain provider

You can create enumerated domains based on data stored in an Excel file. The Excel domain provider handles the link between the data stored in Excel and your BOM.

Creating dynamic domains from Excel

Create a dynamic domain and populate it with data from an Excel file.

Updating a single dynamic domain

If Rule Designer is customized to use an Excel file or other external data sources for BOM domain values, you can update these values from the BOM Editor.

Updating all dynamic domains

After populating or modifying the values of your enumerated domain, you can update all the BOM classes.

Parent topic: [Configuring the BOM for rule authoring](#)

Domains

Make the BOM more specific by setting domains on members. Domains can be static, dynamic, enumerated, or complex.

A domain places a restriction on type elements in the BOM. You can set a domain on classes, attribute types, method return types, and arguments.

The main domains include:

- Static domains, comprising [Literals](#), [Static references](#), [Bounded](#), [Collection](#), and [Other](#) domain types.
- [Dynamic](#) domains, with values stored in an Excel file.
- Enumerated domains, comprising [Literals](#), [Static references](#), and [Dynamic](#) domain types.
- Complex domains, comprising [Bounded](#), [Collection](#), and [Other](#) domain types.

Note:

Not all kinds of BOM domain are enforced in BAL rules or other business rules. Business rules use only enumerated domains (literal, static reference, or dynamic). A semantic check is done to check that the business rule does not use a value outside its domain, and the Intellirule editor suggests values from the enumerated domains. However, the semantic check done at the business rule level is primitive and does not detect complex patterns of incorrect usage that involves operators other than `is` or `is not`. Other kinds of domain, such as bounded domains, are not enforced at business rule level.

You can use domains when working with business rules:

- Editing business rules: Code completion in the Intellirule Editor uses enumerated domains to propose valid settings.
- Checking business rules: You use enumerated domains to report errors and warnings that help you to validate the values you specify.
- Analyzing rules: You use all domain types in the BOM to check the consistency of business rule semantics.

If the XOM has a set of `public`, `static`, and `final` attributes typed to the declaring class, they are automatically considered as an enumeration of static references of the class in the BOM.

If the XOM has members of array type, they are automatically considered as a collection of the class in the BOM.

Bounded

A bounded domain specifies an interval between two bounding values, such as `[0, 120]`.

In the designer BOM editor, when defining a bounded domain on an integer attribute, you can specify the `*` (asterisk) character as the lower or upper bound of the domain. When you specify `*`, the bound value is replaced in the corresponding `.bom` file by `-2147483648` for the lower bound, or `2147483647` for the upper bound. These values correspond to `Integer.MIN_VALUE` and `Integer.MAX_VALUE`, respectively.

Note:

Bounded domains are not enforced at business rule level. You can create a bounded domain on a number primitive type.

Collection

A collection domain specifies the cardinality and the type of collection elements, for example, `0,* class Customer`.

If you have members of type `java.util.Collection`, set a collection domain on these members for them to be automatically considered as a collection in business rules. You can also create, add, and remove methods for the items in the collection domain by using the BOM Editor.

For more information about collections, see [Collections](#).

Note:

You can create a collection domain on a collection or an array.

Dynamic

You can populate a domain in the BOM dynamically from a data source, and then synchronize the data source and the domain.

A dynamic domain is an enumerated domain with values from an Excel file.

Note:
You can create a dynamic domain only on a class in the BOM class editor, but not on a Collection subclass.

Literals

A domain set as an enumeration of literals specifies a list of values, for example, {1, 2, 3}.

Note:
You can create a literal domain on a primitive type or a string.

Static references

A domain set as an enumeration of static references specifies a list of references to constants, for example, {static GroupA, static GroupB, static GroupC}.

You can define attribute types and method return types and arguments as follows:

- If you have an attribute of type A, you can define a domain of static references on it using the static attributes of the class A (classic Java™ enumeration pattern).
- If you have an attribute of a primitive type, you can define a literal domain on it.

Note:
You can create a static reference domain on a class, but not on a Collection subclass.

Other

The ‘other’ domain types were introduced to support most domains that come from the XML binding. In an XML schema, you can define pattern domains (even for numbers) and simultaneously an enumeration. You can also define pattern domains or intersection of domains in the business object model.

You can define other types of domains by using the syntax of regular expressions. For example, you can define a pattern for Strings as follows:

```
"a*n"
```

You can define an intersection of domains as follows:

```
({1, 3, 5, 7, 9}, [0,6])
```

Note:
Other domains are not enforced at business rule level. You can set this type of domain on an array by using the following syntax: '{ (String)"a", (String)"b"}'.

Parent topic: [Working with domains](#)

Related concepts:
[Collections](#)

Related tasks:
[Defining domains](#)

Defining domains

Use domains to extend the business object model (BOM).

About this task

You can create domains of different types to extend the business object model (BOM).

Note:

You can create dynamic domains only on a BOM class.

Procedure

To define a domain:

1. In the Outline view, click the element to which you want to add the domain.
2. In the BOM Editor, in the Domain section, click **Create a domain**.

The Domains wizard displays the options that are available for the type of artifact that you are creating. The options can include items from the following list:

- **Bounded:**

To create a bounded domain, double-click **Bounded**, and then specify the bound values and whether they are included in the domain.

- **Collection:**

To create a collection, double-click **Collection**, and then click **Browse** to select the type of the collection. Note that a member of type `java.util.Collection` is not automatically treated as a collection in business rules until you set a collection domain on it.

- **Literals:**

To create an enumeration of literals, double-click **Literals**, and then click **Add** to add new values to the enumeration of literals.

- **Static References:**

To create an enumeration of static references, double-click **Static References**, and then click **Add** to add new static references to the list.

- **Other:**

To create any other type of domain, double-click **Other**, and then enter the domain definition in the field provided.

- **Excel:**

To create a dynamic domain from an Excel file, double-click **Excel**. For more information, see [Creating dynamic domains from Excel](#).

3. Click **Finish**.

The domain is added to the business element. You can view its definition in the Domain section of the BOM Editor.

You can now use the values of your domain when editing your rules.

Parent topic: [Working with domains](#)

Related concepts:

[Domains](#)

Related tasks:

[Creating dynamic domains from Excel](#)

[Updating a single dynamic domain](#)

[Updating all dynamic domains](#)

Excel domain provider

You can create enumerated domains based on data stored in an Excel file. The Excel domain provider handles the link between the data stored in Excel and your BOM.

Note: You can use the Excel domain provider only on a BOM class.

Some properties must be defined on the BOM class to retrieve the information from the Excel file. To ensure that the properties are mapped correctly, you must create an Excel file with the required structure. The Excel file must contain a column for the values of the domain provider, a column for the label, and a column for the BOM to XOM mapping.

You must create one row for each value of the domain provider. You cannot merge cells.

The Excel file must have the following structure:

Value column

You must create a value column to enter the values of the domain provider.

Label column

You must create a label column to enter the verbalization for the domain value. This label is the name displayed for the value when editing a rule.

BOM to XOM column

You must create a column to add the BOM to XOM mapping for each domain value.

Optional columns

The documentation column is optional. You can create a column to enter documentation on a value.

You can add additional label and documentation columns for other locales. The default locale is the locale of your Eclipse application. The values for the other locales are used when changing the locale of your Eclipse application.

You can also add columns if you want to use custom properties for the values in your dynamic domain. The custom properties that you define through the Domains wizard apply to all the values in your dynamic domain.

Sheets

You can have several domain providers in the same Excel file. Each worksheet corresponds to one domain provider.

For example, you could have a sheet for a domain called Currency, and another sheet for a domain called Status.

The following table is an example of the columns and values in a configured Excel file. In this example, the domain defines the status of the loan:

Values	BOM to XOM	English label	Documentation (En)	Nom français	Documentation (Fr)
BLOCKED	return "Blocked";	Blocked	The loan is blocked	Bloqué	Le prêt est bloqué
ACCEPTED	return "Accepted";	Accepted	The loan is accepted	Accepté	Le prêt est accepté
REJECTED	return "Rejected";	Rejected	The loan is rejected	Rejeté	Le prêt est rejeté
PENDING	return "Pending";	Pending	The loan is pending	En attente	Le prêt est en attente

After creating your Excel file with the required columns and data, you must add the file to the resources folder of your rule project.

You can then map the domain properties to the columns in your Excel file, see [Creating dynamic domains from Excel](#).

Parent topic: [Working with domains](#)

Related concepts:
[Domains](#)

Related tasks:
[Creating dynamic domains from Excel](#)

Related information:

Creating dynamic domains from Excel

Create a dynamic domain and populate it with data from an Excel file.

Before you begin

You can create a dynamic domain with values extracted from an Excel file.

Make sure that the Excel file containing the values to populate the enumerated domain has the correct structure, see [Excel domain provider](#).

Important:

Make sure that the Excel file is in the resources folder of your rule project.

Procedure

To create a dynamic domain from an Excel file:

1. Open the BOM class in the BOM editor.
2. In the Domain section, click **Create a domain**.
3. Select **Dynamic Domains > Excel**, and then click **Next**.
4. In the **Excel File** field, select the Excel file that you added to the resources folder of your rule project.
5. In the **Sheet** field, select the sheet for the domain provider.
6. Select the **Table with header** check box if you have created a header in your Excel file.

Selecting this check box displays the name of the columns in the drop-down lists instead of the default column letters.

7. In the **Value column**, **BOM to XOM column**, and **Label column** fields select the corresponding columns in your Excel file.

In the Label column, you must select at least the label column for the default locale.

8. Click **Finish**, or click **Next** to add custom properties to the values.
9. Optional: Click **Add** to add a custom property.
 - a. Enter the name of your custom property.
 - b. Select the corresponding column in your Excel file.

Results

The values of the dynamic domain are displayed in the BOM editor.

The values are also available in the completion menu of the rule editor. You can now use them when editing your rules.

If you have defined labels or documentation for other locales, you must update the BOM for each locale. To do so, you must restart Rule Designer in the locale to update, and synchronize the BOM with the values in the Excel file. For more information, see [Updating a single dynamic domain](#) and [Updating all dynamic domains](#).

Parent topic: [Working with domains](#)

Related concepts:

[Excel domain provider](#)

[BOM properties](#)

Related tasks:

[Updating a single dynamic domain](#)

[Updating all dynamic domains](#)

Updating a single dynamic domain

If Rule Designer is customized to use an Excel file or other external data sources for BOM domain values, you can update these values from the BOM Editor.

About this task

You can populate and update the values of an enumerated domain on the BOM classes that have the required properties for the dynamic domain provider.

If you are working on a single BOM class, you can update the values from the BOM editor using the **Synchronize** link.

If you want to update several classes at the same time, see [Updating all dynamic domains](#).

Procedure

To update a single BOM class:

1. Open the BOM Editor.
2. In the BOM Editor, open the class that defines the dynamic domain.
3. Click **Synchronize** to populate the enumerated domain.

If the value of the domain provider property is invalid, the Domain section displays following message:
The value provider is invalid. Check the custom properties.

4. Save the changes in the BOM.

Results

If a value is no longer used, the value is set to deprecated, and a warning is displayed in the Problems view.

Parent topic: [Working with domains](#)

Related tasks:

[Updating all dynamic domains](#)

[Defining domains](#)

[Creating dynamic domains from Excel](#)

Updating all dynamic domains

After populating or modifying the values of your enumerated domain, you can update all the BOM classes.

About this task

You can populate and update the values of an enumerated domain on all BOM classes that have the required properties for the dynamic domain provider.

You can update the values for all the dynamic domains at the same time.

If you do not want to update all the dynamic domains at once, see [Updating a single dynamic domain](#).

Procedure

To update all dynamic domains in your BOM at once:

1. Open the BOM Editor.
2. In the BOM Editor, in the Tasks section of the Package page, click **Update the dynamic domains of this BOM entry**.

The Dynamic Domains dialog opens. This dialog shows all the BOM classes that have a dynamic domain defined.

3. Select the classes for which you want to update the enumerated domain and then click **Finish**.
4. Save the changes in the BOM.

Results

If a value is no longer used, the value is set to deprecated, and a warning is displayed in the Problems view.

Parent topic: [Working with domains](#)

Related tasks:

[Updating a single dynamic domain](#)

[Defining domains](#)

[Creating dynamic domains from Excel](#)

Translating an existing business vocabulary

To translate an existing vocabulary that is defined on the BOM, you can copy the vocabulary file or edit the verbalizations.

About this task

You can translate an existing business vocabulary by either making a translated copy of the .voc file or editing the class and member verbalizations.

Procedure

1. To translate a vocabulary by using the .voc file:

- a. In Rule Designer, switch to the Resource Perspective by clicking **Window > Open Perspective > Other > Resource**.

In the Navigator view, the BOM is displayed as a set of three files:

- model_en.voc
- model.b2x
- model.bom

- b. Right-click the .voc file, and then click **Copy**.
- c. Right-click the .bom folder, and then click **Paste**.

A Name Conflict dialog opens, prompting you to rename the file.

- d. In the Name Conflict dialog, replace the language identifier of the file with the language identifier of the translation language. For example, if your BOM entry is named model and you want to translate it to French, name the file model_fr.voc.
- e. Click **OK**.
- f. In the Navigator view, right-click the new .voc file, and then click **Open With > Text Editor**.

The text definition of the vocabulary opens in the Text Editor.

- g. You can now translate the vocabulary by translating the values of the vocabulary keys into the target language.
- h. Save the BOM.

Now when you launch Rule Designer in your target locale, the specified locale becomes the locale for both the vocabulary and the business rule project items.

2. To translate a vocabulary by using the class and member verbalizations:

- a. Open Rule Designer in the target locale.
- b. Open the BOM editor.
- c. Edit the class and member verbalizations.
- d. Save the file.

You now have a vocabulary for the target locale.

Parent topic: [Configuring the BOM for rule authoring](#)

Related information:

[Options for customizing rule authoring](#)

[Concrete syntax properties](#)

[Business Action Language \(BAL\)](#)

Authoring business rules

You use Rule Designer to create and edit different types of rules. You can use automatic variables, ruleset variables, templates, and categories to simplify the rule creation process.

Overview: Ways to express business rules

You write business rules such as action rules and decision tables by using the Business Action Language (BAL).

Working with action rules

You can create action rules by using the Intellirule editor or the guided editor.

Working with decision tables

You use the decision table editor to create and update decision tables.

Working with variables

You can use automatic variables and ruleset variables to define variables for use in business rules.

(Deprecated) Working with decision trees

You use the decision tree editor to create a workflow for the execution of your rules.

(Deprecated) Working with templates

You can create partially written rules and partially defined decision tables for use as templates, to simplify the task of creating rules and decision tables.

Working with technical rules

Use the Technical Rule Editor to write technical rules in the ILOG® Rule Language.

Working with functions

You can create a function in a rule project to share code procedures across more than one element of a ruleset. You express a function in ILOG Rule Language (IRL), and its code is evaluated when the ruleset runs.

Applying a category filter

You can apply a category filter to specify which categories of elements can be used in action rules, decision tables, and decision trees.

Correcting errors by using Quick Fix

You can use the Eclipse Quick Fix feature in the Intellirule Editor, the decision table editor, and the decision tree editor. Quick Fix messages offer suggestions to automatically correct detected errors.

Applying verbalization changes to business rules

If you change the verbalization of a business element used in a rule, you can refactor the business rules that use the business element to take your changes into account.

Parent topic: [Designing projects for rule authoring](#)

Overview: Ways to express business rules

You write business rules such as action rules and decision tables by using the Business Action Language (BAL).

The BAL provides a simple if-then syntax that you use with a vocabulary to write business rules. The BAL defines the syntax and provides constructs for expressing business rule conditions and actions, and the vocabulary defines the terms that you use in the business rules.

Modifiable building blocks make up the business rules. The building blocks represent vocabulary elements, ruleset parameters, ruleset variables, and BAL constructs and operators. For example, in the following action rule statement, the building block the `current load` uses a business term from the vocabulary, `is more than` is a BAL operator, and `5000` is a value:

```
if
  the current load is more than 5000
```

A phrase in a business rule can use a business term. For example, to complete the following phrase, you must select the business term `<a customer>`:

```
the age of <a customer>
```

You can create and maintain business rules in Rule Designer or Decision Center. You can deploy the business rules through decision services.

Operational Decision Manager provides various ways to express business rules:

Action rules

Action rules are sentence-like statements that express a set of conditions followed by the actions to take if the conditions are true. With action rules, you can state business policies in a predefined business vocabulary that a computer can interpret.

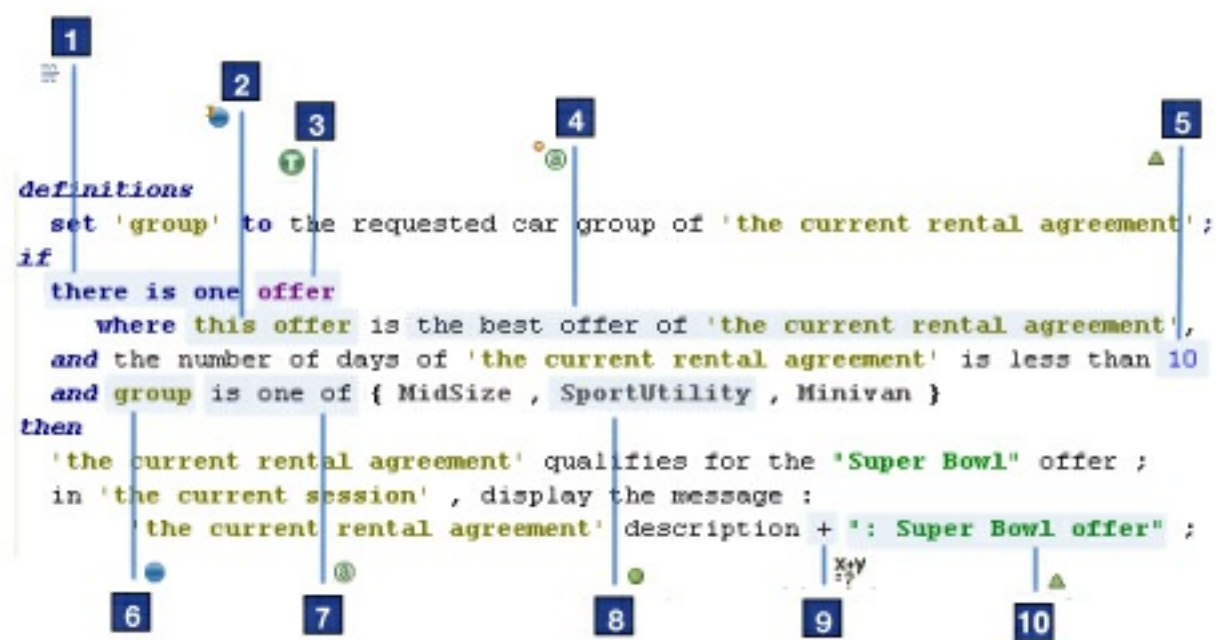
For example, you might state the policy "change customers in the Gold category to the Platinum category when they spend more than \$1,500 in a single transaction" as follows in an action rule:

```
If
  All of the following conditions are true:
    - the customer category is Gold
    - the value of the shopping cart is more than $ 1,500
Then
  Change the customer category to Platinum
```

In an action rule, you can use all the BAL constructs and operators, and all the elements in the vocabulary.

You can find these elements in the following figure:

- BAL construct **1**
- Implicit variable **2**
- Term **3**
- Phrase **4**
- Number value **5**
- Rule variable **6**
- BAL operator **7**
- Constant **8**
- Arithmetic operator **9**
- Text value **10**



Decision tables

Decision tables represent decision logic as a table. Each row in a decision table corresponds to an action rule.

The rows and columns of the tables show the possible situations that a business decision might encounter, and specify which action to take in each situation. You use decision tables to view and manage large sets of action rules with identical conditions.

As shown in the following decision table, you can see business terms, navigation phrases, BAL operators **1**, and action phrases **2** when you edit the condition columns. You can see constants **3**, number values **4**, and text values **5** in the cells of the table.

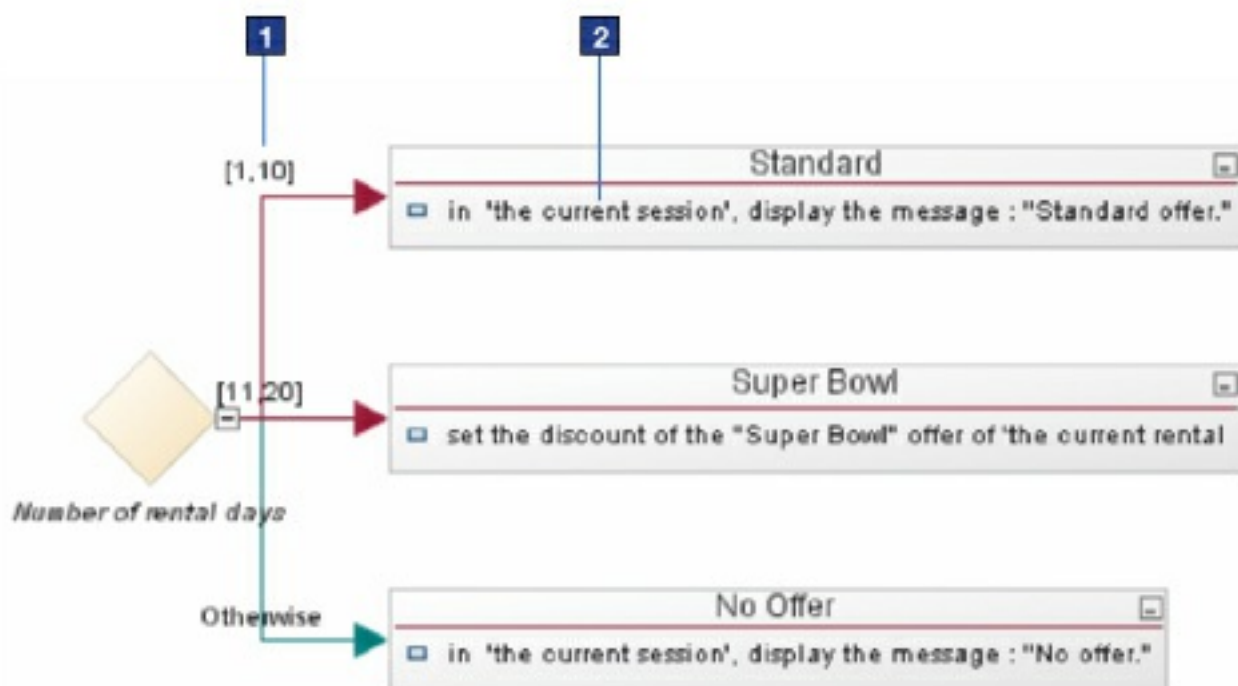
	Coverages	Car group	Surcharge	Comment
1	CDW	Economy, Compact, MidSize	\$15	Add surcharge of 15 for CDW [Economy, Compa.
2		FullSize, Luxury, SportUtility, Minivan	\$25	Add surcharge of 25 for CDW [FullSize, Luxury, .
3	LDW	Economy, Compact, MidSize	\$21	Add surcharge of 21 for LDW [Economy, Compa.
4		FullSize, Luxury, SportUtility, Minivan	\$21	Add surcharge of 21 for LDW [FullSize, SportUtili.
5	PAI		\$3	Add surcharge of 3 for PAI
6	PEI		\$2	Add surcharge of 2 for PEI
7	ALI	Economy, Compact, MidSize	\$11	Add surcharge of 11for ALI [Economy, Compact.
8		FullSize, Luxury, SportUtility, Minivan	\$12	Add surcharge of 12 for ALI [FullSize, SportUtilit..
9				

Decision trees

Decision trees graphically represent decision logic. In a decision tree, the branches represent conditions, and the leaves represent the actions to take if the conditions are true.

Decision trees provide the same function as decision tables. However, with decision trees, you can manage a large set of rules with some common conditions, but not all.

When you edit branches in a decision tree, you can see business terms, phrases, BAL operators, values, and constants. The following diagram shows values **1**, and constants and action phrases **2**:



Technical rules

You write technical rules in IRL, which is similar to Java™ code. You use technical rules to represent specific loops in the action part of your rules.

In the following example, the technical rules contain IRL keywords **1**, BOM elements in their code form **2**, values **3**, and constants **4**.

```

1
when {
    financialEvent: FinancialEvent(type equals 2 EventType.ACKNOWLEDGEMENT);
}
1
then {
    dispatcher.send(financialEvent, "Global Ledger");
    financialEvent.processedDate = Helper.getCurrentDate();
    3 update financialEvent;
}

```

Simplifying rule editing for business users

When you design a rule project in Rule Designer, set up management and rule authoring environments that meet the needs of your business users. You want to avoid differences between the system architecture and the needs of business users.

For example, rule packages can reflect decision points in your application, but not the geography to which the business rules apply. The business rule vocabulary is ultimately implemented as an object model. The constraints that are involved in developing an object model can result in a verbose vocabulary. For example, your model might contain inheritance relationships or utility classes that business users cannot understand. You can hide these classes in the business object model.

Filtering the vocabulary with categories

When you have a large vocabulary, you can use categories to filter the terms and phrases that are available to you when you edit rules in Rule Designer or Decision Center. When a category is assigned to a business rule, only the vocabulary terms and phrases of that category are visible in the business rule.

Adding custom properties to rule artifacts

To add properties to existing types of rule artifacts, you must extend the rule model. You can set up BAL and rule model extensions, and then deploy them to both Rule Designer and Decision Center. For example, to identify the country for which a rule is applicable, you can create a property that is called geography.

Business users can use custom properties to manage business rules. For example, they can use queries in Decision Center to set up views that are based on a property. With these views, business users have a different view of the rule project that is compared to the physical rule package organization that you set up in Rule Designer.

Parent topic: [Authoring business rules](#)

Related concepts:[Action rules](#)[Technical rules](#)[\(Deprecated\) Decision tree overview](#)**Related tasks:**[Applying verbalization changes to business rules](#)[Applying a category filter](#)[Creating an action rule type](#)**Related information:**[Decision tables](#)[Rule project item naming conventions](#)[Working with action rules](#)[Working with decision tables](#)[\(Deprecated\) Working with decision trees](#)[Working with technical rules](#)[Rule languages](#)

Working with action rules

You can create action rules by using the Intellirule editor or the guided editor.

Action rules

You express business policies in rules that match actions to conditions.

Action rule editing errors and warnings

Rule Designer searches for problems when you edit an action rule. Errors and warnings are displayed in the Problems view.

Creating an action rule

When you create a new action rule, you specify the rule project to which it belongs. You can also store the rule in a specific package.

Selecting an editing mode for your action rules

You can choose a predefined editing profile or define a custom profile that best suits your preferences for editing action rules.

Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

Building action rules using the guided editor

When you use the guided editor to build rules, you construct predefined rule fragments by entering text or numbers in fields or by selecting fragments from drop-down lists.

Creating an action rule type

When you create a new rule model class to define a new type of action rule, the new rule type becomes available in the creation wizards.

Parent topic: [Authoring business rules](#)

Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
    the credit score of 'the borrower' is less than 200
then
    in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
    set applicant to a customer
        where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

Rule variables

You create a rule variable to define the scope of a rule.

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

Rule actions

Rule actions define what to do when the if part of the rule is true or false.

Parent topic: [Working with action rules](#)

Related concepts:

[Technical rules](#)
[\(Deprecated\) Decision tree overview](#)

Related tasks:

[Creating an action rule](#)
[Creating an action rule template](#)
[Defining a folder structure for rule project items](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Overview: Ways to express business rules](#)
[Decision tables](#)
[Building rules using the Intellirule editor](#)
[Building action rules using the guided editor](#)

Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
    the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
    set 'the cart' to the shopping cart of customer;
if
    the value of 'the cart' is less than $100
then...
```

One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
    set Smith to a customer;
if
    the category of Smith is Gold
then
    apply 10 % discount to the shopping cart of Smith;
```

When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
()	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation

marks:

<div>R u l e d i t o r</div>	<div></div> <div>Description</div>
<div>In t e l l i r u l e e d i t o r</div>	<div>If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.</div>
<div>G u i d e d e d i t o r</div>	<div>When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.</div>

Types of rule variables

You can assign different types of values to your rule variables.

Parent topic: [Action rules](#)

Related concepts:

[Dependency of rule actions on rule variables](#)

[Rule actions](#)

Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, customer). Once you set a variable, you can use it in any part of the rule that declares the variable.

Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and then parts of the rule use the same value:

```
definitions
    set maxAmount to 1000000;
if
    the amount of 'the loan' is at least maxAmount
then
    in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

Restrictions on rule variables

You can further restrict a variable in the definitions part of a rule by using the operator `where`.

Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
    set 'loyal customer' to a customer
        where the category of this customer is Gold;
if
    the value of the shopping cart of 'loyal customer' is more than $200
then
    apply the super discount;
```

Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
    set 'senior Gold customer' to a customer
        where all the following conditions are true:
            - the category of this customer is Gold
            - the age of this customer is at least 65;
```

Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the definitions part of the rule, for example:

```
definitions
    set applicant to a customer;
    set 'loyal customer' to a customer;
if
    all of the following conditions are true:
        - applicant is married to 'loyal customer'
```

```
        - 'loyal customer' is insured
then
    upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
    'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
    set 'customer 1' to a customer;
    set 'customer 2' to a customer;
if
    'customer 1' is married to 'customer 2'
    and 'customer 2' is insured
then
    upgrade 'customer 1''s rating;
```

Note: When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
    set 'gold customers' to all customers
        where the category of this customer is gold;
    set 'junior gold customer' to a customer in 'gold customers'
        where the age of this customer is at most 15;
    set 'senior gold customer' to a customer in 'gold customers'
        where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

Parent topic: [Rule variables](#)

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
    the customer category is Gold
then
    redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

Combinations of conditions

You can apply conditions to groups, and test nested groups.

Condition negation

You can set a rule to perform an action when a condition is not true.

Parent topic: [Action rules](#)

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
    the customer's maintenance number starts with "TX"
then
    redirect the call to call center A;
```

is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
    the purchase value of the shopping cart is more than $100
then
    apply a 10% discount to the value of the shopping cart
```

after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
    the return date of 'rented car' is after 'pickup date' and before
    'scheduled return date'
then...
```

Parent topic: [Rule conditions](#)

Related concepts:

[Conditions that test for existence](#)

[Combinations of conditions](#)

Related reference:

[BAL operators](#)

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
  there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
  there are 10 customers where the category of each customer is Gold,
then...
```

there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
  there are at most 3 customers
    where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
  set 'silver customers' to all customers
    where the category of each customer is Silver;
  set 'silver count' to the number of 'silver customers'
if
  there are at least ('silver count' + 1) customers
    where the category of each customer is Gold,
then...
```

there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:

```
if
    there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
    set 'gold customers' to all customers
    where the category of each customer is Gold;
if
    there is at least one customer in 'gold customers'
    where the age of this customer is at least 65,
then...
```

Parent topic: [Rule conditions](#)

Related concepts:

[Combinations of conditions](#)

[Conditions that compare business terms and values](#)

Related reference:

[BAL operators](#)

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
    the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
    set 'luxury car groups' to all car groups where the daily rate of each
    car group is at least 50;
if
    'luxury car groups' contain the car group of 'the current rental
    agreement'
```

Parent topic: [Rule conditions](#)

Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting  
longer than 5 minutes
```

Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if  
    the customer is older than 60  
    or the customer is younger than 21  
    and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if  
    the customer is older than 60  
    or (the customer is younger than 21 and the category of the customer  
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if  
    (the customer is older than 60 or the customer is younger than 21)  
    and the category of the customer is Student
```

Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true:` This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true:` This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if  
    all of the following conditions are true:  
        - the category of the customer is Gold  
        - a member of the Gold team is available  
then  
    redirect the call to a member of the Gold team;
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
    any of the following conditions is true:
        - the category of the customer is Gold
        - the customer has been waiting longer than 5 minutes
then
    redirect the call to a member of the Gold team;
```

Parent topic: [Rule conditions](#)

Related concepts:

[Conditions that test for existence](#)

[Conditions that compare business terms and values](#)

Related reference:

[BAL operators](#)

Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

it is not true that

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
    the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
    it is not true that the category of the customer is Gold
then...
```

none of the following conditions are true

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
    all the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
    none of the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

Parent topic: [Rule conditions](#)

Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
    the value of the shopping cart is more than $100
then
    apply a 15% discount on the shopping cart;
```

Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
    'the loan report' is approved
    and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
    in 'the loan report', accept the loan with the message
    "Congratulations! Your loan has been approved";
else
    in 'the loan report', refuse the loan with the message "We are sorry.
    Your loan has not been approved";
```

Example of an action that calls methods

An action statement can invoke a method on a business term. You perform the method by using an action phrase from the rule vocabulary. The method can be a static method or a member of a class.

The following action statement uses `apply` to call the `applyDiscount` method, and `display` to call the `displayMessage` method.

```
then
    apply a 10% discount;
    display the message: "You received a discount!";
```

[Rule actions for lists of business terms](#)

You can define actions that a rule executes for each item in a list.

[Dependency of rule actions on rule variables](#)

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

Parent topic: [Action rules](#)

Related concepts:

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
    for each item in expensive items:
        - apply 5% discount to this item;
        - display the message: "A 5% discount has been applied";
```

Parent topic: [Rule actions](#)

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
    all of the following conditions are true:
        the value of the customer's shopping cart is more than $100
        the category of the customer is Gold
then
    apply a 15% discount
else
    apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
    set applicant to a customer
        where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Parent topic: [Rule actions](#)

Related concepts:

[Rule variables](#)

[Rule actions](#)

Action rule editing errors and warnings

Rule Designer searches for problems when you edit an action rule. Errors and warnings are displayed in the Problems view.

The types of errors and warnings that Rule Designer reports when you edit an action rule, where the error or warning is reported, and the remedial steps you can take are described in the following sections.

Lexical errors

Lexical errors occur when a word in the rule is not recognized and are reported when you type a rule in the Intellirule editor.

Modify the text of the rule to follow the syntax specified by the language grammar and by the vocabulary defined in the application. Error messages might help, but not all times. Code completion and vocabulary use might be used to learn the exact syntax of phrase of the vocabulary.

If you have lexical errors in your rules, you cannot use the guided editor.

Syntactic errors

Syntactic errors are reported when a rule statement is not well formed as soon as you type a rule in the Intellirule editor.

Modify the text of the rule to follow the syntax specified by the language grammar and by the vocabulary defined in the application. Error messages might help, but not all times. Code completion and vocabulary use might be used to learn the exact syntax of phrase of the vocabulary.

If you have syntactic errors in your rules, you cannot use the guided editor.

Semantic errors

Semantic errors are reported when the text of the rule is syntactically correct but its interpretation is wrong as soon as you type a rule in the Intellirule editor.

For semantic errors, you should consider revising the text of the rules to comply with the language semantics and with semantic rules that might have been set up (for example, BOM domains).

Consistency errors

Consistency errors are reported by the consistency checker at build time. For example, a consistency error is raised if a rule contains incompatible tests, or a test that is always false.

You can disable the consistency checker in the Rule Designer build preferences.

Ambiguities

Ambiguities can occur in the following cases:

- A part of the rule has at least two interpretations that are semantically correct. You can use parentheses, intermediate variables, or punctuation if it is available in the statement, to make the rule non-ambiguous.
- There are two identical terms or phrases in the vocabulary. Changing one of the terms or phrases in the vocabulary can remove the ambiguity.

The following tables list error messages that you can see when editing an action rule.

Table 1. Action rule editing errors quick reference

Message	Description
A ruleset parameter <parameter name> is already declared.	Ruleset Parameter Already Declared A declared variable has the same name as a ruleset parameter declared on a type.
A ruleset variable <variable name> is already declared.	Ruleset Variable Already Declared A declared variable has the same name as a ruleset variable declared on a type.
Ambiguous sentence.	Ambiguous Sentence Part of the rule has at least two interpretations that are semantically correct. Use parentheses, intermediate variables, or punctuation if available in the statement.

	There are two identical terms or phrases in the vocabulary. Change one of the terms or phrases in the vocabulary.
An automatic variable <variable name> is already declared.	Automatic Variable Already Declared A declared variable has the same name as the automatic variable declared on a type.
Can't specify else clause without if clause.	Else Without If An else part has been specified, but there is no if part in the rule.
Invalid cardinality <cardinality>, it is incompatible with <cardinality>	Invalid Cardinality The given cardinality (single or multiple) is not compatible with the one requested.
Invalid type <type name>, it is not assignable from type <type name>	Not Assignable From A given type is cannot be assigned from a requested type. For example: <ul style="list-style-type: none">the title of a CD is one of {1}The requested type to be assigned String (the title of a CD) cannot be assigned from Number (1).
Invalid value (<type name>) <value text>	Invalid Value An error on the value has been reported by a value descriptor associated to the value type (checking of value).
Invalid variable <variable name> for expected type <a type>.	Invalid Variable The type of a variable is not compatible with the requested type.
Number must be integer.	Non Integer Value An integer value must be specified.
Value is not in domain.	Value Not In Domain A BOM domain has been specified and the value used in the rule is violating this domain.
Value is out of the range	A range BOM domain has been specified and the value used in the rule is out of this range.
Only values are expected.	Invalid Expression Expressions such as sentences, variables and grammar constructions are not valid. Only value expressions are accepted.
Incompatible sentence, due to incompatible precedence	Some operators are incompatible with the types used in the sentence. For example, in the sentence print "Hello world!" +2 -3, the operators are incompatible, because "Hello world!" +2 is evaluated as a String, and the operator - is incompatible with the String type. To solve the problem, use explicit parentheses: print "Hello world!" + (2 - 3)".
Origin type: <type name> is not a super type of target type: <type name>.	Not Super Type Of A type is not a subtype of the requested type. For example: the name of the customer is 3 The requested type is String (the name of the customer) and the given type is Number (3). String is not a super type of Number.

The variable <variable name> cannot be used, a variable of that type is already declared.	Automatic Variable Bound Cannot use the automatic variable of a type, where there is a global variable of that type already declared.
The variable <variable name> is not assignable.	Variable Not Assignable The variable cannot be assigned.
The word <word> is expected in place of <word>.	Token Mutation A word in the rule is wrong, and the error recovery mechanism suggests another one.
The word <word> is missing.	Missing Token A word is missing in the rule.
The word <word> is not required.	Inserted Token A word has been inserted in the wrong place in the rule.
Unknown word: <word>.	Unknown Token A word is not recognized. This usually occurs with a malformed value.
The phrase <phrase> is not required.	Not Implemented A complete phrase is in the wrong location and must be removed to correct the rule.
The phrase <phrase> is missing.	Not Implemented A complete phrase is missing in the text. Complete the phrase to correct the rule.
No term <term> found.	Term Does Not Exist There is a reference to a term that is not in the vocabulary.
No navigation phrase <phrase> (of subject <subject>) found on term <term>.	Navigation sentence on a given term does not exist in the vocabulary.
No action phrase <phrase> (of subject <subject>) found on term <term>.	Action sentence on a given term does not exist in the vocabulary.
No automatic variable <variable> found.	The automatic variable is not declared automatic on a term, or a term does not exist in the vocabulary.
Too many errors. Cannot recover from errors.	The rule contains too many errors to correct.
Invalid value for <value>: <value>	Invalid Value Invalid value for the Guided Editor.
Invalid variable name: <variable>, the characters: <character> are not allowed.	Invalid Name/Character The variable name is invalid and contains one or more of the following invalid characters:

	' " \n \r \t / () , ;
Unknown phrase, did you mean: <phrase>?	Unknown Phrase A sentence in the text is incorrect but similar to an existing correct sentence. The message suggest the correct sentence.
Value (<value type>) <value text> is incorrect. <a message>	Value Error An error on the value has been reported by a value info associated to a role in the phrase (checking of value). If the value is used in a place where a BOM domain is defined, this error is generated if the value violates the BOM domain.
Variable <variable name> already declared.	Variable Already Declared A variable of that name is already declared.
Variable <variable name> is not declared.	Variable Not Declared A reference to a variable that it is not declared.

The following table lists the most current warnings you might see when editing an action rule.

Table 2. Action rule editing warnings quick reference

Message	Description
<an element> is deprecated	Deprecated Element A deprecated element (type, constant or phrase) has been used in the rule. Deprecated is a property defined on a business element.
Can not bind value types.	A value type such as number, string, and so on cannot be used in a binding.
Category(ies) (<categories>) defined on phrase <a phrase> does not match the rule category filter.	Invalid Phrase Category The phrase is not intended to be used on the rule, because the categories of that phrase do not match the rule category filter.
Category(ies) (<categories>) defined on term <a term> does not match the rule category filter.	Invalid Term Category The term is not intended to be used on the rule, because the categories of that term do not match the rule category filter.
Category(ies) (<categories>) defined on type of variable <variable name> does not match the rule category filter.	Invalid Variable Category The variable is not intended to be declared on the rule, because the categories of the variable type do not match the rule category filter.
Category(ies) (<categories>) defined on value <a constant> does not match the rule category filter.	Invalid Constant Category The constant is not intended to be used on the rule, because the categories of that constant do not match the rule category filter.
Predicate not applicable to value types.	Predicate Not Applicable The predicate is not applicable to the value type, such as number, string, and so on.
The construct <construct> is	

The construct <construct> is deprecated.	Construct Deprecated A deprecated grammar construct (issued from the XML schema definition of the language) has been used in the rule.
The (part of the) rule execution may be unsafe: <message>.	Rule Execution Server The rule tries to set an attribute to a value outside its domain.
The (part of the) rule is never applicable because <message>.	The rule never applies because its conditions can never be met. This is usually caused by simple logic errors in the rule.
The rule is incomplete, fill all the placeholders.	Incomplete Rule A rule has at least one empty placeholder.
Value (<value type>) <value text> is incorrect. <a message>	Value Warning A warning on the value has been reported by a value info associated to a role in the phrase (checking of value).
Variable <variable name> is not used.	Variable Not Used A variable is declared but not used.

Parent topic: [Working with action rules](#)

Related concepts:
[Consistency checking](#)
[Action rules](#)
[Technical rules](#)

Related information:
[Vocabulary errors and warnings](#)
[Working with action rules](#)
[Business Action Language \(BAL\)](#)

Creating an action rule

When you create a new action rule, you specify the rule project to which it belongs. You can also store the rule in a specific package.

Procedure

To create an action rule:

1. Select your rule project.
2. On the **File** menu, click **New** > **Action Rule**.

The New Action Rule wizard opens.

3. If you want to store the rule in a rule package, click **Browse** in the **Package** field, and then select the required package.
4. In the **Name** field, type the name of your action rule.

Note: Avoid using any of the following characters in the name: * ? " < > | \ /

5. In the **Template** field, click **Browse** if you want to use an action rule template. (Rule templates are deprecated.)
6. Click **Finish**.

The new action rule is displayed in the Rule Explorer view and the Intellirule Editor opens. This is the default editor: any time you create or open an action rule, you open the Intellirule editor.

Results

Rule Designer provides the following editors for you to edit action rules:

- **Intellirule Editor:** a text editor that enables you to build rules using a Content Assist system.
- **Guided Editor:** a point-and-click editor that enables you to build rules by entering text or numbers in fields, or by selecting items from drop-down lists.

You can choose which editor you want to use when you build a rule.

Parent topic: [Working with action rules](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Building rules using the Intellirule editor](#)

[Building action rules using the guided editor](#)

Selecting an editing mode for your action rules

You can choose a predefined editing profile or define a custom profile that best suits your preferences for editing action rules.

About this task

You can choose the appropriate editing mode for your profile when you create and edit action rules.

Procedure

To select your mode for editing action rules:

1. On the **Window** menu, click **Preferences**. (On Mac, click **Eclipse** > **Preferences**.)
2. In the Preferences dialog, click **Rule Designer** > **Action Rule Editing**.
3. On the Action Rule Editing page, select a user profile. Each profile has its own set of editing options:
 - **Guided**: Provides assistance by enabling most of the editing options automatically.
 - **Developer**: Enables some options automatically, which you can deselect.
 - **Custom**: You choose the options that you want to enable.
4. Select the editing options that you need.
5. Click **Apply**, and then click **OK**.

Parent topic: [Working with action rules](#)

Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

Using action rules to open business elements

You can use the text in an action rule or in the vocabulary browser to open a business element for editing in the BOM editor.

Parent topic: [Working with action rules](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating an action rule](#)

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Building action rules using the guided editor](#)

Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

When you create or open an action rule, the Intellirule Editor opens by default. If you are currently using the guided editor in a Designer, you can switch to the Intellirule editor: close the guided editor, right-click the action rule you want to edit in the Rule Explorer view, and then click **Open With > Intellirule Editor**.

You can add terms and phrases to a rule by typing or pasting in text, or by selecting them from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, value, or other phrase.

Note:

See the [Rule editor flash demo](#) on the IBM® Customer Support site for an English-only tour of the rule-editing features.

Terms and phrases

The rule editor provides an editing area for building rules. You can add terms and phrases in the following ways:

- Type directly in the editing area.
- Copy text from another editor or application, and paste it into the editing area.
- Select predefined terms and phrases from a completion menu.

Completion menu

The business vocabulary of your company defines the terms and phrases in the completion menu, and the ways you can combine them.

Placeholders

Placeholders indicate places in a term or phrase that you must complete. You must insert the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain more placeholders that you must also complete to construct a valid expression.

Parent topic: [Building rules using the Intellirule editor](#)

Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

About this task

An action rule can comprise some or all of the following parts. The parts must be defined in the following order: definitions, if, then, and else.

Procedure

1. Click anywhere in the editing area and press Ctrl+Spacebar. The completion menu displays a list of available insertion options that are based on your current position in the rule.
2. Select an insertion option from the completion menu.
3. Press Esc or click anywhere in the editing area to hide the completion menu.

Parent topic: [Building rules using the Intellirule editor](#)


Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Ambiguities occur when part of a rule has at least two interpretations that are semantically correct, or when there are two identical terms or phrases in the vocabulary.

For example, the following statement is ambiguous:

```
if
all of the following conditions are true :
  - the category of the customer is Gold
  - the age of the customer is at most 15
and
  the salary of the customer is more than 100
```

Rule Designer raises an ambiguity error (which appears in green ) because it cannot tell whether the salary of the customer is more than 100 is associated with the age of the customer is at most 15, or whether it is the third condition of the rule.

To remove the ambiguity, you can add a comma at the end of the the age of the customer is at most 15 condition statement:

```
if
all of the following conditions are true :
  - the category of the customer is Gold
  - the age of the customer is at most 15,
and
  the salary of the customer is more than 100
```

An alternative way to remove ambiguity is to use parentheses to group expressions.

In cases where there are two identical terms or phrases in the vocabulary, change one of the terms or phrases to remove the ambiguity.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:

[Rule variables](#)

[Condition negation](#)

[Action rules](#)

Related tasks:

[Creating rule parts](#)

[Correcting errors by using Quick Fix](#)

Related information:

[Action rule editing errors and warnings](#)

[Building rules using the Intellirule editor](#)

[Business Action Language \(BAL\)](#)

[Using action rules to open business elements](#)

Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

Procedure

1. In the toolbar above the editing area, click **Completion Menu Options**. A window opens.
2. Select the check boxes for the options you want to set.

Table 1. Completion menu options

Option	Description
Enable on Spacebar	<p>Select to open the completion menu whenever you press Spacebar while you are typing in the editing area.</p> <p>Clear this check box if you want the completion menu to open only when you press Ctrl+Spacebar.</p>
Enable on double-click	<p>Select to open the completion menu by double-clicking a word in the editing area.</p> <p>Clear this check box if you want double-clicking to select the current word instead.</p>
Enable auto-restart	<p>Select to automatically reopen the completion menu when a term or phrase is selected.</p> <p>Clear this check box if you want to open the completion menu by pressing Ctrl+Spacebar.</p>
Enable smart mode	<p>Select to filter the completion menu entries in a smart way to reduce the number of entries.</p>
Enable template mode	<p>Select if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable this mode if you want to insert longer phrases, and then substitute placeholders.</p> <p>Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate this mode if you prefer to build a complete rule in the same way that you might compose an email message.</p>
Use Hierarchical View	<p>Select if you want the completion menu to group terms and phrases by type, for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.</p> <p>Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.</p>
Filter unreachable phrases	<p>Select if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable.</p>
Display toolbar	<p>Select if you want the completion menu toolbar to be displayed above the list of available terms and phrases.</p>

	Clear this check box if you want to hide the completion menu toolbar.
Display documentation	<p>Select if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.</p> <p>Clear this check box if you want to hide the hover help.</p>

3. Click outside the pop-up window to save your settings.

Parent topic: [Building rules using the Intellirule editor](#)

Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

About this task

Correct errors manually in rule and business model files. You can use the Problems view to view the error messages and navigate to the errors in your files.

Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
 - Hover the mouse over a highlighted error in the rule editor to display an error tip icon.
 - Double-click the text that is displayed in the error description in the Problems view to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

Parent topic: [Building rules using the Intellirule editor](#)

Using action rules to open business elements

You can use the text in an action rule or in the vocabulary browser to open a business element for editing in the BOM editor.

Editing a business element from the text of a rule

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements from the text of the rule.

Editing a business element from the Vocabulary Browser

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements by using a Vocabulary Browser, which shows the vocabulary elements available in the current rule.

Parent topic: [Building rules using the Intellirule editor](#)

Editing a business element from the text of a rule

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements from the text of the rule.

Procedure

To edit a business element from the text of a rule:

1. In the Intellirule editor, press Ctrl.
The terms and phrases used in the rule are displayed as links.
2. Click the term or phrase corresponding to the business element you want to edit.



The BOM Editor opens on the selected business element.

3. Make the required changes, and then from the **File** menu, click **Save**.

Results

Business elements can be edited in the Vocabulary Browser.

Parent topic: [Using action rules to open business elements](#)

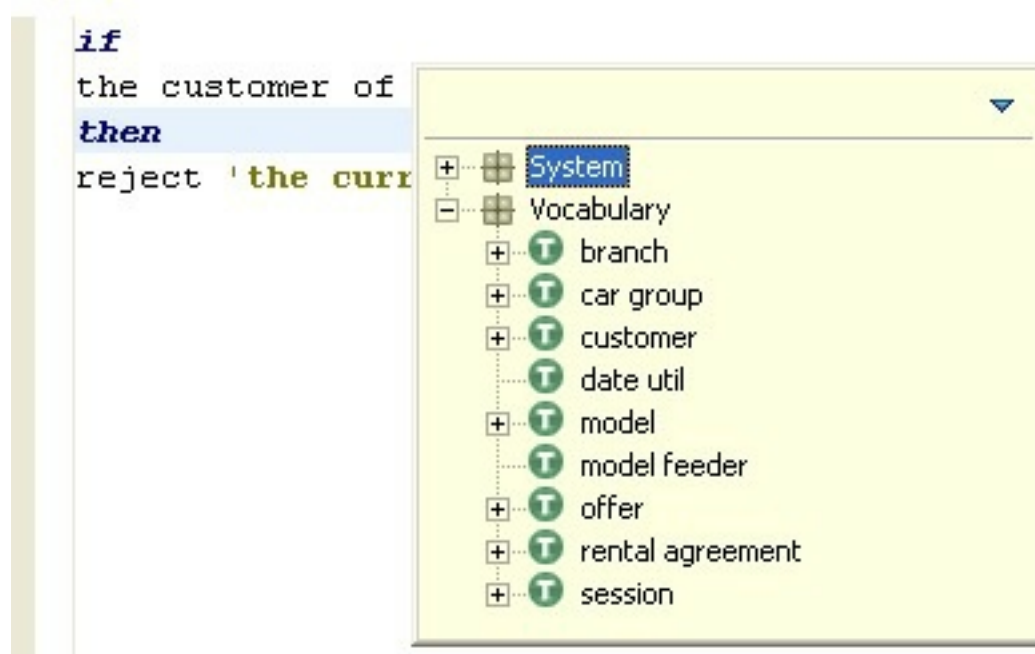
Editing a business element from the Vocabulary Browser

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements by using a Vocabulary Browser, which shows the vocabulary elements available in the current rule.

Procedure

To edit a business element from the Vocabulary Browser:

1. In the Intellirule editor, press Ctrl+B.
2. In the Vocabulary Browser, either type the first letters of the business element you want to access, or browse to it.



3. Double-click the business element.

The BOM Editor opens on the selected business element.

4. Make the required changes, and then from the **File** menu, click **Save**.

Parent topic: [Using action rules to open business elements](#)

Building action rules using the guided editor

When you use the guided editor to build rules, you construct predefined rule fragments by entering text or numbers in fields or by selecting fragments from drop-down lists.

[Switching to the guided editor](#)

When you create or open an action rule, the Intellirule editor opens by default. If you want to work in a more guided environment, you can switch to the guided editor.

[Controls in the guided editor](#)

The guided editor displays controls that you can click to build rule statements.

[Creating rule parts](#)

You can create action rule definitions, conditions, and actions.

[Adding or removing rule statements](#)

You can add or remove statements in any part of an action rule.

[Defining a variable](#)

You define variables in the definitions part of an action rule.

[Choosing whether to enter values or select items](#)

You can build fragments by entering values in fields or by selecting items from a drop-down list.

[Inserting an arithmetic expression](#)

Use the arithmetic operator icon to insert arithmetic expressions.

[Adding a statement to a group of statements](#)

Use the hyphen icon to add a statement to a group of statements.

[Controlling how condition statements are combined](#)

Use logical operators to control the way condition statements are combined.

[Changing the grouping order of condition statements](#)

Use opening and closing parentheses to control the grouping order of condition statements.

[Negating a condition statement](#)

Use the Negate command to negate the sense of a condition statement.

Parent topic: [Working with action rules](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating an action rule](#)

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Building rules using the Intellirule editor](#)

Switching to the guided editor

When you create or open an action rule, the Intellirule editor opens by default. If you want to work in a more guided environment, you can switch to the guided editor.

Procedure

To switch from the Intellirule editor to the guided editor:

1. Save any changes that you made to your rule and close the Intellirule editor.
2. In the Rule Explorer view, right-click the action rule that you want to edit, and then click **Open With > Guided Editor**.



The guided editor opens.

Parent topic: [Building action rules using the guided editor](#)

Controls in the guided editor

The guided editor displays controls that you can click to build rule statements.

The following table displays the controls that you can click to add fragments and build rule statements.

Control	Description
[definitions], [if], [else]	Add a new rule part. See Creating rule parts .
	Add a definition, condition, or action statement. See Adding or removing rule statements .
	Select a value from a drop-down list. See Choosing whether to enter values or select items .
[#]	Insert arithmetic operators into arithmetic expressions. See Inserting an arithmetic expression .
[]	Add a statement to a group of statements. See Adding a statement to a group of statements .
and, or	Control how condition statements are combined. See Controlling how condition statements are combined .
[where]	Add restrictions to definition or condition statements. See where <test> .
[from/in]	Use from to specify the origin of objects of a one-to-many relation used in a definition statement (see from <object>). Use in to specify the source of objects used in a definition and condition statements (see in <list>).

Parent topic: [Building action rules using the guided editor](#)

Related tasks:
[Freezing and unfreezing parts of an action rule template](#)

Creating rule parts

You can create action rule definitions, conditions, and actions.

Overview

An action rule can include some or all of the following parts. The parts must be defined in the order listed.

- definitions
- if
- then
- else

Square brackets [] around the title of a part of a rule means that it is empty and optional. The then part of the rule is not shown between brackets because it is a required part of an action rule.

Building an optional part of a rule

To build an optional part of a rule:

1. Click the title between brackets. The part expands to display the placeholders and fragments that you can use.
2. Click placeholders and select the elements that you require to build the rule.

Building the mandatory then part of a rule

To build the mandatory then part of a rule:

1. Click <select an action>.
2. Select the required action.

The following example shows a complete rule:

```
definitions
  set minimum score to ▼ the number: 200 [4]
  [where]
  ↩
if
  the credit score of the borrower [4] is less than minimum score [4]
  ↩
then
  in the loan report , refuse the loan with the message ▼ "Credit score below" [4] + minimum score [4] [4]
  ↩
[else]
```

Parent topic: [Building action rules using the guided editor](#)

Related information:


[Building action rules using the guided editor](#)
[Business Action Language \(BAL\)](#)

Adding or removing rule statements

You can add or remove statements in any part of an action rule.

Procedure

You can do this as follows:

1. To add a statement after the last existing statement, click the blue arrow icon  below the last existing statement.
2. To add a statement before an existing statement, right-click the first building block of an existing statement and click **Insert Definition**, **Insert Condition**, or **Insert Action**.
3. To delete a statement:
 - a. Select the statement or statements you want to delete.
 - b. Right-click the first building block of the statement and click **Delete Definition**, **Delete Condition**, or **Delete Action**.

Parent topic: [Building action rules using the guided editor](#)

Defining a variable

You define variables in the definitions part of an action rule.

Procedure

To define a variable:

1. Open the definitions part of the rule.
2. Click the **variable1** field and type a name for the variable.
3. Select an item from the **<select a choice >** drop-down list. You can choose whether you want to bind the variable to:
 - a sentence from the vocabulary
 - **an <object >** : an instance of an object. If you choose **an <object >** , the placeholder **<select an object >** is shown. Click the placeholder and select the type of object you want to bind.
 - **all <objects >** : all instances of the same type. If you choose **all <objects >** , the placeholder **<select objects >** is shown. Click the placeholder and select the type of object you want to bind.

Results

Your variable is now defined. You can use it in all parts of the action rule.

Parent topic: [Building action rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Adding or removing rule statements](#)

[Inserting an arithmetic expression](#)

[Adding a statement to a group of statements](#)

[Controlling how condition statements are combined](#)

[Negating a condition statement](#)

Related information:

[Changing the grouping order of condition statements](#)

[Building action rules using the guided editor](#)

[Business Action Language \(BAL\)](#)

[Choosing whether to enter values or select items](#)

[Creating rule parts](#)


Choosing whether to enter values or select items

You can build fragments by entering values in fields or by selecting items from a drop-down list.

Selecting items from a drop-down list

By default, whenever a value can be typed, a placeholder **<enter a string>** is displayed with a black arrow next to it.

Follow these steps to select from a drop-down list:

1. Click the black arrow icon  next to the placeholder to display the list of options.
2. Select an item from the drop-down list.

Entering values in fields

Follow these steps to change from a drop-down list back to a field:

1. Click the building block and select **<enter a string>** from the drop-down list. A field is inserted instead of the building block at that location in your rule statement.
2. Click the field and enter a value.

Parent topic: [Building action rules using the guided editor](#)

Inserting an arithmetic expression


Use the arithmetic operator icon to insert arithmetic expressions.

About this task

You can build complete arithmetic expressions.

Procedure

To insert an arithmetic operator:

1. Click the arithmetic operator icon  in a rule statement. The following expression are shown next to the first statement, preceded by the addition operator [+]:

+ <enter a string>

2. If you want to change the addition operator to another operator, click the addition operator and select the one you want from the drop-down list.
3. To complete the arithmetic operation, click **<enter a string>** and choose whether you want to enter a string or select an item from the drop-down list (click the arrow next to it to display the list).

Parent topic: [Building action rules using the guided editor](#)

Adding a statement to a group of statements

Use the hyphen icon to add a statement to a group of statements.


About this task

You can create groups of condition statements and action statements by using the following building blocks:

- Condition statements:
 - all of the following conditions are true
 - any of the following conditions is true
 - none of the following conditions are true
- Action statements:
 - for each

You can then add or remove statements to or from a group.

Procedure

1. To add a statement to a group of statements:
 - a. Click the hyphen icon . A new statement is added to the group of statements.
2. To remove a statement from a group of statements:
 - a. Right-click the hyphen icon next to the statement, and then click **Delete Definition**, **Delete Condition**, or **Delete Action**.

Parent topic: [Building action rules using the guided editor](#)

Controlling how condition statements are combined

Use logical operators to control the way condition statements are combined.

About this task

When there is more than one condition statement in the definitions or if part of a rule, you can control whether the rule conditions are met if all (represented by the logical operator and) or any (represented by the logical operator or) of its condition statements are valid.

Procedure

To change how condition statements are combined:

1. Click the **and** or the **or** building block.
2. Select the appropriate logical operator from the drop-down list.

Parent topic: [Building action rules using the guided editor](#)

Changing the grouping order of condition statements

Use opening and closing parentheses to control the grouping order of condition statements.

By default, the and logical operator takes precedence over the or logical operator when you have several condition statements.

However, you can use parentheses to group condition statements to change the order in which they are processed.

Action	Procedure
Add an opening parenthesis to a condition statement	Right-click the first building block of the statement and click ... (.
Add a closing parenthesis to a condition statement	Right-click the last building block of the statement, or the next operator, and click)
Remove a parenthesis	Right-click the parenthesis and click Delete .
Insert both parentheses in one step	Select the entire phrase, right-click it, and then click (...) .

Parent topic: [Building action rules using the guided editor](#)

Negating a condition statement

Use the Negate command to negate the sense of a condition statement.

About this task

You can negate the sense of a condition statement by adding `it is not true that` at the beginning of the statement.

Procedure

1. Right-click the first building block of the statement.
2. Click **Negate**.

Note:

To remove `it is not true that` from the statement, right-click it and select **Delete**.

Parent topic: [Building action rules using the guided editor](#)

Creating an action rule type

When you create a new rule model class to define a new type of action rule, the new rule type becomes available in the creation wizards.

Procedure

To create an action rule of type MyRule:

1. Select your rule project and on the **File** menu click **New** > **Action Rule**.
2. In the New Action Rule wizard, type the name of your action rule in the **Name** field.
3. In the **Type** field, select the custom rule type, in this example, **MyRule**.
4. Click **Finish**.

The new action rule is shown in the Rule Explorer view and the Intellirule editor opens.

Parent topic: [Working with action rules](#)

Working with decision tables

You use the decision table editor to create and update decision tables.

[Decision tables](#)

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

[Creating a decision table from scratch](#)

Decision tables represent in tabular form all possible situations that a business decision might encounter.

[\(Deprecated\) Creating a decision table from a template](#)

You can create decision tables using a decision table template.

[Defining columns](#)

You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.

[Adding and removing columns](#)

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.

[Sorting and moving columns](#)

You can sort condition columns and move action columns.

[Adding and populating rows](#)

You can add blank rows, partially completed rows, and otherwise rows to a decision table. You can also move, split, merge and delete rows, and populate them with values from an enumerated domain.

[Specifying values and operators](#)

You can specify values and operators in condition statements.

[Disabling action cells](#)

You can disable an action cell by clicking an icon on the menu bar or by using the Enable / Disable Action command.

[Populating a decision table from an Excel spreadsheet](#)

You can copy the contents of an Excel spreadsheet into a decision table provided the structure of the Excel spreadsheet and the decision table are the same.

[Defining preconditions](#)

You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.

[Formatting a decision table](#)

You can define format settings for a table, a column, or individual cells.

[Defining checking options](#)

You can specify tests that Rule Designer conducts to identify problems with a decision table.

[Value test conditions](#)

When you construct conditions that test values, the way you complete the associated input fields depends on the type of test.

[Using decision table locking facilities](#)

You can apply a variety of locks to protect your decision tables against editing by others.

[Viewing rule details](#)

You can view the details of a rule in a decision table row by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision table editor.

[Viewing the decision table as a spreadsheet](#)

You can display a decision table in a spreadsheet view if the decision table has two condition columns and one action column and the structure of the decision table is symmetric.

[Documenting a decision table](#)

You can document each row in a decision table.

Parent topic: [Authoring business rules](#)

Decision tables

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥600,000		true	0.005
5	B	<100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		≥600,000		true	0.0075

When an application calls a decision table, the action rules are executed. If the conditions in a row are met, the rule that is formed by the row performs the actions in the row.

You can add rows to the decision table and enter values in their cells to create new rules. You can also define preconditions that apply to all the rules in a table. Error markers help you find overlaps and gaps in your rules.

Columns

Each column in a decision table represents a condition or an action.

Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

Preconditions

You can pretest data before using it with a decision table.

Decision table errors and warnings

Rule Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.

Parent topic: [Working with decision tables](#)

Related concepts:

[Action rules](#)

[Technical rules](#)

[\(Deprecated\) Decision tree overview](#)

Related information:

[Overview: Ways to express business rules](#)

Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. The top cell of each column identifies the object of a condition or the target of an action.

Row	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001

Each numbered row in the table forms a rule. The rule performs the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of the loan is between 100000 and 300000
then
  set the Insurance required to true
  set the Insurance rate to 0.001
```

Condition operators

You can split a condition across columns in a decision table when a rule statement contains more than one value. For example, the following condition requires you to specify values for <min> and <max>:

```
if
  the age of the customer is between <min> and <max>
```

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [Decision tables](#)

Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

Note: If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In Rule Designer, you group cells by merging them.

In the following table, the Grade A cells of rows 1 and 2 are grouped into one cell. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 100000 and 300000
then
  - set the insurance required to true
  - set the loan rate to 0.001
```

- Rule 2

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row that has the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3		300,000	600,000	true	0.003

The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.005
5	B	< 100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		600,000	800,000		
9		≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003
3	A			12	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.004

Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003
3		Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3		Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 100000 and 300000
then
  - set the insurance required to true
  - set the loan rate to 0.001
```

- Rule 2

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.002
```

- Rule 3

```
if
  all of the following conditions are true:
    - the loan grade is A
    - it is not true that the amount of loan is between 300000 and
600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

Parent topic: [Decision tables](#)

Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
  set 'wealthy customer' to a customer
    where the average monthly balance of this customer
      is more than $1 000 000
if
  the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

Parent topic: [Decision tables](#)

Decision table errors and warnings

Rule Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.

As you submit values to the cells of your decision table, Rule Designer checks the column automatically for errors. As shown in the following table, when there is a problem, the column header displays a warning symbol **1** and the affected cells are highlighted:

- A red line for an error **2**
- A yellow line for a warning **3**

State	Age of the customer ⚠ 1	
	<min>	<max>
Rhode Island	18	22
	21	25

Visual cues: A red line under 'Rhode Island' is labeled **2**. A yellow line between the two rows of the 'Age of the customer' column is labeled **3**.

The errors and warnings are also listed in the Problems view.

You can define different checking properties for different condition columns.

Overlap warnings

Rule Designer can verify that the values you enter for a given condition throughout the different rows do not overlap or are not identical.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 29 and 40

In this example, customers that are 29 years old satisfy the condition of both rows. Rule Designer reports this as an overlap warning.

Rule Designer checks for overlaps for all the entries in a column, and within partitioned groups of cells.

The overlap checker supports the following types:

- Number (discrete and continuous): int, long, float and double.
- The various types of Date defined in the Boot BOM: Date, SimpleDate, UniversalDate, Time, UniversalTime, DayOfWeek, Month, and Year.

The extended BAL also supports:

- Percentage (ilog.rules.brl.Percentage)
- Currency (ilog.rules.brl.Currency)

The following operators are supported for all types of object: is, is not, is one of, is not one of.

Hierarchical overlap warnings

Rule Designer checks for hierarchical overlaps (that is, overlaps involving more than one column) and distinguishes between real overlaps (duplications in multiple columns) and local overlaps (duplications of a value in one column).

In the following example, Rule Designer identifies a real overlap because the adjoining cells in the Age and Category columns contain identical values:

Age	Category
18	Gold
18	Gold

For real overlaps, Rule Designer does the following:

- Displays a warning triangle in the Age column header to highlight the duplication of the value 18.
- Displays a wavy yellow line on the side of each row.
- Adds a warning in the Problems view.

In the following example, Rule Designer identifies a local overlap because although the adjoining cells in the Age column are identical, the Category column contains distinct values (Gold in row 1 and Silver in row 2).

Age	Category
18	Gold
18	Silver

For local overlaps, Rule Designer does the following:

- Displays a warning triangle in the Age column header to highlight the duplication of the value 18.
- Displays a wavy blue line on the side of each row.
- Does not add a warning in the Problems view.

Gap warnings

Rule Designer can verify that the cells in a condition column consider all possible cases, which helps you make sure there are no gaps in your tables.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 32 and 40

In this example, customers that are 31 years old are never be taken into account. Rule Designer reports this as a gap warning.

Rule Designer evaluates gaps for all the entries in a column, and within partitioned groups of cells.

The gap checker supports the following types:

- Number (discrete and continuous): int, long, float and double.
- Types that describe concepts. For example, type Category, which might have items such as Silver, Gold and Platinum.
- The various types of Date defined in the Boot BOM: Date, SimpleDate, UniversalDate, Time, UniversalTime, DayOfWeek, Month, and Year.

There are two modes for numbers: interval and infinity.

- Interval checks for gaps from the smallest value to the largest value.
- Infinity checks for gaps from minus infinity to plus infinity.

Gap and overlap warning limitations

The detection of gaps and overlaps in decision tables is too complex to do under all circumstances. Rule Designer consequently applies limits in the following areas:

- Complex column expressions
- Hierarchical overlaps

Complex column expressions: Rule Designer only detects gaps and overlaps for columns with simple expressions. If an expression is too complex (for example, expressions that involve operators), no gaps or overlaps are computed.

The following expressions are considered to be simple:

- `the amount of the loan is <a number>`
- `myParam is <a number>`

where myParam is a ruleset parameter or variable.

- `var is <a number>`

where var is a variable defined as a precondition and assigned a simple expression.

The following expressions are considered to be too complex for gap and overlap calculations:

- `the amount of the loan + 1,000 is <a number>`
- `var is <a number>`

where var is a variable defined as a precondition and assigned a complex expression.

Hierarchical overlaps: Rule Designer checks for hierarchical overlaps on a best-effort basis to avoid the high computational cost of detecting all possibilities. Hierarchical overlaps might not be computed accurately when

conditions are added or removed; that is, when structural changes occur on the partition where a hierarchical overlap is computed.

Symmetry information

Rule Designer can verify that all partitions use the same set of items. You can check symmetry for the whole table or for specified columns. Such checking is useful to make sure that all choices are covered at the same level in the decision table.

When enabled, symmetry checking is just given as information, allowing you to create non symmetric tables.

Parent topic: [Decision tables](#)

Related information:

[Vocabulary errors and warnings](#)

[Business Action Language \(BAL\)](#)

Creating a decision table from scratch

Decision tables represent in tabular form all possible situations that a business decision might encounter.

About this task

Decision tables comprise rows and columns, and each row corresponds to a rule. They are used to represent in tabular form all possible situations that a business decision might encounter, and to specify which action to take in each of these situations.

Procedure

To create a decision table from scratch:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Table**.

The New Decision Table wizard opens.

3. Optional: Add the decision table to an existing rule package:
 - In the **Package** field, click **Browse** and click the required rule package.
4. In the **Name** field, type the name of the decision table.
5. Click **Finish**.

The new decision table is displayed in the Rule Explorer view and the decision table editor opens.

The default decision table has three condition columns and one action column. Action columns have a shaded background.

Results

You can now use the decision table editor to define the condition and action columns for the decision table, and to specify values for each row, that is, for each rule.

Parent topic: [Working with decision tables](#)

Related tasks:

[Correcting errors by using Quick Fix](#)
[Setting up a decision table template](#)
[Populating a decision table from an Excel spreadsheet](#)
[Defining preconditions](#)
[Formatting a decision table](#)
[Defining checking options](#)
[Using decision table locking facilities](#)
[Viewing rule details](#)
[Viewing the decision table as a spreadsheet](#)
[Documenting a decision table](#)
[\(Deprecated\) Creating a decision table from a template](#)
[Defining columns](#)

Related information:

[Decision tables](#)
[Business Action Language \(BAL\)](#)
[Adding and populating rows](#)
[Specifying values and operators](#)

(Deprecated) Creating a decision table from a template

You can create decision tables using a decision table template.

About this task

If you have a number of sets of rules that are similar, rather than creating a new decision table each time, you can use a decision table template.

Procedure

To create a decision table from a template:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Table**.

The New Decision Table wizard opens.

3. Optional: Add the decision table to an existing rule package:
 - In the **Package** field, click **Browse** and then click the required rule package.
4. In the **Template** field, click **Browse**. The Select a template dialog opens.
5. Click the template you want to use and then click **OK**.
6. In the **Name** field, type the name of the decision table.
7. Click **Finish**.

Results

The new decision table is displayed in the Rule Explorer view and the decision table editor opens. You can now use the Decision Table Editor to define the condition and action columns for the decision table and to specify values for each row; that is, for each rule.

Parent topic: [Working with decision tables](#)

Defining columns

You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.

About this task

A decision table must have at least one condition column and one action column. You can define condition and action columns using the decision table editor.

In a condition column, you enter a condition statement that defines a comparison but not its values.

A condition statement is an incomplete BAL predicate expression whose placeholders are mapped to specific subcolumns. Placeholders are specific tokens enclosed within < and >. They can represent a concept from the vocabulary (for example, <an object>, <a loan>, or <a string>), or empty text defined in the vocabulary phrase, such as is between <min> and <max>.

If you create a condition statement that has more than one placeholder, the required subcolumns are automatically created. If the expression does not contain any placeholders, Rule Designer completes the expression with an ... is a <boolean> statement so that it has a placeholder.

Note: An expression can have placeholders anywhere. However, any ambiguity about the expected definition might trigger a warning.

In an action column, you specify each action to take in each of the specified situations.

Procedure

To define condition and action columns:

1. Use the decision table editor to define the required columns, as summarized in the following table:

To define a condition column:	To define an action column:
<div><div>a. Double-click the top cell of the condition column. The Condition Column window opens.</div><div>b. In the Title field, type the required column title.</div><div>c. In the Test field, double-click <condition> and construct your condition statement in the Column Expression Editor using the available elements from the vocabulary.</div></div>	<div><div>a. Double-click the top cell of the action column. The Action Column window opens.</div><div>b. In the Title field, type the required column title.</div><div>c. In the Action field, double-click <action> and construct your action statement in the Action Expression Editor using the available elements from the vocabulary.</div></div> <div>To define an action column in the decision table editor edit bar, click a cell in the action column you want to define, and select the required action.</div>

Note: Press Ctrl+Spacebar to open the Content Assist box when constructing your condition or action statement.

2. When you have constructed your condition or action statement, click **OK**.

Results

Your columns are now defined. If you need to redefine a column at any time, double-click the column header and change the required details.

When you have defined at least one condition column and an action column, you can begin to specify values for the placeholders in the condition and action definitions. See [Specifying values and operators](#) for details.

Parent topic: [Working with decision tables](#)

Related tasks:

- [Formatting a decision table](#)
- [Documenting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)
- [Adding and removing columns](#)
- [Sorting and moving columns](#)

Related information:

- [Decision tables](#)
- [Adding and populating rows](#)

Adding and removing columns

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.





About this task

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.

Procedure

To add or remove columns:

- 1. Use the decision table editor to add or remove the required columns, as summarized in the following table:

To add a condition column:	To add an action column:	To remove a column:
<div><div>a. Right-click a condition column next to the place you want to insert the new condition column.</div><div>b. Click  Insert Condition Column Before or  Insert Condition Column After.</div><div>A new condition column is added to the decision table.</div></div>	<div><div>a. Right-click an action column next to the place you want to insert the new action column.</div><div>b. Click  Insert Action Column Before or  Insert Action Column After.</div><div>A new action column is added to the decision table.</div></div>	<div><div>a. Right-click the column you want to remove.</div><div>b. Click Remove Condition Column or Remove Action Column.</div><div>The column and any dependent cells are removed from the decision table.</div><div>If you want to remove an action column that is visible, but all other action columns are hidden, you must first make at least one other action column visible: right-click the header of the column that you want to make visible, click Edit action column and check the Visible check box.</div></div>

Note: If you insert a column at the first position in the table, you create a new Root partition whose children are all non-empty conditions of the existing first column.

- 2. Save your changes.

Attention: You can remove or add sub columns by changing the definition of the column. When you do, there is no automatic refactoring of the data contained in the sub columns. To avoid losing data, you can copy the content of the sub columns in an excel sheet, add a column with the new definition in Rule Designer, and paste the data in the new sub columns.

Parent topic: [Working with decision tables](#)

Related tasks:

- [Formatting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)

Related information:

- [Adding and populating rows](#)

Sorting and moving columns

You can sort condition columns and move action columns.

About this task





You can sort only condition columns. You can sort an entire column or a group of related cells. Sorting a condition column impacts the order in which the rules in the corresponding rows are executed.

You can move only action columns. When you move an action column, it changes the order in which the actions are executed.

Procedure

To sort or move columns:

1. Use the decision table editor to sort or move the required columns, as summarized in the following table:

To sort a condition column:	To move an action column:
<div>a. In the decision table editor, right-click a cell in a condition column.</div> <div>b. Click  Sort Ascending or  Sort Descending.</div>	<div>a. In the decision table editor, right-click the action column you want to move.</div> <div>b. Click  Move Action Column Left or  Move Action Column Right.</div>

2. Save your changes.

Parent topic: [Working with decision tables](#)

Related tasks:

- [Formatting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)
- [Adding and removing columns](#)
- [Defining columns](#)

Related information:

- [Decision tables](#)
- [Adding and populating rows](#)

Adding and populating rows

You can add blank rows, partially completed rows, and otherwise rows to a decision table. You can also move, split, merge and delete rows, and populate them with values from an enumerated domain.

Adding a blank row

You add and populate a row for each rule you want to define.

Adding a partially completed row

You can add partially completed rows to a decision table.

Adding an otherwise row

You can add an otherwise row to a condition column for when none of the other conditions in the column are correct.

Populating rows with values from an enumerated domain

You can add new rows and populate them with values from an enumerated domain.

Adding rows that contain values

You can add rows that already contain values to a decision table to create or extend a series of rules.

Moving rows

You can move rows up or down in a decision table, using the decision table editor.

Splitting rows

You can split a row if one of its columns has two subcolumns.

Merging rows

You can merge rows in a decision table.

Deleting rows

You can delete decision table rows.

Parent topic: [Working with decision tables](#)

Related information:



[Working with decision tables](#)

Adding a blank row

You add and populate a row for each rule you want to define.

Procedure

To add a blank row to a decision table:

- 1. In the decision table editor, right-click a cell in the first column.
- 2. Click **Add**, and then click either **Insert New Row Before**  or **Insert New Row After** .

A new row is added either above or below the row containing the cell you selected. All the values for the row are empty.

	State	Age of the customer		Rental Rejected
		Minimum Age	Maximum Age	
1	New York	18	21	true
2		21	25	false
3				
4	Rhode Island	16	21	true
5		21	25	true

Parent topic: [Adding and populating rows](#)



Related information:
[Working with decision tables](#)

Adding a partially completed row

You can add partially completed rows to a decision table.

Procedure

To add a partially completed row:

- 1. In the decision table editor, right-click the condition cell immediately next to the cell containing the value or values you want to copy to the new row.
- 2. Click **Add**, and then click either **Insert New Row Before**  or **Insert New Row After** .

A new row is added either above or below the row containing the condition cell you selected. The values next to the cell you right-clicked are automatically copied into the new row.

	State	Age of the customer		Rental Rejected
		Minimum Age	Maximum Age	
1	New York	18	21	true
2		21	25	false
3				
4	Rhode Island	16	21	true
5		21	25	true

Parent topic: [Adding and populating rows](#)

Related information:
[Working with decision tables](#)

Adding an otherwise row

You can add an otherwise row to a condition column for when none of the other conditions in the column are correct.

Procedure

To add an otherwise row:

Right-click the condition cell and click **Add > Otherwise**.

A new, partially completed row is added below the row you right-clicked. The cell below the condition cell you selected contains the text otherwise.

	State
1	New York
2	
3	
4	Rhode Island
5	
6	Massachusetts
7	
8	New Hampshire
9	
10	Otherwise

The otherwise row runs when the conditions next to the otherwise statement are true, but none of the cases in the same group as the otherwise cell are true.

Results

In a condition column, you can automatically create and complete rows for all the values from an enumerated domain.

Parent topic: [Adding and populating rows](#)

Related information:
[Working with decision tables](#)

Populating rows with values from an enumerated domain


You can add new rows and populate them with values from an enumerated domain.

About this task

You can add new rows and populate them with the values provided through a BRL Value Provider or from an enumerated domain, if you have set one up. An enumerated domain is a list of predefined elements (classes assigned to a particular group), defined in the BOM.

Procedure

To add new rows and populate them with values from an enumerated domain:

1. In the decision table editor, right-click a cell in a condition column whose values are enumerated domain items.
2. Click **Add** >  **All Domain Values**.

A row is created for each value of the enumerated domain.

	length of rental (days)	pickup branch state
1	10	Massachusetts
2		New Hampshire
3		New York
4		Rhode Island

Results

Note: The decision table editor can detect missing domain values, and allows you to add them automatically using the Quick Fix function. Look for the light bulb icon  in the editor marker bar. See [Correcting errors by using Quick Fix](#).

You can use a drag-and-drop function to add a number of rows at a time and have them automatically populated with values. Rule Designer automatically calculates the content of the cells for days of the week, months, intervals, and numbers. For any values next to the selected cell, it copies the value of the cells in the first row into each of the new rows.

Parent topic: [Adding and populating rows](#)

Related information:
[Working with decision tables](#)

Adding rows that contain values

You can add rows that already contain values to a decision table to create or extend a series of rules.

Procedure

To add rows containing values:

- 1. In the decision table editor, select the first value in the series you want to create or extend.
- 2. Press Ctrl and place your cursor over the cell handle.

The cursor changes into a hand with a + symbol.

age	
number	number
15	30

- 3. Drag the hand down to select the number of rows that you want to insert.

A tooltip displays the values that Rule Designer places in the cells.

pickup branch state	age	
	number	number
New York	15	30
Rhode Island		
Massachusetts		
New Hampshire		

+ 47 - 62

- 4. Release the cursor.

The rows are inserted and populated with the automatically computed values, and any values copied from the original row.

length of rental (days)	pickup branch state	age	
		number	number
10	New York	15	30
		31	46
		47	62
	Rhode Island		

Note: If you do not want the values of the cells next to the inserted values to be automatically filled in, do not press Ctrl when clicking the cell handle. If you just drag the cell handle, values are inserted only in the selected column.

Parent topic: [Adding and populating rows](#)



Related information:
[Working with decision tables](#)

Moving rows

You can move rows up or down in a decision table, using the decision table editor.

Procedure

To move a row in a Decision Table:

1. In the decision table editor, right-click a cell in the row you want to move.
2. Click  **Move Up** or  **Move Down**.

Results

The row swaps places with the row above or below it.

Parent topic: [Adding and populating rows](#)

Related information:

[Working with decision tables](#)

Splitting rows

You can split a row if one of its columns has two subcolumns.

Procedure

To split a row:

1. Right-click a cell in the row you want to split. The chosen cell must be in a condition column that has subcolumns.

State	Age of the customer	
	Min	Max
Rhode Island	18	21
	21	25

2. Click  **Split**.

When a row made up of grouped cells is split, the values of the column are split across two rows. The other values in the row are duplicated in the new row:

State	Age of the customer	
	Min	Max
Rhode Island	18	21
	21	25

3. To specify the empty values, double-click an empty cell and enter or select the required value, as appropriate.

Parent topic: [Adding and populating rows](#)

Related information:
[Working with decision tables](#)

Merging rows

You can merge rows in a decision table.

Procedure

To merge rows:

- 1. Right-click a cell in the row you want to merge. The chosen cell must be in a condition column.

length of rental (days)	pickup branch state
10	New York
	Rhode Island
	New Hampshire

- 2. Click  **Merge**.

When you merge the rows, you place the values of the merged condition cells together. However, this action does not merge the values of the action cells in the rows.

length of rental (days)	pickup branch state
10	in "New York","Rhode Island"
	New Hampshire

Parent topic: [Adding and populating rows](#)


Related information:
[Working with decision tables](#)

Deleting rows

You can delete decision table rows.

Procedure

To delete rows:

1. In the decision table editor, right-click the row you want to delete.
2. Click  **Remove selected row(s)**.

Parent topic: [Adding and populating rows](#)

Related information:

[Working with decision tables](#)

Specifying values and operators

You can specify values and operators in condition statements.

Specifying values

You can enter values in the cell itself, or by using the edit bar.

Setting default operators for columns

When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

Parent topic: [Working with decision tables](#)

Specifying values

You can enter values in the cell itself, or by using the edit bar.

About this task

You must complete at least one condition cell for a rule before you can complete the associated action cell.

Note: When you edit a cell directly in the table, you enter values, not complex expressions. You can enter complex expressions using the edit bar located above the Decision Table.

Procedure

To specify a value in a cell:

1. In the decision table editor, double-click the required cell and specify a value using one of the following options.

To specify a value using a value editor:	To specify a value without using a value editor:
<p>If the cell type allows for the use of a dedicated value editor, a button is displayed next to the cell.</p> <ol style="list-style-type: none">a. Click the button to open the value editor.b. Enter the required value, and then press Enter. <p>Move the cursor away from the cell you are editing to validate, or press Esc to cancel</p>	<ol style="list-style-type: none">a. In the decision table editor, double-click a cell. <p>A cursor or a list of allowed values displays, depending on the type of the value you specify.</p> <ol style="list-style-type: none">b. Type or select the required value, and press Enter. <p>Move the cursor away from the cell you are editing to validate.</p>

2. Save your changes.

Parent topic: [Specifying values and operators](#)

Related tasks:
[Setting default operators for columns](#)

Related information:
[Decision tables](#)
[Business Action Language \(BAL\)](#)
[Adding and populating rows](#)
[Working with decision tables](#)

Setting default operators for columns

When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

About this task

When you define a condition column statement, you set the default operator for all the cells in that column.

For example, if you define the following condition column statement, you set the default operator to `is more than (>)`:

the age of the customer is more than <enter a string>



The operators you can choose from match the value type for the cell. You can change an operator later by editing the cell. You can edit a cell using the decision table editor, or in the edit bar.

You can set the cell value to an expression. You can specify expressions only by using the edit bar.

Procedure

To define a condition column statement:

1. Set the default operator for all the cells in that column in one of the following ways:

To change an operator using the decision table editor:	To change an operator using the edit bar:	To set the cell value to an expression:
<div>a. In the decision table editor, right-click the cell.</div> <div>b. Click Operator, and then click an operator.</div>	<div>a. In the decision table editor, click the cell.</div> <div>b. In the edit bar, modify the operator part of the statement.</div> <div>c. Click Enter  next to the edit bar.</div> <div>The operator displays as a symbol next to the cell value.</div>	<div>a. In the decision table editor, click the cell.</div> <div>b. In the edit bar, modify the value part of the statement.</div> <div>c. Click Enter .</div> <div>The expression is shown in italics in the cell, as in the example.</div>

2. Save your changes.

Results

The following example of a decision table shows an expression in italics in the cell:

the state of the pickup branch of 'the current rental agreement' is "Rhode Island" + ", my favorite state!"

	State	Age of the customer		Rental Rejected	Reason
		Minimum Age	Maximum Age		
1	New York	18	21	true	In New Yo.
2		21	25	false	
3					
4	"Rhode Island" + ", my favorite state!"	16	21	true	In Rhode .
5		21	25	true	In Rhode .
6					

Parent topic: [Specifying values and operators](#)

Related tasks:
[Specifying values](#)

Related information:
[Business Action Language \(BAL\)](#)
[Working with decision tables](#)

Disabling action cells

You can disable an action cell by clicking an icon on the menu bar or by using the Enable / Disable Action command.


About this task

When you define an action column, you specify the default action phrase for all the cells in that column. You cannot change the action phrase for a specific cell.

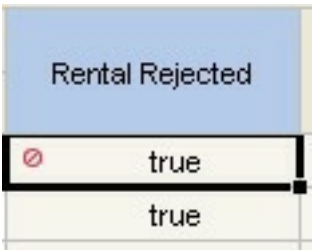
However, you can disable action cells so that the action is not carried out for that particular row. This is equivalent to clearing the contents of the cell, except that you keep the information it contains so that it can be re-enabled later if required.

Procedure

To disable an action cell:

- 1. In the decision table editor, right-click the action cell you want to disable.
- 2. Click  **Enable / Disable Action**.

The cell shows the disabled icon.



Results

This is a toggle function, which means that you do the same action to re-enable the action cell. When you re-enable the cell, the cell no longer shows the disabled icon.

Parent topic: [Working with decision tables](#)

Related tasks:

[Populating a decision table from an Excel spreadsheet](#)
[Defining columns](#)

Related information:

[Adding and populating rows](#)

Populating a decision table from an Excel spreadsheet

You can copy the contents of an Excel spreadsheet into a decision table provided the structure of the Excel spreadsheet and the decision table are the same.

Procedure

To populate a decision table with the contents of an Excel spreadsheet:

1. Define the columns of your decision table. See [Defining columns](#).
2. Make sure that the columns of the Excel spreadsheet correspond to the columns of your decision table.
3. Copy the cells in the Excel spreadsheet.
4. Right-click the first cell of the decision table, that is, the cell on row 0 in the first column, and click **Paste**. If you are inserting a new line and not overwriting an existing line of information, right-click and use **Paste Special**.

Note: This feature is not available in the Decision Center decision tables.

Results

Rule Designer automatically merges cells that have the same value. For example, if the content of your Excel spreadsheet is as follows:

	A	B	C	D
1	CDW	Economy,Compact,MidSize	15	Add surcharge of 15 for CDCW
2	CDW	FullSize,Luxury,SportUtility,Minivan	25	Add surcharge of 25 for CDW [FullSize,Luxury,SportUtility,
3	LDW	Economy,Compact,MidSize	21	Add surcharge of 21 for LDW
4	LDW	FullSize,Luxury,SportUtility,Minivan	21	Add surcharge of 21 for LDW [FullSize,SportUtility,Luxury,
5	PAI		3	Add surcharge of 3 for PAI
6	PEI		2	Add surcharge of 2 for PEI
7	ALI	Economy,Compact,MidSize	11	Add surcharge of 11 for ALI
8	ALI	FullSize,Luxury,SportUtility,Minivan	12	Add surcharge of 12 for ALI

The resulting decision table looks like this:

	Coverages	Car group	Surcharge	Comment
1	CDW	Economy,Compact,MidSize	\$15	Add surcharge of 15 for CDW...
2		FullSize,Luxury,SportUtility,Mi...	\$25	Add surcharge of 25 for CDW...
3	LDW	Economy,Compact,MidSize	\$21	Add surcharge of 21 for LDW ...
4		FullSize,Luxury,SportUtility,Mi...	\$21	Add surcharge of 21 for LDW ...
5	PAI		\$3	Add surcharge of 3 for PAI
6	PEI		\$2	Add surcharge of 2 for PEI
7	ALI	Economy,Compact,MidSize	\$11	Add surcharge of 11for ALI [E...
8		FullSize,Luxury,SportUtility,Mi...	\$12	Add surcharge of 12 for ALI [...

If you specify a range of values, the range is represented by two subcolumns in the decision table. For example, if you specified the following range in your Excel spreadsheet:

	A	B	C	D
1	CDW	Economy,Compact,MidSize	[15..21]	Add surcharge of 15 for CDCW

The Surcharge column in the decision table would look like this:

Surcharge	
[15	21]
# 25	

Parent topic: [Working with decision tables](#)

Related tasks:

[Creating a decision table from scratch](#)

[\(Deprecated\) Creating a decision table from a template](#)

[Defining columns](#)

Related information:

[Decision tables](#)

[Business Action Language \(BAL\)](#)

Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.

About this task

You can define preconditions for a decision table. Preconditions can contain the following definitions:

- Variables that are available throughout the decision table
- Conditions that must be satisfied before the decision table is executed

Procedure

To define a precondition:

1. In the decision table editor, click the **General** tab.
2. In the **Preconditions** section, press Ctrl+Spacebar.

The Content Assist box opens. You define variables and conditions in the Preconditions section in the same way that you would create variables and conditions for an action rule in the Intellirule editor. See [Building rules using the Intellirule editor](#) for details.

3. Save the decision table.

The variables you defined are now available for creating condition definitions in the decision table, which means that the rows of the decision table cannot be evaluated before the conditions defined in the precondition are satisfied.

Parent topic: [Working with decision tables](#)

Formatting a decision table

You can define format settings for a table, a column, or individual cells.

About this task

- Column format settings override decision table format settings.
- Cell format settings override decision table and column format settings.

Procedure

To format a decision table, a column, or an individual cell:

1. Follow one of the following procedures, depending on what you want to do:

To format a decision table:	To format a column:	To format a cell:
<div><div><div>a. Right-click the header part of the decision table, and then click Decision Table Properties.</div><div>b. In the Decision Table Properties dialog, click the Format tab.</div><div>c. Apply the required formatting and font settings.</div></div><div><div>You can set the foreground color and font of the column headers and row headers. You can also set the foreground color, background color, and font for all cells throughout the table.</div></div></div>	<div><div><div>a. Right-click the header part of the decision table, and then click Format.</div><div>b. In the Format Column dialog, apply the required alignment, colors, font, and format.</div></div><div><div>To format the contents of the cells in the column, do one of the following tasks:</div><div><div>■ From the Format Statement list, select one of the preconfigured formats. The formats available depend on the column data type: string, number, date, or time.</div><div><div>Use the Preview field to verify the selected format.</div><div>For example, in a column of cells that contain numbers, select the format {0, number, currency} to display the number in currency format using the currency of the currently active locale.</div></div><div><div>■ Enter the required format in the Format Statement field. Use a zero enclosed in curly braces to represent the cell contents. For example, to display numbers in currency format</div><div>using a specific currency symbol such as £, enter</div></div></div></div></div>	<div><div><div>a. Right-click the cell you want to format, and then click Format cell.</div><div>b. In the Format Cell dialog, apply the cell tooltip, alignment, colors, font, and format. You can override the formatting options at subcell level.</div></div><div><div>To format the cell contents, do one of the following tasks:</div><div><div>■ From the Format Statement list, select one of the preconfigured formats. The formats available depend on the cell data type: string, number, date, or time.</div><div><div>Use the Preview field to verify the selected format.</div><div>For example, in a cell displaying a number, select the format {0, number, currency} to display the number in currency format using the currency of the currently active locale.</div></div><div><div>■ Enter the required format in the Format Statement field. Use a zero enclosed in curly braces to represent the cell contents. For example, to display the number in currency format</div><div>using a specific currency symbol such as £, enter</div></div></div></div></div>

	<p>the following format: £{0}.</p> <p>Rule Designer applies the settings you specify to all the cells in the column and its subcolumns.</p>	<p>the following format: £{0}.</p> <p>Rule Designer applies the settings you specify to the cell.</p>
--	---	---

2. Click **OK** to save your settings.

Parent topic: [Working with decision tables](#)

Related information:
[Business Action Language \(BAL\)](#)

Defining checking options

You can specify tests that Rule Designer conducts to identify problems with a decision table.

About this task

You can define global checking options for condition columns, or you can set up your own value checks to verify cell values in individual condition or action columns.


Attention:

Checking is an intense CPU activity and might increase processing time for large decision tables and, as a result, for the build process.

Procedure

To define checking options for condition columns or cell values:

- 1. Use the decision table editor to define the type of checking you require, as outlined in the following table:

To define global checking options	To define your own cell value checks:
<div><div>a. In the decision table editor, right-click a column or a cell and then click  Decision Table Properties.</div><div>b. Click the General tab of the Decision Table Properties dialog.</div><div>c. In the “Table checks” area, select the options you want to check.</div><div>You can choose which checks you want to be done against which column, and the severity of the checking reports.</div></div>	<div><div>a. In the decision table editor, double-click the required column.</div><div>b. In the Condition Column or Action Column dialog, in the Properties section, select a placeholder in the Expression Parameters list, and then select the Check Value check box.</div><div>c. Select the required test condition from the drop-down list next to the check box.</div><div>The contents of the drop-down list differs, depending on the type of value check. See Value test conditions for details.</div><div>d. When you make your selection, an input field opens (or more than one, depending on your selection) for you to enter the value(s) to check against.</div><div>The check value input field provides you with a content assist facility. To activate it, click Ctrl+Spacebar.</div><div>Note that you can enter only values, not complex expressions.</div><div>The check applies to all the cells in the column. Warnings are reported if any values do not match the test.</div></div>

- 2. Click **OK** to save your settings.

Parent topic: [Working with decision tables](#)

Value test conditions

When you construct conditions that test values, the way you complete the associated input fields depends on the type of test.

A numeric value check displays a list of logical operators (greater than, less than and so on), to which you add the required numeric value in the associated input field. A true or false check displays a choice of equal to or not equal to.

You might also see these options. You should complete the associated input fields as follows:

- `is in`: enter a comma-delimited set of values, enclosed in curly brackets. For example: {Massachussets, New York, Rhode Island}
- `is not in`: enter a comma-delimited set of values, enclosed in curly brackets.
- `is between`: enter a range of values in square brackets, separated by three periods. For example: [19...25]
- `match`: enter the value, character or string to be matched.

Values in a set, range or match might be numeric or non-numeric.

Parent topic: [Working with decision tables](#)

Related information:

[Decision tables](#)

[Business Action Language \(BAL\)](#)

Using decision table locking facilities

You can apply a variety of locks to protect your decision tables against editing by others.

About this task

Decision tables have locking facilities that allow you to protect your decision tables against editing by others. The locking rules should be specified in the context of a template library to be applied on instances of this decision table. For both condition columns and action columns, you can lock the structure, data, or both.

Procedure

To define decision table locking requirements:

1. Right-click any cell in your decision table, and then click **Decision Table Properties**. The Decision Table Properties Editor opens.
2. Click the Lock tab.
3. Choose which locks you want to apply:

Apply locking rules to this table

The default is off to allow you to edit your decision table and lock rules. Set to on for decision tables used in a package on which you want to enforce locking rules.

Show lock icons on columns

If on, the Lock icon is displayed on any locked condition and action items.

Prevent addition and removal of columns

Prevents condition or action columns from being added or deleted.

4. In the **Condition columns** section, select any of the following locks:

Lock test

The test expression associated with the condition column cannot be changed. Note that cell content, including cell-specific test operators, can still be changed.

Lock partitions

Partition items cannot be added to or removed from the selected condition column. The values of existing partition items can still be changed, but no rows can be added to or removed from this column.

Locking partitions locks partitions in all preceding columns because adding a new partition item to a previous column can create a new row in the locked column.

Lock values

The values of existing partition items or the cell operator cannot be changed.

5. In the **Action columns** section, select any of the following locks:

Lock action

The expression associated with the column cannot be changed.

Lock status

The status of the action cell cannot be changed. The action status specifies whether an action is enabled or disabled. See [Disabling action cells](#) for details on disabling action columns.

Lock values

The values in the action cell of the selected column cannot be changed.

6. Click **OK** to apply your selections.

Parent topic: [Working with decision tables](#)

Viewing rule details

You can view the details of a rule in a decision table row by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision table editor.


About this task

In a decision table, a row corresponds to one rule.

Procedure

To view the rule details:

Use the decision table editor to view the rule details in a tooltip or in a viewing pane, as follows:

To view the rule details in a tooltip:	To view rules in a dedicated viewing pane:
<div>a. In the decision table editor, place the cursor over the row header number on the side.</div>	<div><div>a. In the decision table editor, right-click a column or a cell, and then click </div><div>Decision Table Properties.</div><div>b. In the Decision Table Properties dialog, go to the Table view section of the General page.</div><div>c. Select Show rules.</div><div>d. Click OK.</div><div>A new viewing pane opens at the bottom of the decision table editor.</div><div>e. In the decision table editor, click the row header number on the left.</div></div>

Results

The rule is shown in a tooltip or the viewing pane, depending on which procedure you followed. If you defined preconditions for the decision table, these are also shown.

Parent topic: [Working with decision tables](#)

Viewing the decision table as a spreadsheet

You can display a decision table in a spreadsheet view if the decision table has two condition columns and one action column and the structure of the decision table is symmetric.

About this task

You can display a decision table in a spreadsheet view, if the decision table meets the following conditions:

- The decision table has two condition columns and one action column.
- The structure of the decision table is symmetric. A decision table is symmetric if, for each row in the first condition column, the same rows are defined for the second condition column.

	length of rental (days)	pickup branch state	message
0	≤ 10	New York	City fare
1		Rhode Island	
2		New Hampshire	Rent this car one more day and get a discount!
3	≥ 11	New York	
4		Rhode Island	Special discount
5		New Hampshire	You get a discount!

The spreadsheet view of a decision table presents the first condition column as the row header, and the second condition column as the column header. The action column values are represented as cells in the spreadsheet.

Note: This feature is not available in the Decision Center decision tables.

Procedure

To view a decision table as a spreadsheet:

1. In the decision table editor toolbar, click  **Switch to Spreadsheet View.**

The decision table is displayed as a spreadsheet.

×

✓

	"New York"	"Rhode Island"	"New Hampshire"
10	City fare		Rent this car one more day and get a ...
11		Special discount	You get a discount!

2. To switch back to the decision table view, in the spreadsheet view toolbar, click  **Switch to Decision Table View.**

Parent topic: [Working with decision tables](#)

Documenting a decision table

You can document each row in a decision table.


About this task

You can create a column to document the rows in a decision table.

Note: This feature is not available in the Decision Center decision tables.

Procedure

To document a decision table:

1. In the decision table editor, right-click a column or a cell, and then click **Decision Table Properties** .
2. In the Decision Table Properties dialog, go to the **Table view** section of General page.
3. Select **Show comments**.
4. Click **OK**.

A Comments column is added to the side of the table. You can use this column to document the rows.

Parent topic: [Working with decision tables](#)

Working with variables

You can use automatic variables and ruleset variables to define variables for use in business rules.

Variables available for rule authoring

A variable is any entity that can have different values.

Setting up automatic variables

You set up automatic variables to define a variable that can be accessed anywhere in business rules.

Defining ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project.

Parent topic: [Authoring business rules](#)

Variables available for rule authoring

A variable is any entity that can have different values.

A variable comprises three elements:

- Name
- Type
- Value

You define variables so that you can use the name independently of the value it represents. Variables can make your business rules less ambiguous and easier to understand.

A variable name must not contain any of the following characters:

Character type	Example
Tabulation	TAB
Line break	\n
Single quotation mark	'
Double quotation mark	"
Opening and closing parentheses	()
Slash	/
Comma	,
Semicolon	;

A variable can have one of the following value types:

Variable value type	Example
Constant	<ul style="list-style-type: none">• A number: 12• A literal character string: "Gold customer"• A Boolean, an object domain element: STATUS_APPROVED
Expression	<ul style="list-style-type: none">• An arithmetic expression whose result is a number: the price of the product * 1.20• A text concatenation or formatting expression: "The loan request for " + the name of the applicant + " is rejected"• A Boolean expression: the customer is married• A navigation expression to an object: the last transaction of the checking account of the customer
Object	For example, a reference to a predefined business term: the customer
Collection of values of the types already listed	For example, a list of numbers { 1, 3, 5 }, a list of objects { CHINA, FRANCE, UK, USA }

Important: The type of value that you use must match or be compatible with the variable type.

The following tables list the different types of variables that you can use in your business rules.

An automatic or implicit variable has a predefined name and type, and is set automatically or implicitly by the system:

Variabl e	Scope	Definition and use
Aut om atic vari able	A variable that is available in all the business rules that use the BOM class where the variable is declared.	
Impl icit vari able	A variable that is automatically declared by a language construct like a binding. For example: set 'var-name' to <a type> where ..., there is a ..., there are ..., for each....	Built into the Business Action Language. The name, type and value are controlled by the system. You use implicit variables to refer to a current instance of an object that is being tested or declared, or to a collection that is being

		<p>iterated.</p> <p>Implicit variables are visible only in the code completion menu of the Intellirule editor.</p>
--	--	--

You can define your own rule variables, ruleset variables, and ruleset parameters:

V a r i a b l e	Scope	Definition and use
R u l e v a r i a b l e	<p>A variable that can be referenced only in the action rule in which it is defined.</p>	<p>You define a rule variable in the definitions part of the rule.</p> <p>Rule variables are local to the rule in which they are defined. They are not available in other rules. The variable name must be unique within the rule.</p>
R u l e s e t v a r i a b l e	<p>A variable that can be referenced by all the business rules in a rule project.</p> <p>Ruleset variables are not available outside of the rule project scope.</p>	<p>You define a ruleset variable in a variable set in the rule project. Before you define a ruleset variable therefore, you must create a variable set in which to group your ruleset variables.</p> <p>You define a ruleset variable by its name, type, and verbalization. Ruleset variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.</p> <p>Make sure that you set default values to ruleset variables before you use them in rules.</p> <p>You can also use ruleset variables to pass data between tasks in a ruleflow.</p>
R u l e s e t p a r a m e t e r	<p>A ruleset parameter is a variable that is defined as the interface between the calling application of a ruleset and the ruleset itself.</p> <p>It has a direction: in, out, or in-out.</p> <p>The direction indicates whether the caller passes data in, expects data out of the ruleset, or passes data in and expects some modifications to the data passed in.</p>	<p>You define a ruleset parameter in the signature of a decision operation in a decision service.</p> <p>You use ruleset parameters to exchange data between a ruleset and an application, and as the main data elements on which the rule operates.</p> <p>Ruleset variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.</p>

Parent topic: [Working with variables](#)

Setting up automatic variables

You set up automatic variables to define a variable that can be accessed anywhere in business rules.

About this task

To define a variable that can be accessed anywhere in business rules, you can set up automatic variables. Automatic variables are bound to the instance of the class on which it is defined. Automatic variables are variables that you declare as an instance of a specific BOM class. They are available in all the business rules that use the BOM class where the variable was declared.

When you specify an automatic variable, it is defined for the corresponding business term, and identified with the article “the”. For example, if customer is a business term, a corresponding variable called the customer is available in the rule editors.

Note: This is valid only for the locales that use this type of article.
--

You cannot define one variable to be the same as another variable of the same type. If your rule requires you to refer to more than one occurrence of customer, you have to define the other variables explicitly, in the definitions part of the rule. In this case, the automatic variable is no longer available in this rule.

The automatic variable is known internally as customer. The variable the customer is the automatic variable in a given verbalization context (with definite article the) and is the form that you can use in rules to reference this variable. Consequently, you cannot declare another variable named customer in the definition part of a rule:

```
definitions set 'customer' to a customer;
```

This statement generates the following error:

"An automatic variable 'customer' is already declared"

For example, in the following rule, ceiling is an explicit variable declared in the definitions part of the rule, and the customer is an automatic variable.

```
definitions set ceiling to 10,000;  
if the value of the shopping cart of the customer is more than ceiling  
then...
```

You cannot declare a ruleset parameter or variable with the same name as an automatic variable. If you do, Rule Designer redirects references to automatic variables such as the customer to the ruleset parameter. For example, if the rule project contains a ruleset parameter called customer, in the following condition, the customer references the ruleset parameter instead of the automatic variable:

```
if  
    the salary of the customer is more than 100 ...
```

Procedure

To generate an automatic variable for a business class:

1. In the Outline view, click the class for which you want to generate an automatic variable.
2. In the Class Verbalization section of the BOM Editor, select the **Generate automatic variable** box.
3. Save the BOM.

An instance of the class is now available as an automatic variable from business rules.

Parent topic: [Working with variables](#)

Related concepts:

[Rule variables](#)
[Vocabulary](#)

Related information:

[Overview: Ways to express business rules](#)
[Working with action rules](#)
[Business Action Language \(BAL\)](#)

Defining ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project.

Creating a variable set

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables.

Defining a ruleset variable

After you have created a variable set, you can define a ruleset variable.

Modifying the verbalization of a ruleset variable

After defining a ruleset variable, you can modify its verbalization.

Parent topic: [Working with variables](#)

Related concepts:

[Action rules](#)

Related information:

[Overview: Ways to express business rules](#)

[Decision operations](#)

Creating a variable set

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables.

About this task

Ruleset variables are variables that you can use in all the business rules of the ruleset. You can also use ruleset variables to pass data between tasks in a ruleflow. Ruleset variables are accessible from business rules only if you verbalize them.

Procedure

To create a variable set:

1. Select your rule project.
2. On the **File** menu, click **New** > **Variable Set**.

The New Variable Set wizard opens.

3. If you want to select a rule package, click **Browse** next to the **Package** field.
4. In the **Name** field, type the name of your variable set.
5. Click **Finish**.

The new variable set is shown in the Rule Explorer view and the Variable Set Editor opens.

Results

You can now define your ruleset variables.

Parent topic: [Defining ruleset variables](#)

Defining a ruleset variable

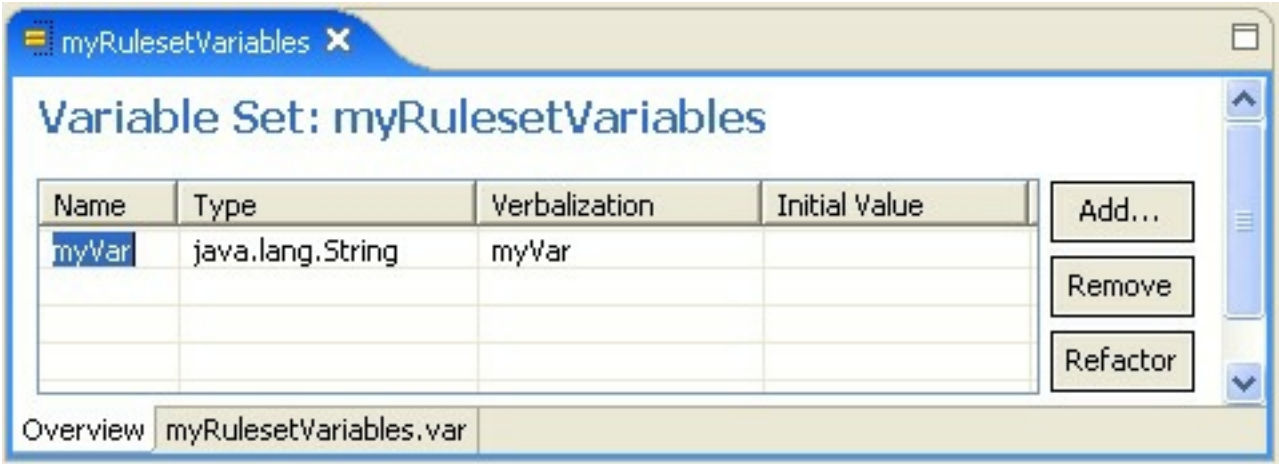
After you have created a variable set, you can define a ruleset variable.

Procedure

To define a ruleset variable:

1. In the Variable Set editor, click **Add**.

Default values for a new ruleset variable are displayed in a table row.



2. Specify the name, type and (optionally) the initial value of the ruleset variable.
3. If you do not want the ruleset variable to be visible from business rules, delete the default verbalization from the Verbalization column. Otherwise, modify the verbalization.

Results

The ruleset variable is now defined and you can start using it in business rules.

Parent topic: [Defining ruleset variables](#)

Modifying the verbalization of a ruleset variable

After defining a ruleset variable, you can modify its verbalization.

Procedure

To modify the verbalization of a ruleset variable:

1. In the Variable Set editor, select the variable in the table and click **Refactor**.

The Refactor Variable window opens.

2. In the **Variable verbalization** field, type a new verbalization for the variable.

Restriction:

Do not use the following characters in the verbalization of ruleset variables:

Variable	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
()	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon

3. If you want to see the impact of the change on the business rules, click **Preview**.

The Refactor dialog shows which business rules (if any) need to be refactored.

4. Click **OK**.

The business rules are refactored.

What to do next

Note: You must refactor the rows individually. If you change the verbalization in a row without refactoring and then refactor another row, the new value of the first row is not refactored.

Parent topic: [Defining ruleset variables](#)

(Deprecated) Working with decision trees

You use the decision tree editor to create a workflow for the execution of your rules.

Decision trees

Decision trees provide a concise view of a set of action rules that are not sufficiently symmetric to present in a decision table.

Creating a decision tree

You create a decision tree by specifying a name and optionally specifying a package.

Labeling and defining condition nodes

You label condition nodes to make condition tree diagrams easier to read. You define a condition node by constructing a condition statement in the edit bar of the decision tree editor.

Labeling and defining branches

When you define or insert a new branch, you must label it. You can insert Otherwise branches, and branches on nodes that define an enumerated domain. You can also duplicate and delete branches.

Labeling and defining actions

Actions are made up of two elements: the action set header (title) and the action set.

Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision tree and conditions that must be satisfied before the decision tree is executed.

Defining checking options

You can specify the tests that you want Rule Designer to conduct to identify problems with a decision tree.

Viewing rules

You can view the details of a rule in a decision tree by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision tree editor.

Parent topic: [Authoring business rules](#)

Decision trees

Decision trees provide a concise view of a set of action rules that are not sufficiently symmetric to present in a decision table.

[\(Deprecated\) Decision tree overview](#)

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

[Decision tree editing errors and warnings](#)

Rule Designer checks for overlaps and gaps in decision tree conditions, and indicates problems by using visual cues.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Related information:

[Overview: Ways to express business rules](#)

[\(Deprecated\) Working with decision trees](#)

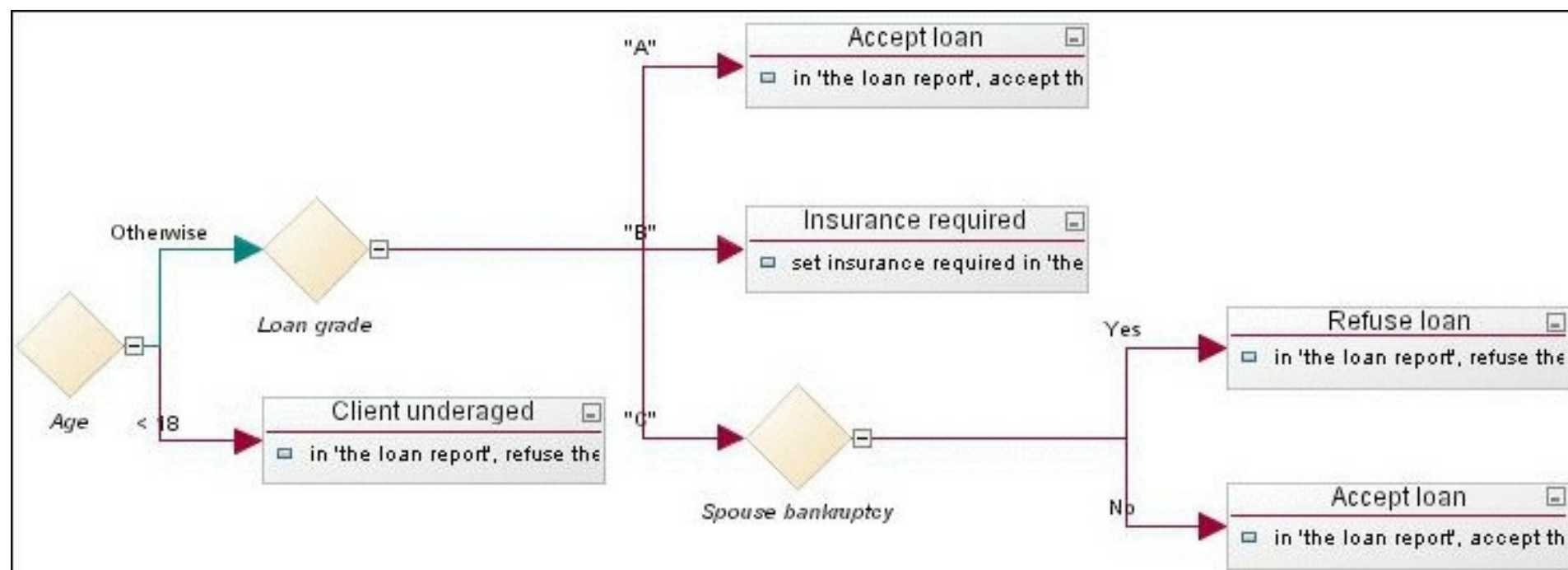
[Business Action Language \(BAL\)](#)

(Deprecated) Decision tree overview

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

Decision trees provide a way to view and manage large sets of business rules in diagrams.

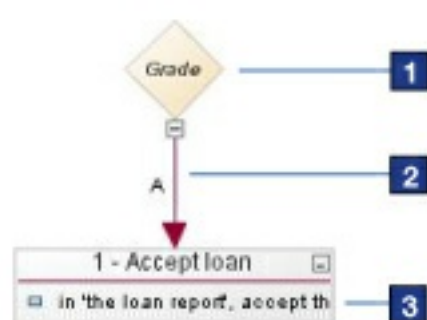
Decision trees make the interaction of nonsymmetrical rules easier to understand. The path from the first condition to the end of the actions along any branch represents one rule.



Looking at the following figure, you can see why decision trees are easy to understand:

- A condition is declared in its diamond-shaped node **1**.
- The possible values for the condition are represented by branches **2**.
- The actions are declared at the end of each branch **3**.

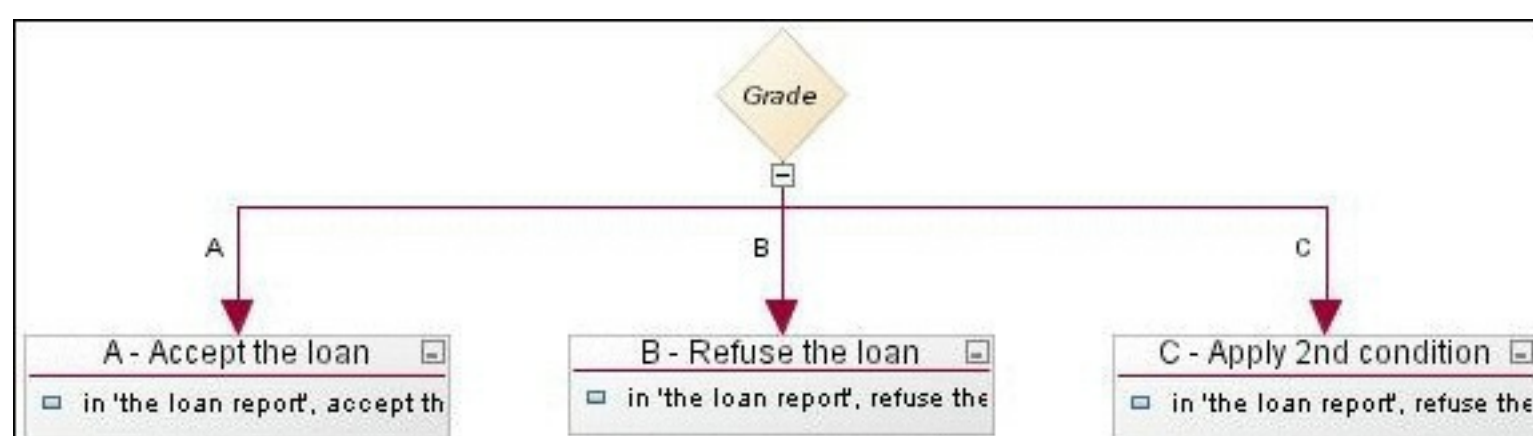
This simple decision tree corresponds to the following rule:



if the grade in the loan report is 'A'
then in the loan report, accept the loan with the message "Loan accepted"

By adding branches, you add new rules that have different values for the condition.

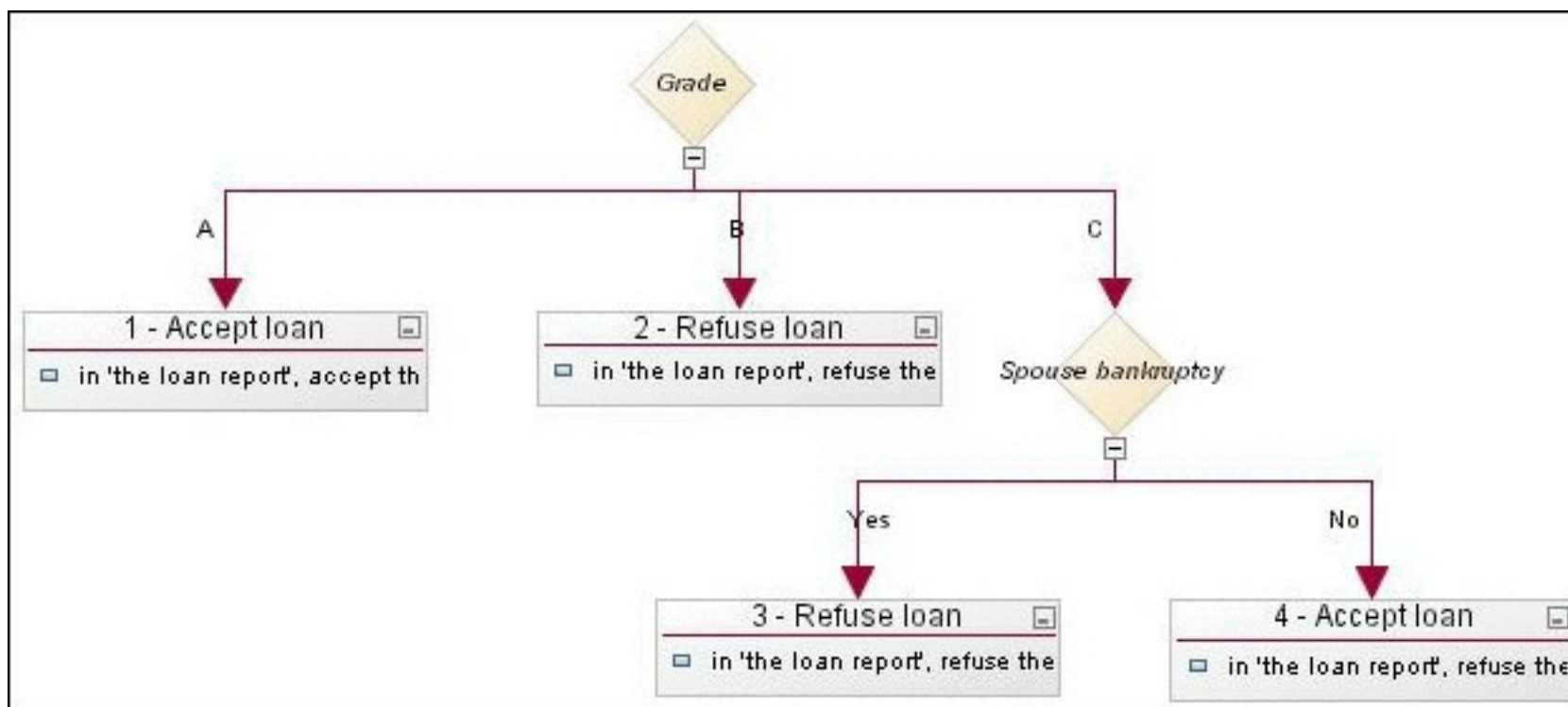
For example, the following decision tree forms three rules for a loan application (A, B, and C):



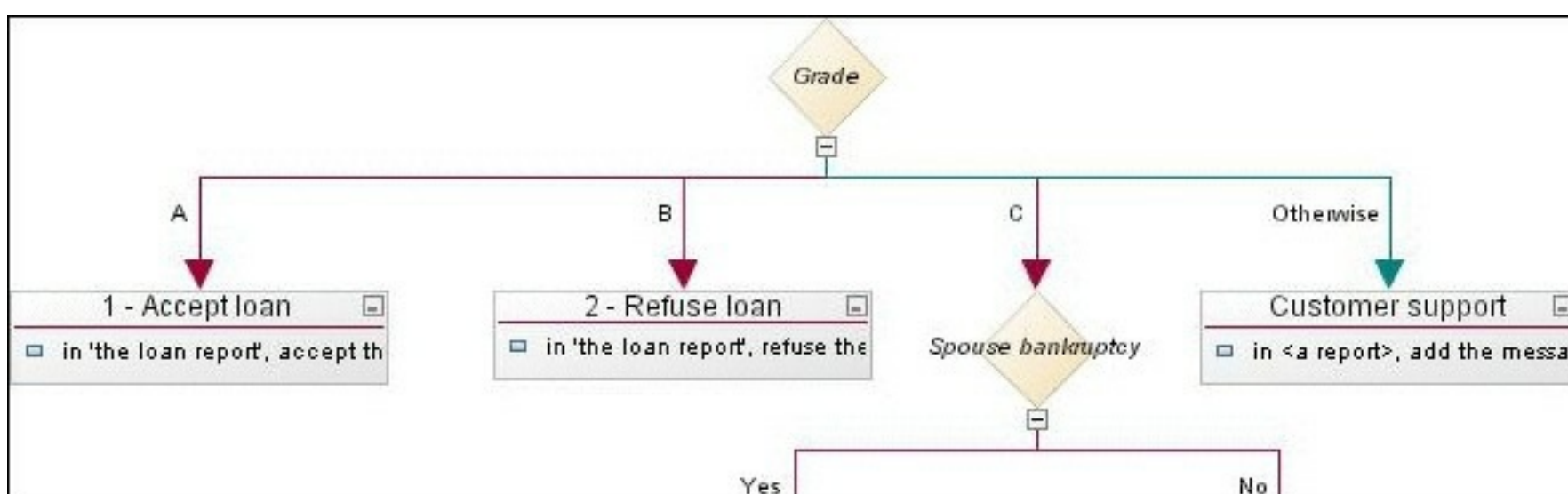
You can put as many actions as you want at the end of a branch, and add another condition to a branch.

For example, rules 1 and 2 in the following decision tree have only one condition (Grade), while rules 3 and 4

have a second condition to check (Spouse bankruptcy) before they do the actions:



Finally, you can add an Otherwise branch for condition values that are not covered by any of the other branches:



You can lay out your decision tree vertically or horizontally for optimal viewing, and set or remove consistency checking.

Preconditions

You can set the following elements in a precondition section of a decision tree:

- Variables that can be used in the decision tree.
- Condition that is applied to an entire decision tree.

If the precondition is not satisfied, none of the rules in the decision tree can be evaluated.

For example, you can apply the following precondition to a decision tree:

definitions

set 'wealthy customer' to a customer
where the average monthly balance of this customer
is more than \$1,000,000

if

the state of residence of 'wealthy customer' is NY

Applying this precondition to a decision tree limits the scope of the decision tree rules to only those customers who have an average monthly balance of \$1,000,000 and live in New York. You can also use the variable `wealthy customer` in the tree.

Preconditions are tested before each rule in a tree is run.

Parent topic: [Decision trees](#)

Related concepts:

[Action rules](#)
[Technical rules](#)

Related reference:


[BAL operators](#)

Related information:

[Overview: Ways to express business rules](#)

Decision tree editing errors and warnings

Rule Designer checks for overlaps and gaps in decision tree conditions, and indicates problems by using visual cues.

If there is an error or warning in a decision tree, the warning symbol  identifies the affected nodes and branches.

When you save the decision tree, the Problems view also displays the errors and warnings.

A decision tree that has a checking error cannot be executed, unless you disable checking.

The rules that make up your decision tree can be checked for gaps and overlap. You can specify your checking preferences at node level and at tree level.

Gaps

Rule Designer checks that the branches in the decision tree are uninterrupted. For example, if one branch defines the interval [21 - 30] and another branch defines the interval [33 - 40], an error is generated since the interval [31 - 32] is not accounted for.

Overlap

Rule Designer checks that no branches overlap in the decision tree. For example, if one branch defines the interval [21 - 30] and another branch defines the interval [28 - 40], an error is generated since the interval [28 - 30] overlaps in that node.

Note:

You cannot check dates for gaps and overlap.

Parent topic: [Decision trees](#)

Related concepts:

[\(Deprecated\) Decision tree overview](#)

Related tasks:

[Applying verbalization changes to business rules](#)

[Applying a category filter](#)

Related reference:

[BAL operators](#)

Related information:

[Overview: Ways to express business rules](#)

[Vocabulary errors and warnings](#)

[\(Deprecated\) Working with decision trees](#)

[Business Action Language \(BAL\)](#)

Creating a decision tree

You create a decision tree by specifying a name and optionally specifying a package.

About this task

Decision trees are composed of branches that have a condition node as their root, and end with actions. Every node is a condition node, except for leaf nodes. Decision trees allow you to manage a large set of rules with some conditions in common but not all.

Procedure

To create a decision tree:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Tree**. The New Decision Tree wizard opens.
3. Optional: Add the decision tree to a rule package:
 - a. In the **Package** field, click **Browse**.
 - b. Expand the folder structure, click the package to which the decision tree should be added, and then click **OK**.
4. In the **Name** field, type a name for the decision tree.
5. Click **Finish**. The new decision tree is displayed in the Rule Explorer view and the Decision Tree Editor opens. The default decision tree has one condition node, one branch, and one action.

Results

You can now use the decision tree editor to label and define the condition node and action, and to create further nodes, branches and actions as appropriate.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Related tasks:

[Correcting errors by using Quick Fix](#)

[Defining preconditions](#)

[Defining checking options](#)

[Viewing rules](#)

Related information:

[Decision trees](#)

[Business Action Language \(BAL\)](#)

[Labeling and defining condition nodes](#)

[Labeling and defining branches](#)

[Labeling and defining actions](#)

Labeling and defining condition nodes

You label condition nodes to make condition tree diagrams easier to read. You define a condition node by constructing a condition statement in the edit bar of the decision tree editor.

Labeling a condition node

You label condition nodes to make condition tree diagrams easier to read.

Defining a condition node

When you have created a label for the condition node, you can define the node.

Inserting a condition node

You can insert condition nodes before or after an existing condition node.

Inserting a new condition node after a branch

You can insert condition nodes after an existing branch.

Deleting a condition node

When you delete a condition node, all its branches are also deleted.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Labeling a condition node

You label condition nodes to make condition tree diagrams easier to read.

About this task

Before you can define a condition node, you must give it a name (label). Labels make the condition tree diagrams easier to read.

Procedure

To label a condition node:

1. In the decision tree editor, double-click the condition node. The Node Editor dialog opens.
2. In the **Label** field, type a title for the condition node.
3. Click **OK**.

Results

Your title replaces the default label.

Parent topic: [Labeling and defining condition nodes](#)

Related tasks:

[Defining a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Defining a condition node

When you have created a label for the condition node, you can define the node.

Procedure

To define a condition node:

1. In the decision tree editor, click the condition node.
2. In the edit bar, build a condition statement.

The process is the same as using the guided editor, except that you cannot use the edit bar to edit the values.

3. To save the definition, click the **Enter** button  next to the edit bar.

Results

The condition node is defined. You can now add other condition nodes to the tree and create branches to specify different values for the condition statement defined for the node.

Parent topic: [Labeling and defining condition nodes](#)

Related tasks:

[Labeling a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Inserting a condition node


You can insert condition nodes before or after an existing condition node.

About this task

You can insert condition nodes before an existing condition node, or after it. This procedure describes how to insert a condition node before an existing condition node. After you insert a condition node, you must label and define it.

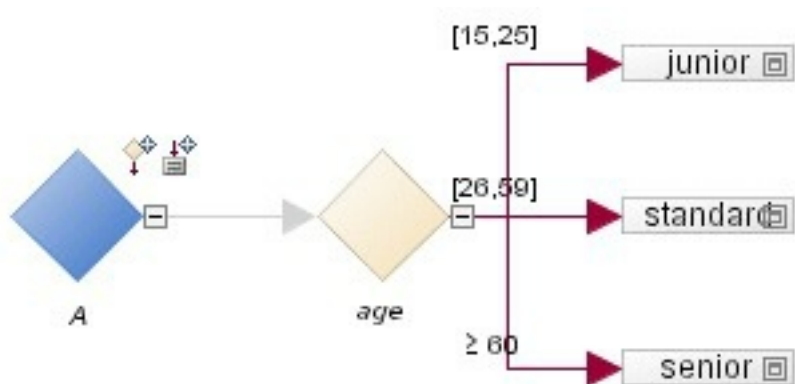
Procedure

To insert a condition node before an existing condition node:

1. Right-click the condition node before which you want to insert a new condition node.
2. Click  **Insert Condition Node**.

Results

The new condition node is added to the decision tree. You must now label and define it, as described in [Labeling a condition node](#) and [Defining a condition node](#).



Parent topic: [Labeling and defining condition nodes](#)

Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Inserting a new condition node after a branch


You can insert condition nodes after an existing branch.

About this task

After you insert a condition node, you must label and define it.

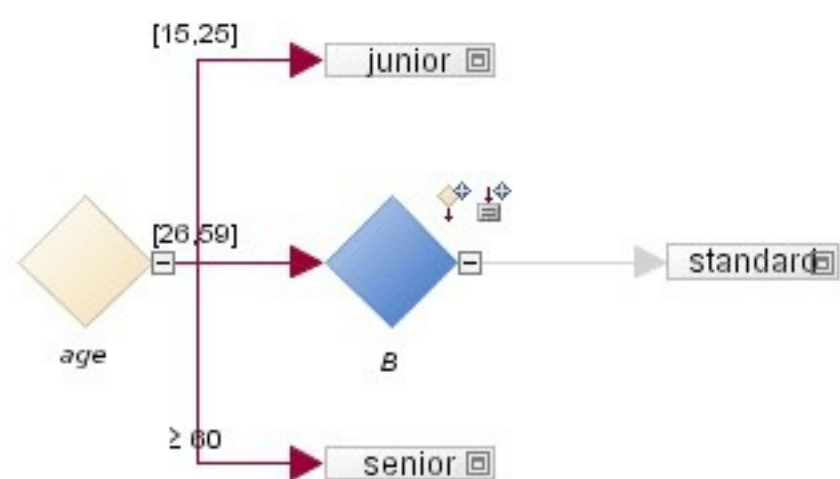
Procedure

To insert a new condition node after a branch:

1. Right-click the branch after which you want to insert a new condition node.
2. Click  **Insert Condition Node**.

Results

The new condition node is added to the decision tree. You must now label and define it, as described in [Labeling a condition node](#) and [Defining a condition node](#).



Parent topic: [Labeling and defining condition nodes](#)

Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a condition node](#)

[Deleting a condition node](#)

Related information:

[Decision trees](#)


[\(Deprecated\) Working with decision trees](#)

Deleting a condition node

When you delete a condition node, all its branches are also deleted.

Procedure

To delete a condition node:

Right-click the node in the decision tree editor and click  **Delete**.

Results

The condition, together with its associated branches, is deleted.

Parent topic: [Labeling and defining condition nodes](#)

Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Labeling and defining branches

When you define or insert a new branch, you must label it. You can insert Otherwise branches, and branches on nodes that define an enumerated domain. You can also duplicate and delete branches.

Defining a branch

Branches are represented by links. You must define the branch before you can label it.

Labeling a branch

You label a branch.

Inserting a new branch

You can attach new branches to condition nodes.

Inserting an Otherwise branch

You use an Otherwise branch if none of the other possibilities for a condition are correct.

Changing a branch to and from Otherwise

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch.

Inserting and defining branches for enumerated domain values

You can insert and define branches automatically on nodes that define an enumerated domain.

Duplicating a branch

You duplicate a branch by using drag and drop in the Decision Tree Editor.

Deleting a branch

When you delete a branch, you also delete all its associated actions.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Related tasks:

[Correcting errors by using Quick Fix](#)

Related information:

[Decision trees](#)

[Business Action Language \(BAL\)](#)


[Labeling and defining condition nodes](#)

Defining a branch

Branches are represented by links. You must define the branch before you can label it.

Procedure

To define a branch:

1. In the decision tree editor, click the branch.
2. In the edit bar, edit the operator and values of the condition statement in the same way as in the guided editor.
3. Click the **Enter** button  next to the edit bar to save the definition.

Results

The newly-defined branch shows up in dark red.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

Labeling a branch

You label a branch.

Procedure

To label a branch:

1. In the decision tree editor, double-click the required branch. The Branch Editor dialog opens.
2. In the **Label** field, type a label for the branch.
3. Click **OK**.

Results

Your label replaces the default label. You can add other branches, duplicate and delete branches, and insert and define branches on nodes that define an enumerated domain.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)


[Deleting a branch](#)

Inserting a new branch

You can attach new branches to condition nodes.

Procedure

To insert a new branch:

In the decision tree editor, right-click the condition node to which you want to attach a new branch, and then click **Add** >  **New Branch**.

Results

A new branch is displayed under the node. You can now label and define it, as described above.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

Inserting an Otherwise branch

You use an Otherwise branch if none of the other possibilities for a condition are correct.

About this task

You can also change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch. Note that each node can have only one Otherwise branch.

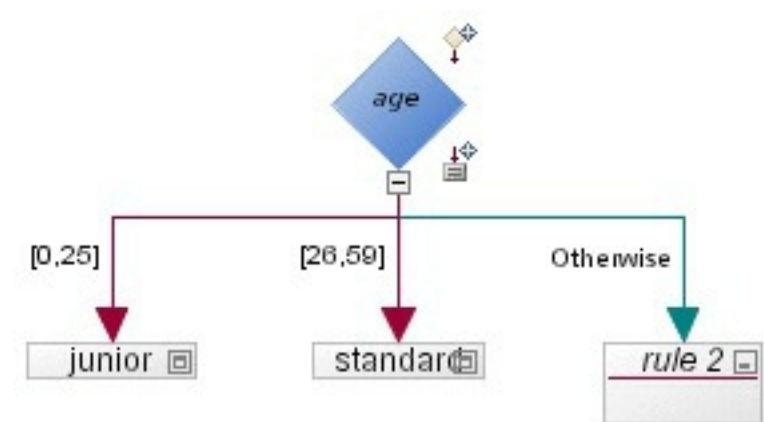
Procedure

To insert an Otherwise branch:

In the decision tree editor, right-click the condition node to which you want to attach an Otherwise branch and then click **Add > Otherwise**.

Results

A new Otherwise branch is added under the node. The Otherwise branch is green.



Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

Changing a branch to and from Otherwise

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch.

About this task

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch. Note that each node can have only one Otherwise branch.

Procedure

To change a branch to and from Otherwise:

1. In the decision tree editor, right-click the Otherwise branch.
2. Click **Set / Unset as Otherwise**.

Results

The status of the branch changes to or from an Otherwise branch, as appropriate.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Duplicating a branch](#)

[Deleting a branch](#)

Inserting and defining branches for enumerated domain values


You can insert and define branches automatically on nodes that define an enumerated domain.

About this task

In the decision tree editor, create branches for the enumerated domain values.

Procedure

To insert and define branches for all the values of an enumerated domain:

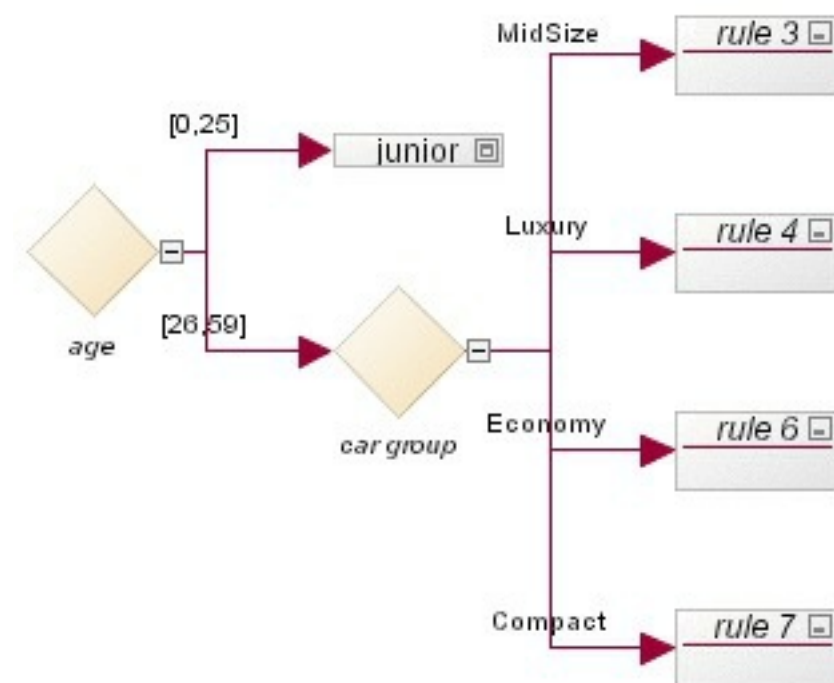
1. In the decision tree editor, right-click the condition node to which you want to attach the branches.
2. Click **Add** >  **All Domain Values**.

Note:

The Decision Tree Editor can detect missing domain values. You can add missing values automatically using the quick fixes. See [Correcting errors by using Quick Fix](#).

Results

The new branches are displayed under the node.



Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)
[Labeling a branch](#)
[Inserting a new branch](#)
[Inserting an Otherwise branch](#)
[Changing a branch to and from Otherwise](#)
[Duplicating a branch](#)
[Deleting a branch](#)

Duplicating a branch

You duplicate a branch by using drag and drop in the Decision Tree Editor.

Procedure

To duplicate a branch:

1. In the decision tree editor, click the branch that you want to duplicate.
2. Press Ctrl and drag the branch to a location at the same level.
3. Release Ctrl.

Results

If overlap checking is active on the parent condition node, an error is displayed on the duplicated branch.

Without using the Ctrl key, you can use drag and drop to reorder nodes at the same level. A drag and drop without using the Ctrl key is equivalent to a cut and paste. You can also drop a branch onto a node to replace it with a copy of the selected branch.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)


[Deleting a branch](#)

Deleting a branch

When you delete a branch, you also delete all its associated actions.

Procedure

To delete a branch:

1. In the decision tree editor, right-click the branch.
2. Click **Delete** .

Results

The branch, together with its associated actions, are deleted.

Parent topic: [Labeling and defining branches](#)

Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

Labeling and defining actions

Actions are made up of two elements: the action set header (title) and the action set.

Labeling an action set

You label actions by entering a title for the actions in the action set header.

Defining an action

You define an action in the action set part of the action.

Adding an action to an action set

When you add an action to an action set, you must then define it.

Changing the order of actions

You change the order of actions in an action set by moving them up or down the list of actions.

Deleting actions

You can delete individual actions from an action set, or the whole action set.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Labeling an action set

You label actions by entering a title for the actions in the action set header.

About this task

You label action set headers. The label (title) must match the name of the corresponding rule.

Procedure

To label an action set:

1. In the decision tree editor, double-click the action set header.
2. In the Action Set Editor dialog, type a title for the actions in the **Title** field.
3. Click **OK**.

Results

Your title replaces the default title.

Parent topic: [Labeling and defining actions](#)

Related tasks:

[Defining an action](#)

[Adding an action to an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Defining an action


You define an action in the action set part of the action.

About this task

The action set can contain one or more actions.

Procedure

To define an action:

1. In the decision tree editor, in the action set part of the action, click **edit action**.
2. In the edit bar, define the action. The edit bar operates in the same way as the guided editor.
3. Click **Enter** .

Results

The action is now defined, and the action set displays the action text.

Parent topic: [Labeling and defining actions](#)

Related tasks:

[Labeling an action set](#)

[Adding an action to an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Adding an action to an action set


When you add an action to an action set, you must then define it.

About this task

When you add an action to an action set, you define it by using the **edit action** entry.

Procedure

To add an action to an action set:

1. In the decision tree editor, right-click the action set header.
2. Click **Add Action** .

Results

The action set displays a new **edit action** entry, for you to define the new action.

Parent topic: [Labeling and defining actions](#)

Related tasks:

[Defining an action](#)

[Labeling an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Changing the order of actions



You change the order of actions in an action set by moving them up or down the list of actions.

About this task

Move the actions up or down to change their order.

Procedure

To change the order of actions:

1. In the action set, right-click the action you want to move.
2. Click  **Move Up** or  **Move Down**.
3. Save your changes.

Results

Actions are now listed in the new order.

Parent topic: [Labeling and defining actions](#)

Related tasks:

[Defining an action](#)

[Adding an action to an action set](#)

[Labeling an action set](#)

[Deleting actions](#)

Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

Deleting actions

You can delete individual actions from an action set, or the whole action set.



About this task

When you delete the whole action set, you also delete the associated branch.

Procedure

To delete actions:

1. Decide whether you want to delete an individual action, or an action set:

<div><div>a. In the action set, right-click the action.</div><div>b. Click  Delete.</div><div>The action is removed from the action set.</div></div>	<div><div>a. In the decision tree editor, right-click the action set header.</div><div>b. Click  Delete.</div><div>The action set and its associated branch is deleted.</div></div>

2. Save your changes.

Parent topic: [Labeling and defining actions](#)

Related tasks:

- [Labeling an action set](#)
- [Defining an action](#)
- [Adding an action to an action set](#)
- [Changing the order of actions](#)

Related information:

- [Decision trees](#)
- [\(Deprecated\) Working with decision trees](#)

Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision tree and conditions that must be satisfied before the decision tree is executed.

About this task

You can define preconditions for a decision tree. Preconditions can contain the following definitions:

- Variables that are available throughout the decision tree
- Conditions that must be satisfied before the decision tree is executed

Procedure

To define a precondition:

1. In the decision tree editor, click the **General** tab.
2. In the **Preconditions** section, press Ctrl+Spacebar.

The Content Assist box opens. You define variables and conditions in the Preconditions section in the same way as you would create variables and conditions for an action rule in the Intellirule editor. See [Building rules using the Intellirule editor](#) for details.

3. Save the decision tree.

The variables you defined are now available for creating condition nodes and branches in the decision tree, which means the decision tree cannot be evaluated before the conditions defined in the precondition are satisfied.

Parent topic: [\(Deprecated\) Working with decision trees](#)

Defining checking options

You can specify the tests that you want Rule Designer to conduct to identify problems with a decision tree.


About this task

You can define checking options at tree level, or at individual condition node level.

Procedure

To define checking options for the whole tree:

Choose whether you want to check options for the whole tree, or just a condition node:

To define checking options for the whole tree:	To define checking options on a condition node:
<div><div>a. Right-click anywhere in the decision tree editor, and then click  Decision Tree Properties. The Decision Tree Properties dialog opens.</div><div>b. In the Checking section, select the checks you require.</div><div>You can select whether you want gaps and/or overlaps to be checked, and the severity of checking reports.</div></div>	<div><div>a. In the decision tree editor, double-click the condition node. The Node Editor dialog opens.</div><div>b. In the Checking section, select which checks you require.</div><div>You can select whether you want gaps and/or overlaps to be checked.</div></div>

Parent topic: [\(Deprecated\) Working with decision trees](#)

Related information:

[Decision trees](#)

[Business Action Language \(BAL\)](#)

Viewing rules

You can view the details of a rule in a decision tree by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision tree editor.


About this task

You can view the text of a rule corresponding to an action. You can view the rule text in a tooltip, or in a dedicated viewing pane.

Procedure

To view rule text:

Choose whether you want to view the rule text in a tooltip or in a viewing pane:

To view rule text in a tooltip:	To view rules in a dedicated viewing pane:
<div>a. In the decision tree editor, place the cursor over the action set.</div> <div>The rule is displayed in a tooltip.</div>	<div>a. In the decision tree editor toolbar, click  Decision Tree Properties. The Decision Tree Properties dialog opens.</div> <div>b. In the Tree view section, select Show rules.</div> <div>c. Click OK.</div> <div>A new viewing pane opens at the bottom of the decision tree editor.</div> <div>d. In the decision tree editor, click the header of an action set.</div> <div>The corresponding rule is shown in the viewing pane.</div>

Results

Whichever way you choose to view the rules, if you defined preconditions for the decision tree, these are also shown.

Parent topic: [\(Deprecated\) Working with decision trees](#)

(Deprecated) Working with templates

You can create partially written rules and partially defined decision tables for use as templates, to simplify the task of creating rules and decision tables.

(Deprecated) Templates

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

Setting up an action rule template

You use the guided editor to create action rule templates. You leave placeholders to cater for variations, and you freeze the parts that should remain fixed.

Setting up a decision table template

You create decision table templates to enable the efficient creation of multiple decision tables that have similar content and structure.

Parent topic: [Authoring business rules](#)

(Deprecated) Templates

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

By defining it as such in the template, parts of rules that are created from a template can be frozen so that they cannot be edited.

Templates simplify the task of writing rules and decision tables, and are useful for:

- Creating action rules and decision tables that are already partially written.
- Restricting users from editing certain parts of partially completed rules.

A template applies only at the moment you create a rule or a decision table. If you subsequently modify the template, this will have no impact on rules or decision tables previously created from this template. You would need to update these rules or decision tables manually.

Parent topic: [\(Deprecated\) Working with templates](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Setting up an action rule template](#)

Setting up an action rule template

You use the guided editor to create action rule templates. You leave placeholders to cater for variations, and you freeze the parts that should remain fixed.

Overview

If you want to create a number of action rules with similar content and structure, you can create an action rule template, and protect (freeze) parts or all of a template so that they cannot be edited in the action rules created from the template.

Creating an action rule template

You can create an action rule template if you want to create a number of action rules with similar content and structure.

Freezing and unfreezing parts of an action rule template

You can protect certain parts of the template so that they cannot be edited in the action rules created from the template.

Freezing or unfreezing the whole template

How to freeze or unfreeze the whole template.

Propagating changes to a template's freeze/unfreeze settings

How to propagate changes to a template's freeze/unfreeze settings.

Parent topic: [\(Deprecated\) Working with templates](#)

Related concepts:

[\(Deprecated\) Templates](#)

Related reference:

[Business Action Language \(BAL\)](#)

Overview

If you want to create a number of action rules with similar content and structure, you can create an action rule template, and protect (freeze) parts or all of a template so that they cannot be edited in the action rules created from the template.

If you change the freeze/unfreeze settings in the template, you can use a specific command to reflect the change in the rules created from it. However, if you change the template content, you can reflect these changes only by manually updating the associated action rules.

You build action rules in a template in the same way that you create a normal action rule, except that:

- You leave parts of the rule empty to cater for variations in the action rules that are created from it
- You can use only the guided editor to build template rules and the action rules created from the template, as only this editor has the capability to set and apply freeze information. However, if text in a rule template or a rule derived from a template needs to be edited at a later date, you must use the Intellirule Editor.

Parent topic: [Setting up an action rule template](#)

Creating an action rule template

You can create an action rule template if you want to create a number of action rules with similar content and structure.

Before you begin

Procedure

To create an action rule template:

1. Select your rule project and on the **File** menu click **New** > **Action Rule Template**.

The New Action Rule Template wizard opens.

2. The default directory for templates to be saved is the templates folder in your rule project. If you want to save your template to a folder inside the templates folder, click **Browse** in the **Folder** field and navigate to the required folder.
3. Type the name of your action rule in the **Name** field.
4. Click **Finish**.

The new action rule template is shown in the folder you specified, and the Action Rule Template guided editor opens.

5. Using the guided editor, build the contents of the template, leaving those parts of the rule empty where variations are likely to occur.

In the Documentation section, you can enter information specific to the template. This information is not copied to the rules created from the template. To enter documentation that applies to all the rules created from the template, use the **documentation** property in the Properties view.

6. Save your work.

Results

You freeze parts of the rule template through the Action Rule Template guided editor.

Parent topic: [Setting up an action rule template](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating an action rule](#)

[Defining a folder structure for rule project items](#)

Related reference:

[Business Action Language \(BAL\)](#)

Related information:

[Building rules using the Intellirule editor](#)

[Building action rules using the guided editor](#)

Freezing and unfreezing parts of an action rule template

You can protect certain parts of the template so that they cannot be edited in the action rules created from the template.

About this task

In the guided editor, you can define a specific structure for the rules by freezing some controls in the template.

Procedure

To freeze and unfreeze controls:

1. In the Action Rule Template guided editor, right-click each control that you want to freeze, and click **Freeze**.

The controls that you have frozen are grayed out. Note that place holders are still shown in blue, indicating that you can still edit them.

Note:

If you keep the same structure, the freeze information is kept. However, the freeze information is lost if you change the structure of the rule.

2. To unfreeze a control, in the Action Rule Template Guided Editor, right-click the control that you want to unfreeze, and click **Unfreeze**.

The controls that you have unfrozen are shown in black.

Results

You can also freeze the whole template. This action freezes everything except the place holders. These are still shown in blue, indicating that you can still edit them.

Parent topic: [Setting up an action rule template](#)

Related information:

[Controls in the guided editor](#)

Freezing or unfreezing the whole template

How to freeze or unfreeze the whole template.

About this task

You can protect (freeze) the whole template so that it cannot be edited in the action rules created from the template.

Procedure

To freeze or unfreeze the whole template:

1. Right-click anywhere in the Action Rule Template Guided Editor, and click **Freeze All**.
2. To unfreeze the whole template, right-click anywhere in the Action Rule Template guided editor, and click **Unfreeze All**.

Note:

The **Freeze** and the **Freeze All** functions also freeze the black arrow icon ▼ that allows policy managers to change whether they edit a value or select an item. To allow them to make this choice, unfreeze the symbol.

Parent topic: [Setting up an action rule template](#)

Propagating changes to a template's freeze/unfreeze settings

How to propagate changes to a template's freeze/unfreeze settings.

About this task

If you change the freeze/unfreeze settings in the template, you can use a specific command to reflect the change in the rules created from it.

However, if you change the template content, you can reflect these changes only by manually updating the associated action rules.

Procedure

To propagate changes to a template's freeze/unfreeze settings:

1. In the Rule Explorer, select the template.
2. Right-click the template and click **Action Rule Template > Apply Freeze Information**.

The new freeze or unfreeze setting in the template is propagated through to the action rules created from the template.

Parent topic: [Setting up an action rule template](#)

Setting up a decision table template

You create decision table templates to enable the efficient creation of multiple decision tables that have similar content and structure.

About this task

You build decision tables in a template in the same way that you create a normal decision table, except that you leave parts of the decision table empty to cater for variations in the decision tables that are created from it.

Procedure

To create a decision table template:

1. Select your rule project and on the **File** menu click **New > Decision Table Template**.

The New Decision Table Template wizard opens.

2. The default directory for templates to be saved is the templates folder in your rule project. If you want to save your template to a folder inside the templates folder, click **Browse** in the **Folder** field and navigate to the required folder.
3. Type the name of your decision table in the **Name** field.
4. Click **Finish**.

The new decision table template is shown in the folder you specified, and the decision table editor opens.

5. Using the decision table editor, build the contents of the template, leaving those parts of the decision table empty where variations are likely to occur.

In the Documentation section, you can enter information specific to the template. This information is not copied to the decision tables created from the template. To enter documentation that applies to all the decision tables created from the template, use the **documentation** property in the Properties view.

6. Save your work.

Parent topic: [\(Deprecated\) Working with templates](#)

Related tasks:

[Creating a decision table from scratch](#)

Working with technical rules

Use the Technical Rule Editor to write technical rules in the ILOG® Rule Language.

Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping.

Creating a technical rule

You create a technical rule by specifying a name and source folder, and optionally specifying a package.

Using the Technical Rule Editor

You use the Technical Rule Editor to build technical rules.

Writing technical rules

Your technical rules must conform to the ILOG Rule Language (IRL) structure. Technical rules can contain variable definitions, conditions, and actions.

Parent topic: [Authoring business rules](#)

Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping.

Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

Variables in technical rules

You can define variables in the condition part of a technical rule and use it subsequently.

Technical rule conditions

Conditions are expressed using the class of object being tested followed by an operator. Actions are only executed if all the tests are satisfied.

Technical rule actions

Technical rule actions control execution, make loops, branch, handle exceptions and discriminate between execution branches.

Parent topic: [Working with technical rules](#)

Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
 - `evaluate`
 - `exists`
 - `from`
 - `in`
 - `instanceof`
 - `not`
 - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

Note: Rule Designer does not refactor technical rules. If you rename, move, or delete a technical rule, the references in other rule project artifacts are not updated.
--

Parent topic: [Technical rules](#)

Related concepts:

[Action rules](#)

[\(Deprecated\) Decision tree overview](#)

Related information:

[Overview: Ways to express business rules](#)

[Decision tables](#)

[Action rule editing errors and warnings](#)

Variables in technical rules

You can define variables in the condition part of a technical rule and use it subsequently.

```
when
{
    ?x: Fish(color==yellow; type==angel);
}
then
{
    retract ?x;
    insert Fish(yellow, shark);
}
```

The presence of the ?x variable at the beginning of the condition serves as a marker, which identifies any object that matches this condition. You can then refer to the matched object in other conditions or in the action part of the rule. Consequently, when the first action asks for the removal of the object referenced by the ?x rule variable, the yellow angel fish object is removed from the set of objects provided to the rule engine for execution.

Parent topic: [Technical rules](#)

Related concepts:
[Technical rule conditions](#)

Related tasks:
[Applying verbalization changes to business rules](#)

Related information:
[Technical rule actions](#)
[Working with technical rules](#)
[ILOG Rule Language \(IRL\)](#)

Technical rule conditions

Conditions are expressed using the class of object being tested followed by an operator. Actions are only executed if all the tests are satisfied.

The condition part of a technical rule is composed of a set of tests on Java™ objects. Tests are applied to each object provided to the rule engine for execution, and an object is said to match the condition when it passes all the tests in the condition.

Syntax of conditions

A condition starts with the name of the class concerned by the condition, such as `Film`. This class name is followed by tests, enclosed in parentheses, that express constraints on the attribute values for objects of this class.

For example, suppose that you want the rule to apply only to films showing after 8 in the evening. This condition is expressed as follows:

```
Film(showTime > 20.00);
```

This condition matches objects of the `Film` class whose `showTime` attribute is greater than `20.00`. Suppose that you then require the location of the film to be in Paris. This compound condition is expressed as follows:

```
Film(showTime > 20.00; location == Paris );
```

Operators

Each test starts by naming the attribute, followed by one of the operators in the following table.

Table 1. Condition operators

Operator	Description
<code>==</code>	equals
<code>!=</code>	does not equal
<code><</code>	is less than
<code>></code>	is greater than
<code><=</code>	is less than or equal to
<code>>=</code>	is greater than or equal to
<code>in</code>	is in
<code>!in</code>	is not in
a method call	method invocation

The test ends with an expression whose value is compared with that of the attribute.

A condition can contain several tests involving the same object attribute. For example, the following condition tests for films whose show times are between 8 and 10 in the evening:

```
Film(showTime > 20.00; showTime < 22.00);
```

Combination sign: &

The ampersand sign `&` is used as a combination sign, to associate several tests with the same object attribute involved in a condition. Therefore, to test for films whose show times are between 8 and 10 in the evening, you can express the condition as follows:

```
Film(showTime > 20.00 & < 22.00);
```

Here are two equivalent conditions, the second of which uses the `&` sign:

```
Film(produced > 1980; produced < 1990; showTime > 20.00; showTime < 22.00);
Film(produced > 1980 & < 1990; showTime > 20.00 & < 22.00);
```

Conjunction

Implicit AND relationships exist between all the tests in a rule. In other words, all the tests in the condition part of a rule must be satisfied before actions can be executed.

For example, consider the following conditions: “a film whose spoken language is English” and “a cinema whose location is Paris.” These conditions might be written in the following way:

```
Film(language == English);  
Cinema(location == Paris);
```

These two conditions are matched by pairs of Film and Cinema objects for which the language of the film object is equal to English AND the location of the cinema object is equal to Paris.

Parent topic: [Technical rules](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Technical rule actions

Technical rule actions control execution, make loops, branch, handle exceptions and discriminate between execution branches.

Action part of a technical rule

Actions can create, remove, and modify objects.

If/Else control

Use `if` statements to apply actions selectively. Use `else` statements to do a different set of actions when the expression is false.

Loops

You can add loops in actions to repeat the execution of action statements. You control the repetitions using `for` and `while` statements.

Branching statements

You can add `break` and `continue` branching statements in actions.

Exception handling statements

You can add statements that handle exceptions in actions.

Else clause with an evaluate statement

Technical rules with an evaluation statement in the condition part can have an optional `else` statement in the action part.

Parent topic: [Technical rules](#)

Action part of a technical rule

Actions can create, remove, and modify objects.

The keyword `then` is used to specify the start of the action part of a rule. The action part of a rule is composed of IRL statements that can, for example, create, remove, and modify objects.

For controlling the flow of the action part of a rule, the ILOG® Rule Language (IRL) includes statements for:

- controlling execution (`if/else`)
- making loops (`for`, `while`)
- branching (`break`, `continue`)
- handling exceptions (`try`, `catch`, `finally`)

These statements follow the Java™ language specification. You can also add an optional `else` clause in the action part of a technical rule.

Parent topic: [Technical rule actions](#)

Related concepts:

[If/Else control](#)

[Loops](#)

[Branching statements](#)

[Exception handling statements](#)

[Else clause with an evaluate statement](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

If/Else control

Use `if` statements to apply actions selectively. Use `else` statements to do a different set of actions when the expression is false.

The `if` statement enables rule actions to selectively execute other statements based on specific criteria.

For example, suppose that your rule prints debugging information based on the value of a Boolean variable named `DEBUG`. If `DEBUG` is `true`, your rule prints debugging information, such as the value of a variable `x`. Otherwise, your rule proceeds normally. A segment of code to implement the action might look like this:

```
then {
  if (?DEBUG) {
    System.out.println("DEBUG: x = " + ?x);
  }
}
```

This is the simplest version of the `if` statement: the block governed by the `if` is executed if a condition is true. Generally, the simple form of `if` can be written like this:

```
then {
  if (test) {
    statements
  }
}
```

If you want to do a different set of statements if the expression is false, use the `else` statement. For example, suppose your rule needs to do different actions depending on whether the user clicks the OK button or another button in an alert window. Your rule could do this by using an `if` and an `else` statement:

```
then {
  int ?i = 1;
  if ( ?i == 1 ) {
    System.out.println( " i = 1 " ) ;
  }
  else {
    System.out.println( " i <> 1 " ) ;
  }
}
```

The `else` block is executed only if the `if` part is false.

Parent topic: [Technical rule actions](#)

Related concepts:

[Branching statements](#)

[Exception handling statements](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Loops

You can add loops in actions to repeat the execution of action statements. You control the repetitions using `for` and `while` statements.

IRL includes statements for making loops in the action part of technical rules.

For loop

The `for` statement provides a compact way to iterate over a range of values.

The general form of the `for` statement can be expressed like this:

```
then{
    for (initialization; test; increment) {
        statements
    }
}
```

The expression `initialization` is used to initialize the loop. It is executed once at the beginning of the loop. The test expression determines when to terminate the loop. This expression is evaluated at the top of each iteration of the loop. When the expression evaluates to false, the loop terminates. Finally, `increment` is an expression that is invoked after each iteration through the loop. Here is an example:

```
then {
    int ?i = 1;
    int ?j = 1;
    for ( ?i = 1; ?i <= 3; ?i++ ) {
        for ( ?j = 1; ?j <= 3; ?j++ ) {
            System.out.println(" i =" + ?i + " j = " + ?j ) ;
        }
    }
}
```

Often `for` loops are used to iterate over the elements in an array, or the characters in a string. The following example uses a `for` statement to iterate over the elements of an array and print them:

```
then {
    for (?i = 0; ?i < ?arrayOfInts.length; ?i++) {
        System.out.println(?arrayOfInts[?i] + " ");
    }
}
```

While loop

You use a `while` statement to continually execute a block of statements while a condition remains true. The general syntax of the `while` statement is:

```
then {
    while (test) {
        statements
    }
}
```

First the `while` statement evaluates the `test` expression, which must return a boolean value. If the test returns true, then the `while` statement executes the statements associated with it. The `while` statement continues testing the test expression and executing its block until the test returns false. Here is an example of a `while` loop:

```
then {
    int ?i = 0 ;
    while ( ?i <= 3 ) {
        System.out.println( ?i ) ;
        ?i++ ;
    }
}
```

The following example uses a while statement in the action part of the rule to iterate over an array of elements in a variable named elements. The variable elements is bound to an attribute of a variable c, also named elements. The variable c is declared in the condition part of the rule as a collection of ManagedObject().

```
rule foundManagedObject {
  when {
    ?c: collect ManagedObject();
  }
  then {
    Enumeration ?elements = ?c.elements();
    while (?elements.hasMoreElements()) {
      System.out.println(elements.nextElement());
    }
  }
}
```

Parent topic: [Technical rule actions](#)

Related concepts:

[Branching statements](#)

[Exception handling statements](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Branching statements

You can add break and continue branching statements in actions.

IRL supports two branching statements: the break statement and the continue statement.

Break

The break statement terminates a for or a while loop when the statement is found. In a for loop, for example, the flow of control transfers to the statement following the enclosing for, as shown here:

```
then {
    int ?i = 1;
    int ?j = 1;
    for ( ?i = 1; ?i <= 3; ?i++ ) {
        System.out.println("\ni= " + ?i + ": ");
        for ( ?j = 1; ?j <= 3; ?j++ ) {
            System.out.println(" j = " + ?j) ;
            if ( ?i == ?j ) {
                break;
            }
        }
    }
}
```

The break statement ends only the loop in which it exists. If two loops are nested, a break in the inner loop exits the inner loop but not the outer loop. The output of the above example is shown here:

```
i= 1:
 j = 1

i= 2:
 j = 1
 j = 2

i= 3:
 j = 1
 j = 2
 j = 3
```

Continue

You use the continue statement to skip the current iteration of a for or while loop. Instead of ending the loop like the break statement, the continue statement skips all following statements in the loop body and executes the next iteration of the loop. The continue statement is demonstrated in the following example.

```
then {
    StringBuffer ?whitePaper = new StringBuffer(
        "The Case for Business Users of Information Technology");
    int ?max = ?whitePaper.length();
    int ?numSs = 0;
    int ?i = 0;
    for (?i = 0; ?i < ?max; ?i++) {
        if (?whitePaper.charAt(?i) != 's'){
            continue;
        }
        ?numSs++;
        ?whitePaper.setCharAt(?i, 'S');
    }
    System.out.println("Found " + ?numSs + " s's in the string.\n");
    System.out.println(?whitePaper);
}
```

Here is the output:

```
Found 6 s's in the string.
```


The CaSe for BuSineSS USerS of Information Technology

The example steps through a string buffer checking each letter. If the current character is not an s, the continue statement skips the rest of the statements in the loop and proceeds to the next iteration to test the next character. If it is an s, the rule increments a counter and converts the s to uppercase.

Parent topic: [Technical rule actions](#)

Related concepts:

[Loops](#)

[Exception handling statements](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Exception handling statements

You can add statements that handle exceptions in actions.

IRL provides mechanisms for reporting and handling exceptions. When an error occurs, the rule throws an exception. This means that the normal flow of the rule is interrupted, and the rule engine attempts to find an exception handler, that is, a block of code that handles a particular type of error. The exception handler generally does the necessary actions to recover from the error.

Exception setup

Before you can catch an exception, code somewhere must throw one. Any Java™ code can throw an exception.

A throw expression is any kind of expression whose type is assignable to the Java Throwable type or subclass. A throw expression can be specified either in the API or in a rule. When an exception is thrown it can be caught by the API or a rule—with a try-catch-finally statement or by any Java code—which causes the IRL code to execute.

Note:

A thrown exception is an `IlrUserRuntimeException` subtype of `IlrRunTimeException` which encapsulates the exception thrown by the throw statement.

- `IlrSystemRuntimeException` is thrown when Decision Server detects an error at runtime. This kind of exception cannot be recovered; it is used only for debugging purposes.
- `IlrUserRuntimeException`, on the other hand, could be used to reset the context and relaunch the rules.

The try-catch-finally statements

The try-catch-finally statements are:

- The try statement identifies a block of statements within which an exception might be thrown.
- The catch statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block. There can be as many statements following a try statement as needed. Each statement handles any and all exceptions that are instances of the class listed in parentheses, of any of its subclasses, or of a class that implements the interface listed in parentheses.
- The finally statement must be associated with a try statement and identifies a block of statements that is executed regardless of whether or not an error occurs within the try block. The finally statement is generally used to clean up after the code in the try statement. If an exception occurs in the try block and there is an associated catch block to handle the exception, control transfers first to the catch block and then to the finally block.

Here is the general form of these statements:

```
try {
    statements
}
catch (ExceptionType1 name) {
    statements
}
catch (ExceptionType2 name) {
    statements
}
...
finally {
    statements
}
```

IRL allows general exception handlers that handle multiple types of exceptions. However, more specialized exception handlers can determine what type of exception occurred and assist in the recovery of these errors. Handlers that are too general can make code more error prone by catching and handling exceptions that were not anticipated and for which the handler was not intended.

The following try statements demonstrate how to create an exception and throw it.

```
then {
```

```

try {
    method1(1) ;
    System.out.println("Call method1(1) was OK") ;
}
catch ( Exception e ) {
    System.out.println("Catch from method1(1) call") ;
}
try {
    method1(2) ;
    System.out.println("Call method1(2) was OK") ;
}
catch ( Exception e ) {
    System.out.println("Catch from method1(2) call") ;
    System.out.println("Exception details :-") ;
    System.out.println("Message: " + e) ;
}
}

```

The try-catch statements use a method call from a Java class to throw an exception when the variable passed into method1 is not equal to 1. Here is method1:

```

public static void method1(i) throws Exception{
    if (i == 1) {
        System.out.println("method1 - Things are fine \n") ;
    }
    else {
        System.out.println("method1 - Somethings wrong! \n") ;
        throw new Exception("method1 - Its an exception! \n") ;
    }
}

```

Exceptions thrown by any rule or Java code within the scope of the rule engine is caught by the try-catch statements for that specific exception.

The following CatchCustomer rule provides an example of a try-catch statement block capable of catching exceptions thrown by the ILOG Rule language and/or Java code. The rule uses two classes, Customer and Item, and two subclasses of the Java Exception class, TooHighExpense and IsNotMember.

```

rule CatchCustomer
{
    when {
        ?c: Customer();
        ?item: Item();
    }
    then {
        try {
            System.out.println("Customer " + ?c.getName() + " wants to buy "
                               + " item " + ?item.getLabel() + " for a price of " +
                               ?item.getPrice() + "$");

            IsMember(?c);
            ?c.buy(?item);
        }
        catch (TooHighExpense ex) {
            System.out.println("M/Mrs/Mz " + ?c.getName() + " you have already
                               bought:");

            int j = 0;
            for(j = 0; j<?c.getItems().size(); j++) {
                java.util.Enumeration ?i = ?c.getItem(j);
                System.out.println("    Item: " + ?i.getLabel() + " costs " +
                                   ?i.getPrice() + "$");
            }
            System.out.println("You have at your disposal " + ?c.getBudget() + "$");
            System.out.println("The current purchase is not allowed");
        }
        catch (IsNotMember ex) {
            System.out.println("M/Mrs/Mz " + ?c.getName() +

```

```

        ", you are not a member; would you like to
        subscribe?");
    }
}
}

```

In the example, a customer is given a budget. The customer is not allowed to buy items that exceed the allocated budget. If the customer attempts to buy items that exceed the budget, a TooHighExpense exception is thrown. In addition, to buy items the customer must be a member. If a nonmember attempts to buy an item, an IsNotMember exception is thrown.

The two Java classes IsNotMember and TooHighExpense used in the CatchCustomer rule are shown here:

```

public class IsNotMember extends Exception
{
    public IsNotMember(String name){
        super();
        this.customer = name;
    }
    public void printStackTrace(PrintWriter s){
        s.println("An IsNotMember exception has been caused by customer "
            + customer + ". We are sorry but you can not make a purchase "
            + "without being a member.");
        super.printStackTrace(s);
    }
    String customer;
};

public class TooHighExpense extends Exception
{
    public TooHighExpense(String name, int expense, int limit, int price){
        super();
        this.expense = expense;
        this.limit = limit;
        this.price = price;
        this.customer = name;
    }
    public void printStackTrace(PrintWriter s){
        s.println("A TooHighExpense exception has been caused by customer "
            + customer + ". For this customer, the current expense is " +
expense +
            " and the authorized budget is " + limit +
            ". The purchase of the item whose price is " + price
            + " is not allowed.");
        super.printStackTrace(s);
    }
    private int price;
    private int limit;
    private int expense;
    String customer;
};

```

Parent topic: [Technical rule actions](#)

Related concepts:

[Loops](#)

[Branching statements](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Else clause with an evaluate statement

Technical rules with an evaluation statement in the condition part can have an optional `else` statement in the action part.

When the condition part of a rule finishes with an `evaluate` statement, the action part of the rule might include an `else` clause. The `else` clause is optional. The last `evaluate` expression in the condition part acts as a discriminator, to choose between the `then` or the `else` execution branches. If the `evaluate` expression is true, the `then` part is executed; if it is false, the `else` part is executed.

If the rule ends with several `evaluate` statements, only the last one is used to discriminate.

Here is an example of a rule with an `else` statement:

```
rule myrule
{
  when
  {
    ?s : String(startsWith("Irl"));
    evaluate(?s.length() > 30);
  }
  then
  {
    out.println("Very long string");
  }
  else
  {
    out.println("String length is reasonable");
  }
};
```

Parent topic: [Technical rule actions](#)

Related concepts:

[Branching statements](#)

[Exception handling statements](#)

Related tasks:

[Applying verbalization changes to business rules](#)

Related information:

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Creating a technical rule

You create a technical rule by specifying a name and source folder, and optionally specifying a package.

About this task

Technical rules are rules written in the ILOG® Rule Language. They allow you to write specific loops in the action part of your rules.

Procedure

To create a technical rule:

1. Select your rule project and on the **File** menu click **New** > **Technical Rule**.

The New Technical Rule wizard opens.

2. Select the required Source folder.
3. If you want to select a rule package, click **Browse** in the **Package Name** field.
4. Type the name of the technical rule in the **Name** field.
5. Click **Finish**.

The new technical rule is shown in the Rule Explorer view and the Technical Rule Editor opens.

Parent topic: [Working with technical rules](#)

Using the Technical Rule Editor

You use the Technical Rule Editor to build technical rules.

Using Content Assist in a technical rule

Use Content Assist to build technical rules by selecting valid code from a list.

Errors in the Technical Rule Editor

The Technical Rule Editor dynamically checks the correctness of the code in a technical rule.

Organizing import statements

You organize import statements to check that all BOM classes that are used in the technical rule are correctly referenced. The Technical Rule Editor inserts missing import statements and removes redundant ones.

Opening declarations from the Technical Rule Editor

You can use the text in a technical rule to browse through BOM classes, methods, attributes, and functions.

Defining IRL editing settings

You can define preferences for editing in IRL. This applies to the editors for functions and ruleflow transitions.

Parent topic: [Working with technical rules](#)

Using Content Assist in a technical rule

Use Content Assist to build technical rules by selecting valid code from a list.

About this task

You use the Technical Rule Editor to write rules in the ILOG® Rule Language (IRL). To open the Technical Rule Editor, double-click a technical rule in the Rule Explorer.

The Technical Rule Editor lets you use Content Assist for the following items:

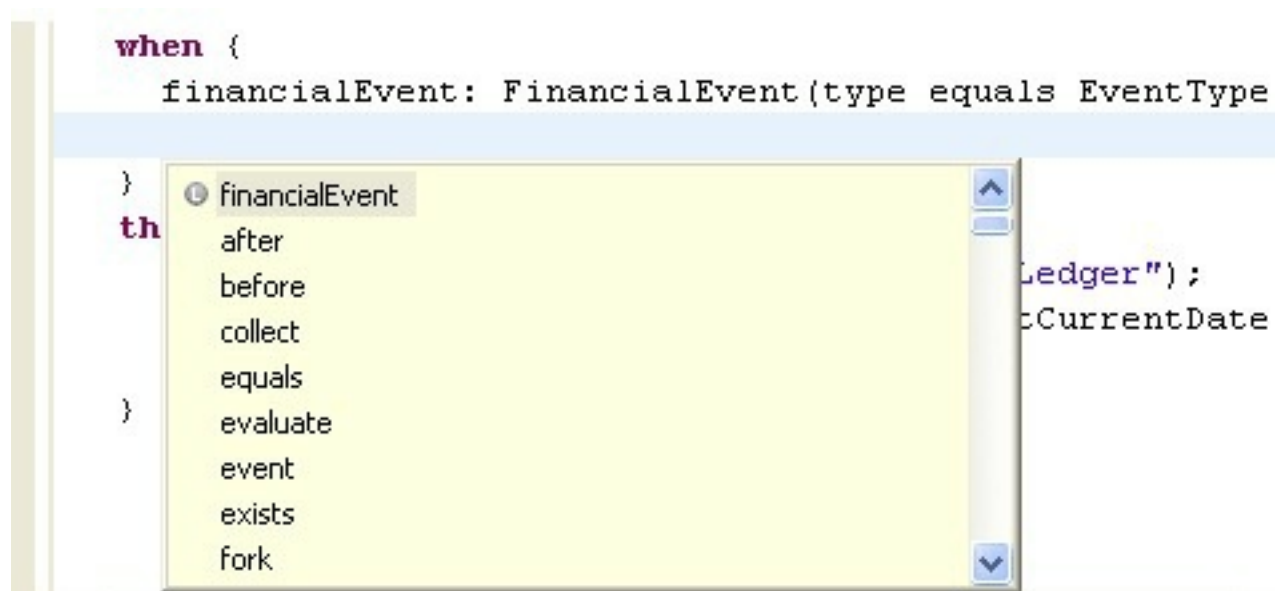
- IRL keywords
- Class names and package names
- BOM attributes, properties and methods
- Functions
- Rules and tasks
- Ruleset variables and parameters

Procedure

To complete your code:

1. Type the first letters of a statement and press Ctrl+Spacebar.

This opens a list from which you can select an item.



2. Select an item and press Enter. The item is added to the statement in the Technical Rule Editor.
3. Continue until you have created your technical rule.

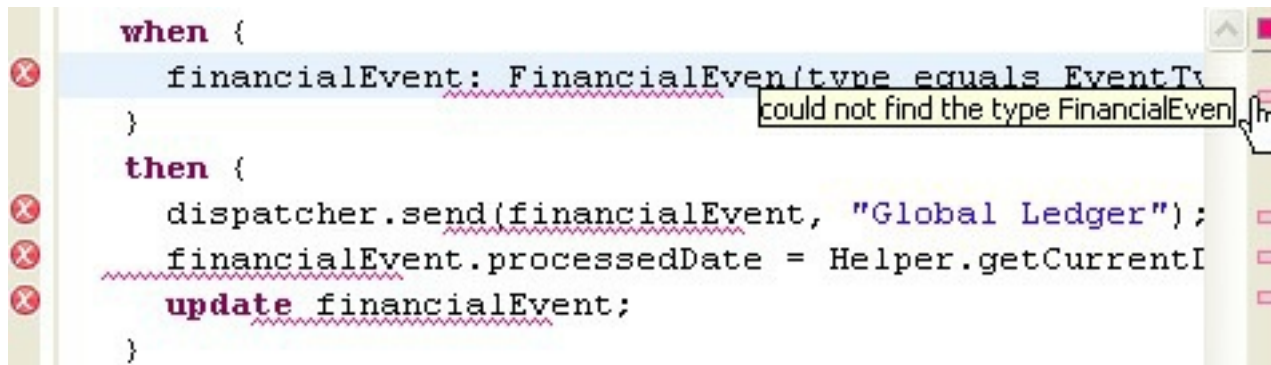
Parent topic: [Using the Technical Rule Editor](#)

Errors in the Technical Rule Editor

The Technical Rule Editor dynamically checks the correctness of the code in a technical rule.

As you type in the Technical Rule Editor, it shows errors and warnings:

- Invalid code is underlined.
- Errors are displayed in the Technical Rule Editor margin.
- Both the Technical Rule Editor margin and the underlined code have tooltips that contain the error messages.



Errors are underlined in red. Warnings are underlined in yellow.

Parent topic: [Using the Technical Rule Editor](#)

Related tasks:

[Organizing import statements](#)

[Opening declarations from the Technical Rule Editor](#)

[Defining IRL editing settings](#)

Related information:

[Technical rules](#)

[Using the Technical Rule Editor](#)

[ILOG Rule Language \(IRL\)](#)

Organizing import statements

You organize import statements to check that all BOM classes that are used in the technical rule are correctly referenced. The Technical Rule Editor inserts missing import statements and removes redundant ones.

About this task

The Technical Rule Editor can help you to write the required import statements for a technical rule. The import statements reference BOM classes that are used in the technical rule.

Procedure

To update the import statements:

Do one of the following actions:

- Select **Source** > **Organize Imports**.
- Press Ctrl+Alt+O.

Results

Either action adds any missing import statements and removes the redundant ones in the Imports section of the Technical Rule Editor.

Parent topic: [Using the Technical Rule Editor](#)

Related tasks:

[Opening declarations from the Technical Rule Editor](#)

[Defining IRL editing settings](#)

Related information:

[Errors in the Technical Rule Editor](#)

[Technical rules](#)

[Using the Technical Rule Editor](#)

[ILOG Rule Language \(IRL\)](#)

Opening declarations from the Technical Rule Editor

You can use the text in a technical rule to browse through BOM classes, methods, attributes, and functions.

Procedure

To browse from an item in the Technical Rule Editor to its declaration in another file:

Do one of the following actions:

- Select the item, and then select **Navigate > Open Declaration**.
- Press F3.
- Press Ctrl and click to switch to link mode.

In link mode, the words in the Technical Rule Editor are displayed as links when you hover over them. When you click a link, the element declaration opens in a separate editor.

Parent topic: [Using the Technical Rule Editor](#)

Defining IRL editing settings

You can define preferences for editing in IRL. This applies to the editors for functions and ruleflow transitions.

Procedure

To modify IRL editing preferences:

1. From the **Window** menu, click **Preferences**. (On Mac, click **Eclipse** > **Preferences**.)
2. Expand **Rule Designer** , and then click **IRL Editing**.
3. Select the required appearance and syntax coloring settings.
4. Select the required options for automatic text modifications.

Parent topic: [Using the Technical Rule Editor](#)

Writing technical rules

Your technical rules must conform to the ILOG® Rule Language (IRL) structure. Technical rules can contain variable definitions, conditions, and actions.

Writing the basic structure of a technical rule in IRL

When you write a technical rule, use the ILOG Rule Language structure.

Defining a variable

When you define a variable, use the ILOG Rule Language syntax.

Writing conditions in IRL

You can write simple, complex, existence, collection, nonexistence, and evaluate conditions. To write conditions, use the ILOG Rule Language syntax.

Adding tests to a condition

You can write standard tests, tests on attribute values, tests on values that are returned by methods, and tests on values that belong to sets of values. You can also write conditions in join tests.

Using objects that are not in memory in a condition

When you write conditions, you can use objects that are not in memory by referring to relations from objects that are available.

Writing actions in IRL

When you write rule actions, use the same syntax as a Java™ method body, except for the syntax for anonymous inner classes.

Writing a rule with an else action clause

When you write a rule with an else action clause, use the ILOG Rule Language syntax.

Notifying the rule engine of object changes

You can notify the rule engine of the availability of new objects, or when an object is updated or retracted.

Parent topic: [Working with technical rules](#)

Writing the basic structure of a technical rule in IRL

When you write a technical rule, use the ILOG® Rule Language structure.

About this task

In technical rules, when introduces conditions, and then introduces actions.

Procedure

To write a technical rule, use the following pattern:

```
rule rulename {
  when {
    // conditions:
    <condition>;
    [<condition>;]*
  }
  then {
    // actions
    [<action>;]*
  }
};
```

Note: Use a valid identifier name in Java™ syntax for the rulename, except when you use dots, which are accepted.

Example

```
rule financial.rules.CheckBalance {
  when {
    ...
  }
  then {
    ...
  }
};
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Defining a variable

When you define a variable, use the ILOG® Rule Language syntax.

Procedure

- To define a simple class condition variable, use the following syntax in the condition part of the rule:
`variable : ClassName();`

It defines a variable of type `ClassName`. This variable is never null and its value is shown in the rule instance tuple. The scope of this variable is the whole rule.

- To define a simple variable in a condition, use the following syntax in the test part of a condition:
`variable : <value>`

To use this syntax in a class condition, follow this pattern: `ClassName (variable : <value>)`. To use this syntax in an evaluate condition, follow this pattern: `evaluate (variable : <value>)`. It defines a variable that has the type of the value. The value of the variable can be null. The scope of this variable depends on the condition in which it is defined: If the variable is defined in an existential condition, its scope is limited to this condition. If the variable is defined in any other type of condition, its scope is the whole rule.

- To define a simple variable in the action part of the rule, use the following syntax: `int variable = <value>;`

It defines a variable. The scope of the variable follows the Java™ language scoping rules.

Example

In this example, the account and contract variables refer to objects in the working memory, objects of the Account BOM class and objects of the Contract BOM class. The balance is defined as the balance of an account in the condition part of the code. Similarly, the expiration is defined as the expiration date of the contract. The variable `b` is an example of a variable defined in the action part of the rule.

```
when {
    account: Account(balance : getBalance());
    contract: Contract(expiration : getExpirationDate());
}
then {
    int b = balance;
    System.out.println(contract + expiration);
    System.out.println(account + b);
}
```

This rule generates a list of all the available instances of each contract and its expiration date, and each account and its balance.

Parent topic: [Writing technical rules](#)

Related information:
[ILOG Rule Language \(IRL\)](#)

Writing conditions in IRL

You can write simple, complex, existence, collection, nonexistence, and evaluate conditions. To write conditions, use the ILOG® Rule Language syntax.

About this task

You can define several types of conditions.

Table 1. Conditions you can write in IRL

Condition name	Condition role
Simple condition	Checks for every instance of a class and runs the rule for every instance.
Existence condition	Checks that at least one instance of a class is available. If it is the case, the existence condition runs the rule once.
Collection condition	Gathers all instances of a class and runs the rule once per collection found.
Nonexistence condition	Checks that no instance of a class is available. If it is the case, the nonexistence condition runs the rule once. This condition is the negation of simple conditions, existence conditions, and collection conditions.
Evaluate condition	Runs global tests on one or several bound variables. <div>Note: You cannot put an evaluate condition at the beginning of the condition part of a rule. You must add one or more conditions that are not evaluate conditions before the evaluate condition.</div>

Procedure

- To write a simple condition, use the following pattern: `variable: ClassName(<tests>);`

In this example, the condition matches for all the accounts in the working memory and the rule runs all instances that match this condition. You can reuse the Account variable in other conditions or actions.

```
when {
  account : Account();
}
```

- To write an existence condition, use the following pattern: `exists ClassName(<tests>);`

In this example, the condition checks that at least one account exists inside the working memory. If it is the case, the rule runs once.

```
when {
  exists Account();
}
then {
  ...
}
```

- To write a collection condition, use the following pattern: `collection: collect ClassName(<tests>);`

In this example, the condition gathers all the accounts in a collection and the rule runs once for all the accounts. You can reuse the *accounts* variable as a collection in other conditions or actions.

```
when {
  accounts : collect Account();
}
then {
  out.println(accounts.size());
  ...
}
```


- To write a nonexistence condition, use the following pattern: `not ClassName(<tests>);`

In this example, the condition checks that no report exists in the working memory. If it is the case, the rule runs once.

```
when {  
    not Report();  
}  
then {  
    ...  
}
```

- To write an evaluate condition, use the following pattern: `evaluate (<test> [<test>]*);`

In this example, the condition gathers all the sales and purchases of a stock. If the result of a statistics computation of these two groups is less than two percent of the stock value, the condition runs the rule.

```
when {  
    stock : Stock();  
    purchases : collect Purchase(getStock()==stock);  
    sales : collect Sales(getStock()==stock);  
    evaluate ( Statistics.variance(purchases, sales) < 0.02 *  
              stock.getValue());  
}
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Defining a variable](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Adding tests to a condition

You can write standard tests, tests on attribute values, tests on values that are returned by methods, and tests on values that belong to sets of values. You can also write conditions in join tests.

About this task

You can add tests to conditions when you write technical rules in IRL.

Procedure

- To add standard tests to a condition, use the following pattern:

```
[variable:] ClassName(<test> [; <test>]*);  
not ClassName(<test> [; <test>]*);  
collect ClassName(<test> [; <test>]*);  
exists ClassName(<test> [; <test>]*);
```

- To write a test is based on the attribute value of an object that is bound in the current condition, use the following pattern:

```
attribute <operator> <value>  
attribute booleanmethod <value>  
attribute.method (<arguments>)  
<operator> <value>
```

In this example, the rule applies to all accounts whose balance is over 2,000, with a currency in Euros (EUR) and a date greater than the current date.

```
when {  
  account: Account(  
    balance > 2000;  
    currency equals Currency.EUR;  
    date.compareTo(currentDate) > 0);  
}
```

- To write a test that is based on the return value of the method that is called on an object that is bound in the current condition, use the following pattern:

```
non-void-method(<arguments>) <operator> <value>  
non-void-method(<arguments>) booleanmethod <value>  
// only in case of a method with a single argument, returning a Boolean:  
attribute.method(<arguments>)<operator> <value>
```

In this example, the rule applies to all accounts whose balance is over 2,000, with a currency in Euros and a date greater than the current date. This test is the same as the test on attribute values, but it uses methods instead.

```
when {  
  account: Account(  
    getBalance()>2000;  
    getCurrency() equals Currency.EUR;  
    getDate().compareTo(currentDate) > 0);  
}
```

- To write a test based on a value that belongs (in) or not (!in) to a set of values, use the following pattern:

```
<value> in|!in <group>  
<value> in|!in {value1 [,value]*}
```

Where <group> is a collection or an array.

In this example, the rule applies for all accounts whose currency is in Euros or U.S. dollars (USD), and for all contracts whose status does not belong to the list of returned methods.

```
when {
```

```
account : Account(getCurrency() in { Currency.EUR,
                                     Currency.USD});
contract : Contract( status !in getBlockedStatusList() );
}
```

- To write a condition with a join test, use the following pattern:

```
variable1: ClassName1(<tests>);
variable2: ClassName2(<tests using variable1>);
[variableP: ClassNameP(<tests using variable1, 2, ...>);]
```

Note: A join test uses one or more variables that were defined in previous conditions.

In this example, the rule applies for all purchases that are related to a contract, and for which an account exists in the same currency as the purchase.

```
when {
  purchase : Purchase(pcurrency : getCurrency());
  contract : Contract(getID() equals
                      purchase.getContractID());
  Account(getContractID() equals contract.getID();
          getCurrency() == pcurrency);
}
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Writing conditions in IRL](#)

Using objects that are not in memory in a condition

When you write conditions, you can use objects that are not in memory by referring to relations from objects that are available.

Procedure

To use objects that are not in memory when you write a condition, use the following pattern:

```
<condition> in <group>  
<condition> from <instance>
```

In this pattern, *<group>* is an expression that returns an object array, a collection, or an enumeration, and *<instance>* is an object.

Example

In this example, `contract` is a group of objects that is in the working memory. The condition can apply to the object `account.getCurrency` because of its relation with `contract.getAccounts`. With this condition, the rule executes all the objects.

```
when {  
    contract : Contract();  
    account : Account() in contract.getAccounts();  
    currency : Currency() from account.getCurrency()  
}
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Writing actions in IRL

When you write rule actions, use the same syntax as a Java™ method body, except for the syntax for anonymous inner classes.

All bound variables that are defined in the rule, all ruleset parameters, all public static methods and attributes, and all functions are visible in the action scope. Some specific variables are always available in the action scope:

- `?context` is the current rule engine
- `?instance` is the current rule instance

As in a Java method body, all public classes of the current class loader are visible (through an import or with their fully qualified name).

Example

```
then {
    System.out.println("The account " + account + " has no
                        money in it");
    Report report = new Report(account);
    dispatcher.send(report); // dispatcher is a ruleset parameter
}
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Writing conditions in IRL](#)

[Writing a rule with an else action clause](#)

Writing a rule with an else action clause

When you write a rule with an else action clause, use the ILOG® Rule Language syntax.

About this task

When the condition part of a rule finishes with an evaluation, the action part of the rule might include an else clause. The last evaluation within the condition part acts as a discriminator to choose between the then or the else execution branches. If the evaluation expression is true, the then part is executed. If it is false, the else part is executed.

If the rule ends with several evaluations, only the last one is used as the discriminator. The else clause is optional, and cannot be used when there is no evaluation at the end of the condition part.

Procedure

To write a rule with an else action clause, use the following pattern:

```
rule rulename {
  when {
    // conditions:
    <condition>;
    [<condition>;]*
    evaluate (<then/else discrimination test>);
  }
  then {
    // "then" actions
    [<action>;]*
  }
  else {
    // "else" actions
    [<action>;]*
  }
}
```

Example

In this example, if the evaluation expression is true, the rule executes the then action and charges a penalty. If the evaluation is false, the rule executes the else action and computes a bonus.

```
rule financial.rules.CheckBalance {
  when {
    account: Account();
  }
  evaluate (account.getBalance() < 0 );
  then {
    account.chargePenalty();
  }
  else {
    account.computeBonus();
  }
};
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Notifying the rule engine of object changes](#)

Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

Notifying the rule engine of object changes

You can notify the rule engine of the availability of new objects, or when an object is updated or retracted.

About this task

Notification of object changes depends on the execution mode:

- When the rule engine runs in RetePlus mode, it instantly recomputes the matching rules and reschedules an internal agenda.
- When the rule engine runs in sequential or Fastpath mode, it uses the new status of the objects when you enter a new rule task.

Procedure

1. Use the following code phrase structure: `insert object;`

This statement notifies the RetePlus algorithm that a new object is available. Rules that depend on this kind of object run when the objects meet the rules conditions.

2. Use the following code structure: `insert ClassName(<constructor_arguments>) [{statement1; ... statement}] ;`

3. If the rule engine runs in RetePlus mode, use the following code phrase structure: `update object;`

This statement notifies the RetePlus algorithm that rules that depend on this object must be reevaluated. Some of them no longer run, and others start running.

4. If the rule engine runs in RetePlus mode, use the following code phrase structure: `retract object;`

This statement notifies the RetePlus algorithm that this object is no longer available. The rules that depend on this object no longer run.

Results

The following code presents the global pattern to notify the rule engine of object changes:

```
insert object;
insert ClassName(<constructor_arguments>)
  [{statement1; ... statement}] ;
update object;
retract object;
```

Example

In the then action part of this example, the RetePlus algorithm is notified that the new object penalty is available, and that the account object is no longer available. As a result, rules can run for penalty objects, but not for account objects anymore. In the else action part, the RetePlus algorithm is notified that the rules that depend on the account object must be reevaluated. As a result, some of these rules stop running and others start running.

```
rule financial.rules.CheckBalance {
  when {
    account: Account();
    evaluate (account.getBalance() < 0 );
  }
  then { // compute a penalty
    Penalty penalty = new Penalty(account);
    insert penalty;
    retract account;
  }
  else { // compute a bonus
    account.computeBonus();
    update account;
  }
};
```

Parent topic: [Writing technical rules](#)

Related tasks:

[Adding tests to a condition](#)

Related information:

[Technical rules](#)

Working with functions

You can create a function in a rule project to share code procedures across more than one element of a ruleset. You express a function in ILOG® Rule Language (IRL), and its code is evaluated when the ruleset runs.

Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

Creating functions

You write a function by using the ILOG Rule Language (IRL).

Parent topic: [Authoring business rules](#)

Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG Rule Language (IRL) and its code is evaluated when the ruleset runs.

Note: Rule Designer does not refactor functions. If you rename, move, or delete a function, the references in other rule project artifacts are not updated.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)
{
    code
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the return keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of `return` that does not return a value:

```
return;
```

Note: A function is not required to declare in its `throws` statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

Parent topic: [Working with functions](#)

Related tasks:

[Creating functions](#)

Related reference:

[function](#)

Related information:

[Default constructor dynamic methods](#)

Creating functions

You write a function by using the ILOG® Rule Language (IRL).

About this task

A function is a procedural item in a ruleset. You write functions in the IRL by using the Function Editor in Rule Designer.

Procedure

To write a function:

1. Right-click an item in the Rule Explorer, and then on the **New** menu, select **Function**.
2. In the New Function dialog box, give the following information:
 - **Function Name**
 - **Source Folder**
 - **Package Name**

Decision Server creates the function file and opens the Text view of the new function. The Rule Explorer tree displays the new function name.

3. Click **Finish**.
4. Click the **Edit Signature** link to open and edit the function signature. You can add parameters and change the function type.
5. Click the **Text** tab to return to the function text view.
6. Enter the function code in the **Content** area.

Parent topic: [Working with functions](#)

Related concepts:
[Functions](#)

Related reference:
[function](#)

Related information:
[Default constructor dynamic methods](#)

Applying a category filter

You can apply a category filter to specify which categories of elements can be used in action rules, decision tables, and decision trees.

Before you begin

You must first define categories at the rule project level before you can apply them to rule artifacts.

About this task

Apply a category filter to specify the categories to use on a specific rule artifact.

Procedure

1. Open the action rule editor, the decision table editor, or the decision tree editor.
2. On the first page of the editor, in the Category Filter section, click **Edit**.
3. In the Category Filter dialog, select a category in the **All categories** field, and then click > > to move it to the **Selected categories** field. You can also double-click it to move it from one field to the other.
4. Click **OK**.

The categories you selected are now applied to the business rules and are listed in the Category Filter section of the editor.

Parent topic: [Authoring business rules](#)

Related concepts:

[Categories](#)

Related tasks:

[Defining and assigning categories](#)

Correcting errors by using Quick Fix

You can use the Eclipse Quick Fix feature in the Intellirule Editor, the decision table editor, and the decision tree editor. Quick Fix messages offer suggestions to automatically correct detected errors.

About this task

The Quick Fix feature in the Intellirule editor is supported only in the English version of the product.

In some cases, Quick Fix messages let you make the following changes:

- Replace one element by another
- Remove incorrect elements
- Declare new variables in the definitions part of the rule
- Add elements to the BOM

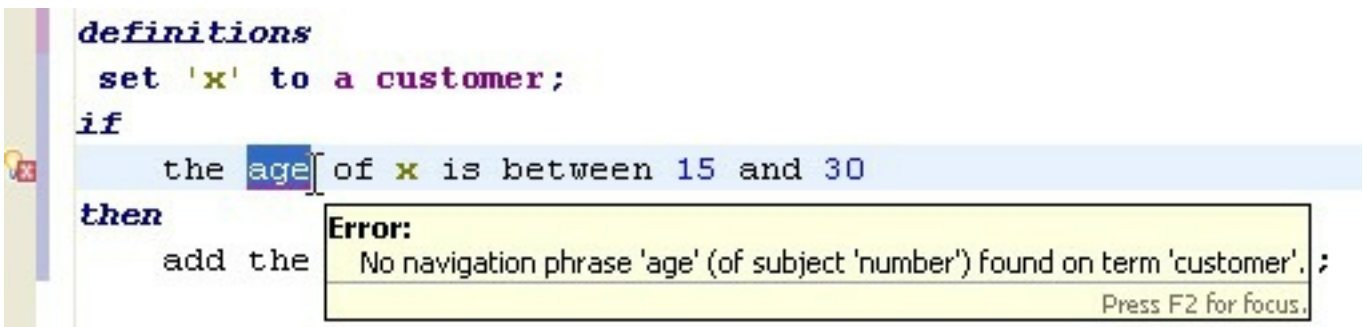
In the decision table editor and the decision tree editor, Quick Fix can also detect and let you add missing domain values.

To access the proposed quick fixes, click the light bulb icon  or place your cursor on the error and press Ctrl-1. The messages provide a list of possible corrections from which you can select the appropriate fix.

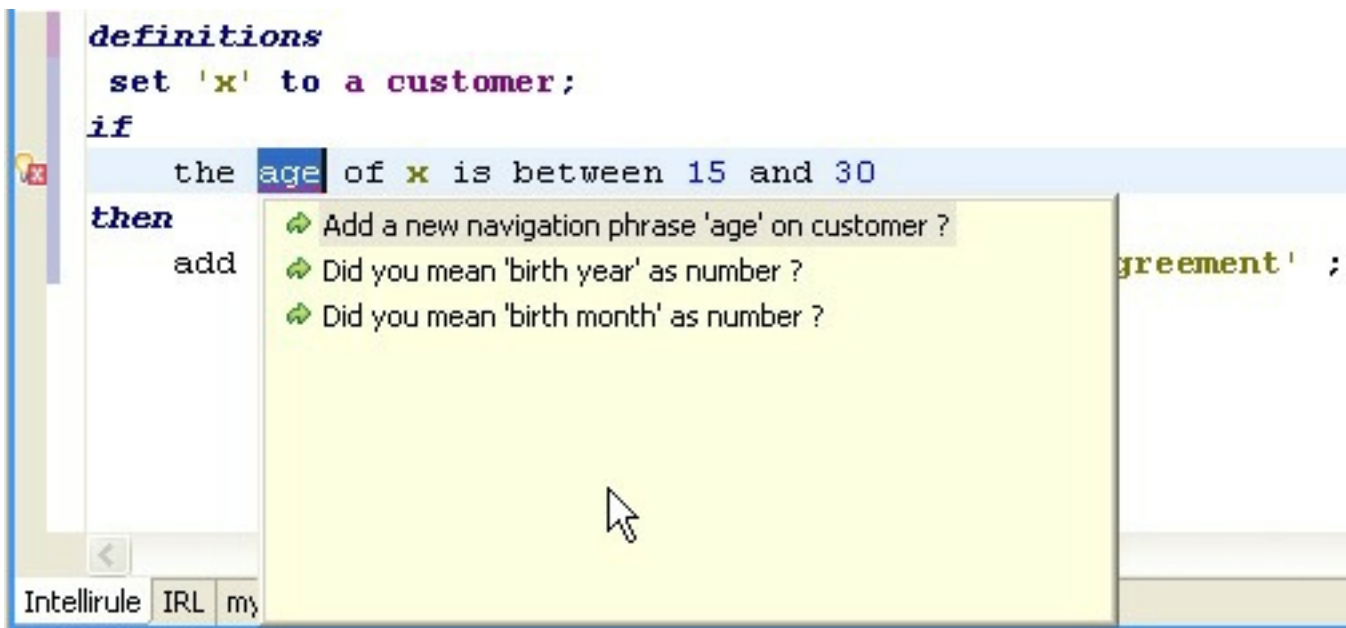
Procedure

To auto-correct an error from the quick fix message:

1. Put your cursor on the error indicator to display the error messages.



2. Click the light bulb icon or move your cursor to the error in the rule, and then press Ctrl-1 to open the quick fix message.



In this example, the **customer** object does not have an **age** attribute, so it must be added to the BOM in order for it to be used in the rule.

Note: To access quick fixes, you must correct errors in the order in which they are shown.

3. From the list of suggestions, click the appropriate fix.

In this example, selecting **Add a new navigation phrase 'age' on customer** opens the Add New Phrase wizard. In the wizard, you can choose to create a navigation phrase or an action phrase for the new element when you add it to the BOM.

New Phrase Wizard

Add New Phrase Wizard
Add a new Phrase to the selected BOM.

BOM Entry:

Class:

Type:

Attribute:

☒ Navigation Phrase
i Create a navigation phrase corresponding to the selected attribute.

☐ Action Phrase
i Create an action phrase corresponding to the selected attribute.

When you finish, the error is resolved and you can continue building your rule.

Parent topic: [Authoring business rules](#)

Applying verbalization changes to business rules

If you change the verbalization of a business element used in a rule, you can refactor the business rules that use the business element to take your changes into account.

About this task

When you change the verbalization of a business element used in a rule, Rule Designer prompts you to specify whether you want the rules that use the business element to be refactored to take your changes into account. If you decide not to do refactoring, the affected rules are not modified and syntax errors are reported in the Problems view.

If you want to change the subject of a phrase (referenced in curly braces in the navigation phrase template in the Member Verbalization area on the Member page of the BOM Editor) and benefit from the refactoring capability to update the rules that use them, you should make the change in the Edit Term dialog box, which you access by clicking **Edit the subject used in phrases** in the Member Verbalization area. If you make the change directly in the **Template** field, Rule Designer treats it as a change of the phrase template, which might cause an incorrect refactoring that leads to erroneous rules.

Refactoring is also executed after you modify ruleset parameters and variables that are used in business rules.

Note: Rule refactoring relies on business rules being compiled by the Eclipse build after modification. If you disable the Eclipse automatic build option, refactoring might not be executed when you modify the vocabulary.

Refactoring does not preserve rule formatting such as white spaces, tab spaces and new lines. Refactoring results in the text in a rule being regenerated, and this process might not preserve existing formatting.

Rules that have errors or that have not been saved might not have been refactored properly.

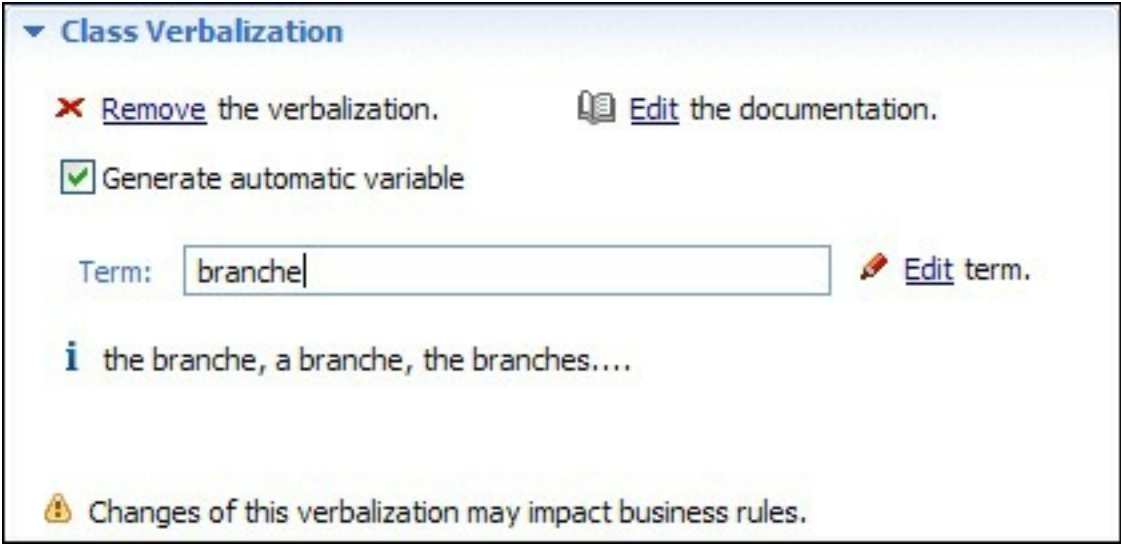
Rule refactoring is automatically activated when you save the BOM. You can also explicitly execute it from the Package page of the BOM Editor.

Procedure

To execute rule refactoring:

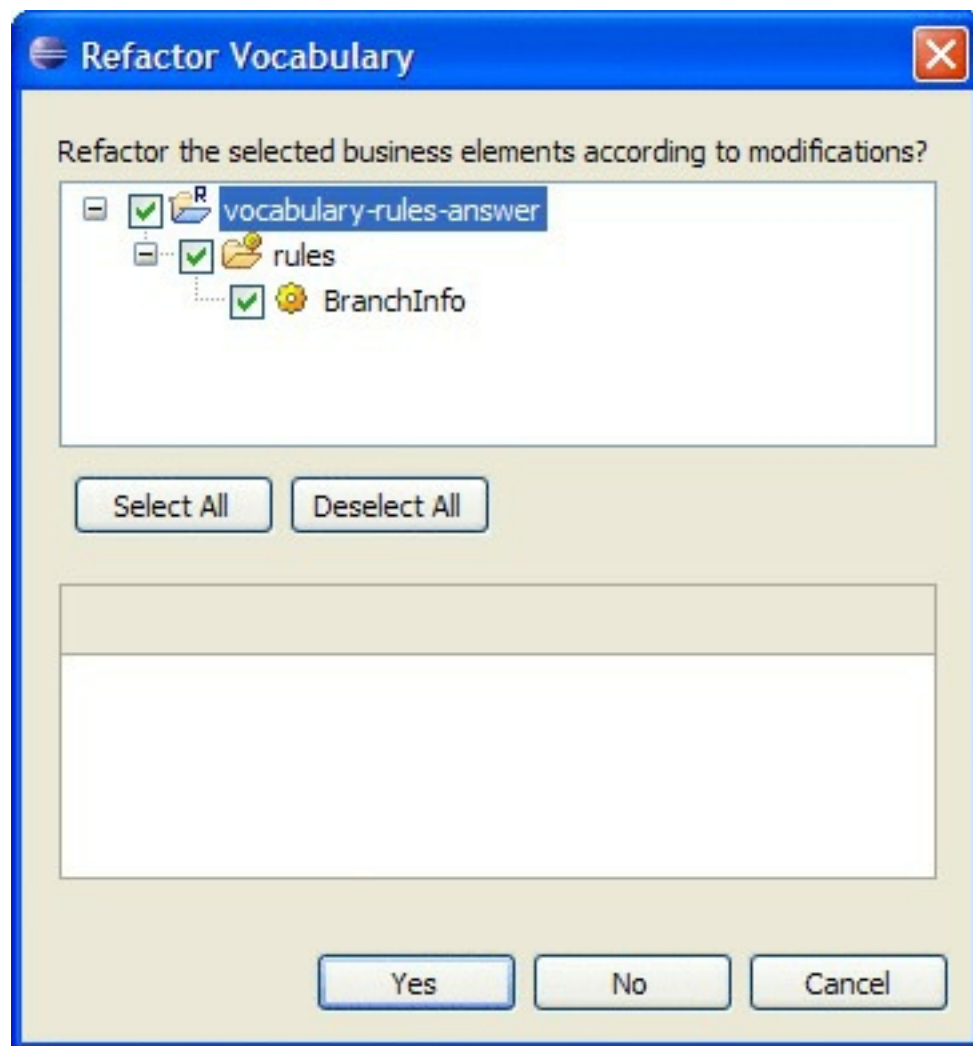
1. In the BOM Editor, modify the verbalization of a term or phrase that is used in a business rule.

The BOM Editor displays the following warning:



2. Save the BOM.

The Refactor Vocabulary dialog opens.



3. In the Refactor Vocabulary dialog, select the business rules you want to be refactored, and click **Yes**.

Results

If you want to see refactoring history, in the **Project** menu click **Properties > Refactoring History**.

Parent topic: [Authoring business rules](#)

Related information:

[Overview: Ways to express business rules](#)

[Updating the BOM when the XOM changes](#)

Reviewing a rule project

Manage rule project items and run queries and reports using the features provided in Rule Designer.

Overview: Querying, analyzing and reporting

Rule Designer provides functionality to search, query, and analyze rule projects, and generate reports on them.

Searching

Using the Search feature of Rule Designer, you can search for rule artifacts, XOM elements, and BOM elements of a selected project. You can view the results in the Search view.

Querying

You can filter your rules by creating and running queries in Rule Designer.

Analyzing

Rule analysis relies on consistency checking and completeness analysis. Analyze your rule projects to find ambiguities, conflicts, and missing rules, and fix them. Using Rule Analysis view options, you can fine tune the process.

Generating Rule Project Statistics reports

You can easily generate a report to get an overview of rule projects and of the artifacts that they contain or reference.

Parent topic: [Designing projects for rule authoring](#)

Overview: Querying, analyzing and reporting

Rule Designer provides functionality to search, query, and analyze rule projects, and generate reports on them.

After you have created your rule project or if a rule project has been modified, you can review it by exploring and analyzing its contents.

In Rule Designer, you can do handle rule projects in the following way:

- Search through your project: see [Searching](#) and [Querying](#).
- Analyze the project rules using the Rule Analysis view: see [Analyzing](#).
- Generate reports using the three BIRT report templates provided in Rule Designer.
- Obtain statistics on the rule projects that are opened in your workspace, and on the artifacts that the rule projects contain: see [Generating Rule Project Statistics reports](#).

Parent topic: [Reviewing a rule project](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Analyzing a rule project](#)

Searching

Using the Search feature of Rule Designer, you can search for rule artifacts, XOM elements, and BOM elements of a selected project. You can view the results in the Search view.

[Searching rule project elements](#)

To search for packages, rule files, or rule project elements in a rule project, you can specify some scoping options such as case sensitiveness, project dependencies, and type of rule artifact.

[Searching the execution object model](#)

You can search for XOM classes and packages or limit the search to the current XOM.

[Searching the business object model](#)

You can search through BOMs or for a specific BOM.

Parent topic: [Reviewing a rule project](#)

Searching rule project elements

To search for packages, rule files, or rule project elements in a rule project, you can specify some scoping options such as case sensitiveness, project dependencies, and type of rule artifact.

[Searching for packages and rule files](#)

Using the search feature, you can search for packages and rule files throughout a rule project.

[Searching for a project element](#)

Using the search feature, you can define a scope to search only some rule project elements.

Parent topic: [Searching](#)

Related tasks:

[Generating Rule Project Statistics reports](#)

[Finding rule dependencies](#)

Searching for packages and rule files

Using the search feature, you can search for packages and rule files throughout a rule project.

About this task

You can use the Search dialog to search packages and rule files for some rule artifacts that you specify by selecting the appropriate options.

Procedure

1. In the **Search** menu, select **Search** to open the Search dialog, and then click the **Rule Search** tab to open the Rule Search page.

The Rule, BOM, and XOM searches are Eclipse-specific.

2. Specify patterns to find rules, tasks, functions, and rule packages in the **Search string** field.

Use * for a multi-character wildcard and ? for a single-character wildcard. Using wildcards, you can specify all or part of the name of the rule, task, function, or rule package that you are searching for.

3. Select the **Case sensitive** box if you want your search to differentiate lowercase from uppercase characters in names.
4. Select **Include referenced projects** if you want to conduct the search on the current project and its dependencies.

If you do not select this option, the search is only done on the current project.

5. In the **Search for** area, select the type of element that you want to locate.
6. In the **Limit to** area, click **Name** if you want the search to be carried out on the names of rule, functions, tasks, and rule packages, or click **Body** for rules and functions.
7. Click **Search**.

Results

The results are displayed in the Search view of the Rule Designer Eclipse window.

Parent topic: [Searching rule project elements](#)

Related tasks:
[Searching for a project element](#)

Related information:
[Searching](#)

Searching for a project element

Using the search feature, you can define a scope to search only some rule project elements.

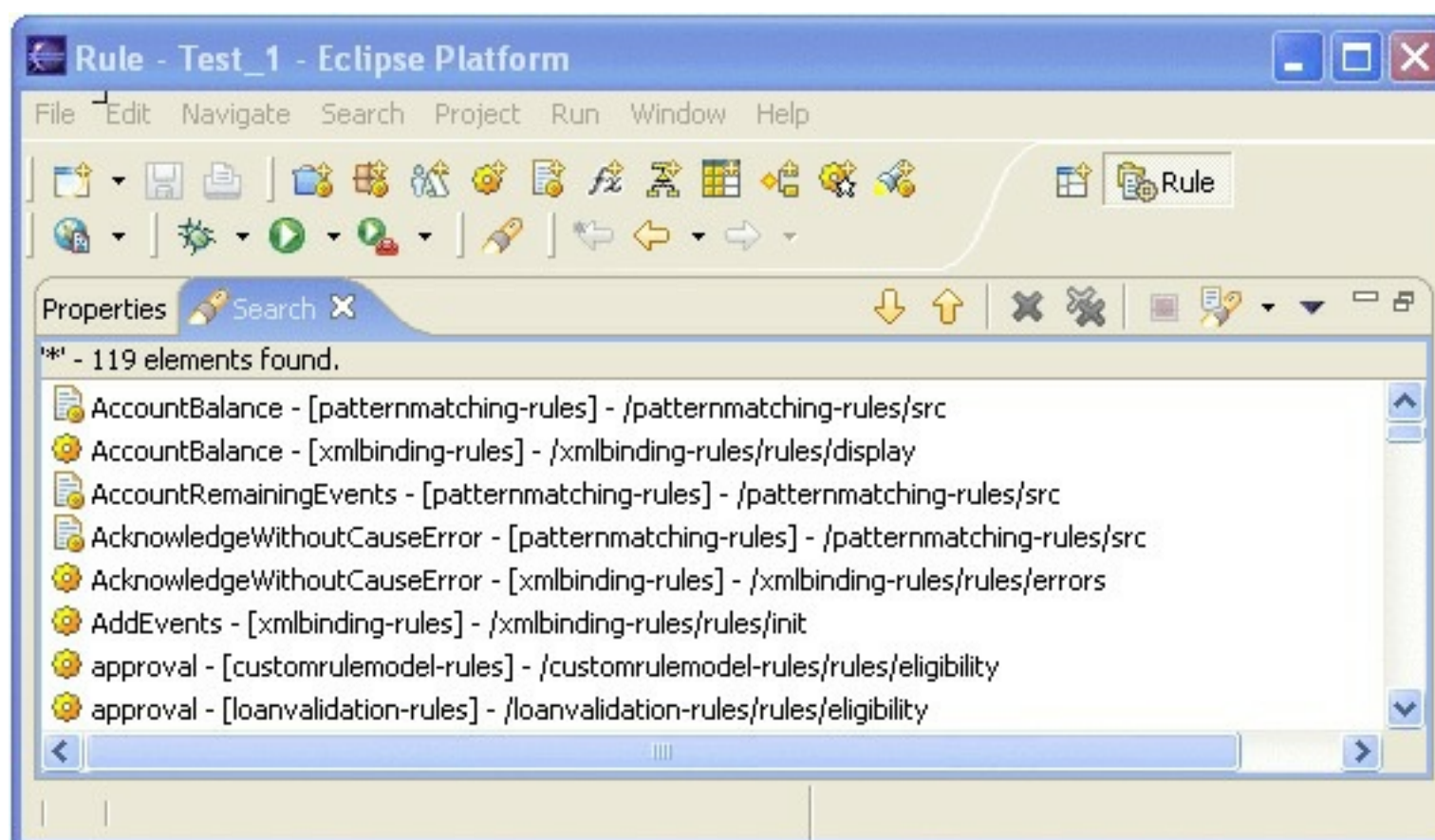
About this task

In the Search dialog, you can define a scope to search specific rule project elements in a project.

Procedure

1. In the Rule Explorer, select the rule project element.
2. Open the Rule Search page in the Search dialog and set up the search.
3. Select the scope in the Scope area.
 - To search in all the rule projects in the workspace, select **Workspace**.
 - To search on specific rule projects, select **Selected Resources**.
4. Click **Search**.

The results are displayed in the Search view.



For every match, the Search view shows the full name and the path to the element found.

5. To open the file where an element is defined, double-click the item in the Search view.

Parent topic: [Searching rule project elements](#)

Related tasks:

[Searching for packages and rule files](#)

Related information:

[Searching](#)

Searching the execution object model

You can search for XOM classes and packages or limit the search to the current XOM.

[Searching for XOM classes and packages](#)

In the Search dialog, you can search for XOM classes and packages in the workspace.

[Searching the current XOM](#)

You can limit the search to the current XOM.

Parent topic: [Searching](#)

Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

Related tasks:

[Generating Rule Project Statistics reports](#)

Related information:

[Searching the business object model](#)

[Searching rule project elements](#)

Searching for XOM classes and packages

In the Search dialog, you can search for XOM classes and packages in the workspace.

About this task

You can search for XOM classes and packages.

Procedure

1. In the **Search** menu, select **Search**, and then click the **XOM Search** tab.
2. Specify patterns to find classes and packages in the **Search string** field.

Use * to specify any string and ? to specify any character. Using wildcards, you can specify all or part of the name of the classes or packages that you want to find.

3. Select the **Case sensitive** box if you want your search to differentiate lowercase from uppercase characters in names.
4. Click **Search**.

The results are displayed in the Search view.

Parent topic: [Searching the execution object model](#)

Related tasks:

[Searching the current XOM](#)

Related information:

[Searching](#)

Searching the current XOM

You can limit the search to the current XOM.

About this task

If you do not want to search for all the XOM classes and packages in the workspace, you can search only the current XOM.

Procedure

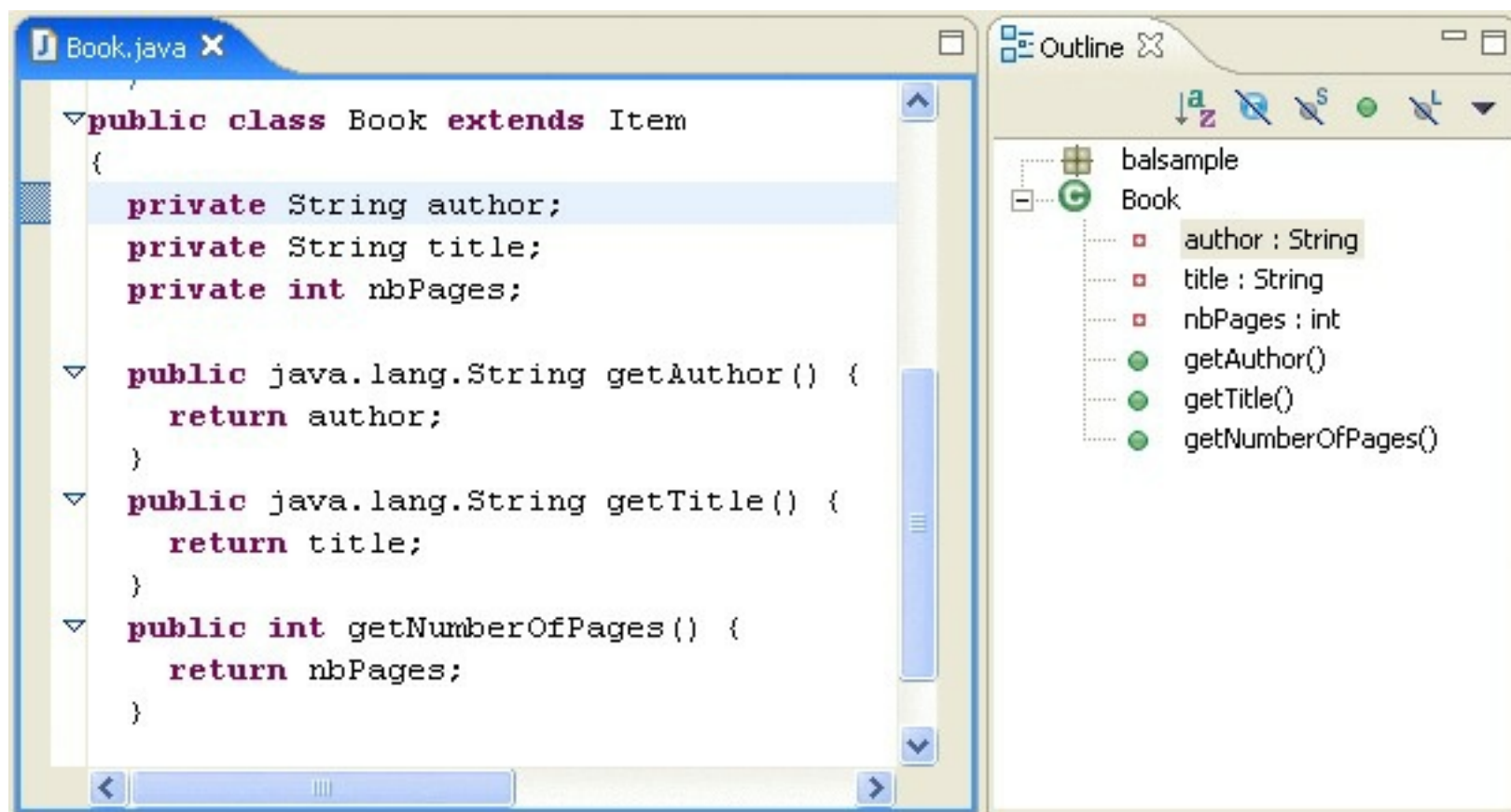
1. Highlight the rule project in the Rule Explorer.
2. Open the XOM Search page in the Search dialog and set up the search.
3. Select the scope in the Scope area:
 - To search on all XOMs in the workspace, select **Workspace**.
 - To search on XOMs for specific rule projects, select **Selected Rule Projects**.
4. Click **Search**.

The results display in the Search view.

For every match, the Search view shows the fully qualified name, the rule project, and the JAR file or class folder where the match was found.

5. To open the source file of a match for a class, double-click the match item in the Search view.

If the source is attached to the corresponding XOM reference, the source file is opened. Otherwise, an editor opens for you to set the source.



Parent topic: [Searching the execution object model](#)

Related tasks:

[Searching for XOM classes and packages](#)

Related information:

[Searching](#)

Searching the business object model

You can search through BOMs or for a specific BOM.

[Searching the BOM classes and packages](#)

You can search for BOM classes and packages or limit the search to the current BOM.

[Searching the current BOM](#)

In the Search dialog, you can limit the search to the BOM of the selected rule project.

Parent topic: [Searching](#)

Related tasks:

[Generating Rule Project Statistics reports](#)

Related information:

[Searching the execution object model](#)

[Searching rule project elements](#)

[Business object model \(BOM\)](#)

Searching the BOM classes and packages

You can search for BOM classes and packages or limit the search to the current BOM.

About this task

You can search the BOM for classes and packages.

Procedure

1. In the **Search** menu, select **Search**, and then click the **BOM Search** tab.
2. Specify patterns to find classes and packages in the **Search string** field.

Use * to specify any string and ? to specify any character. Using wildcards, you can specify all or part of the names of the classes or packages that you want to find.

3. Select the **Case sensitive** box if you want your search to differentiate between lowercase and uppercase characters in names.
4. Click **Search**.

Results

The results display in the Search view.

Parent topic: [Searching the business object model](#)

Searching the current BOM

In the Search dialog, you can limit the search to the BOM of the selected rule project.

About this task

You can search for classes and packages in the BOM of a specific rule project, or in all the rule projects contained in the workspace.

Procedure

1. Highlight the rule project in the Rule Explorer.
2. Open the **BOM Search** page in the Search dialog and set up the search.
3. Select the scope in the Scope area:
 - To search on BOMs in all the rule projects in the workspace, select **Workspace**.
 - To search on BOMs for specific rule projects, select **Selected Resources**.
4. Click **Search**.

The results display in the Search view.

For every match, the Search view shows the name of the class or package, its container, and the BOM file where it is located.

5. To open the BOM editor corresponding to the match, double-click the match.
 - If the match is a class, the BOM editor opens on the Class tab with the selected class.
 - If the match is a package, the BOM editor opens on the Package tab with the selected package in the BOM tree.

Parent topic: [Searching the business object model](#)

Querying

You can filter your rules by creating and running queries in Rule Designer.

Queries

Using queries, you can extract rule artifacts or project elements by defining criteria as statements in the query condition part and display the found artifacts or, optionally, modify them with Do statements in the action part. Queries synchronize between Rule Designer and Decision Center, except for module-specific constructs.

Creating a query

To create a query, you select the project folder, name the query, and select the appropriate project element, conditions, and actions in the Query Editor. To make the query synchronizable, avoid any module-specific construct.

Running a query

You run your own queries or predefined queries on the rule project and, optionally, on referenced rule projects.

Finding rule dependencies

You can run predefined queries to find rules that can enable or disable another rule or ruleflows that can select a rule.

Customizing queries

To create business-specific query predicates that process customized rule project items and properties, you extend the query BOM files, and then integrate your extensions into Rule Designer and Decision Center.

Automating queries with the Rule Designer API

You can run a query in four steps using the Rule Designer API.

Parent topic: [Reviewing a rule project](#)

Queries

Using queries, you can extract rule artifacts or project elements by defining criteria as statements in the query condition part and display the found artifacts or, optionally, modify them with Do statements in the action part. Queries synchronize between Rule Designer and Decision Center, except for module-specific constructs.

Introducing queries

Using queries, you can search through your workspace for rule artifacts and enforce actions on the artifacts found. You enter the search criteria in the condition part and, in the action part, the modification to carry out.

Query conditions

In the condition part of a query, you specify the type of project element that you want to search and, if applicable, the “such that” filters to apply to the selected project elements.

Query actions

A query action specifies the action to be taken if the conditions of a query are met.

Query synchronization between Rule Designer and Decision Center

Queries work both in Rule Designer and Decision Center. When you synchronize a rule project between Rule Designer and Decision Center, queries in both modules are also synchronized, except for certain module-specific query constructs.

Parent topic: [Querying](#)

Introducing queries

Using queries, you can search through your workspace for rule artifacts and enforce actions on the artifacts found. You enter the search criteria in the condition part and, in the action part, the modification to carry out.

For example, if you have a modification to apply to a certain type of rule, you can create a query that searches for the rules corresponding to this type and modifies them. You can also create queries to find which rules have been affected by any modification you made.

Queries can be categorized according to the rule part that they search:

- The condition part of a rule
- The action part of a rule
- Both the condition and action parts of a rule

You can use queries to evaluate the impact of changes to the object model or changes to rules.

You construct queries in the same way that you construct rules with a condition and an action. As the query condition, you define the criteria to match (see [Query conditions](#)) and as the action, you specify what to do. The default query action is to display the results in the Search window, but it is possible to have the query carry out other actions such as move the results of a query to another package (see [Query actions](#)).

You can use queries in Rule Designer and Decision Center. However, if you are synchronizing your rule projects between the two modules, you must be aware of some differences (see [Query synchronization between Rule Designer and Decision Center](#)).

Parent topic: [Queries](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Automating queries with the Rule Designer API](#)

Query conditions

In the condition part of a query, you specify the type of project element that you want to search and, if applicable, the “such that” filters to apply to the selected project elements.

When you write a query condition, you start by selecting a project element type, then you refine the search by filtering on the project element properties, definition, or semantic content.

By default, queries are run on business rules, but you can also search on other project elements:

- Specific types of business rule:
 - Action rules
 - Decision tables
 - Decision trees (Decision trees are deprecated)
 - Your own rule class if you extended the rule model
- Other project elements:
 - Rule packages
 - Rule artifacts
 - Technical rules
 - Ruleflows
 - Templates (Templates are deprecated)
 - Functions
 - Variables
 - Variable sets

After you have selected the project element that you want to query, you can refine the query by adding a filter in the form of a `such that` statement, followed by an appropriate condition statement, or statements. The statements available for you to select depend on the type of project element that you have chosen and on the elements defined in the BOM.

Examples

*Find all ruleflows
such that each ruleflow is in package "accounts"*

Or:

*Find all business rules
such that the effective date of each business rule is after 3/31/2016*

You can further refine queries by specifying more than one statement:

*Find all business rules
such that the effective date of each business rule is after 3/31/2016
and the effective date of each business rule is before 4/30/2016*

Note: Queries on rule artifacts such as “Find all rule artifacts” do not include functions.
--

Queries on properties

You can refine a query by filtering on one of the properties of the project elements being queried, such as name, status, or effective date.

Queries on definitions

You can refine a query by filtering on project element definitions.

Queries on semantic content

You can refine a query by filtering on the semantic content of a rule, that is on its behavior.

Queries on rule dependencies

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule affects the applicability of other rules and how a rule is affected by the execution of

other rules.

[Queries on categories](#)

When you write a business rule, you can specify a category filter on the rule. The default category is Any.

Parent topic: [Queries](#)

Related concepts:

[Query actions](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Automating queries with the Rule Designer API](#)

[Query synchronization between Rule Designer and Decision Center](#)

Queries on properties

You can refine a query by filtering on one of the properties of the project elements being queried, such as name, status, or effective date.

Name is a standard rule property. Status and effective date are extended properties, provided by the default extension model.

You can run this type of query against any type of project element. This type of query is useful for carrying out a rule audit.

Example 1

The following query returns all business rules in your workspace with a status of new.

```
Find all business rules  
such that the status of each business rule is new
```

Example 2

The following query returns all variables in a package called accounts.

```
Find all variables  
such that each variable is in package "accounts"
```

Parent topic: [Query conditions](#)

Related concepts:

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

[Queries on categories](#)

Queries on definitions

You can refine a query by filtering on project element definitions.

Queries on definitions find rules that use or modify the value of a member or call a method. You can run this type of query against any type of rule. It is useful for identifying rules that are affected by policy change.

You can run queries on definitions against any type of rule. It is useful for identifying rules that are affected by policy change:

- The [uses the phrase](#) and [uses the phrase ... where](#) predicates find rules that call a method.

Although their naming is similar, the `uses the phrase` and `uses the phrase ... where` queries behave differently. The first query (`uses the phrase`) looks at the syntactic form of the rule, whereas the second query (`uses the phrase ... where`) looks at the behavior of a rule. That is to say that the `uses the phrase` query checks if the textual representation of a rule contains a method call; and the `uses the phrase ... where` query checks whether the execution of a rule calls the method and checks that the arguments of the method satisfies the constraints. The first query can return never-applicable rules, whereas the second query does not return never-applicable rules as they are not executed.

- The [uses the value of](#) and [modifies the value of](#) predicates find rules that use or modify the value of a member.
- The [is using the BOM class](#) and [is using the BOM member](#) predicates find rules that use a BOM class or BOM member.

Queries with the **is using BOM class** or the **is using BOM member** predicate work only if the workspace has been rebuilt since the last modification. If not, the results might not be accurate. The selected BOM class or member must be verbalized.

uses the phrase

The predicate `uses the phrase <a method verbalization>` returns rules that call the specified method.

Example

The following query returns all the action rules that call the method that adds 'a number' to the corporate score in 'a report':

```
Find all action rules
such that each action rule uses the phrase [ add 'a number' to the
corporate
score in 'a report' ]
```

uses the phrase ... where

The predicate `uses the phrase ... where <a method verbalization with a constraint on method arguments>` returns rules that call a given method to which a constraint on arguments is applied.

This query filters on the project element definition **and** on the semantic content of a rule.

Example 1

The following query returns all the business rules that call the method that sets the credit score of 'a borrower' to 'a number', where this number equals 100:

```
Find all business rules
such that each business rule uses the phrase [ set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100 ]
```

This query could return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

Example 2

The following query returns all the business rules that call the method that sets the credit score of 'a borrower' to 'a number', where this number is at least 100:

```
Find all business rules
such that each business rule uses the phrase [ set the credit score of 'a
borrower'
to 'a number' , where 'a number' is at least 100 ]
```

This query could return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 200;
```

Example 3

The following query returns all the business rules that call the method that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a number'
to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When a rule does not show the BOM elements involved in the constraint, this rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query could return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10 to
'the borrower';
```

Example 4

The following query returns all the business rules that call the method that sets the loan amount range to <min> and <max>, with a maximum rate of <rate> where the value of the <min> and <max> parameters are lower than 10000, and the <rate> parameter is greater than 15%:

```
Find all business rules
such that each business rule uses the phrase [ set the loan amount
range to 'a number' and
'another number', with a maximum rate of 'a number 3', where 'a number' is
lower than 10000,
and 'another number' is lower than 10000, and 'a number 3' is greater than 15
% ]
```

You can select a predefined variable for each placeholder. Each variable has the type of the corresponding placeholder. If the type of the variable is String or Number, you can select up to 10 variables: 'a number', 'another number', 'a number 3', 'a number 4', and so on. For any other type, you can select up to three variables.

You can also select the predefined variables in the constraint part of the query and apply a constraint to them.

uses the value of

The predicate uses the value of <an attribute, ruleset parameter, or variable> returns rules that use the values of certain members.

Example

The following query returns all the business rules that use loan grade values.

```
Find all business rules
such that each business rule uses the value of the loan grade in 'a report'
```

modifies the value of

The predicate modifies the value of <an attribute, ruleset parameter, or variable> returns rules that modify certain members.

Example

The following query returns all the decision tables that modify the insurance rate of the loan report ruleset

parameter:

*Find all decision tables
such that each decision table **modifies the value of** the insurance rate of
'the loan report'*

is using the BOM class

The predicate **is using the BOM class** returns the rules that reference a certain class either by using a binding to an instance of this class or by using the automatic variable linked to this class (Rule Designer only).

This query returns only business rule artifacts, it does not return technical rules.

Example

The following query returns all the business rules that use the class Borrower:

*Find all business rules
such that each business rule **is using the BOM class** "loan.Borrower"*

You can also use the predicate **is not using BOM class**.

is using the BOM member

Returns the rules that reference a certain BOM member (Rule Designer only).

This query returns only business rule artifacts, it does not return technical rules.

Example

The following query returns all business rules that read or modify the amount attribute of the Loan class:

*Find all business rules
such that each business rule **is using the BOM Member** "loan.Loan.amount"*

You can also use the predicate **is not using BOM member**.

Note: You can use other queries such as **uses the value of**, **modifies the value of**, and **uses the phrase** to obtain more accurate results.

Parent topic: [Query conditions](#)

Related concepts:

[Queries on properties](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

[Queries on categories](#)

Queries on semantic content

You can refine a query by filtering on the semantic content of a rule, that is on its behavior.

This type of query uses the phrases of the vocabulary that refer to members. It finds rules that could be applicable when certain conditions are true or could become true. It is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can run this type of query against the following project elements:

- **Action rules:** a type of business rule whose purpose is to do an action
- **Business rules:** any rule written using the Business Action Language (BAL)
- **Decision tables:** business rules written in table format, where each row corresponds to a single rule
- **Decision trees:** business rules written in the form of a tree
- **Technical rules:** rules written using the ILOG® Rule Language (IRL)
- **All rules:** any type of rule, including decision tables or decision trees

You can run a query on all rules of the current project, or all rules of the current project and its dependent projects.

Note:

- When you create a semantic query, for dates as well as for strings, only the “is” and “is not” constructs are valid. The following constructs are not available: “is before”, “is after”, “is between”, “contains”, “starts with”, and “ends with”.
- Semantic queries run on the IRL code generated from the BAL rule. If a value translator is attached to a BOM member, the generated IRL uses this translation. Therefore, the semantic queries cannot find the BOM member because it is not shown in the IRL code.

You can use the following query predicates:

- [may apply when](#)
- [may become applicable when](#)
- [may lead to a state where](#)

may apply when

The predicate `may apply when <a condition>` returns all rules whose condition part could meet the query condition, or the rules in which nothing in the rule condition contradicts the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

Example

Rule 1:

If the score of the borrower is at least 10 then...

Rule 2:

If the age of the borrower is at least 21 then...

Rule 3:

If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

Query 1:

*Find all business rules
such that each business rule **may apply when** [the score of the borrower is 20]*

Query 2:

*Find all business rules
such that each business rule **may apply when** [the score of the borrower is 5]*

Query 1 returns Rule 1 and Rule 2. It returns Rule 1 because if the borrower's score is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the borrower's score could well be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

Query 2 returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to below 10. In Rule 2, there is nothing that specifically stops the rule from being applicable when the score is 5. In Rule 3, at least one of the conditions could apply and there is nothing in the other condition that negates the fact that the score could be 5.

may become applicable when

The predicate `may become applicable when` <a condition> is a more specific query that returns only those rules in which the condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition. The query returns rules that would be applicable if the query condition was true. For example, if a change occurs in the elements of the rule condition, and the changed rule condition matches the query condition.

Example

Rule 3:

If the category of the customer is Platinum then...

Rule 4:

If the category of the customer is not Platinum then...

Rule 6:

If the age of the customer is at most 65 and the category of the customer is not Platinum...

Query 1:

*Find all business rules
such that each business rule may become applicable when [the category of
'a customer' is Gold]*

This query returns Rule 4 and Rule 6. It returns Rule 4 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 6 for the same reason. The additional condition relating to the customer's age does not contradict the condition in which the category can be Gold.

The query does not return Rule 3 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 5 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer's category is Gold.

Query 2:

*Find all business rules
such that each business rule may become applicable when [the age of
'a customer' is at least 21]*

This query does not return Rule 5 because it searches for rules that could "become" applicable when the customer's age is over 21. Rule 5 is applicable even if the customer's age is under 21. Therefore Rule 5 does not "become" applicable, but it "remains" applicable even if the customer's age changes from under 21 to over 21.

may lead to a state where

The predicate `may lead to a state where` <a condition> returns the rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of the rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

Example

Rule 7:

If the age of the borrower is at least 25 then set the credit score of the borrower to 60

Rule 8:

If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

Query:

Find all business rules
such that each business rule **may lead to a state where** [the credit score of the borrower is more than 50]

This query returns Rule 7 but not Rule 8 because, after the age of the borrower has been checked and the credit score set, only Rule 7 shows a result of over 50.

Note: You can also run a query on the action part of a rule using the `uses the` phrase predicate, see [uses the phrase](#).

Parent topic: [Query conditions](#)

Related concepts:

[Queries on properties](#)

[Queries on definitions](#)

[Queries on rule dependencies](#)

[Queries on categories](#)

Queries on rule dependencies

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule affects the applicability of other rules and how a rule is affected by the execution of other rules.

You can use the following predicates:

- [may select](#)
- [may enable rule](#)
- [may disable](#)
- [may be enabled by](#)
- [may be disabled by](#)

may select

The predicate `may select <a rule>` returns the ruleflows and rule tasks that might select a given rule.

When you define a ruleflow, you can specify the rules that make up a rule task either explicitly or by using a filtering function. The rule overriding mechanism is also likely to determine how rules are selected for execution at run time (for more information, see [Rule overriding](#)).

Example

The following query returns the ruleflows and rule tasks that might select "Rule 1".

```
Find all ruleflows  
such that each ruleflow may select "Rule 1"
```

may enable rule

The predicate `may enable rule <a rule>` returns rules that might make a given rule applicable.

Example

Rule 1:

```
if the age of the customer is 25  
then set the category of the customer to Bronze ;
```

Rule 2:

```
if the category of the customer is Bronze  
then give Champagne to the shopping cart of the customer with message:  
"Congratulations" ;
```

Query:

```
Find all business rules  
such that each business rule may enable "Rule 2"
```

This query returns Rule 1 because this rule sets the category of the customer to Bronze and therefore makes the condition of Rule 2 valid.

may disable

The predicate `may disable <a rule>` returns rules that might make a given rule inapplicable.

Example

Rule 1:

```
if the age of the customer is 18  
then set the category of the customer to Copper ;
```

Rule 2:

```
if the category of the customer is Bronze  
then give Champagne to the shopping cart of the customer with message:  
"Congratulations" ;
```

Rule 3:

```
if the age of the customer equals 25 and the category of the customer is Gold
then set the category of the customer to Diamond ;
```

Query:

```
Find all business rules
such that each business rule may disable "Rule 2"
```

This query returns Rule 1 because if the category of the customer is set to Copper, it cannot be Bronze. The query does not return Rule 3 because the action is to set the category of the customer to Diamond and this rule can be executed only from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

may be enabled by

The predicate may be enabled by <a rule> returns rules that might be made applicable by a given rule.

Example

Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

Rule 2:

```
if the category of the customer is not Copper
then set the discount of the shopping cart of the customer to 5;
```

Rule 3:

```
if the age of the customer equals 25 and the category of the customer is
Bronze
then set the category of the customer to Diamond ;
```

Query:

```
Find all business rules
such that each business rule may be enabled by "Rule 1"
```

This query returns Rule 2 and Rule 3. It returns Rule 2 because if the category of the customer is not Copper, it could be Bronze. It returns Rule 3 because the condition specifies that the category of the customer should be Bronze.

may be disabled by

The predicate may be disabled by <a rule> returns rules that might be made inapplicable by a given rule.

Example

Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

Rule 2:

```
if the age of the customer equals 25 and the category of the customer is
Copper
then set the category of the customer to Gold
```

Query:

```
Find all business rules
such that each business rule may be disabled by "Rule 1"
```

This query returns Rule 2 because this rule is made inapplicable by Rule 1. In Rule 1, the category of a customer whose age is 25 is set to Bronze, therefore it cannot be Copper.

Note:

You can run these queries by right-clicking a rule in the Rule Explorer and clicking:

- **Find Rule Dependencies > Rules which may enable or disable this rule**
- **Find Rule Dependencies > Rules which may be enabled or disabled by this rule**
- **Find Rule Dependencies > Ruleflows which may select this rule**

Parent topic: [Query conditions](#)

Related concepts:

[Queries on properties](#)

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on categories](#)

Queries on categories

When you write a business rule, you can specify a category filter on the rule. The default category is Any.

You can query rules that use a specific category or rules that use the default category Any.

Attention: In the localized versions of the product, the Any category is translated in the editor. However, to create a query on the rules containing the category Any, you must use the English word Any as the name of the category, instead of the translated name even if the query is in a different locale.

Parent topic: [Query conditions](#)

Related concepts:

[Queries on properties](#)

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

Query actions

A query action specifies the action to be taken if the conditions of a query are met.

The default action of a query is to display the results of the query in the Search window. However, you can extend the actions of a query by adding a Do statement to it.

You can manage rules by specifying the following actions:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add a category to, or remove a category from, the displayed elements
- Set any of the properties (such as effective date, status, priority) of the displayed elements to a value that you specify

The actions available depend on the project element specified in the query. For example, there are more actions to choose from for a query on business rules than for a query on variables.

Example 1

In the following example, you change the status of the business rules from new to validated.

```
Find all business rules
such that the status of each business rule is new
Do set the status of each business rule to validated
```

Example 2

In the following example, you collect the decision tables that modify the value of 'the insurance rate' and group them into the 'insurance' package.

```
Find all decision tables
such that each decision table modifies the value of 'the insurance rate'
Do move each decision table to package "insurance"
```

Example 3

In the following example, you delete the rules that could lead to a certain credit score and in which the expiration date is after a certain date.

```
Find all business rules
such that each business rule may lead to a state where [the credit score of
'a borrower' is less than 5 ]
and the expiration date of each business rule is after 6/21/2007
Do delete each business rule
```

Parent topic: [Queries](#)

Related concepts:

[Query conditions](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Automating queries with the Rule Designer API](#)

[Query synchronization between Rule Designer and Decision Center](#)

Query synchronization between Rule Designer and Decision Center

Queries work both in Rule Designer and Decision Center. When you synchronize a rule project between Rule Designer and Decision Center, queries in both modules are also synchronized, except for certain module-specific query constructs.

You can create and run queries in either Rule Designer or Decision Center. However, you do not have to write the same query twice: when you synchronize rule projects between Rule Designer and Decision Center, queries in both modules are also synchronized.

The query BOM and vocabulary files are shared between Rule Designer and Decision Center. However, certain query constructs are not: if you write queries using module-specific constructs, these queries are synchronized but you cannot run them in the other module. If this is the case, you receive an error message. To make sure that a query written in one module can be run in the other, avoid using module specific constructs.

Rule Designer query constructs

The following condition predicates are specific to Rule Designer:

- `{this}` is editable
- `{this}` is not editable
- `{rule artifact}` is using BOM class
- `{rule artifact}` is not using BOM class
- `{rule artifact}` is using BOM member
- `{rule artifact}` is not using BOM member
- the parent package of `{a rule package}`
- `{rule package}` is in package `{rule package}`
- `{variable}` is in package
- the rule query class

The following action predicate is specific to Rule Designer:

set the editable status of `{this}` to `{0}`

Decision Center query constructs

The following condition predicates are specific to Decision Center:

- the creator of `{this}`
- the creation date of `{this}`
- the last modifier name of `{this}`
- the last modification date of `{this}`
- the view class
- the group of

The following action predicate is specific to Decision Center:

set the group of `{project element}` to `{group}`

Parent topic: [Queries](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Automating queries with the Rule Designer API](#)

[Synchronization commands in Designer](#)

Creating a query

To create a query, you select the project folder, name the query, and select the appropriate project element, conditions, and actions in the Query Editor. To make the query synchronizable, avoid any module-specific construct.

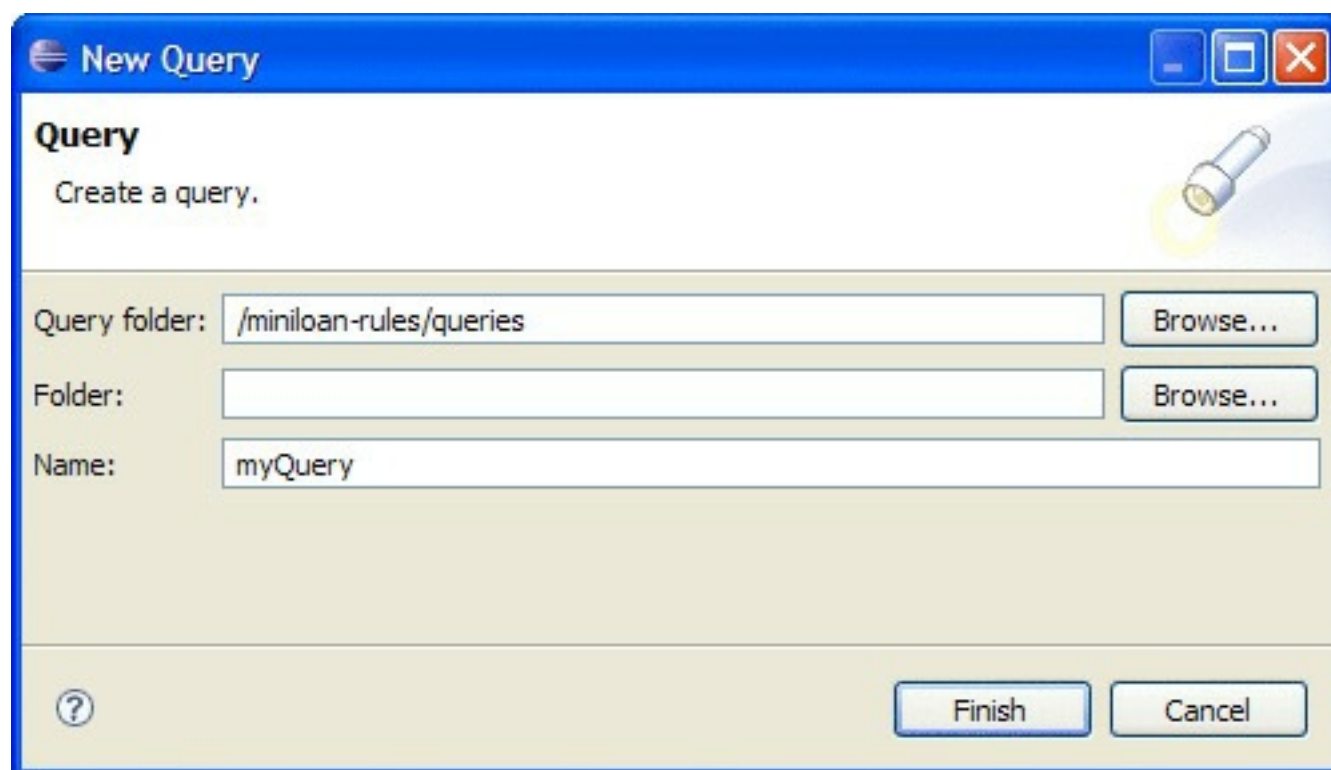
About this task

Not all query constructs are available in Rule Designer and Decision Center. To make sure that a query can run in both Rule Designer and Decision Center, use only query constructs that are available in both. For more information, see [Query synchronization between Rule Designer and Decision Center](#).

Procedure

To create a query:

1. In the Rule Explorer, right-click the queries project folder and click **New** > **Query** to open the New Query wizard.



Use the **Folder** field to specify a subfolder within the queries folder. Click **Browse** to select this subfolder or create it.

2. Type a name for the query in the **Name** field.
3. Click **Finish**.

You have now created an entry for the query in the Rule Explorer in the location that you specified. The Query Editor opens.

4. In the Content part of the Query Editor, define your query by selecting the required project elements, conditions, and actions.

You build a query in the same way as you build a rule. The project elements, conditions, and actions are displayed in drop-down lists when you click the building blocks. The conditions and actions available in the lists depend on the elements that you select (see [Query conditions](#) and [Query actions](#)).

5. If you want to enter a description or other information about the query, expand the Documentation section and type the required comments.
6. Save the query.

Note:

Some queries are based on a rule or a package. In this case, a tree view lets you select the rule or package to use in your query. After you have selected the rule or package, its fully qualified name is shown in the query, for example:

Find all rules
such that each rule may enable "validation.loan.checkAmount"

To modify the query and select another rule, double-click the existing rule.

Parent topic: [Querying](#)

Related concepts:
[Query conditions](#)

Related information:

Running a query

You run your own queries or predefined queries on the rule project and, optionally, on referenced rule projects.

About this task

After you have written a query, you can run it on the rule project that contains it and, optionally, on rule projects referenced from the current rule project.

Procedure

To run a query:

1. In the Rule Explorer, double-click the query to open it.
2. If you want to run the query on the current rule project and on the elements of any rule project on which the current project depends, select **Include dependencies**.
3. In the Query Editor, click **Run query**.

All rules that meet the query search criteria are listed in the Search window.

Note: In some cases, when you run queries that return decision tables or ruleflows, the rows or rule tasks that match the search criteria are highlighted.

4. If you have created queries with actions, you can run them using the **Run query actions** option.

Results

Note: Queries that include the **is using BOM member** or the **is using BOM class** predicate work only if the workspace has been rebuilt since the last modification. If it has not, the results might not be accurate.

Parent topic: [Querying](#)

Related concepts:

[Query conditions](#)

Related information:

[Automating queries with the Rule Designer API](#)

Finding rule dependencies

You can run predefined queries to find rules that can enable or disable another rule or ruleflows that can select a rule.

About this task

In Rule Designer, you can run predefined queries on rule dependencies. Rule Designer provides predefined queries so that you do not have to write the queries that enable you to search for rules that affect or are affected by other rules. You can also run a predefined query to find the ruleflows that select a given rule.

You can search for the following artifacts:

- Rules that are affected by the execution of other rules
- Rules that affect the execution of other rules
- Ruleflows that can select a given rule

For more information, see [Queries on rule dependencies](#).

Procedure

To run queries on rule dependencies:

1. Right-click a rule in the Rule Explorer and click **Find Rule Dependencies**.
2. Click one of the following queries:
 - **Rules which may enable or disable this rule**
 - **Rules which may be enabled or disabled by this rule**
 - **Ruleflows which may select this rule**

Results

The rules or ruleflows that meet the query search criteria are listed in the Search window.

Parent topic: [Querying](#)

Customizing queries

To create business-specific query predicates that process customized rule project items and properties, you extend the query BOM files, and then integrate your extensions into Rule Designer and Decision Center.

[Integrating query extensions into Rule Designer](#)

You use query extensions to create business-specific query predicates that process customized rule project items and properties. You integrate them into Rule Designer by creating a Rule Designer plug-in with a query model extension point.

Parent topic: [Querying](#)

Integrating query extensions into Rule Designer

You use query extensions to create business-specific query predicates that process customized rule project items and properties. You integrate them into Rule Designer by creating a Rule Designer plug-in with a query model extension point.

Before you begin

Before you integrate query extensions into Rule Designer, make sure you have all the necessary files:

- A BOM file
- A vocabulary file
- A BOM extender mapping file
- One or more extender classes

About this task

You use query extensions to create business-specific query predicates that process customized rule project items and properties. To integrate them into Rule Designer, you create a Rule Designer plug-in with a query model extension point.

Procedure

To integrate query extensions into Rule Designer:

1. Create a plug-in project that depends on the plug-in: `ilog.rules.studio.model.query`.
2. In this plug-in, create an extension for `ilog.rules.studio.model.query.queryModelExtension`.
3. Place your query extension files and classes in the `src` directory of the plug-in.
4. In the extension details, set the following attributes:

`bom_url` - the location of the BOM file

`voc_url` - the base name of the vocabulary files (without the locale suffix)

`bom_extender_mapping_file` - the location of the BOM extender mapping file

For example, if you create three extension files named `ext.bom`, `ext.voc`, and `ext.properties` and place them at the root of the `src` directory in the plug-in project, the extension details are as follows:

- **`bom_url`**: `ext.bom`
 - **`voc_url`**: `ext.voc`
 - **`bom_extender_mapping_file`**: `ext.properties`
5. Save the plug-in project.
 6. Deploy the plug-in.

Results

Your query extensions are now available in your Rule Designer installation.

Parent topic: [Customizing queries](#)

Related information:

[Business object model \(BOM\)](#)
[Queries](#)

Automating queries with the Rule Designer API

You can run a query in four steps using the Rule Designer API.

Services on top of the rule model API allow you to automate queries. You can automate queries by API with the entry point `IlrQueryService`.

You run a query in two or three steps, depending on whether you want to run the actions of the query or not:

1. Fetch the query service.
2. Initialize the query.
3. Run the query (this can be done repeatedly).
4. Run the actions of the query.

Fetching the query service

Use the following to fetch the query service:

```
IlrQueryService queryService = (IlrQueryServiceImpl)
IlrStudioQueryPlugin.getQueryService(ruleProject);
```

If you want to set a BOM parser other than the default, use:

```
queryService.setBom(IlrBOMGetter.getInstance().getBOM());
```

Initializing the query service

Depending on the elements you have, you can initialize the query service from:

- an `IlrQuery` project element
- a string containing the text of the query (for example, `Find all business rules such that...`)

Initializing from `IlrQuery`

The following example shows how to initialize a query service with an `IlrQuery` element obtained from an `IlrProject`:

```
IlrQuery myQuery;
queryService.initQuery(myQuery, null);
```

To pass your own `IlrRuleQueryMatchCollector` as second argument:

```
IlrQuery myQuery;
IlrRuleQueryMatchCollector matchCollector;
queryService.initQuery(myQuery, matchCollector);
```

Initializing with query text

The following example shows how to initialize a query service with query text:

```
String myQuery = "Find all business rules such that the name of each business
rule contains \"foobar\"";
boolean useLocalScope = false;
IProject currentProject;
queryService.initQuery(myQuery, useLocalScope, currentProject, null);
```

The second Boolean parameter determines whether the query scope is restricted to the specified project, or if it includes project dependencies.

As with an `IlrQuery`, you can also pass an `IlrRuleQueryMatchCollector` as last parameter:

```
queryService.initQuery(myQuery, useLocalScope, currentProject,
matchCollector);
```

Running the query

To run a query, use:

```
List results = queryService.runQuery();
```

The contents of the list returned depends on the implementation of `IlrRuleQueryMatchCollector`.

Using the default provided, a list of objects of one of the following type is returned:

- `IlrQueryProjectElementWrapper`
- `IlrQueryParameterWrapper`
- `IlrQueryVariableWrapper`

Running query actions

If the query has actions (checked using `IlrQueryService.queryHasActions()`), you can run the actions with:

```
queryService.runQueryActions(results)
```

The results are those returned by `runQuery()`. If needed, you can change the list before passing it to `runQueryActions()`.

Parent topic: [Querying](#)

Related concepts:

[Query conditions](#)

Related tasks:

[Creating a query](#)

[Running a query](#)

Related information:

[Queries](#)

Analyzing

Rule analysis relies on consistency checking and completeness analysis. Analyze your rule projects to find ambiguities, conflicts, and missing rules, and fix them. Using Rule Analysis view options, you can fine tune the process.

Rule analysis

Running in the background, rule analysis carries out completeness and consistency checking on a rule project to warn you of semantically redundant, conflicting or missing rules and let you fix them interactively.

Analyzing a rule project

To check the integrity of a rule project, you generate an analysis report on it, you resolve the ambiguities and conflicts, you instantiate the rules that were found missing, and you check for remaining errors.

Fine tuning rule project analysis

Use the advanced options of the Rule Analysis view to customize the process by filtering out some of the rules, by locking the results display of the latest analysis while another one is running, or by creating multiple views to run several concurrent cycles.

Parent topic: [Reviewing a rule project](#)

Related tasks:

[Creating a query](#)

Related information:

[Rule analysis](#)

Rule analysis

Running in the background, rule analysis carries out completeness and consistency checking on a rule project to warn you of semantically redundant, conflicting or missing rules and let you fix them interactively.

[Introducing rule analysis](#)

Rule analysis is a feature for checking that your rule project contains consistent rules with no conflict or redundancy.

[Consistency checking](#)

The consistency analysis mechanism finds semantically conflicting rules such as rules that are never selected, rules that never apply, equivalent rules, redundant rules, conflicting rules, and conflicts between decision tables or decision trees.

[\(Deprecated\) Completeness analysis](#)

The completeness analysis mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

Parent topic: [Analyzing](#)

Related information:

[Analyzing a rule project](#)

[Fine tuning rule project analysis](#)

Introducing rule analysis

Rule analysis is a feature for checking that your rule project contains consistent rules with no conflict or redundancy.

Using the rule analysis feature, you can analyze your rule project to check that it contains no ambiguities or conflicts, and that no rules are missing.

Rule analysis can be very useful at different stages in the elaboration of your rules. For instance, after you have created your rule project, you want to make sure that the ruleset is consistent and that there are no conflicting or redundant rules. You also want to check that every possible case is covered, for example, that there is a rule for every age range.

If your project has been modified, you can use rule analysis to make sure that no inconsistencies or gaps have been introduced.

Rule Analysis view

The Rule Analysis view can run as a background task. It raises a warning when an ambiguity, a conflict, or a missing rule is identified. It lets you access the rules directly and fix the errors found and fill the gaps interactively.

If you do not want to analyze the whole rule project, the Rule Analysis view provides different options that filter the rules in your rule project.

Note: You can run an analysis of the rules only on classic rule projects that were built with the classic rule engine.

Parent topic: [Rule analysis](#)

Related concepts:

[Consistency checking](#)

[\(Deprecated\) Completeness analysis](#)

Consistency checking

The consistency analysis mechanism finds semantically conflicting rules such as rules that are never selected, rules that never apply, equivalent rules, redundant rules, conflicting rules, and conflicts between decision tables or decision trees.

Consistency checking is a mechanism for checking whether rules do not contain semantically conflicting elements.

Ambiguities can be found either in a single rule or in a set of rules. For example:

- A single rule can contain self-contradictory conditions and therefore never apply.
- Two rules can apply to the same object and set a given attribute to two different values. Such rules are conflicting.

Consistency checking goes beyond syntax aspects to consider semantics as well. That is, how the rule behaves during execution. Using Rule Designer , you can choose which checks are carried out.

Consistency checks belong to one of the following categories:

- Checks that analyze an individual rule. These checks are activated when you build the rule and when you run the Consistency checking analysis:
 - [Rules that are never selected](#)
 - [Rules that never apply](#)
 - [Rules with range violation](#)
- Checks that analyze rules in relation to other rules. These checks are activated only when you run the Consistency checking analysis.
 - [Rules with equivalent conditions](#)
 - [Equivalent rules](#)
 - [Redundant rules](#)
 - [Conflicting and self-conflicting rules](#)

Consistency checking reports problems on rules:

- If there is a ruleflow in your rule project, the consistency checking mechanism reports problems on the rules that are included in a rule task and that might be selected at run time.

The mechanism compares only rules that could be in the same task. In the case of a rule task with dynamic selection filtering, the mechanism takes into account the rules that are potentially selected by this task. A rule is considered potentially selectable when it cannot be established that it definitely cannot be selected.

- If there is no ruleflow in your rule project, all the rules in the project are likely to be selected.

For more information, see [Runtime rule selection](#) and [Rule overriding](#).

Note: Consistency checking gives an indication of the consistency of your rules but cannot identify all potential problems. An empty consistency checking report is therefore not a guarantee that there are no problems in the analyzed rules.

Rules that are never selected

Rules are reported as "never selected" when they are not part of a rule task and cannot be selected at run time. For more information, see [Runtime rule selection](#) and [Rule overriding](#).

Rules that never apply

A rule is analyzed as never applicable when its conditions can never be met.

Typically, the syntax of such rules is correct but the rules contain common logic errors. For example:

- The wrong operator is used to combine condition statements, for example and instead of or: the category of the customer is Gold and the category of the customer is Platinum.
- Values are inverted, for example, in the following rule: the age of the customer is between 70 and 50.
- Values in the conditions are not within the permitted range.

Rules with range violation

In order to reduce the risk of errors, some members can only be assigned values within a specified range. For example, the yearly interest rate on a loan might be limited to values between 0 and 10.

If a rule contains an action that tries to assign a value that is not within the permitted range, Rule Designer

displays a range violation error in the report and in the Rule Editor.

Rules with equivalent conditions

The consistency checking mechanism also reports rules in which the condition parts have the same meaning, and where the action parts are different but not in conflict.

Rules with equivalent conditions do not necessarily represent an error situation, but they could be good candidates to be merged.

Equivalent rules

Equivalent rules are reported when both their conditions and actions are the same.

In the following example, **Rule1** and **Rule2** are equivalent:

Rule1

```
definitions
  set minDiscount to 5
  set ageDiscount to 10
if
  the age of the borrower is more than 65
then
  set the discount to minDiscount + ageDiscount
```

Rule2

```
if
  the age of the borrower is at least 66
then
  set the discount to 15
```

Although the syntax of these two rules is different, rule analysis evaluates the numeric expressions and reports that the rules are equivalent. You can therefore delete one of them.

Note: Equivalent rules often arise between a decision table that you create and an existing rule.
--

Redundant rules

When two rules have the same actions, one of them becomes redundant when its conditions are included in the conditions of the other.

In the following example, the Else part of **Rule2** makes **Rule1** redundant:

Rule1

```
if
  the category of the customer is Gold
then
  set the discount to 10
```

Rule2

```
if
  the category of the customer is Platinum
then
  set the discount to 15
else
  set the discount to 10
```

Although **Rule1** is correct, it is redundant and can therefore be deleted.

Note: Redundant rules often arise between a decision table that you create and an existing rule.

Conflicting and self-conflicting rules

Rules **conflict** when the actions of two different rules set a different value for the same business term (member). Conflicts occur between two rules when the conditions are equivalent or cover the same values. For example, consider the following two rules:

Rule1

```
if
  the loan report is approved
  and the amount of the loan is at least 300 000
then
  set the category of the borrower to Gold
```

Rule2

```
if
  the age of the latest bankruptcy of the borrower is less than 1
  and the category of the borrower is not Platinum
then
  set the category of the borrower to No Category
```

Rule1 and **Rule2** conflict when the following situation arises:

- The loan report is approved.
- The amount of the loan is 300 000 or more.
- The borrower has not had a bankruptcy in the last year.
- And the category is anything but Platinum.

In these specific circumstances, the rules sets the category of the borrower to different values. You can correct conflicting rules by changing the conditions, deleting one of the rules, or setting different priorities on the rules.

A rule is **self-conflicting** when two executions of a rule assign different values to the same member. For example, a rule that can apply twice on a given working memory (and ruleset parameters) and sets different values to a common attribute is self-conflicting. For example:

```
if
  the customer category is Gold
then
  set the discount of the cart to the bonus points of the customer
```

With this rule, if there are two customer objects with different bonus points in the working memory, the rule is executed twice and a conflict occurs because the two executions of the rule set different values to the discount of the cart.

Decision table conflicts

To check decision tables and decision trees, you must enable the option **Include decision tables and decision trees in the inter-rule checks**.

When this option is enabled, you can check rules between and within decision tables or decision trees. The rule analyzer checks for conflicts, redundancies, and equivalences between the following elements:

- Two rows of the same decision table
- Two rows of two different decision tables
- Two leaves of the same decision tree
- Two leaves of two different decision trees
- A row of a decision table and a leaf of a decision tree
- A row of a decision table and an action rule
- A leaf of a decision tree and an action rule

When this option is disabled, these checks are not done on decision tables and decision trees, but only on action rules.

However, decision tables and decision trees are checked for never-applicable rules and rules with domain violation even if the option is disabled. This option does not impact overlapping and gap checks which are detected when you write the rules.

Parent topic: [Rule analysis](#)

Related concepts:[Introducing rule analysis](#)[\(Deprecated\) Completeness analysis](#)**Related information:**[Action rule editing errors and warnings](#)[Querying](#)

(Deprecated) Completeness analysis

The completeness analysis mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

Completeness analysis is a mechanism for checking whether there are rules missing from a set of rules.

A ruleset is considered complete if for any possible case at least one rule applies.

You run completeness analysis for the following purposes:

- Check whether a ruleset is complete.
- Obtain a report of the missing rules.
- Add the missing rules to the ruleset.

For more information, see [Analyzing a rule project](#).

Completeness mechanism

The completeness mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

To check that a ruleset is complete, the completeness mechanism proceeds as follows to identify the cases that are missing:

- If the rule project contains a ruleflow, the completeness mechanism analyzes the ruleflow task by task, to check that each rule task is complete.

For example, if a rule is missing in several rule tasks, the report lists the rule tasks where the rule is missing.

- If the rule project does not contain any ruleflow, the completeness mechanism takes into account all the rules in the project.

Note:

A ruleflow must contain at least one rule task.

Example of completeness analysis

In the following example, the ruleset is composed of five rules that define the customer and that set the category of the customer to Silver, Platinum, or Gold.

Rule 1

```
if
    the value of the customer is at least 500 and the value of the
customer is less than 1000
then
    set the category of the customer to "Silver";
```

Rule 2

```
if
    the age of the customer is at least 65 and the value of the customer
is less than 500
then
    set the category of the customer to "Platinum";
```

Rule 3

```
if
    the value of the customer is at least 1000 and the value of the
customer is less than 1500
then
    set the category of the customer to "Gold";
```

Rule 4

```
if
    the value of the customer is at least 1500
then
```

```
    set the category of the customer to "Platinum";
```

Rule 5

```
if
    the value of the customer is less than 500 and the age of the customer
is less than 50
then
    set the category of the customer to "Platinum";
```

Ruleset completeness analysis detects that no rule applied in the case where the customer is between 50 and 65 and has a value of less than 500.

This missing case is presented as a rule skeleton that you can use to create the missing rule:

```
definitions
    set 'Customer1' to a customer ;
if
    the value of Customer1 is less than 500
    and the age of Customer1 is less than 65
    and the age of Customer1 is at least 50
then

<action>
```

Missing rules

If completeness analysis shows that your ruleset is not complete, it proposes additional rules to complete the ruleset. These new rules are skeleton rules. You must edit the missing rule to define the action to be done.

The completeness mechanism analyzes the conditions of each rule in the ruleset to define the set of conditions for the proposed rule. If the rules match a Customer object but test different attributes, for example age, value, and shopping cart, the set of conditions in the proposed rule takes into account the attributes tested in the analyzed rules. Therefore, the condition part in the suggested rule might seem complex and include many condition statements. It can be simplified to reflect a more natural way of expressing the rule conditions.

Note:

The analysis is based on the objects used in the existing rules and that match the working memory. For more information, see [Working memory](#).

In the following example, the suggested rule tests the value and the age attributes. You can simplify the rule by removing either condition.

Example

If completeness analysis proposes the following rule:

```
definitions
    set 'Customer1' to a customer ;
if
    the value of Customer1 is less than 500
    and the age of Customer1 is less than 65
    and the age of Customer1 is at least 50
then

<action>
```

you can remove the condition on the age of the customer, and keep only the condition on the value:

```
definitions
    set 'Customer1' to a customer ;
if
    the value of Customer1 is less than 500
then

<action>
```

If the ruleset contains only action rules, the proposed missing rules are verbalized. However, if the ruleset includes technical rules or decision tables, or if there is too little information on verbalization, the missing rules are presented in IRL.

Note:

The completeness mechanism might generate rules that overlap. You can modify the conditions of these rules.

Performance

The performance of completeness analysis depends on several factors such as the number of rules in the ruleset, the number of rule tasks, and most importantly the number of missing rules.

To improve performance and reduce execution time of completeness analysis, set a limit to the number of missing rules to propose. For example, limit the results to ten missing rules. Once you have integrated the first ten missing rules, run completeness analysis again to identify other missing rules.

Parent topic: [Rule analysis](#)

Related concepts:

[Introducing rule analysis](#)

[Consistency checking](#)

Analyzing a rule project

To check the integrity of a rule project, you generate an analysis report on it, you resolve the ambiguities and conflicts, you instantiate the rules that were found missing, and you check for remaining errors.

Analyzing a rule project

You run a rule analysis cycle to check the integrity of a rule project.

Resolving ambiguities and conflicts

From the Rule Analysis view, you can directly access the rules that contain ambiguities and conflicts.

Instantiating the missing rules

When a missing rule is reported in the Rule Analysis view, you can create the missing rule and define a package for the new rule.

Checking for remaining errors

You look for new or remaining errors.

Parent topic: [Analyzing](#)

Related information:

[Rule analysis](#)

[Fine tuning rule project analysis](#)

Analyzing a rule project

You run a rule analysis cycle to check the integrity of a rule project.


About this task

When you run a rule analysis cycle to check the integrity of a rule project, you go through the following steps:

1. Generate an analysis report by selecting the rule project and the extraction options.
2. Resolve each ambiguity and conflict listed.
3. Instantiate the missing rules found, if applicable: select a package, name the rule, and edit the conditions and actions as necessary.
4. Rerun the analysis cycle to check for any remaining errors and repeat the process as appropriate.

Procedure

To run a report on a rule project:

1. In the Rule Explorer, right-click a rule project and click **Open Rule Analysis**.
2. Click  **Select Rule Project and Extraction Type** in the Rule Analysis view toolbar.
3. Select an extraction option in the Type of Extraction section.

The default option is **Use all rules contained in project**. If you want to streamline the results of the analysis, see [Filtering the results](#).

4. Click **OK** to close the Select Rule Project and Extraction Type dialog.
5. In the Analysis Options section, select the type of checks that you want to activate.

To check decision trees and decision tables, select the **Include decision trees and decision tables in the inter-rule checks** check box.

For more information on the checks available, see [Consistency checking](#) and [\(Deprecated\) Completeness analysis](#).

Note:

If you start the rule analysis and modify the checks selected before the analysis is complete, the changes are not taken into account until the next time you run the analysis.

6. To search for missing rules in your rule project, select the **Completeness** check box.

By default, the analysis stops looking for missing rules when it has found 100 missing rules. If you have a large rule project and if you want to make sure that the analysis does not take too long, you can modify this number and start searching for 10 missing rules, for example.

7. To start the analysis, click  **Run Rule Analysis** in the view toolbar.

The report is shown in the Results section of the Rule Analysis view.

Note:

- You can also analyze rules by clicking the **Analyze rule project** link in the Rule Project Map. This link activates the analysis and refreshes the results.
- You can cancel the analysis by clicking the **Cancel Operation** icon in the Progress view (**Window > Show view > General > Progress**).

Parent topic: [Analyzing a rule project](#)

Related tasks:

[Resolving ambiguities and conflicts](#)

[Instantiating the missing rules](#)

[Checking for remaining errors](#)

Resolving ambiguities and conflicts


From the Rule Analysis view, you can directly access the rules that contain ambiguities and conflicts.

About this task

The Rule Analysis view displays the ambiguities or conflicts found in your rule project and provides a link to the rules that contain these potential errors.

Procedure

To resolve ambiguities or conflicts:

1. Expand each ambiguity or click  **Expand all** on the top of the **Results** section.
2. To open the rule, click the link that points to it.
3. Modify the rules as appropriate and save your rule project.

Rule analysis reports ambiguities or conflicts. You can choose to fix them, or to ignore them if you think they are not relevant. For more information on the type of ambiguities and how to resolve them, see [Consistency checking](#).

Note: If you close or delete your rule project, the results are no longer shown in the Rule Analysis view.

Parent topic: [Analyzing a rule project](#)

Related tasks:

[Analyzing a rule project](#)
[Instantiating the missing rules](#)
[Checking for remaining errors](#)

Instantiating the missing rules

When a missing rule is reported in the Rule Analysis view, you can create the missing rule and define a package for the new rule.

About this task

The Rule Analysis view identifies rules that are potentially missing in the rule project and enables you to instantiate the missing rules. For more information, see [\(Deprecated\) Completeness analysis](#).

Procedure

To instantiate missing rules:

1. In the Results section of the Rule Analysis view, expand the reported missing rule.
If the rule project contains a ruleflow, rule analysis lists all the rule tasks in which the rule is missing.
2. Click **Instantiate missing rule**.
3. In the New Action Rule dialog, in the **Package** field, select the package to which you want to add the new rule.
4. In the **Name** field, enter a name for the new rule and click **OK**.
The new rule is created in the package that you selected.
5. Edit the new rule to define the action, and modify the condition statements if appropriate.

Note: If the ruleset contains a ruleflow, make sure the missing rule is included in all the rule tasks that report the rule as missing.
--

Parent topic: [Analyzing a rule project](#)

Related tasks:

[Analyzing a rule project](#)

[Resolving ambiguities and conflicts](#)

[Checking for remaining errors](#)

Checking for remaining errors


You look for new or remaining errors.

About this task

After you have resolved the ambiguities or conflicts and included the missing rules, check whether there are any new errors or remaining errors.

Procedure

To rerun the analysis:

1. To start the analysis, click  **Run Rule Analysis** in the view toolbar.
2. Check if there are new or remaining errors and repeat the correction process as necessary.

Parent topic: [Analyzing a rule project](#)

Related tasks:

[Analyzing a rule project](#)

[Resolving ambiguities and conflicts](#)

[Instantiating the missing rules](#)

Fine tuning rule project analysis

Use the advanced options of the Rule Analysis view to customize the process by filtering out some of the rules, by locking the results display of the latest analysis while another one is running, or by creating multiple views to run several concurrent cycles.

Filtering the results

You can use the Rule Analysis view to filter your rule project using extractors.

Displaying the results

You can lock the results display and choose when you want to view the new results.

Creating multiple views

You can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

Parent topic: [Analyzing](#)

Related tasks:

[Creating a query](#)

Related information:

[Analyzing a rule project](#)

[Rule analysis](#)

Filtering the results

You can use the Rule Analysis view to filter your rule project using extractors.

About this task


If you are working on a large rule project for which a full analysis might take some time to complete, you can use the advanced options of Rule Analysis view to filter the results.

- You can select an extractor to retain a subset of rules and filter out the rest of the project.
- In automatic run mode, you can lock the display to the results of a given analysis cycle while another is running in the background.
- To work on different analyses concurrently, you can create multiple views.

When you analyze your rule project, you might not want results for all the rules in your rule project. If you prefer to analyze only a subset of your rules, you can use the Rule Analysis view to filter your rule project using extractors. Extractors are often based on the queries in your rule project.

Procedure

To select a filter:

1. Click  **Select Rule Project and Extraction Type** in the view toolbar.
2. In the Select Rule Project and Extraction Type dialog, select **Use an extractor to filter rules contained in project** and click **Browse**.
3. In the Extractor Selection dialog, select an extractor from the list, and then click **OK**.

Parent topic: [Fine tuning rule project analysis](#)

Related tasks:


[Displaying the results](#)

[Creating multiple views](#)

Displaying the results

You can lock the results display and choose when you want to view the new results.

About this task


If you activate the  **Set Automatic Run** option, the Rule Analysis view updates the results every time you save your rule project. However, you can lock the results display and choose when you want to view the new results. This is useful to start fixing the existing errors while another analysis is running.

Note:

Do not use the **Set Automatic Run** option during the development of a rule project.

Procedure

To lock the results display:

1. In the view toolbar, click  **Lock Results Display**.
This option is available only in Set Automatic Run mode.
2. To view the latest results when the analysis is complete, click the **New results available: refresh** link.

Parent topic: [Fine tuning rule project analysis](#)

Related tasks:

[Filtering the results](#)

[Creating multiple views](#)

Creating multiple views

You can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

About this task

While an analysis is running, you cannot change the rule project or select a different extractor. If you modify the selected checks, they are be taken into account until the next time you run the analysis. However, you can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

It is useful to create multiple views for the following reasons:

- You can reduce the time needed to complete the analysis by activating only the checks in which you are interested.
- You can activate different checks for the same rule project. For example, you can have a view displaying consistency problems while another view displays missing rules found in the rule project.
- You can run rule analysis on different rule projects at the same time.
- You can compare the number of errors between two analysis reports.

Note:

- If you have different views set on Automatic Run for the same project, each view is updated each time you save your rule project.
- In some cases, you might get more errors even when you select fewer checks. Some checks, such as equivalent rules and redundant rules, are aggregated.

Procedure

To create multiple views:

1. In the view toolbar, click **Add New Rule Analysis View**.
2. To select a rule project and a type of extraction, click **Select Rule Project and Extraction Type** as described in [Analyzing a rule project](#).

Parent topic: [Fine tuning rule project analysis](#)

Related tasks:

[Filtering the results](#)

[Displaying the results](#)

Generating Rule Project Statistics reports

You can easily generate a report to get an overview of rule projects and of the artifacts that they contain or reference.

About this task

The Rule Project Statistics report provides an overview of the rule projects that are opened in your workspace and indicates their scale.

The report shows statistics for each rule project and for the rule artifacts that it contains, for example:

- The number of ruleset parameters and categories defined in the rule project
- The number of BOM classes, methods and attributes
- The minimum and maximum numbers of rows in decision tables
- The number of rule tasks in a ruleflow, and the execution mode they use
- The number of phrases and business terms in the vocabulary

Procedure

To generate a Rule Project Statistics report:

1. Click **File > Export**.
2. In the Export wizard, select **Rule Designer > Rule Project Statistics**, and then click **Next**.
3. To save the report in the workspace directory, keep the default location and click **Finish**.

The report opens as an HTML page in your browser.

Tip: Alternatively, you can right-click a rule project, and click **Export > Rule Project Statistics**.

Results

Rule Designer creates a <rule-statistics-report>.xml file in the workspace directory, along with a rs4j-export.xsl file to view the report in HTML. The generated report uses the locale of Rule Designer.

- The .xml file contains the statistics.
- The .xsl file is based on the eXtensible Stylesheet Language standard and converts the XML information into a series of HTML tables. You can customize the .xsl file or replace it with your own .xsl file to fit specific needs. If you modify this file manually, it cannot be regenerated by exporting the rule project statistics in Rule Designer. To override the modified file with a new generated one, you must first delete the file.

Note: If you want to open the report after closing it, go to the directory where you saved the report and open the <rule-statistics-report>.xml file in a web browser.

Parent topic: [Reviewing a rule project](#)

Related information:

[Developing rule projects](#)

[Searching](#)

Building and running rules

In Operational Decision Manager, building rules refers to the compilation of your rule projects and, depending on your level of automation, can include creating the corresponding rulesets or RuleApps. Running rules refers to the use of launch configurations to run and debug rulesets in Rule Designer. Rule execution refers specifically to topics related to the engine.

[Overview: Building and running rules](#)

Building rules involves the compilation of your rule projects and can include creating the corresponding rulesets or RuleApps.

[Building rule projects interactively](#)

Each time you modify a rule and save it, Rule Designer builds the rule project by compiling its content.

[Automating builds](#)

You can automate the build of your projects in a continuous deployment pipeline by building your rule and Java projects as rule applications.

[Running and debugging](#)

Rule Designer provides a running and debugging environment to test the execution of a ruleset.

[Executing rulesets in the decision engine](#)

In Rule Designer, you can extract and package your rules for execution on different platforms.

[Optimizing execution](#)

You can improve performance and scalability of your application through the way Decision Server integrates with Java™, and settings for execution modes, algorithms, and various associated configuration files.

Overview: Building and running rules

Building rules involves the compilation of your rule projects and can include creating the corresponding rulesets or RuleApps.

Rule Designer proposes launch configurations to build a rule project into a ruleset archive, which you can run and debug in Rule Designer using a basic predefined engine.

You can use Rule Designer to build rule projects into RuleApps, which contain sets of ruleset archives that you can deploy to Rule Execution Server.

You can also automate the build process using Rule Designer in headless mode.

Parent topic: [Building and running rules](#)

Building rule projects interactively

Each time you modify a rule and save it, Rule Designer builds the rule project by compiling its content.

About this task

During the build process, Rule Designer detects problems and reports them in the Problems view:

- Errors: Syntax or structural defects
- Warnings: Indicates awkward or deprecated syntax in the valid parts of the rules

Some errors result from the way in which rules interact in a ruleset archive, which is the standard artifact that a rule engine can read. For example, if a project contains two rule with the same name, the resulting ruleset would contain an error because there cannot be two rules with the same name in a single ruleset. A typical case of this is when a rule generated from a decision table conflicts with a BAL rule of the same name.

Procedure

1. To build a rule project, click **Save**.

When you modify a rule and save it, Rule Designer builds the project.

2. To rebuild a rule project, click **Project > Clean**.

Results

After a successful build, you can use a launch configuration to generate a ruleset archive, and run and debug the ruleset.

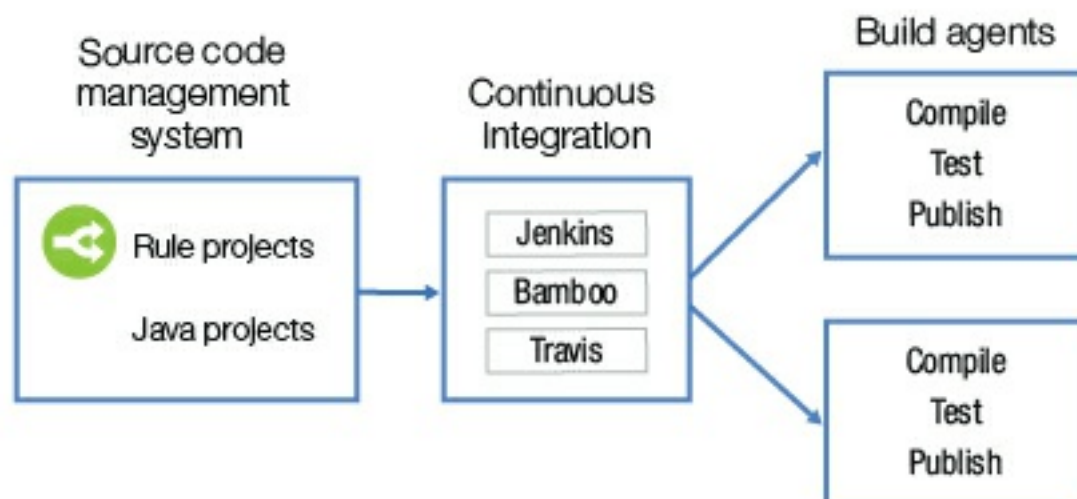
Parent topic: [Building and running rules](#)

Automating builds

You can automate the build of your projects in a continuous deployment pipeline by building your rule and Java projects as rule applications.

The following diagram shows an example of a continuous deployment pipeline architecture that uses Jenkins, Bamboo and Travis for continuous integration, and that describes the tasks that build agents do.

Figure 1. Diagram of a continuous deployment architecture



You can use Rule Designer in headless mode to automate the compilation tasks in build agents.

Adding global variables

Describes how to add a global variable.

Rule Designer build automation tool

Rule Designer provides a tool to automate builds and ruleset extraction from the command line. The build automation tool starts Rule Designer in headless mode, that is, with no user interface for the development environment.

Parent topic: [Building and running rules](#)

Adding global variables

Describes how to add a global variable.

About this task

You can use the `ilog.rules.studio.javascript` plug-in in conjunction with Eclipse to automate tasks on a preconfigured workspace or a folder containing the rule projects.

The following example demonstrates how to add a global variable named `out` to replace a fully qualified call to:

```
java.lang.System.out.println(<text message>);
```

Procedure

To add global variables:

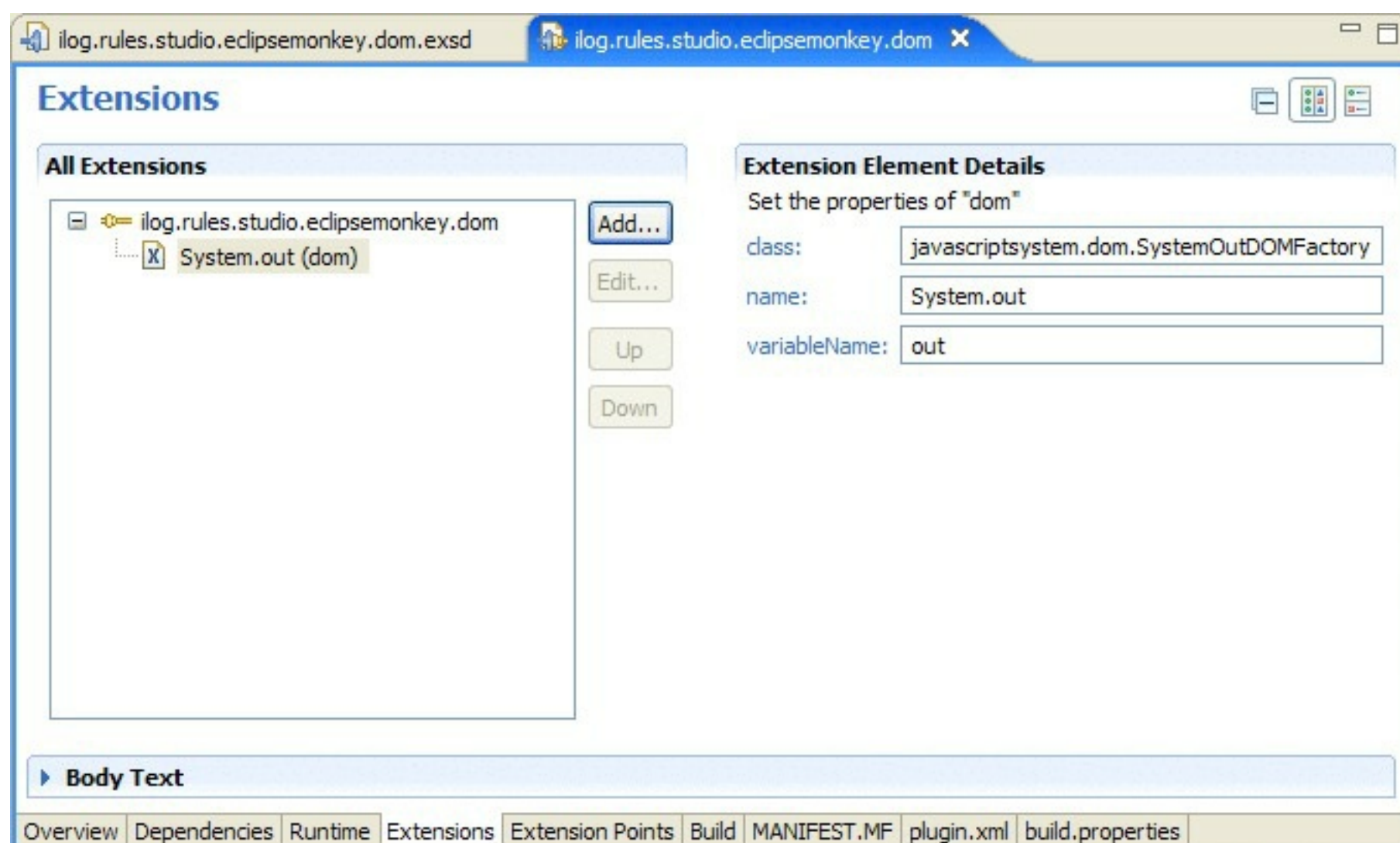
1. On the **File** menu, click **New** > **Project**.
2. Under the **Plug-in Development** category, select **Plug-in Project** and then click **Next**.
3. In the **project name** field, enter a name and then click **Next**.
4. Click **Finish**.
5. Click **Yes** to open the Plug-in Development perspective.
6. Click the Dependencies tab, and under **Required Plug-ins** click **Add**.
7. Add `ilog.rules.studio.eclipsemonkey` to the dependencies.
8. Click the Extension Points tab and then click **Add**.
9. In the **Extension Point ID** and **Extension Point Name** fields, enter `ilog.rules.studio.eclipsemonkey.dom` and then click **Finish**.
10. Click the `plugin.xml` tab and then enter the following text:

```
<extension point="ilog.rules.studio.eclipsemonkey.dom">
  <dom
    class="javascriptsystem.dom.SystemOutDOMFactory"
    name="System.out"
    variableName="out"/>
</extension>
```

The extension declares the factory that creates the Java™ instance on which the JavaScript method calls are forwarded when used with the `out` variable.

Results

The plug-in Extensions tab now includes the new plug-in project:



You must now add the implementation of the factory. Factories contain a single method named `getDOMroot`.

This method creates the Java instance of the class to which the method calls are forwarded.

The following SystemOutDOMFactory example returns System.out static instances:

```
package javascriptsystem.dom;

import ilog.rules.studio.eclipsemonkey.dom.IMonkeyDOMFactory;

public class SystemOutDOMFactory implements IMonkeyDOMFactory {
    public Object getDOMroot() {
        return System.out;
    }
}
```

By adding this global variable, you simplify each print command to the standard output. You can also create a more complex object that provides methods such as formatting text messages with parameters, as shown in the following example:

```
function main() {
    // instead of java.lang.System.out.println("my message");
    out.println("my message");
}
```

Parent topic: [Automating builds](#)

Rule Designer build automation tool

Rule Designer provides a tool to automate builds and ruleset extraction from the command line. The build automation tool starts Rule Designer in headless mode, that is, with no user interface for the development environment.

For IBM® AIX® platforms, only headless Eclipse mode is supported. Installing Rule Designer on IBM AIX is useful if you want to run automated builds or ruleset extraction from rule projects that are maintained on a platform where Rule Designer runs in full mode. You can also automate routine operations such as building rule projects or running queries. For advanced automation tasks, you can customize the automation using JavaScript.

In the same build, you import the projects into the workspace, apply an extension model, build the projects, and generate ruleset archives.

You can use the build automation tool to test builds in Rule Designer, and then run automated builds from the command line. The `ilog.rules.studio.automation` plug-in contains the `ilog.rules.studio.automation.builder` application.

You use arguments in the command line to specify the different steps that are involved in the build. For example, you use arguments to define the paths to the projects to import and to specify whether to remove existing projects from the workspace before building. All arguments are optional, but you must at least specify the name of the rule project to build.

Note:

You can use an extra argument on the command line: the name of the decision operation to use when you generate the ruleset archive for a decision service.

The list of rule projects to build is mandatory except when you use the help option **-h**.

Running the build automation tool from the command line

The build automation tool supports command-line arguments, which you can use to set up automated builds.

In the command line, you specify the arguments and the rule projects to build. You can choose the decision operation that you want to use for generating the ruleset archive. All arguments and extractors are optional, but you must at least specify the name of the rule projects to build.

To filter the rules when you generate a ruleset archive, use a question mark (?) between the rule project name and the extractor name or decision operation. For example, if you are using a decision service, write `<rule_project_name>?<operation_name>`.

To run the build automation tool for different operating systems, write command lines similar to the following examples:

Windows

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
```

You can then add the arguments, the rule projects to build, and the extractor to filter the rules. If your rule project name contains spaces, use double quotation marks as separators.

If you are using a decision service, use the following syntax:

```
-importPath "<path_to_xom_project>;<path_to_rule_project>"
<rule_project_name>?<operation_name>
```

Here is an example of a complete command line with no filter:

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
-importPath
"my_workspace/rulesetextraction/rulesetextraction_xom;my_workspace/rulesetextraction/rulesetextraction_rules"
-cleanBuild
-buildOptions BUILD_CHECK_IRL|BUILD_INCLUDES_RULE_VALIDATION
rulesetextraction_rules
```

UNIX

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
```

You can then add the arguments, the rule projects to build, and the extractor to filter the rules. If your rule project name contains spaces, use double quotation marks as separators.

If you are using a decision service, use the following syntax:

```
-importPath <path_to_xom_project>:<path_to_rule_project>
<rule_project_name>?<operation_name>
```

Here is an example of a complete command line with no filter:

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
-importPath
"my_workspace/rulesetextraction/rulesetextraction_xom:my_workspace/rulesetextraction/rulesetextraction_rules"
-cleanBuild
-buildOptions BUILD_CHECK_IRL|BUILD_INCLUDES_RULE_VALIDATION
rulesetextraction_rules
```

Command-line arguments for the automated build

The following table lists the command line arguments that you can use to define your automated build.

Table 1. Build command line arguments

Argument name	Description	Default
-? (-h)	Prints usage information and then exits.	-
-autoRefresh	Refreshes the workspace automatically if Eclipse determines that the contents of the workspace are changed. For example, set this argument to true when modify the contents of the workspace outside Eclipse.	false
-brdx <file>	Specifies the path to the .brdx extension model file to use. If you specify the .brmx file, you must also specify a .brdx file.	-
-brmx <file>	Specifies the path to the .brmx extension model file to use. If you specify the .brdx file, you must also specify a .brmx file.	-
-buildOptions	Specifies the build options to use, separated by the pipe () character. Build options that you do not specify are set to false, except for BUILD_GENERATE_IRL , which is always set to true. You can use the following build options:	BUILD_GENERATE_IRL is always set to true

	<ul style="list-style-type: none"> • BUILD_GENERATE_IRL: Generates IRL during build • BUILD_CHECK_IRL: Performs IRL checks during build • BUILD_INCLUDES_B2X_VALIDATION: Performs BOM to XOM checks during build • BUILD_INACTIVE_RULES: Builds rules for which the property "active" is false • BUILD_INCLUDES_RULE_VALIDATION: Performs rule analysis during build <p>The build is incremental.</p>	
-cleanBuild	Cleans the results of prior workspace builds before it rebuilds.	false
-clearWorkspace	Removes all projects from the workspace before it imports projects. Project contents remain intact.	false
-failFast	Stops the build if any rule project contains errors.	false
-importPath <val>	<p>Specifies the list of projects to import into the workspace before building. The list uses the <code>File.separator</code> system property.</p> <p>The paths of projects to import are separated by a semicolon (;) for Windows and by a colon (:) for UNIX.</p>	-
-nl <language>	Sets the language for the applications, for example, -nl en_US for American English.	-
-noSplash	Bypasses the splash screen on startup.	-
-output or -o <val>	<p>Specifies the name of the output directory to store the generated ruleset archive.</p> <p>You must specify an existing folder. If you do not specify an output directory, the ruleset archive is stored in the output directory of the rule project (<code>./output</code>). If you specify a relative path, the path is relative to the parent rule project that is being processed.</p>	-
-refresh	Refreshes the entire workspace prior to building projects.	false

For more information about the arguments supported by the [IlrAutomator](#) class, refer to [IlrBuildCommandLineArguments](#).

Parent topic: [Automating builds](#)

Running and debugging

Rule Designer provides a running and debugging environment to test the execution of a ruleset.

[Running and debugging decision operations](#)

To run a decision operation in standard or debugging mode, you must create a launch configuration by selecting the project in which the operation is stored.

[Debugging rulesets](#)

Rule Designer provides a set of tools to inspect the execution of the rules and the state of the engine, and set breakpoints on classes, objects, rules, decision tables, and ruleflows. The debugger monitors rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and running all the rules.

Parent topic: [Building and running rules](#)

Running and debugging decision operations

To run a decision operation in standard or debugging mode, you must create a launch configuration by selecting the project in which the operation is stored.

About this task

To run or debug a decision operation, the rule engine needs values for all the IN and IN_OUT parameters for your ruleset.

Procedure

To start creating a run or debug configuration:

- On the **Run** menu, click **Run Configurations**.
- On the **Run** menu, click **Debug Configurations**.

To complete the new launch configuration:

- Double-click the **Decision Operation** header to further configure the new run or debug configuration, and set up the parameters.

To run the decision operation in standard or debugging mode:

- According to your initial choice, click **Run** to start the execution, or click **Debug** to start the execution in debug mode.

Results

The ruleset is run in either standard or debugging mode, and you now have a launch configuration that is ready to be reused.

If a particular decision operation already contains values for all IN and IN_OUT parameters, you can also run or debug it with default settings as follows:

1. Select the decision operation or the rule project in which it is stored.
2. Choose **Run As > 2 Decision Operation** or **Debug As > 2 Decision Operation** from the **Run** menu.

If you select a rule project that has more than one decision operation, you are prompted to select one in the wizard.

Parent topic: [Running and debugging](#)

Debugging rulesets

Rule Designer provides a set of tools to inspect the execution of the rules and the state of the engine, and set breakpoints on classes, objects, rules, decision tables, and ruleflows. The debugger monitors rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and running all the rules.

[Ruleset debugging tools](#)

Use the Rule Designer debug tools to make sure that your rule application projects run as expected.

[Starting a debugging session](#)

You start a debugging session to test your rules and evaluate the runtime performance of a ruleset.

[Setting breakpoints in rules](#)

You can set breakpoints on rule expressions to do a step-through analysis of the rules.

[Setting breakpoints in the working memory](#)

You can set breakpoints on objects in working memory if you want execution to stop when the objects are retracted or updated.

[Setting breakpoints in the BOM](#)

You can set breakpoints on business object model (BOM) classes to stop execution when an object is added, retracted, or updated.

[Stepping through the rule execution code](#)

You can use the Debug Perspective to step into the code to debug a rule.

[Inspecting the working memory](#)

You can inspect the objects in working memory when the execution mode is RetePlus.

[Inspecting the agenda](#)

You can inspect the rules that are placed in the agenda when you debug a rule or Java™ project for rules.

[Viewing and evaluating expressions](#)

You can inspect expressions in the context of a stack frame when the virtual machine (VM) suspends a thread at a breakpoint or it is stepping through code.

[Viewing variable values](#)

You can view the variables in a stack frame.

[Viewing breakpoints](#)

You can view the breakpoints that you inserted.

Parent topic: [Running and debugging](#)

Ruleset debugging tools

Use the Rule Designer debug tools to make sure that your rule application projects run as expected.

[Debug mode](#)

You can execute a ruleset in debug mode to test rules, inspect execution of rules and engine state, and set breakpoints on classes, objects, rules, decision tables and trees, and rule flows.

[Debug view](#)

The Debug view displays the stack frame for the suspended threads for each target you are debugging. You can get a stack trace of the execution history.

[Working memory](#)

The Working Memory view display the objects in working memory for the RetePlus execution mode. Use it with the Agenda view to easily check whether the rules behave as expected.

[Agenda](#)

The Agenda view displays any rule instances scheduled for execution in RetePlus mode. Use it with the Working Memory view to quickly debug your applications.

[Variables](#)

The Variables view displays the names of the variables currently bound to a business rule and the parameters currently visible in the engine.

[Breakpoints](#)

The Breakpoints view displays the breakpoints that stop the execution of rules at any point to examine the state of variables, the agenda, and working memory.

[Console](#)

The Console view displays messages that have been sent to the output stream associated with the engine.

Parent topic: [Debugging rulesets](#)

Debug mode

You can execute a ruleset in debug mode to test rules, inspect execution of rules and engine state, and set breakpoints on classes, objects, rules, decision tables and trees, and rule flows.

In Rule Designer, you can debug both rule code and Java™ code in a project.

The Debug views are useful when you start testing and debugging the execution of your rules. The debugger monitors business rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and executing all the rules.

Debug actions

With the debugger, you can control the process of business rule execution:

- Step into, step over, and step return in any rule or code statement:
 - Step over: to go to the next rule statement.
 - Step into: to go to the next rule statement or stop in the next Java statement.
 - Step return: to go to the next rule statement that is not defined in the current rule artifact.
- Drop to frame or use step filters. Use **Window > Preferences > Java > Debug > Step Filtering** to set debug filters. (On Mac, click **Eclipse > Preferences > Java > Debug > Step Filtering**.)
- Resume, suspend, and terminate.
- Set breakpoints on rules, classes, and objects.

Debug mode performance

You can improve the performance of ruleset parsing when you are debugging a rule project in Eclipse by disabling this option: **Window > Preferences > Java > Debug > Suspend execution on compilation error** (On Mac, click **Eclipse > Preferences > Java > Debug > Suspend execution on compilation error**)

Note: The improvement factor depends on your Java virtual machine. Even when this preference is turned off, the debug mode remains slower than the regular run mode.

Parent topic: [Ruleset debugging tools](#)

Related information:

[Debugging rulesets](#)

Debug view

The Debug view displays the stack frame for the suspended threads for each target you are debugging. You can get a stack trace of the execution history.

From the Debug view, you can manage the debugging of a rule project or any Java™ program in the workbench. It displays the stack frame for the suspended threads for each target you are debugging. Each thread is shown as a node in the tree.

The Debug view does not provide a history of all the events occurring in the rule engine. However, a stack trace is a useful debugging tool, particularly when an exception has been thrown in the code or when a method is called from a variety of places within your code and you want to learn from where it is called when a particular problem occurs.

You can get stack trace information of the execution history. The stack lists the classes, and the methods within those classes, that are called at the point where the exception occurs through programmatic access. For example, you can use the `StackTraceElement` class and the `getStackTrace` method of the `Throwable` class.

You can also generate a partial Java stack trace by using the static `Thread.dumpStack` method or the `printStackTrace` method of the `Throwable` class.

If the JVM experiences an internal error, it calls its own signal handler to print out the threads and monitors information.

Parent topic: [Ruleset debugging tools](#)

Working memory

The Working Memory view display the objects in working memory for the RetePlus execution mode. Use it with the Agenda view to easily check whether the rules behave as expected.

The view can display the value and type of the various fields of the object. From that view, you can dynamically inspect the objects as they affect the rules. This is important to check whether a rule behaves correctly. For example, you can easily check the behavior of the rules currently in the agenda that use a specific value.

Parent topic: [Ruleset debugging tools](#)

Related information:

[RetePlus algorithm](#)

[Debugging rulesets](#)

Agenda

The Agenda view displays any rule instances scheduled for execution in RetePlus mode. Use it with the Working Memory view to quickly debug your applications.

The values in the Working Memory view are updated when the engine is updated. The view can display the priority of a rule in the agenda and the objects used by it. In conjunction with the Working Memory view, it provides you with a powerful way to debug an application quickly.

Parent topic: [Ruleset debugging tools](#)

Related concepts:

[RetePlus mode](#)

Related information:

[RetePlus algorithm](#)

Variables

The Variables view displays the names of the variables currently bound to a business rule and the parameters currently visible in the engine.

For example, if the ShoppingCart object is bound to a variable called ?shoppingCart in one of the rules, the Variables view shows this relationship. The values in the Variables view are updated after the evaluation of each expression.

Parent topic: [Ruleset debugging tools](#)

Related information:
[Debugging rulesets](#)

Breakpoints

The Breakpoints view displays the breakpoints that stop the execution of rules at any point to examine the state of variables, the agenda, and working memory.

You can set breakpoints on the following elements:

- Rule expressions
- XOM classes and class members
- Objects in the working memory

You can set breakpoints in the following rule artifacts:

- BAL rules (in the action part)
- Technical rules
- Decision tables and decision trees (in an action cell)
- Functions
- Ruleflows
- Rule tasks (in the Ruleflow Editor)
- Working memory of the rule engine (when executing with the RetePlus execution mode)
- Business object model (BOM)
- BOM-to-XOM mapping code

Parent topic: [Ruleset debugging tools](#)

Console

The Console view displays messages that have been sent to the output stream associated with the engine.

The Console shows the following text:

- Standard output
- Standard error
- Standard input

Parent topic: [Ruleset debugging tools](#)

Related information:
[Debugging rulesets](#)

Starting a debugging session

You start a debugging session to test your rules and evaluate the runtime performance of a ruleset.

About this task

Before deploying your rules, you should always test and debug them, and evaluate the runtime performance of a ruleset.

Procedure

To start a debug session in Rule Designer:

1. On the **Run** menu, select **Debug Configurations**.
2. In the **Debug Configurations** dialog, click a previously defined configuration in the **Configurations** pane.
3. Click **Debug**.
4. Click **Yes** to confirm the switch to the Debug perspective.

Results

You can then add and remove breakpoints, inspect the agenda and working memory, and take other debugging actions. This way, you can test and fix bugs in a decision operation for decision services in a Java™ application.

Parent topic: [Debugging rulesets](#)

Related information:
[Ruleset debugging tools](#)

Setting breakpoints in rules

You can set breakpoints on rule expressions to do a step-through analysis of the rules.

Setting breakpoints

You can set breakpoints on rule expressions to facilitate a step-through analysis of the rules.

Removing a breakpoint

When you no longer need a breakpoint, you can remove it.

Removing all breakpoints

You can remove all the breakpoints together.

Parent topic: [Debugging rulesets](#)

Setting breakpoints

You can set breakpoints on rule expressions to facilitate a step-through analysis of the rules.

About this task

You can set breakpoints only on the parts of the rule that are run, that is, function statements and rule actions. If you try to set a breakpoint on a line where breakpoints are meaningless, the breakpoint is set at the next valid line.

Procedure

To set a rule breakpoint in a rule:

1. In the Intellirule editor, select the line where you want to set a rule breakpoint.
2. Do one of the following actions:
 - Select **Toggle Breakpoint** either from the **Run** menu or after right-clicking in the shaded border at the rule.
 - Double-click the Intellirule editor margin.
 - Press Ctrl+Shift+B.

The margin of the Intellirule editor displays a blue dot.


Parent topic: [Setting breakpoints in rules](#)

Removing a breakpoint

When you no longer need a breakpoint, you can remove it.

Procedure

To remove a breakpoint:

1. Select **Toggle Breakpoint** either from the **Run** menu or by right-clicking the breakpoint in the shaded border at the rule.
2. Double-click the breakpoint in the Intellirule editor margin.
3. In the Breakpoints view, select the breakpoint that you want to remove and click **Remove Selected Breakpoints** .

Parent topic: [Setting breakpoints in rules](#)

Removing all breakpoints

You can remove all the breakpoints together.

Procedure

To remove all the breakpoints:

Select **Remove All Breakpoints**.

You can do this operation in only the Breakpoints view.

Parent topic: [Setting breakpoints in rules](#)

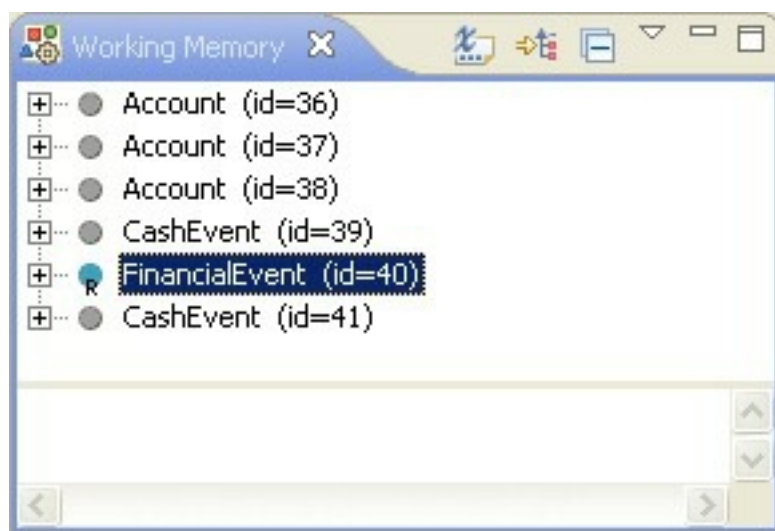
Setting breakpoints in the working memory

You can set breakpoints on objects in working memory if you want execution to stop when the objects are retracted or updated.

Procedure

To set a breakpoint:

1. Start a debug session.
2. In the **Working Memory** view, right-click the object on which you want to set a breakpoint, and then select one of the following options in the pop-up menu:
 - If you want execution to stop when you retract this object, click **Add/Remove Working Memory Breakpoint > Retract**.
 - If you want execution to stop when you update this object, click **Add/Remove Working Memory Breakpoint > Update**.



3. Continue debugging.

Results

Execution stops when the object is retracted or updated.

Parent topic: [Debugging rulesets](#)

Related information:
[Ruleset debugging tools](#)

Setting breakpoints in the BOM

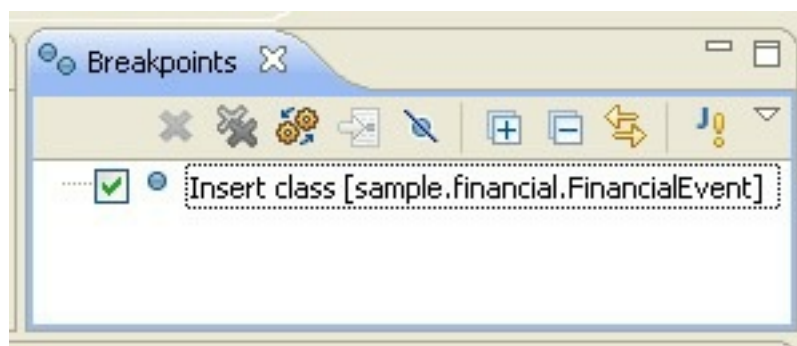
You can set breakpoints on business object model (BOM) classes to stop execution when an object is added, retracted, or updated.

Procedure

To set a breakpoint:

1. Choose the type of BOM class breakpoint you want to set:
 - To stop execution when adding an object of the class, click **Run > Add BOM Class Breakpoint > Insert**.
 - To stop execution when retracting an object of the class, click **Run > Add BOM Class Breakpoint > Retract**.
 - To stop execution when updating an object of the class, click **Run > Add BOM Class Breakpoint > Update**.
2. A dialog opens. In the **Class name** field, type the fully qualified name of the class on which you want to set a breakpoint, then click **OK**.
3. On the **Window** menu, click **Show View > Breakpoints**.

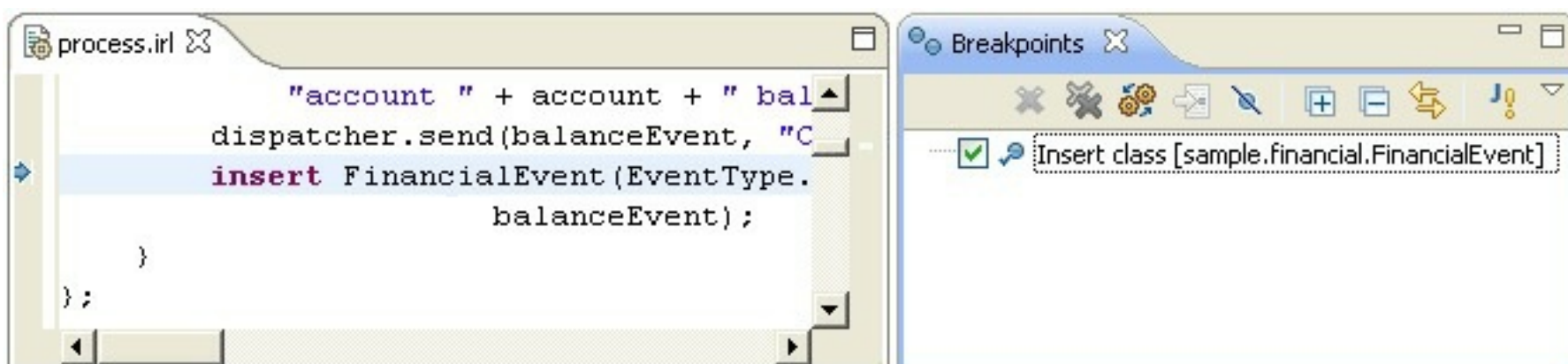
The Breakpoints view opens. You can see that a BOM class breakpoint has been added to the list of breakpoints.



4. Start debugging.

Results

Execution stops when an object of the BOM class is added, retracted, or updated.



Parent topic: [Debugging rulesets](#)

Stepping through the rule execution code

You can use the Debug Perspective to step into the code to debug a rule.

About this task

Several stepping options are available.

Note:

You cannot step into Java™ code if the rules are executed in sequential mode. When debugging, you can step into the BOM-to-XOM code, even if the sequential mode is used.

Procedure

To step into your IRL code:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

2. Select your configuration in the Configurations pane.
3. Make sure that you select the **Stop at first rule statement** check box.
4. Click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be executed.

5. Several stepping options are available to you:

- To call the next expression and suspend execution at the next unfiltered executable line, click **Run** > **Use Step Filters** or click the **Use Step Filters** button. You can also press Shift+F5.
- To call the next Java or rule statement and suspend execution at the next Java or rule line, click **Run** > **Step Into** or click the **Step Into** button. You can also press F5.
- To call the next Java or rule statement and suspend execution at the next rule line in the next rule, function, or rule task, click **Run** > **Step Over** or click the **Step Over** button. You can also press F6.
- To call the next Java or rule statement and suspend execution at the first rule statement of the next rule, function, or rule task, select **Run** > **Step Return** or click the **Step Return** button. You can also press F7.

Parent topic: [Debugging rulesets](#)

Inspecting the working memory

You can inspect the objects in working memory when the execution mode is RetePlus.

About this task

When you debug a rule project or a Java™ project for rules, and the execution mode is RetePlus, you can inspect the objects in the working memory.

Procedure

To inspect the working memory:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

2. Select your configuration in the Configurations area.
3. Make sure that the **Stop at first rule statement** box is selected, and then click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be run.

4. Step into the execution statements and check that the Working Memory view is open.

Results

When objects are added into the working memory, they are shown in the Working Memory view. When you expand objects, you can inspect their values.

Parent topic: [Debugging rulesets](#)

Related information:
[Ruleset debugging tools](#)

Inspecting the agenda

You can inspect the rules that are placed in the agenda when you debug a rule or Java™ project for rules.

Procedure

To inspect the agenda:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

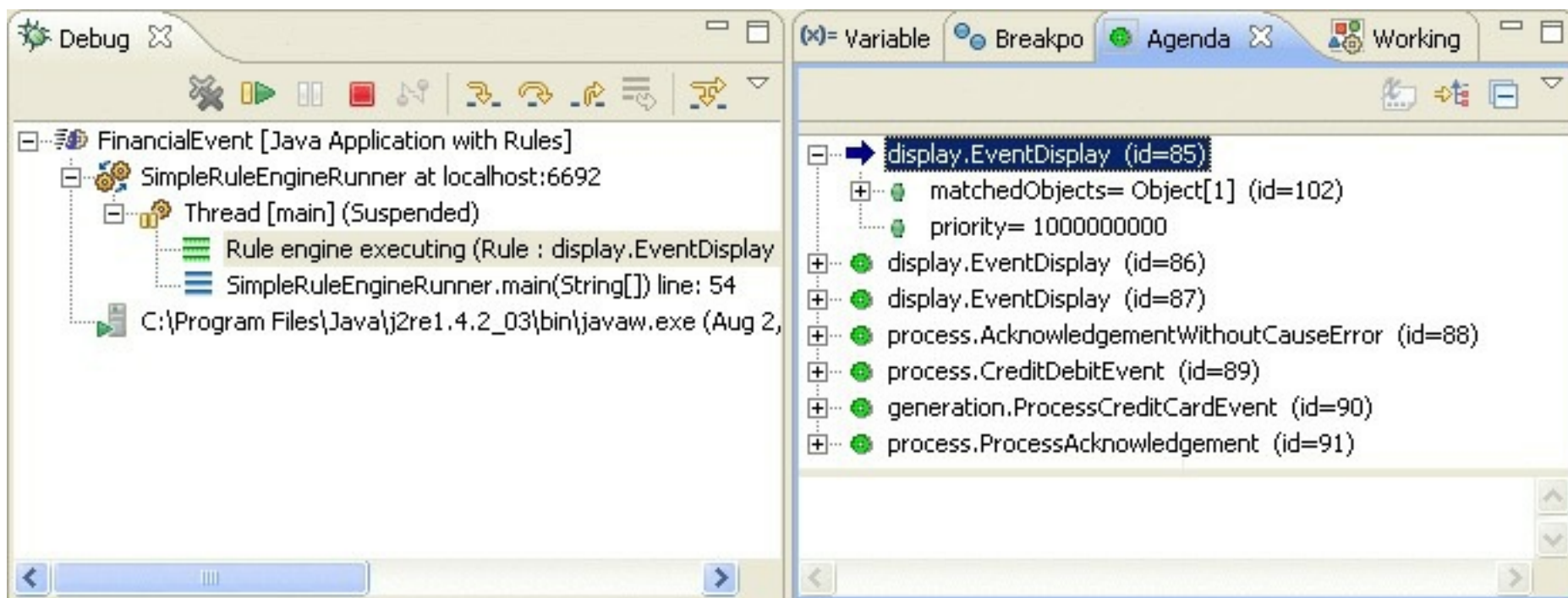
2. In the Configurations area of the Debug Configurations dialog, select your configuration.
3. Make sure the **Stop at first rule statement** box is selected, and then click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be executed.

4. Step into the execution statements and check that the Agenda view is open.

Results

When objects in the working memory match the conditions of a rule, the rule is added to the agenda. The rule is shown in the Agenda view, and when you expand it, you can inspect the matched objects and their values.



Parent topic: [Debugging rulesets](#)

Viewing and evaluating expressions

You can inspect expressions in the context of a stack frame when the virtual machine (VM) suspends a thread at a breakpoint or it is stepping through code.

About this task

When the VM suspends a thread, you can inspect expressions in the context of a stack frame.

Procedure

To view an expression:

1. On the **Windows** menu, click **Show View > Expressions**.

You can then select the expression to do an evaluation.

2. In the **Detail** pane of the **Expressions View**, right-click the expression and click one of the options: **Display**, **Inspect**, or **Execute**.

Results

The result of a Display or Inspect evaluation is shown in a pop-up window. The **Execute** option does not display a result; the expression is run.

Parent topic: [Debugging rulesets](#)

Related information:

[Ruleset debugging tools](#)

Viewing variable values

You can view the variables in a stack frame.

About this task

When the stack frame is selected, you can see the visible variables in that stack frame in the Variables View.

Procedure

To view the variables:

1. On the Windows menu, click **Show View > Variables**.

The Variables View shows the value of primitive types.

2. Expand the variables to examine their values and display their members.

Parent topic: [Debugging rulesets](#)

Viewing breakpoints

You can view the breakpoints that you inserted.

Procedure

To view the breakpoints:

1. On the **Windows** menu, click **Show View > Breakpoints**.
2. Right-click a breakpoint, and then click one of the following options:
 - **Go to File**: Opens the file in which the breakpoint is placed.
 - **Enable/Disable**: Makes the breakpoint active or inactive.
 - **Remove/Remove All**: Removes the selected breakpoint or all the breakpoints.

Parent topic: [Debugging rulesets](#)

Executing rulesets in the decision engine

In Rule Designer, you can extract and package your rules for execution on different platforms.

You define the rule engine to execute your rules in the properties of the rule project. When you change the **Rule Engine** property for a rule project, it also applies to the projects that it references. For decision services, you select the rule engine in the properties of the decision service main project.

In Operational Decision Manager on Cloud, decision services use the decision engine. When you create a decision service in the cloud portal, make sure that the rule project is set to the decision engine.

Decision engine

The decision engine optimizes the execution performance of your ruleset. Before deployment, Decision Center compiles the rules into executable code (Java™ bytecode) by default.

Rule Designer compiles the rules into executable code by default. Clear the **Optimize ruleset loading (Java bytecode generation)** option on the Export a Ruleset Archive page to compile the rules to intermediate code.

You can choose the Fastpath, sequential, or RetePlus execution mode. With the working memory and agenda features, you can store and manipulate application objects. The working memory contains references to the application objects. The agenda lists and orders the rule instances that are eligible for execution.

Workflow for running rules on the decision engine

To run rules with the decision engine, do the following steps:

1. Create a rule project to encapsulate the business logic of your legacy applications.
2. If necessary, change the **Rule Engine** property of the rule project to **Decision engine** instead of **Classic rule engine**. When the rule project contains a BOM-to-XOM mapping in ILOG Rule Language (IRL), the mapping migrates to a BOM-to-XOM mapping in ARL. The migration happens only once. For more information, see [BOM-to-XOM mapping migration](#).
3. Test the execution of the rules in Rule Designer.
4. After you create decision operations and deployment configurations, you can publish the decision service to Decision Center on the cloud.
5. After you update the decision service in Decision Center on the cloud, you can deploy the modified decision service to the Rule Execution Server in the development environment.
6. After you test and validating the decision service, you can deploy it to the production environment.

BOM-to-XOM mapping migration

The migration of the BOM-to-XOM mapping is carried out the first time you change the **Rule Engine** property of the rule project to **Decision engine** instead of **Classic rule engine**. The original BOM-to-XOM mapping that is written in IRL is migrated to the Advanced Rule Language (ARL). In the BOM editor, you can work directly with ARL to define the BOM-to-XOM mapping for rule projects that are designed for decision engine.

The migration does not modify the original BOM-to-XOM mapping in IRL (.b2x file), but creates a new file (.b2xa) to store the migrated code instead, which enables the user to switch back and forth between engines.

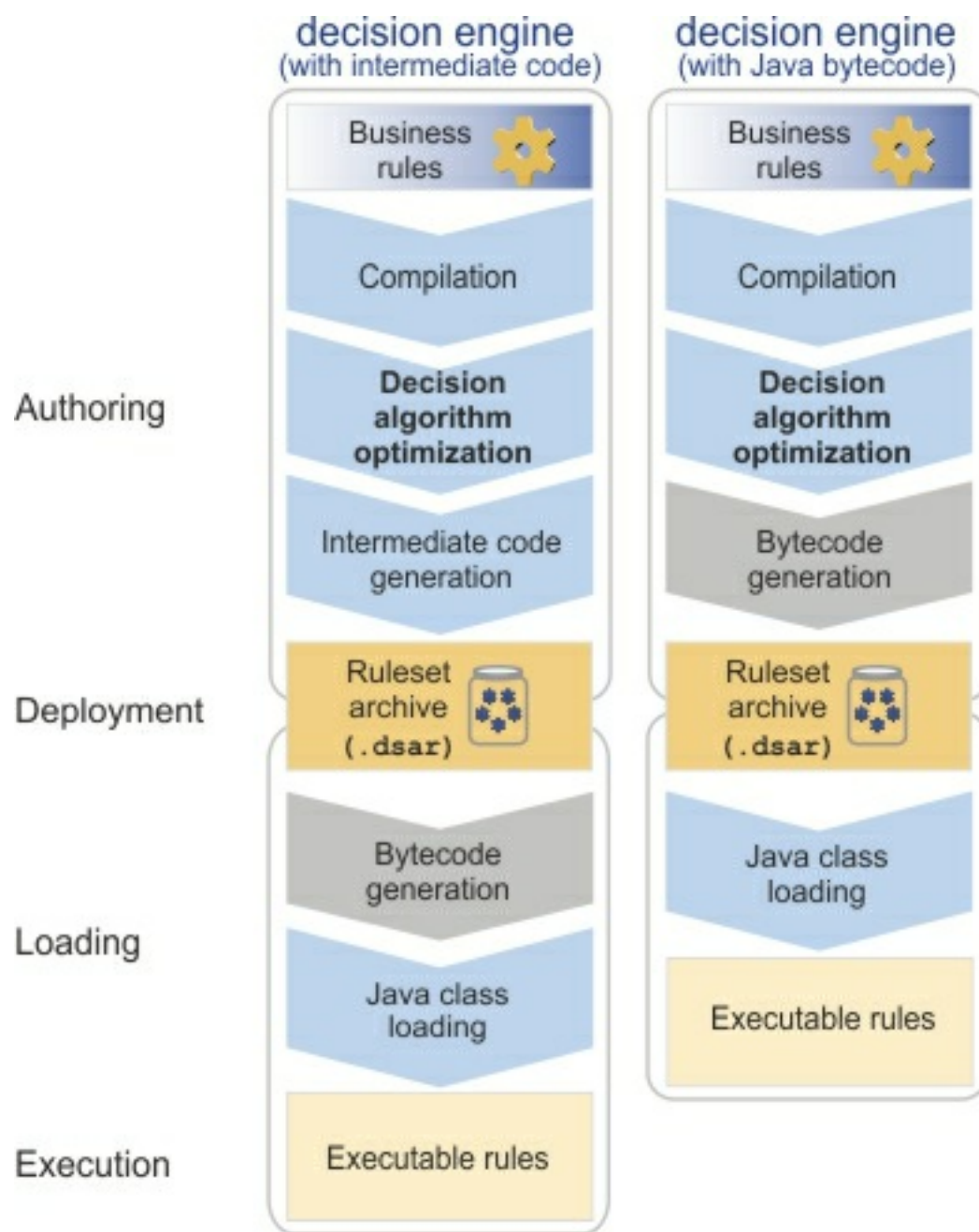
When created, each file separately retains the changes that you make for a particular engine, but there is no synchronization of the changes between the two files.

Migrated code can contain an empty body if the original body contains statements that cannot be migrated.

Compilation and execution

The decision engine compiles rule artifacts into an archive that contains compiled and optimized code that becomes executable when converted to Java bytecode. The ruleset archive .dsar file consists of binary files that contain execution code for the rules and ruleflows. You either deploy intermediate code or Java bytecode.

The following figure shows the process of compilation and execution for the decision engine, with or without Java bytecode generation. This process goes through different stages from the initial compilation of rules until the execution of rules.



Important: Bytecode generation improves the loading time of rulesets that are built with the decision engine. In Rule Designer and Decision Center, the bytecode generation option is selected by default.

Parent topic: [Building and running rules](#)

Related concepts:
[Optimizing the decision engine](#)

Optimizing execution

You can improve performance and scalability of your application through the way Decision Server integrates with Java™, and settings for execution modes, algorithms, and various associated configuration files.

Choosing an execution mode

You can change the execution mode for any task of a rule. Your choice depends on various criteria, and affects the connection between rules and application data.

Exception handling in rules

Exceptions in rule conditions prevent rules from being fired. This default behavior stops rule processing so that you can interpret the cause and severity of the exception. You can alternatively handle exceptions by allowing the automatic processing of some exceptions, or by customizing a response to a specific exception.

BAL building blocks

The **in** and **from** building blocks help you optimize performance.

Optimizing the decision engine

You can optimize the performance of the decision engine by adjusting your rule conditions, and by selecting the most efficient execution mode for your rules.

Parent topic: [Building and running rules](#)

Choosing an execution mode

You can change the execution mode for any task of a rule. Your choice depends on various criteria, and affects the connection between rules and application data.

Engine execution modes

Decision Server supports the RetePlus, sequential, and Fastpath execution modes. The execution mode affects which rules are executed and in which order.

Deciding on an execution mode

You can choose a different execution mode than the default one and provide selection criteria.

Changing the execution mode

You can use the `algorithm` property to change the execution mode for a rule task.

Data accessibility

The execution mode you choose affects the connection between rules and data.

Parent topic: [Optimizing execution](#)

Engine execution modes

Decision Server supports the RetePlus, sequential, and Fastpath execution modes. The execution mode affects which rules are executed and in which order.

RetePlus mode

Use RetePlus optimization techniques to improve performance through reduction of the number of rules and conditions, computation of the rules to run, and prioritization of the rule order.

Sequential mode

The sequential mode provides performance advantages by running all the eligible rules for a rule task in sequence.

Fastpath mode

Fastpath is a sequential execution mode that also detects semantic relations between rule tests in the same way as RetePlus. However, unlike RetePlus, the Fastpath mode does not support inference.

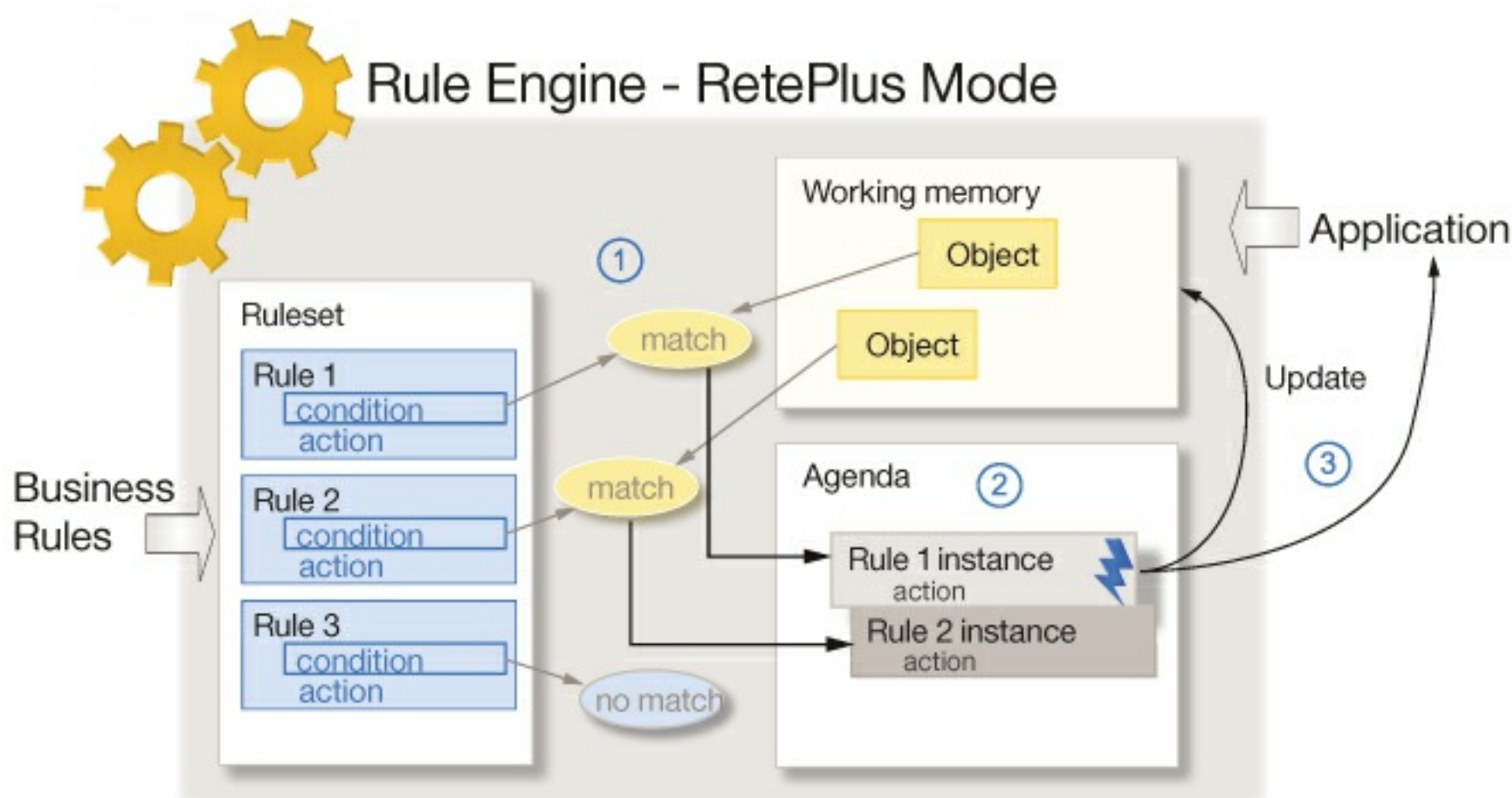
Parent topic: [Choosing an execution mode](#)

RetePlus mode

Use RetePlus optimization techniques to improve performance through reduction of the number of rules and conditions, computation of the rules to run, and prioritization of the rule order.

In RetePlus mode, the rule engine minimizes the number of rules and conditions to be evaluated, computes which rules must be run, and identifies in which order these rules must be run. In RetePlus, the rule engine uses a *working memory* and an *agenda* for storing and manipulating application objects. The working memory contains references to the application objects. The agenda lists and orders the rule instances that are eligible to be run.

The following diagram shows how the RetePlus algorithm works.



The RetePlus algorithm operates as follows:

1. The rule engine matches the conditions of the rules in the ruleset against the objects in working memory. During the pattern matching process, RetePlus creates a network based on semantic relationships between rule condition tests.
2. For each match, a rule instance is created and put into the agenda. Then, based on some ordering principles, the agenda selects the rule instance to be run.
3. When the rule instance is executed, the rule action is executed.

This action modifies the working memory in the following way

- By adding an object to the working memory
- By removing an object from the working memory
- By modifying the attributes of an existing object.

4. The process carries on cyclically until no more rule instances are left in the agenda.

In RetePlus, whenever the working memory is modified, the rule engine repeats the pattern matching process. It reassesses matches after each rule instance is run and modifies the data. As a possible consequence, the list of rule instances in the agenda can change. Thus, RetePlus is incremental and data-driven. The RetePlus algorithm is based on an inference process that the sequential and FastPath algorithms do not support.

Note:

As of V8.9.0, Reteplus for the decision engine considers the value instead of the reference to identify rule instances. For example, a list that includes several strings with the same value does not activate one rule fired for each list instance, as is the case with the classic rule engine.

Parent topic: [Engine execution modes](#)

Related information:

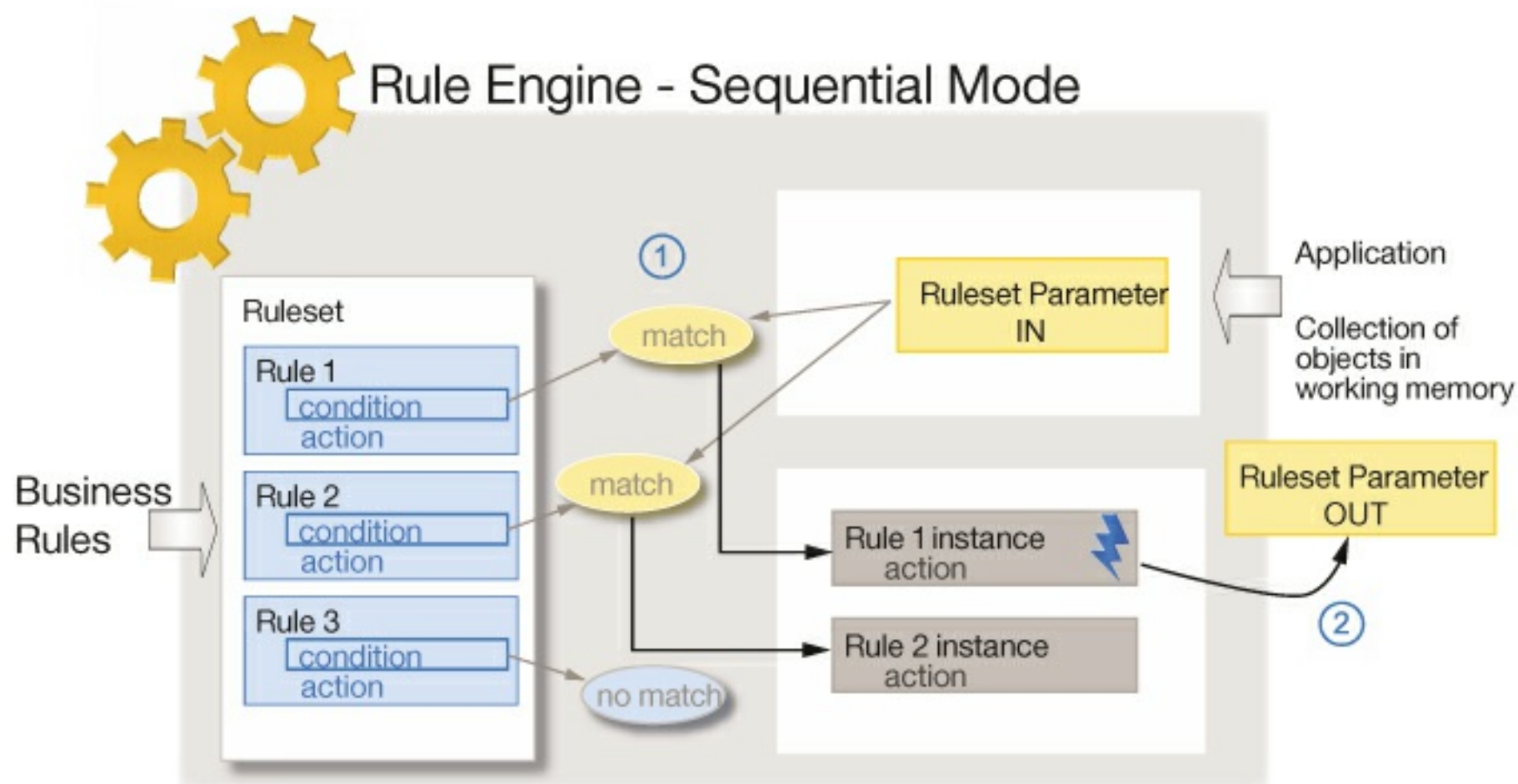
[RetePlus algorithm](#)

[Improving the performance of the RetePlus execution mode](#)

Sequential mode

The sequential mode provides performance advantages by running all the eligible rules for a rule task in sequence.

The following diagram shows how the sequential algorithm works.



The sequential algorithm operates as follows:

1. The rule engine does pattern matching on input ruleset parameters and on the conditions defined on the collections of objects in working memory.
2. For each match, a rule instance is created and immediately run. When a rule instance is run, it sets the value of an attribute or an output ruleset parameter.

Rules that are run with the sequential algorithm are stateless. The sequential algorithm operates rather like an execution stack where pattern-matching rule instances are run once with no reevaluation of the rules. In rules that are run in sequential mode, you cannot use existence conditions, such as `there is at least one`, or the `number of`, in relation to objects in the working memory. However, you can make such conditions refer to a collection within an object in the working memory, see [in](#).

Because of its systematic nature, the sequential execution mode does well on validation and compliance types of applications.

Rules can be processed sequentially by using rule tasks within a ruleflow, see [Ruleflows](#).

Sequential processing is specified in the `algorithm` property of the rule task. You can select it explicitly in Rule Designer. See [Choosing an execution mode](#).

Parent topic: [Engine execution modes](#)

Related information:

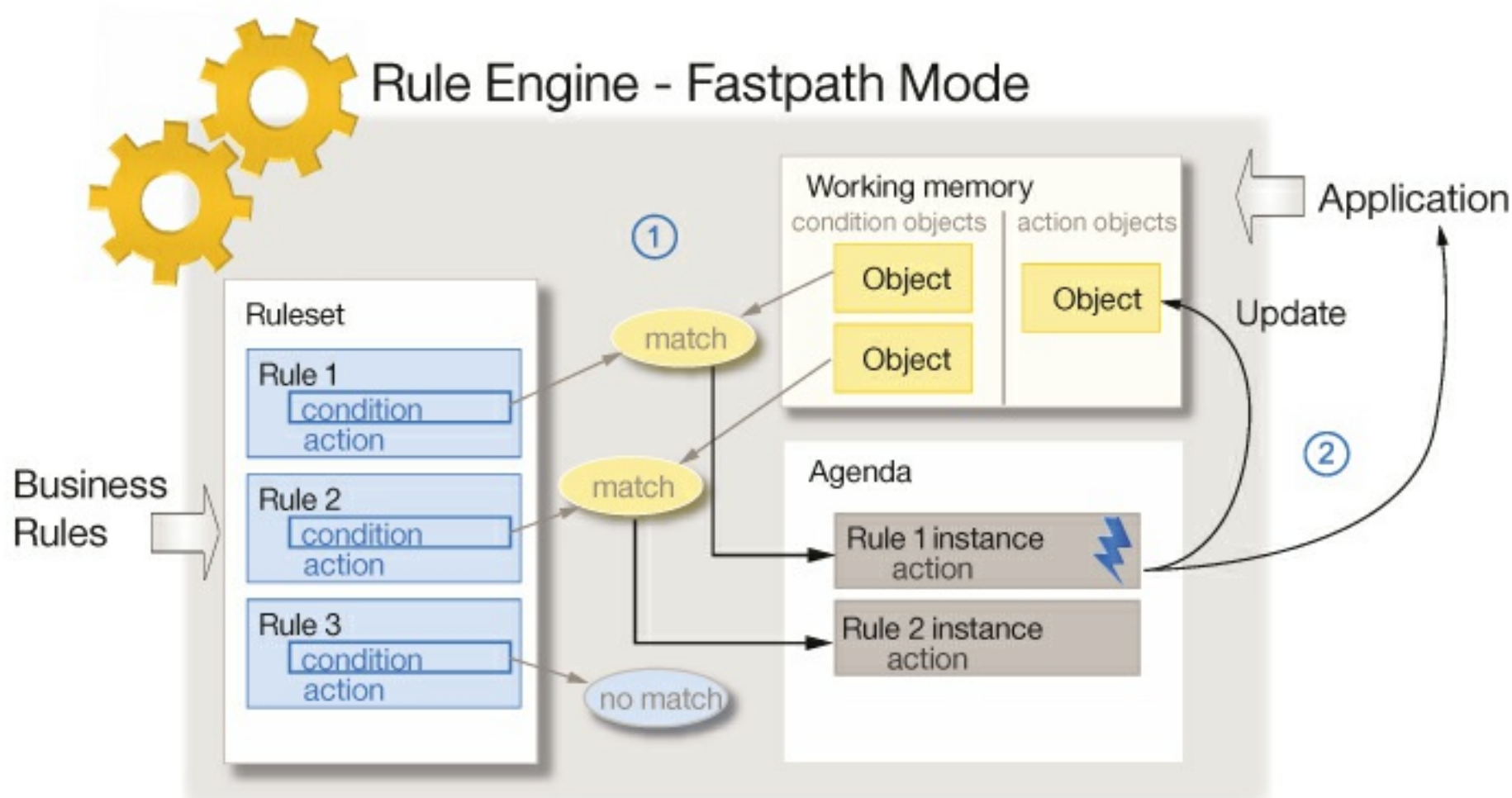
[Sequential algorithm](#)

Fastpath mode

Fastpath is a sequential execution mode that also detects semantic relations between rule tests in the same way as RetePlus. However, unlike RetePlus, the Fastpath mode does not support inference.

The Fastpath execution mode improves the sequential compilation and execution of a rule project. Fastpath is a sequential mode of execution that also detects semantic relations between rule tests during the pattern matching process, like RetePlus.

The following diagram shows how the Fastpath algorithm works.



The Fastpath algorithm operates as follows:

1. The rule engine uses a working memory that references application objects or ruleset parameters. Fastpath does the pattern matching process, as in RetePlus, by creating a tree based on semantic relations between rule condition tests.
2. For each match, a rule instance is created and inserted in the agenda.
3. After the pattern matching process, the rule instances in the agenda are executed.
4. The rule engine stops after the rule instances have been executed. This behavior also depends on the exit criteria of the rule task. The pattern matching process is not repeated.

Fastpath combines features of the RetePlus mode for pattern matching and features of the sequential mode for rule execution. In this sense, it compares well for correlation types of applications as well as for validation and compliance applications.

Like the sequential mode, the Fastpath mode is stateless. As such, it is dedicated to matching objects against a very large number of rules that individually do simple discriminations or light join tests. It is best for very large number of rules to be executed in sequence directly without any inference support. In addition to its advantages as a variant of the sequential mode, the Fastpath execution mode is designed to further optimize the execution of the compliance and validation rules, which constitute a substantial part of business rules.

Parent topic: [Engine execution modes](#)

Related information:
[Choosing an execution mode](#)

Deciding on an execution mode

You can choose a different execution mode than the default one and provide selection criteria.

You can choose an execution mode for each rule task of a ruleflow. By default, a rule task is assigned the Fastpath execution mode when you create it. To achieve optimal performance, you might need to choose another execution mode that is better adapted for the rules in a particular rule task.

Here is a brief table that shows which execution mode to select in what cases.

Table 1. Quick view of execution mode selection

Choose this mode:	In this case:
RetePlus	<ul style="list-style-type: none">• All included semantically.• Stateful application.• Rule chaining.• Useful in the case of many objects and limited changes.
Sequential	<ul style="list-style-type: none">• Covers most cases.• Many rules, few objects. Has limitations. Use with homogeneous rules.• Highly efficient in multi-threaded environment.
Fastpath (the default mode for ruleflow tasks)	<ul style="list-style-type: none">• Rules implementing a decision structure, many objects.• Highly efficient in multi-threaded environment.

To determine which execution mode to use on a rule task, you must analyze the structure of the rules in the rule task and what type of processing they do.

Note:

You cannot use the Fastpath or Sequential algorithm when the rules have actions that create, modify, or remove objects that are matched in the conditions of the rules.

To make the best choice, answer the following questions:

- [What type of application do your rules implement?](#)
- [What types of objects do your rules use?](#)
- [What is the effect of rule actions?](#)
- [What sort of tests do you find in rule conditions?](#)
- [What priorities have you set on your rules?](#)

What type of application do your rules implement?

Depending on the purpose of the business logic that is defined in a rule task, you choose a different execution mode.

Compliance and validation

Loosely interrelated rules that check a set of conditions to yield a go/no go or similar constrained result. Compliance business rule applications are often used in underwriting, fraud detection, data validation, and form validation. Business rules in such applications generally have a yes or no result and provide some explanation on the decision.

For compliance applications, preferably use the **sequential** or **Fastpath** execution modes.

Computation

Strongly interrelated rules that compute metrics for a complex object model. Computation business rule applications are often used for scoring and rating, contracts, and allocation. Business rules in such applications carry out different calculations on an object that is responsible for providing a final value (or rating).

For such applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

Correlation

Strongly interrelated rules that correlate information from a set of objects to compute some complex metrics. Correlation business rule applications are often used for billing. Business rules in such applications insert information.

For correlation applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

Stateful session

Strongly interrelated rules that correlate events in a stateful engine session. Stateful applications are often used in alarm filtering and correlation, GUI customization, and web page navigation.

For such applications, preferably use the **RetePlus** execution mode without a ruleflow.

What types of objects do your rules use?

Depending of the types of objects acted upon by your rules, you choose a different execution mode.

Homogeneous or heterogeneous rules?

Bindings are heterogeneous when rules do not operate on the same classes. When bindings are heterogeneous, the rules might have condition parts of different heights. For example:

Rule	Condition
Rule1	... when{A();B()} ...
Rule2	... when{A()} ...
Rule3	... when{B()} ...

If your rules define heterogeneous bindings, use the **RetePlus** or **Fastpath** execution mode.

Bindings are homogeneous when all the rules operate on the same class (the same kind and number of objects), but introduce different tests, for example:

Rule	Condition
Rule1	... when{Person(age == 12);} ...
Rule2	... when{Person(age > 20);} ...

If your rules define homogeneous bindings, use the **sequential** execution mode.

What is the effect of rule actions?

Depending of the types of effects generated by your rules on execution, you choose a different execution mode.

Modifications to the working memory

If rule actions manipulate working memory objects and use of the IRL keywords insert, retract, or update, use the **RetePlus** execution mode. Because these keywords entail modifications to the working memory, the rule engine reevaluates subsequent rules. If you use another execution mode, the rule engine does not reevaluate subsequent rules after the modifications.

Rule chaining

When rule actions cause modifications in the working memory or in the parameters, and when they do pattern matching on objects that have no relationship, like people and hobbies, there is chaining between your rules. This process is also known as inference.

For example:

SILVER LEVEL CUSTOMER, GREATER THAN \$5000 purchase
promote to GOLD LEVEL
GOLD LEVEL CUSTOMER, GREATER THAN \$5000 purchase
apply 25% discount

You can see there is chaining between these two rules because they do pattern matching on two different objects, customer and purchase, and that the second one modifies the level attribute of the customer object.

Basically, if you know your rule actions cause the execution of other rules, use the **RetePlus** execution mode.

What sort of tests do you find in rule conditions?

Depending on the type of conditions used in your rules, you choose a different execution mode.

Tests that require a working memory

If rule conditions test the existence of an object or gather items of a collection directly in working memory,

with the IRL keywords exists or collect, without in or from constructs, use the **RetePlus** or **Fastpath** execution modes.

Regular pattern for tests

If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, use the **Fastpath** execution mode.

If the order of tests in rule conditions is not regular, use the **RetePlus** or **sequential** execution modes.

What priorities have you set on your rules?

Depending on the priorities you have set on the rules, you choose a different execution mode.

- If you have set static priorities, you can use any algorithm.
- If you set dynamic priorities, that is, priorities that are defined as an expression, you must use the **RetePlus** execution mode.

Summary

You can use the following table as a reference to make your decision when you choose an execution mode for a rule task. The number of stars indicates the degree of performance.

Table 2. Choosing an execution mode

In your rule task:	RetePlus	Sequential	Fastpath
Compliance and validation application	★	★★★★	★★★★
Computation application with inference	★★★★	⊖	⊖
Computation application without inference	★	★	★★★★
Correlation application with inference	★★★★	⊖	⊖
Correlation application without inference	★	★	★★★★
Working memory objects	★★★★	★★★★	★★★★
Rule chaining	★★★★	⊖	⊖
Tests on existence or collection items directly in working memory	★★★★	⊖	★★★★
Shared test patterns	★★	★	★★★★
Heterogeneous bindings	★★★★	⊖	★★★★
Dynamic priorities	★★★★	⊖	⊖
Runtime rule selection that selects a few rules among many	★★★★	★★★★	Decision engine: ★ ★
Numerous rules	★	★★★★	★★★★

Parent topic: [Choosing an execution mode](#)

Changing the execution mode

You can use the algorithm property to change the execution mode for a rule task.

About this task

You can specify the default execution mode by using the Preferences dialog.

Procedure

To select an execution mode on the currently selected task:

1. Click **Rule Task** in the Properties view.
2. Select one of the algorithms: RetePlus, Sequential, or Fastpath.

Note:

Fastpath is the default mode for all newly created rule tasks. You do not have to choose Fastpath unless you are changing the execution mode of a rule task back to Fastpath. For rule tasks from a previous release, where RetePlus was the default mode, consider switching the tasks to Fastpath if you do not need inference.

Parent topic: [Choosing an execution mode](#)

Related information:

[Ruleflows](#)

[Engine execution modes](#)

[Working with ruleflows](#)

Data accessibility

The execution mode you choose affects the connection between rules and data.

In general:

- In sequential mode, preferably use direct data connection with `from` or `in` keywords plus ruleset parameters or variables.
- In RetePlus and Fastpath modes, favor evaluation in the working memory.

The following table summarizes the effect on data accessibility of changing from one execution mode to another.

Table 1. Changing execution modes

From Mode1 to Mode2	Effect on data accessibility
RetePlus -> Sequential	<p>Sequential mode has limitations compared to RetePlus (see Sequential algorithm).</p> <p>When the rules are connected to the data via working memory, executing traces is different in RetePlus/Fastpath and sequential.</p> <ul style="list-style-type: none">• RetePlus: foreach rule by priority, all the tuples.• Sequential: foreach tuple, all the rules by static priority. <p>When the rules are connected to the data via <code>in</code> and <code>from</code> keywords, the data is typically in ruleset parameters and variables. Only the executing trace should differ, as described above.</p>
RetePlus -> Fastpath	<p>Fastpath is a sequential-type algorithm, with static priorities and no support for update. However, the change to Fastpath should not affect the executing trace.</p>
Fastpath -> Sequential	<p>Fastpath has fewer rule condition limitations than the sequential mode. The executing trace of Fastpath is the same as RetePlus.</p> <p>When rules are connected to the data via working memory, there is the following difference:</p> <ul style="list-style-type: none">• Fastpath: foreach rule by priority, all the tuples.• Sequential: foreach tuple, all the rules by static priority.
Fastpath -> RetePlus	No effect.
Sequential -> RetePlus	Only the executing trace differs, as described above for RetePlus to Sequential.
Sequential -> Fastpath	Only the executing trace differs, as described above for RetePlus to Sequential.

Parent topic: [Choosing an execution mode](#)

Exception handling in rules

Exceptions in rule conditions prevent rules from being fired. This default behavior stops rule processing so that you can interpret the cause and severity of the exception. You can alternatively handle exceptions by allowing the automatic processing of some exceptions, or by customizing a response to a specific exception.

Automatic exception handling in conditions and custom exception handler are two distinct features. If both features are activated together:

- Exceptions in conditions that are managed by automatic exception handling are caught and not seen by the `CustomExceptionHandler`
- Exceptions in actions are not handled by the automatic exception handling feature (only in the case of a rule variable defined in the condition part, and reused in the action part), so these exceptions trigger the execution of `handleActionException`

Automatic exception handling in conditions

You can automate exception handling at the decision service project level so that the rule engine continues to process rule conditions that generate Java exceptions. When you activate automatic exception handling in conditions, the engine handles four Java exception types by default, but you can customize the list of Java exception types that the rule engine handles.

Examples of automatic exception handling

When you enable automatic exception handling in conditions, the rule engine handles specific subclasses so that rule processing can continue after exceptions are thrown. You can understand the logic behind automatic exception handling by reviewing examples.

Setting a custom exception handler

You can set an exception handler to customize exception handling in rule conditions and actions. When you implement the `CustomExceptionHandler` API, exceptions can either be ignored or rethrown.

Parent topic: [Optimizing execution](#)

Automatic exception handling in conditions

You can automate exception handling at the decision service project level so that the rule engine continues to process rule conditions that generate Java exceptions. When you activate automatic exception handling in conditions, the engine handles four Java exception types by default, but you can customize the list of Java exception types that the rule engine handles.

The rule engine ignores unknown values during the evaluation of the rule condition when automatic exception handling is enabled. You can enable and disable this feature in Rule Designer in the Rule Engine panel of the rule project properties window. Automatic exception handling is disabled by default.

When you enable automatic exception handling, The following subclasses are handled by the rule engine when automatic exception handling is enabled:

- ArithmeticException
- NumberFormatException
- NullPointerException
- IndexOutOfBoundsException

Important: Exceptions that occur in the transition condition of a ruleflow are not automatically handled by the rule engine.

With automatic exception handling in conditions enabled, expressions in conditions that result in exceptions during their evaluation are treated as unknown values and are handled by the rule engine. The rule engine evaluates the overall rule conditions by using the following three-valued logic:

NOT(A)		AND(A,B)					OR(A,B)				
				B					B		
A	¬A	A ∧ B		F	U	T	A ∨ B		F	U	T
F	T		F	F	F	F		F	F	U	T
U	U	A	U	F	U	U	A	U	U	U	T
T	F		T	F	U	T		T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

If the rule engine evaluates the rule condition as true or false, despite unknown values in the condition expressions, the rule action logic is applied. If the rule engine evaluates the rule condition as unknown, because of unknown values in the condition expressions, the rule is not fired.

For example, in the following rule, when the value of the name of the customer is null, and the age of the customer is 20, the first condition cannot be resolved, and the second condition is met:

```
if
  the name of the customer is 'Paul'
  or the age of the customer is 20
then
  reject the loan;
```

Because the rule uses the logical or operator, the action to reject the loan is taken. If the age of the customer is not 20, the first condition cannot be resolved, and the second condition is not met. In this case, the rule is not fired, and the loan is not rejected.

See [Examples of automatic exception handling](#) for more examples.

Conditions that use collections

When the rule engine assembles [collections](#) of objects, the objects for which the where clause is unknown or false are ignored. Consequently, the test on the number of objects in a collection reflects this behavior: Only objects that have a where clause evaluated true are counted.

The following business language expressions use tests on the number of objects:

- If the number of ...
- If there is no ...
- If there is at least one ...
- If there are at least <number> ...
- If there are more than <number> ...
- If there are less than <number> ...
- If there are <number> ...

The following example shows a condition that uses collections and tests the number of objects. In the example, the collection (the `borrowers`) contains only borrowers for whom the attribute values are known. So, when the count operation is performed (is less than 3), any borrower for whom an attribute is unknown is not included in the count. In other words, if 3 borrowers are evaluated and the name of one borrower is unknown while the name of the two other borrowers is the name of the `'borrower'`, then the condition is `True` and the then action is taken.

```
if
    the number of borrowers where the name of each borrower is the name of 'the
    borrower',
    is less than 3,
then
    add "less than 3 similar borrowers found" to the messages of 'the loan' ;
else
    add "duplicate detected" to the messages of 'the loan' ;
    reject 'the loan' ;
```

In the following example, if the where clause is true for two borrowers and unknown for two borrowers, then the rule engine considers that the rule condition is false. The `else` action is taken:

```
If there are more than 3 borrowers in the past borrowers of 'the loan' where
the name of each borrower is the name of 'the borrower',
then
    reject 'the loan';
    add "more than 3 similar borrowers found" to the messages of 'the loan' ;
else
    add "3 or fewer similar borrowers found " to the messages of 'the loan' ;
```

For more examples, see [Expressions that use collections](#).

Adding exception types

You can customize the list of exception types automatically handled by creating a BOM with the corresponding classes and by adding specific properties, `de.autoCatchExceptionInConditions` and `de.doNotAutoCatchExceptionInConditions`, to each class where appropriate. When automatic exception handling is enabled in Rule Designer, the behavior is to handle automatically exceptions that occur in the condition parts of rules for exceptions that belong to the following subclasses:

- `ArithmeticException`
- `NumberFormatException`
- `NullPointerException`
- `IndexOutOfBoundsException`

You can customize the default behavior to add more exception types to be automatically handled by setting a property named `de.autoCatchExceptionInConditions` to the corresponding class using the BOM editor:

```
package java.lang;
    public class ClassCastException
        extends java.lang.RuntimeException
            property "de.autoCatchExceptionInConditions" "true"
    {
        public ClassCastException();
        public ClassCastException(string arg);
    }
```

Excluding exception types

You can exclude exception types from automatic exception handling in one of the handled classes by overriding the definition of the class in the BOM with the property named `de.autoCatchExceptionInConditions`. For example, you can remove arithmetic exceptions from automatic exception handling by removing the `de.autoCatchExceptionInConditions` property from the `ArithmeticException` class:

```
public class ArithmeticException
    extends java.lang.RuntimeException
{
    public ArithmeticException();
    public ArithmeticException(string arg);
}
```

You can exclude a subclass of the four default classes from automatic exception handling by setting a property named `de.doNotAutoCatchExceptionInConditions` (in bold):

```
public class StringIndexOutOfBoundsException
    extends java.lang.IndexOutOfBoundsException
    property "de.doNotAutoCatchExceptionInConditions" "true"
{
    public StringIndexOutOfBoundsException();
    public StringIndexOutOfBoundsException(string arg);
    public StringIndexOutOfBoundsException(int arg);
}
```

Logging

Automatic exception handling uses the standard Java™ logger logging service, so that you can log messages based on message type and level and control how these messages are formatted and stored at runtime.

In Rule Designer, to capture activity traces that occur in automatic exception handling, you must first define a `logging.properties` file and make it available to the JVM that is used in the rule execution run or debug configuration. Then, in the Run Configurations window, under the Parameters & Arguments tab of your decision operation, enter one of the following VM arguments: -

`Djava.util.logging.config.file=file_path/logging.properties` or -

`Djava.util.logging.config.file="${workspace_loc}/your_project/logging.properties"`.

In the following example, the log level for the logger name `com.ibm.rules.engine.aeh` is set to FINE.

```
handlers = java.util.logging.ConsoleHandler
#####
# Handler specific properties.
# Describes specific configuration info for Handlers.
#####
# Limit the messages that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
com.ibm.rules.engine.aeh.level = FINE
```

To enable automatic exception handling logs in a server environment, define a `logging.properties` file and make it available to the JVM by configuring your application server, the log level for the Logger name `com.ibm.rules.engine.aeh` should be set to FINE.

For an introduction to the Java Logging API, see [Java Logging Overview](#). For more information about the `java.util.logging` Java logger, see [java.util.logging](#) in the Oracle documentation.

Parent topic: [Exception handling in rules](#)

Examples of automatic exception handling

When you enable automatic exception handling in conditions, the rule engine handles specific subclasses so that rule processing can continue after exceptions are thrown. You can understand the logic behind automatic exception handling by reviewing examples.

With automatic exception handling in conditions, unknown values for Boolean expressions are resolved by using the following three-valued logic that applies for all combinations of Boolean expressions that can result in an exception:

NOT(A)

A	¬A
F	T
U	U
T	F

AND(A,B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A,B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

Simple rule condition

In the following example, if the status of the borrower is unknown, no rule action is executed.

```
if
  the status of 'the borrower' starts with "duplicate"
then
  reject 'the loan' ;
  add "duplicate detected" to the messages of 'the loan' ;
else
  add "no duplicate found" to the messages of 'the loan' ;
```

Rule condition that uses the OR operator

In the following example, if the status of the borrower is unknown and the comment of the loan contains the word duplicate, then the loan is rejected and a duplicate detected message is sent.

```
if
  the status of 'the borrower' starts with "Duplicate" or the comment of 'the loan' contains "duplicate"
then
  reject 'the loan' ;
  add "duplicate detected" to the messages of 'the loan' ;
else
  add "no duplicate found" to the messages of 'the loan' ;
```

However, if the status of the borrower is unknown and the comment is unknown, then no rule action is taken. The loan is not rejected and the message no duplicate found is not added to the loan.

Rule condition that uses the AND operator

In the following example, if the status of the borrower is unknown and the comment in the loan contains the word duplicate, neither the then nor the else actions are taken. In other words, the loan is not rejected and the message no duplicate found is not added to the message of the loan.

```
if
  the status of 'the borrower' starts with "duplicate" and the comment of 'the loan' contains "duplicate"
then
  reject 'the loan' ;
  add "duplicate detected" to the messages of 'the loan' ;
else
  add "no duplicate found" to the messages of 'the loan' ;
```

For the loan to be rejected, both expressions must be true.

If there is no ... then ...

If a collection of 4 past borrowers contains 3 borrowers for whom the where clause is false and 1 borrower for whom it is unknown, then the rule action is taken.

```
If there is no borrower in the past borrowers of 'the loan' where the name of
this borrower is the name of 'the borrower',
then
add "no duplicate found" to the message of 'the loan'
```

If there is at least one ... then ...

If a collection of past borrowers contains 3 borrowers for whom the where clause is true in 1 case, false in 1 case, and unknown in 1 case, then the rule action is taken.

```
If there is at least one borrower in the past borrowers of 'the loan' where
the name of this borrower is the name of 'the borrower',
then
    reject 'the loan' ;
    add "duplicate detected" to the message of 'the loan'
```

Rule conditions that use the NOT operator

In the following example, if the status is unknown and the comment does not contain duplicate, the then action is taken and no duplicate found is added to the message of the loan.

```
if it is not true that
    (the status of 'the borrower' starts with "duplicate" and the comment of
'the loan' contains "duplicate")
then
    add "no duplicate found" to the messages of 'the loan' ;
else
    reject 'the loan' ;
    add "duplicate detected" to the messages of 'the loan' ;
```

If the status is unknown and the comment contains duplicate, no rule action is executed.

Decision Table

In the following example, the otherwise row to a condition column is selected when all the other conditions in the column are false. Alternatively, if one of the other conditions is unknown, then the otherwise row is not selected:

	Customer status contains...	duplicate detected		no duplicate detected		reject 'the loan'
1	Similar	duplicate detected	⊗	(nothing detected)		-
2	Duplicate	duplicate detected	⊗	(nothing detected)		-
3	Otherwise	⊗ (nothing detected)		no duplicate detected	⊗	-

Rule variable

If the rule variable 'statusIsNew' is unknown, the rule engine stops the evaluation of the expressions. The condition is considered unknown and no rule action is taken.

```
definitions
    set 'statusIsNew' to the status of 'the borrower' starts with "New";

if it is not true that 'statusIsNew' and the comment of 'the loan' contains
"similar"
then
    reject 'the loan' ;
    add "duplicate detected" to the messages of 'the loan' ;
else
    add "no duplicate found" to the messages of 'the loan' ;
```

Collection of objects

If the result of the test that the object must fulfill is unknown, then this object is not added to the collection

because only the objects tested true are added:

```
definitions
set 'duplicateBorrowers' to all borrowers where the status of each borrower
starts with "duplicate" ;
```

Expressions that use collections

In the following example, if the where clause is true for three borrowers and unknown for one borrower, then the rule engine considers that the condition is true because the borrower for whom the where clause is unknown is not counted. The then action is taken:

```
if
    there are less than 3 borrowers where the name of each borrower is the
name of 'the borrower',
then
    add "3 or less similar borrowers found" to the messages of 'the loan' ;
else
    reject 'the loan';
    add "more than 3 similar borrowers found" to the messages of 'the loan' ;
```

In the following example, if the where clause is true for one borrower and unknown for two borrowers, then the rule engine considers that the rule condition is false because the two borrowers for whom the where clause is unknown are not counted. The else action is taken:

```
if
    there are more than 3 borrowers where the name of each borrower is the
name of 'the borrower',
then
    reject 'the loan';
    add "3 or more similar borrowers found" to the messages of 'the loan' ;
else
    add "less than 3 similar borrowers found" to the messages of 'the loan' ;
```

Expressions that uses contain

The following example expression uses contain, which is an alternative way to test a collection. If the collection ('duplicateBorrowers') contains the object 'the borrower', then the condition is true.

```
definitions
    set 'duplicateBorrowers' to all borrowers where the customer status of
each borrower starts with "duplicate" ;
if
    'duplicateBorrowers' contain 'the borrower'
then
    add "duplicate detected" to the messages of 'the loan' ;
    reject 'the loan' ;
```

Rule Variables

All rule variables should be resolved different from unknown to allow the evaluation of the condition part. If the rule variable statusIsDuplicate is unknown, the rule engine stops the evaluation of the expressions. The condition is considered unknown and no rule action is taken, even if the comment of 'the loan' contains "similar":

```
set 'statusIsDuplicate' to the customer status of 'the borrower' starts with
"Duplicate";

if
    'statusIsDuplicate' or the comment of 'the loan' contains "similar"
then
    reject 'the loan' ;
    add "duplicate detected" to the messages of 'the loan' ;
else
    add "no duplicate found" to the messages of 'the loan' ;
```

Parent topic: [Exception handling in rules](#)

Setting a custom exception handler

You can set an exception handler to customize exception handling in rule conditions and actions. When you implement the CustomExceptionHandler API, exceptions can either be ignored or rethrown.

About this task

The default engine behavior is to stop when an exception is thrown. When you implement the CustomExceptionHandler API, you can choose to enforce the default engine behavior or to ignore the exception so that the engine can continue to process rules.

To activate the [CustomExceptionHandler](#), you must implement the interface in the XOM and then specify the implementation class name in Rule Designer.

Procedure

1. Implement the CustomExceptionHandler interface in the XOM. In the following example, the custom exception handler rethrows exceptions for NumberFormatException:

```
package com.acme;
import ...

public class MyCustomExceptionHandler implements CustomExceptionHandler {

    static Logger logger =
    Logger.getLogger(MyCustomExceptionHandler.class.getName());

    /**
     * The custom exception handler must contain a default constructor.
     */
    public MyCustomExceptionHandler() {
    }

    @Override
    public void handleConditionException(Exception e) throws Exception {
        if (e instanceof NumberFormatException) {
            // ignore number format exception
        } else {
            // rethrow the exception, it will stop the engine execution
            throw e;
        }
    }

    @Override
    public void handleActionException(Exception e, RuleInstance
ruleInstance) throws Exception {
        if (e instanceof NumberFormatException) {
            // ignore number format exception and log
            if (logger.isLoggable(Level.INFO)) {
                logger.log(Level.INFO,
                    "Exception while executing action of rule " +
ruleInstance.getRuleName(), e);
            }
        } else {
            // rethrow the exception, it will stop the engine execution
            throw e;
        }
    }
}
```

2. Open Rule Designer.
3. In the Rule Engine panel of the rule project properties window, enter the custom exception handler name. For example, enter com.acme.MyCustomExceptionHandler in the Custom exception handler field.
4. Click **OK**.

Results

The custom exception handler is created when the engine starts, and removed when the engine is reset.

Parent topic: [Exception handling in rules](#)

BAL building blocks

The **in** and **from** building blocks help you optimize performance.

The rule engine runs on Java EE and integrates Enterprise JavaBeans components. You can optimize performance by using BAL building blocks, and execution modes appropriately.

Rules can also be internally rewritten by the engine to enable automatic optimization of the rules. By using the **from** and **in** building blocks, you reduce the number of objects on which pattern matching is applied, and hence the number of evaluations carried out. The rule engine automatically rewrites rules to use these building blocks whenever possible. For details see [Using objects that are not in memory in a condition](#).

Parent topic: [Optimizing execution](#)

Optimizing the decision engine

You can optimize the performance of the decision engine by adjusting your rule conditions, and by selecting the most efficient execution mode for your rules.

Using the same definition for several decision engines

To reduce memory consumption, you can create several engines for concurrent use from the same engine definition.

Adjusting conditions

Most of the execution time is taken up by the pattern-matching process. Consequently, the most efficient way of improving performance in rulesets consists in modifying the condition part of rules.

Improving the performance of the RetePlus execution mode

You can improve performance by avoiding certain operations in the RetePlus execution mode.

Improving the performance of the sequential execution mode

To use the sequential algorithm, make sure that your rules are homogeneous and that the rules are not chained. In the sequential algorithm, you can define the number of rules to execute per tuple and the order of execution, by using control properties.

Improving the performance of the Fastpath execution mode

You improve the performance of the Fastpath execution mode by modifying the condition part of rules.

Parent topic: [Optimizing execution](#)

Related concepts:

[Executing rulesets in the decision engine](#)

Using the same definition for several decision engines

To reduce memory consumption, you can create several engines for concurrent use from the same engine definition.

About this task

Creating several engines from one engine definition, instead of loading an engine definition to create each engine, reduces memory consumption. It does not degrade performance.

Rule Execution Server directly supports concurrent executions.

If you do not want to use Rule Execution Server, use an engine, as an [Engine](#) instance, in only one thread.

Do not use the same objects in different engines in different threads.

Procedure

To create several engines from the same definition, use the following code:

```
EngineDefinition definition = ...  
Engine engine1 = definition.createEngine();  
Engine engine2 = definition.createEngine();
```

Parent topic: [Optimizing the decision engine](#)

Adjusting conditions

Most of the execution time is taken up by the pattern-matching process. Consequently, the most efficient way of improving performance in rulesets consists in modifying the condition part of rules.

Ordering tests

The order in which tests are carried out in rule conditions affects the performance of the pattern-matching process. To get the best performance for a specific condition, place the most discriminant tests at the start. This position accelerates the selection of objects in the discrimination tree for tests that involve constants. Also, when you place the most discriminant tests at the start, you know sooner when an object does not satisfy the tests between conditions in the joins.

Sharing conditions

If the first N conditions of two rules are identical, these rules share the same portion of the network. As a consequence, searches for objects that satisfy these first N conditions are done only once for all the rules that share these conditions. It is therefore a good idea to modify the order of rule conditions so that they share the network as much as possible.

Note:

You cannot share conditions in the sequential execution mode.

Ordering conditions in rules

Place the most selective conditions, or groups of conditions, at the beginning of tests. In this way, the objects that prevent a rule from being executed are removed sooner in the process. Tokens spend less time in the RetePlus network, joins require less memory, and fewer tests are necessary.

Example of the effect of condition order on RetePlus performance

Here is a rule with three object classes.

```
rule filter {
  when {
    A (a1==3; ?x:a2);
    B (b1==2; ?y:b2; b3==?x);
    C (c1==?y);
  }
  then {
    System.out.println("filter");
  }
}
```

Let's work on this example.

1. Imagine different contents for the working memory:
 - There are five objects of class A and their a2 attribute is 10.
 - There is an object of class B whose b2 attribute is 7 and b3 attribute is 10.
 - There is an object of class B whose b2 attribute is 4 and b3 attribute is 10.
 - There are 28 objects of class B. Their b3 attribute is 10 and their b2 attribute is neither 4 nor 7. In other words, there are thirty class B objects in all.
 - There is one object of class C and its c1 attribute is 4.
2. If you were to draw the corresponding RetePlus network, the contents of the first join node would be quite different.
3. It now contains 155 pairs of objects, which can be described like this:
 - Five pairs are formed by the objects of class A and the object of class B whose b2 is 7 and b3 is 10.
 - 150 pairs (that is, $(5 * (28 + 1 + 1))$) are formed by the objects of class A and the objects of class B whose b3 attribute is 10.
4. If you now add an object of class C whose c1 attribute is 7, the work of the second join node consists of doing 155 tests, between the B object of each of the 155 pairs of the first join and the C object that you

have just added, to verify that attributes b2 and c1 have the same value.

5. Now, suppose you create a filter2 rule by changing the order of the conditions of the filter rule so that it is shown as follows:

```
rule filter2 {
  when {
    C (?y:c1);
    B (b1==2; b2==?y; ?x:b3);
    A (a1==3; a2==?x);
  }
  then {
    System.out.println("filter2");
  }
}
```

For the same initial working memory as before, the first join node, corresponding to the first two conditions, now contains 30 (instead of 155) pairs, formed by the class C object and the 30 class B objects whose b2 attribute is 4 and b3 attribute is 10.

6. If you now add an object of the C class whose c1 attribute is 7, the work of the first join node consists of doing 30 tests between the newly added object and the 30 class B objects in memory. This causes us to add a new pair, formed by the C object that you have just added and the B object whose b2 attribute is 7.
7. After all this is done, the second join node is activated. It does 30 tests, between the B object of each pair from the previous join and the A object in memory, to verify that b3 and a2 have the same value.

These two rules, which produce the same result, have quite different computing costs.

- The filter rule uses 155 pairs and a triple to represent the initial state, whereas the filter2 rule uses only 30 pairs and a triple.
- Adding a C object costs 155 tests and the construction of a triple in the filter rule, whereas the same adding costs only 60 tests, a triple, and a pair in the filter2 rule.

Execution order of conditions and definitions

If a condition depends on a variable, the condition executes as soon as the variable exists and it is the turn of the condition to be executed. In the case where several variables are created in the definition part of the rule, it means that a condition might execute before all the variables are created.

In the following example, the condition the age of 'the winning customer' is at least 18 depends on the variable the winning customer and does not depend on the email. Therefore, this first condition executes as soon as the winning customer exists and before the variable the email is created.

```
definitions
  set 'the winning customer' to a Customer ;
  set 'the email' to the email of 'the winning the customer' ;
if
  the age of 'the winning customer' is at least 18
  and 'the email' is defined
then
  print "Send coupon to customer via email.";
```

Parent topic: [Optimizing the decision engine](#)

Improving the performance of the RetePlus execution mode

You can improve performance by avoiding certain operations in the RetePlus execution mode.

RetePlus algorithm

RetePlus is a rule execution mode based on the Rete algorithm. It relies on a working memory and agenda, and supports negative patterns, object notification, and logical objects.

Optimizing the object model

You can decrease the cost of pattern matching by changing the representation of the object model.

Improving the performance of equality or comparison evaluation

Hashing expressions can improve performance at equality-test stage.

Parent topic: [Optimizing the decision engine](#)

RetePlus algorithm

RetePlus is a rule execution mode based on the Rete algorithm. It relies on a working memory and agenda, and supports negative patterns, object notification, and logical objects.

RetePlus is the Decision Server extension based on the Rete algorithm. In RetePlus mode, rule execution uses an environment based on a working memory and agenda.

- [Working memory and the agenda](#)
- [Negative patterns](#)
- [Object notifications](#)
- [RetePlus network operation](#)

Working memory and the agenda

Under the RetePlus execution mode, the rule engine functions with [Working memory](#) and an [Agenda](#), adding [Rule instances](#) for execution when conditions are met.

Working memory

Under the RetePlus execution mode, a rule engine in Decision Server is paired with a *working memory*. The working memory contains all the objects contained in the associated Engine object. You can modify the working memory by adding a statement in the *action part of a rule* or by using the Application Programming Interface (API). Thus, the rule engine is aware of the objects that are in the working memory and any objects that are linked to them. The rule engine can use only objects that are accessible from the working memory.

Agenda

The *agenda* is where Decision Server stores the rules whose patterns are all matched. Any rule that enters the agenda is said to be instantiated, it becomes a *rule instance*, see [Rule instances](#).

The agenda stores rule instances that are eligible to be executed. If the agenda is empty, execution has no effect. Rule instances placed in the agenda are said to be *eligible*. Often, in the agenda, several rules are eligible. Consequently, the rule engine has to have some way of deciding which particular rule in the agenda should be executed.

In the agenda, rule instances are ordered according to three criteria that determine which rule should be executed first. Additional execution control is offered with the implementation of more complex features, see [Object notifications](#).

Refraction

A rule instance that has been executed cannot be reinserted into the agenda if no new fact has occurred, that is, if none of the objects matched by the rule are modified, or if no new object is matched by the rule.

The refraction principle enforces that only one rule instance is created for a given tuple. Since the data unit for an engine cycle is the working memory, the engine can easily record all the possible tuples and check that there is only one rule instance for each tuple.

For example consider the following ruleset:

```
class Person {
  Person(int age, boolean sick);
  boolean sick;
  int age;
}

rule incrementAge {
when {
  p: Person(!sick; age < 50);
} then {
  p.age += 1;
  update p;
}

rule cure {
when {
  p: Person(sick);
} then {
  p.sick = false;
```

```
    update p;
}
```

If the object `Person(18, true)` is inserted in the working memory, the refraction principle produces the following execution trace:

1. cure
2. incrementAge

You can add the repeatable property in the `incrementAge` rule:

```
rule incrementAge {
    property com.ibm.rules.engine.repeatable=true;
when {
    p: Person(!sick; age < 50);
} then {
    p.age += 1;
}
```

If the object `Person(18, true)` is inserted in the working memory, the repeatable property produces the following execution trace:

1. cure
2. incrementAge
3. incrementAge
4. incrementAge
5. ...
6. incrementAge

Until the condition `age < 50` is false.

Repeatable rules

To break the refraction principle, you can use the Boolean rule property

`com.ibm.rules.engine.repeatable`. In the decision engine, you must use this property to mark the sets of rules that you want to be repeatable.

- When a rule is not repeatable, the refraction principle prevents a rule instance from being posted again to the agenda.
- When a rule is repeatable, it can be inserted again as a rule instance in the agenda if a modification occurs against its condition objects.

Recency

If two rule instances have the same priority, the rule that matches the most recent object (the most recently inserted, modified, or retracted object) is executed first.

Priority and recency are used to resolve conflicts when several rule instances are candidates for execution at the same time. If, after using all specified conflict resolution methods, several rule instances remain candidates, then the engine uses internal metarules; this assures that the same sequence of rule executing is always followed, given the same conditions.

Example of rule executing order

The following technical rule example shows you the order in which rules are executed:

```
rule init {
    when {
        angel();
    }
    then {
        insert clown();
        insert dascyllus();
    }
};
rule last {
    priority = -100;
    when {
        dascyllus();
    }
    then {
        System.out.println("last");
    }
};
```

```

rule first {
    priority = 100;
    when {
        angel();
        clown();
        dascyllus();
    }
    then {
        System.out.println("first");
    }
};
rule second {
    priority = 0;
    when {
        angel();
        dascyllus();
        clown();
    }
    then {
        System.out.println("second");
    }
};
rule third {
    priority = 0;
    when {
        angel();
        clown();
        dascyllus();
    }
    then {
        System.out.println("third");
    }
};
};

```

Suppose that the object `angel` has been inserted from the application using the API, and therefore has been inserted into the working memory, and that we execute all the rules that can be executed.

Here is the order in which the rules are executed:

1. `init` rule: Because the object `angel` was inserted from the application, the `init` rule is the only rule that can be executed. It inserts the objects `clown` and then the object `dascyllus` into the working memory. The most recent object is therefore `dascyllus`.
2. `first` rule: This rule is executed before the second rule. The most recent object matched by these two rules is the same. However, the first object has a higher priority than the second, which determines their execution order.
3. `second` rule: This rule is executed before the third rule. The second and the third rules have the same priority, but the object `dascyllus` is more recent than the object `clown`.
4. `third` rule: This rule is executed before the last rule. Although the object `dascyllus` is more recent than `clown`, the third rule has a higher priority than the last rule.
5. `last` rule.

Rule instances

A *rule instance* is added to the agenda when objects in the working memory satisfy the condition part of that rule.

Let us consider the following technical rule:

```

rule rule1 {
    when {
        Fish(color == green; type == shark);
        Fish(type == trigger);
    }
    then {...}
};

```

The following rule instances will be put into the agenda:

Rule	Rule	Rule	Rule
rule1(A,C)	rule1(A,D)	rule1(B,C)	rule1(B,D)

In this example, you can see that a rule instance is a dynamic concept: the rule instance is produced by any combination of objects in the working memory that match the patterns specified in the rule. In contrast, a rule is a static concept.

Negative patterns

Standard rule conditions might be described as existential, in the sense that they could be matched by one or several objects that actually happen to exist. In the RetePlus execution mode there is a complementary concept known as *negative patterns*. To express the nonexistence of a particular object in the working memory, we place the not keyword before the condition of the object, as shown here:

```
not Fish(type==eel);
```

This negative pattern gives rise to a successful match because there is no object that matches the corresponding positive pattern:

```
Fish(type==eel);
```

Conversely, this negative pattern:

```
not Fish(type==angel);
```

does *not* give rise to a successful match for the simple reason that there is an object that matches the corresponding positive pattern:

```
Fish(type==angel);
```

Object notifications

With the RetePlus execution mode, you have statements to control individual operations:

- [Object insertion](#)
- [Object removal](#)
- [Object update](#)
- [Attribute modification](#)

Object insertion

The insert statement lets you create a new object and insert it into the working memory. This keyword is followed by the name of the class and the values of the attributes of the object to be created.

Here is an example that involves the creation of a new Course object whose department and number attributes take the values History and 324, respectively:

```
insert Course(History,324);
```

There are two possibilities for the way that an insert operation is processed:

- The object is already in the working memory. In this case, the insert operation is equivalent to an update.
- The object is not in the working memory. In this case, the object is inserted into the working memory and the insert daemon of the corresponding class and its superclasses, if any, is called, and the agenda is updated.

Object removal

The retract statement lets you remove from the working memory an object that is bound to a rule variable.

In the following example, the retract primitive removes the object bound to the ?c variable:

```
rule RemoveCourse {
  when {
    ?c: Course(department == History; number == 254);
  }
  then {
```

```
    retract ?c;
  }
};
```

The retract daemon of the corresponding class, if any, is called, and the agenda is updated accordingly. Retracting an object from the working memory might cause a logical object to lose one or several of its justifications.

Object update

When an object is modified by Java™, the modifications are not seen by Decision Server. This means that data maintained by Decision Server will not be consistent with the new contents of the modified object.

To avoid inconsistencies, the update statement should be called for such an object. This statement allows Decision Server to update its internal structures according to the new contents of the modified object.

In the following rule, the first action modifies the number attribute. The update primitive is then called to inform Decision Server that the object has been modified.

```
rule NumberingUpdate {
  when {
    ?c: Course(department equals "history"; number > 300);
  }
  then {
    ?c.updateNumbering(); // This call modifies ?c
    update ?c;
  }
};
```

Important:

If you do not use modify, you must use the update primitive to inform Decision Server that objects have been modified. Forgetting to notify Decision Server of modifications leads to inconsistent states.

Attribute modification

The modify statement lets you modify the values of the attributes of objects in the working memory.

Here is an example:

```
rule ModifyLecturer {
  when {
    ?c: Course(department equals "History"; lecturer equals "Tanner");
  }
  then {
    modify ?c {lecturer = "Chen"};
  }
};
```

RetePlus network operation

The RetePlus network indexes rules so as to minimize the number of rules and conditions that need to be evaluated whenever the working memory is changed. The network minimizes the number of evaluations by sharing tests between rules and propagating changes incrementally. When all the tests have been completed, the network designates a rule.

In most cases, executing a rule modifies the working memory. Objects in the working memory referenced by these rules can be inserted, removed, and modified. The network processes the changes inferred by the execution of the rule and produces a new set of rules to be executed.

When a change occurs in the working memory, two things can happen, depending on the nature of the change:

- If there is an insertion of an object into the working memory (positive), the rule can be executed.
- If there is a removal of an object from the working memory (negative), the rule must be removed from the agenda. The agenda is the place where the current set of rules to be executed is maintained.

This process continues cyclically until there are no further rule instances left in the agenda.

Note:

A not condition returns true for a simple condition if the working memory does not contain any object that can match the condition.

Example of a RetePlus network

To illustrate our description of a RetePlus network, we shall make use of a trivial filter rule, written in the rule language that refers to classes named A, B, and C:

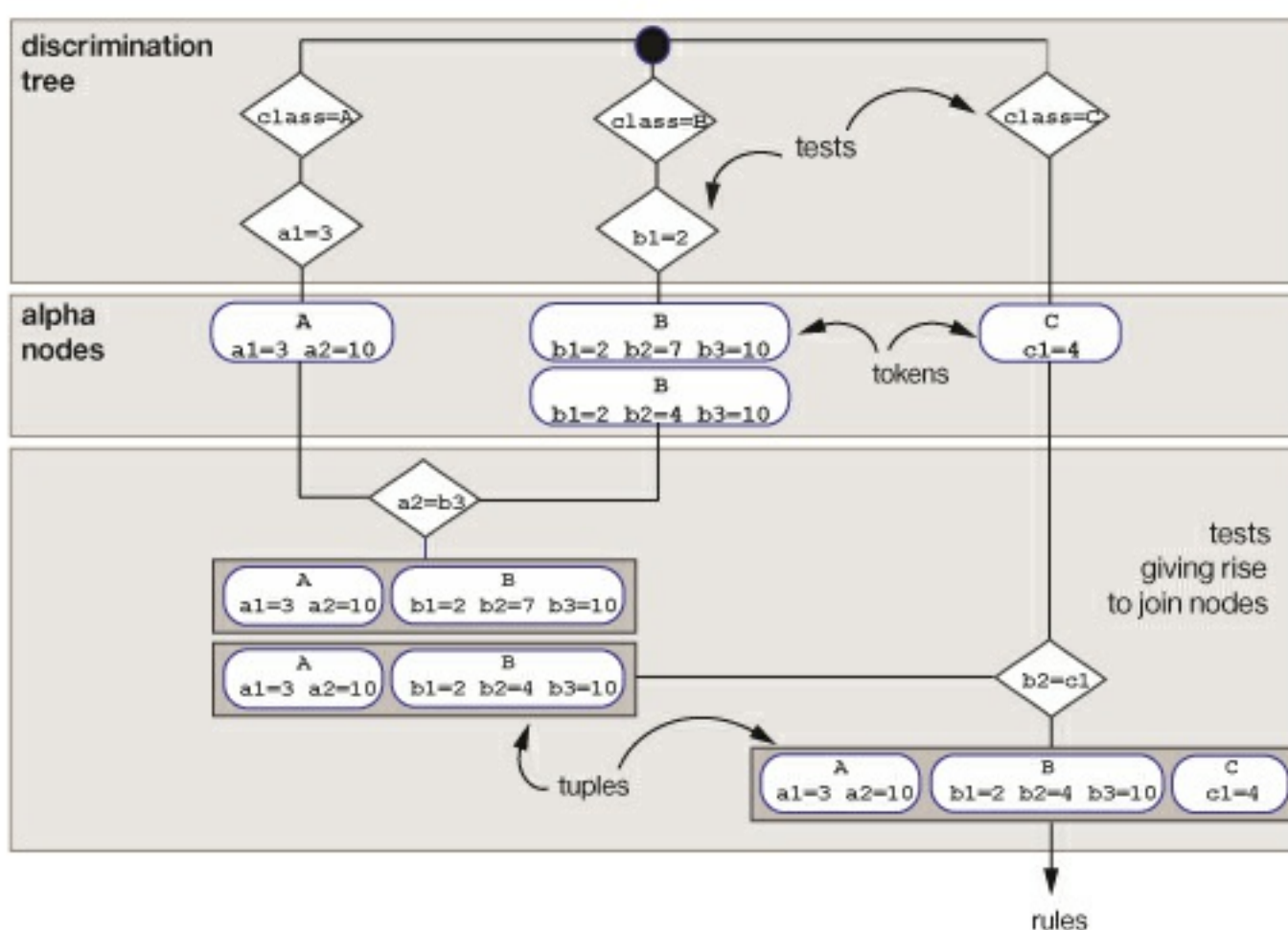
```
rule filter {
  when {
    A (a1==3; ?x:a2);
    B (b1==2; ?y:b2; b3==?x);
    C (c1==?y);
  }
  then {
    System.out.println("filter");
  }
}
```

The class attributes are the following:

- The A class has two attributes, named a1 and a2.
- The B class has three attributes, named b1, b2, and b3.
- The C class has a single attribute, named c1.

Assume that the working memory contains the following objects:

```
A( a1=3 a2=10 )
B( b1=2 b2=4 b3=10 )
B( b1=2 b2=7 b3=10 )
C( c1=4 )
```



RetePlus network structure

A RetePlus network can be represented as a graph composed of three zones:

Discrimination tree

A *discrimination tree* is a pattern-matching process that performs tests. These tests are represented by diamond shapes at the top of the network. The tests concern the classes of objects and the values of their attributes. Input to this tree consists of *tokens* representing each of the current objects in the working memory.

When the pattern deals with only one object and its attributes, it is said to be a discrimination test. When it is

a combination, it is called a join; these appear in the lower part of the graph.

Alpha nodes

An *alpha node* is formed at the next level of the network, for each token that passes the tests of the discrimination tree. Each node is composed of one or several tokens, represented by round-cornered rectangles (there are three alpha nodes in the figure). One alpha node contains two B-class tokens. The other two nodes contain only one class token each—A-class and C-class tokens, respectively.

Tests and tuples

The third zone of the network matches tokens of several classes of objects. The resulting nodes are known as *tuples*, which will be made up of several tokens. The equality test between attributes a2 and b3 gives rise to a node composed of two pairs of tokens, and the test between attributes b2 and c1 then filters out a triplet of tokens. In a RetePlus network, we often refer to tuples of this kind as *join nodes*.

Object addition

To demonstrate further this idea of the RetePlus network, suppose an extra object is inserted into the working memory. The added object is of the C class, and the value of its c1 attribute is 7.

```
C( c1=7 )
```

At this stage, when all the tests of the nodes are carried out, only the third test C (c1==?y); (class of object = C) is satisfied. The tests of the child nodes continue as the token descends through the network.

Here, there is no child node in the discrimination tree. For that reason, the object is stored directly in the alpha node.

When the token reaches the second join node B (b1==2; ?y:b2; b3==?x), the test is carried out between each B object of the pairs stored next to the memory of the join node and the C object contained in the token that just arrived at the join node. The test is satisfied for the B object of the second pair. We can therefore proceed and create a triplet formed by the second pair and the object in the token.

By forming this triplet we have, in fact, descended all the way through the network, and a new instance of the filter rule can be executed.

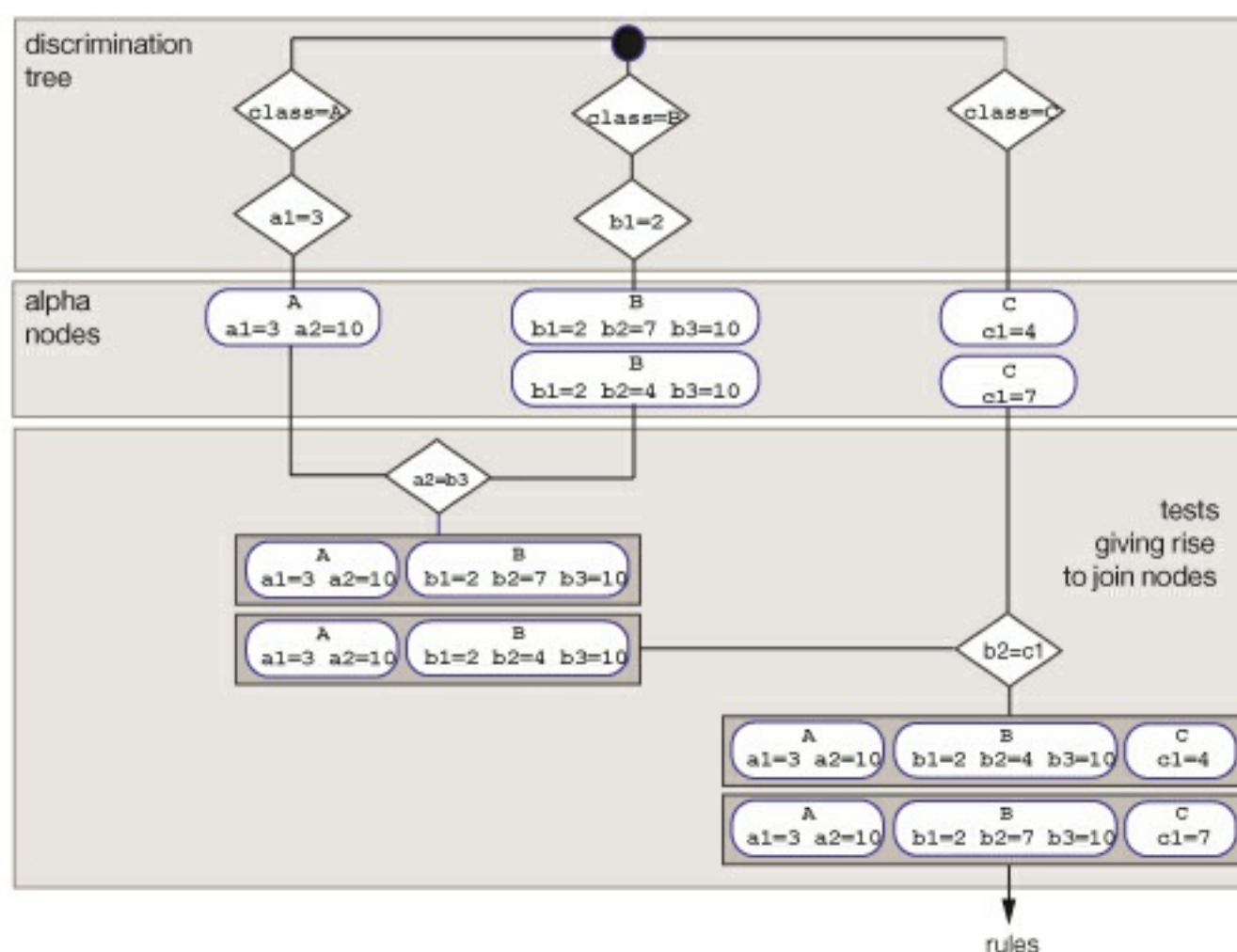
Object removal

Suppose that we now remove the object that we just added to the working memory.

The progress of the token through the discrimination tree is the same as in the case of object addition, but the arrival of the negative token in the alpha node causes the removal of the object contained in the token.

When the token arrives at the second join node, all the triplets that contain the object in the token are removed and the token continues its progression through the network.

As in the object addition example, the rule node is reached. The instance of the filter rule whose objects satisfy the conditions corresponding to the objects in the token are removed from the agenda.



Optimizing the object model

You can decrease the cost of pattern matching by changing the representation of the object model.

Suppose that you are filtering a set of ordered objects in which each object is identified by a rank. If you want to retrieve an object next to another just by the use of the ranks, you can write a complex rule such as the following:

Low performing rule

The next rule expresses the following behavior: If you look for the object of rank ?n and if you find an object whose rank is greater than ?n and there is no object whose rank is between their ranks, then the next object has successfully been found.

```
rule next {
  when {
    element( ?n:rank );
    ?next : element( ?n2:rank & > ?n );
    not element( rank > ?n & < ?n2);
  }
  then {
    do something with ?next
  }
};
```

In such a rule, the object model and the reasoning process are not efficient. Performance is low.

More efficient rule

To improve performance, each object keeps a pointer to the object next to it. You can easily use this representation for the pattern matching process, as in the following example.

```
rule next {
  when {
    element( ?n:rank; ?next: next );
  }
  then {
    do something with ?next
  }
};
```

Parent topic: [Improving the performance of the RetePlus execution mode](#)

Improving the performance of equality or comparison evaluation

Hashing expressions can improve performance at equality-test stage.

Test evaluations that involve variables from two different rule conditions can be time consuming. In the following example, the condition on Account references the variables ?p and ?b bound in the previous conditions.

Example: Bank accounts

```
when {  
  ?p:Person(age > 20);  
  ?b:Bank();  
  ?a:Account(name equals ?p.name; bank == ?b;  
             balance < ?p.minBalance);  
  ...  
}
```

Without optimization, if there are 1,000 instances of Person with age > 20 and 5 instances of Bank, the three tests in the Account condition are executed 5,000 times every time a new instance of Account is added. This is because this new instance has to be tested against all the possible combinations of the ?p and ?b variables.

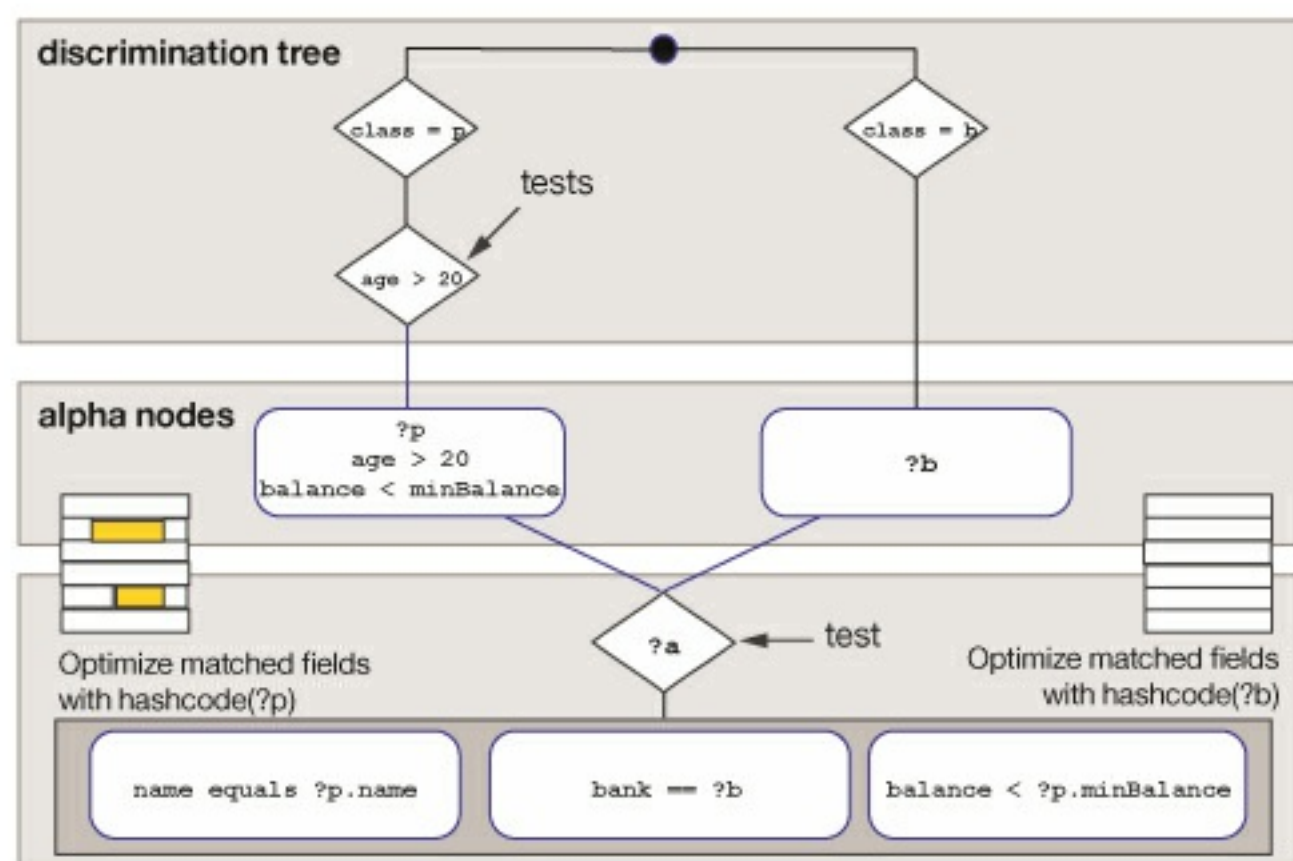
Autohashing optimization minimizes the execution of the tests based on the equality operators by relying on a hash table. A hash table is conceptually a contiguous section of memory with a number of addressable elements in which data can be quickly added, deleted, and found. Hash tables increase memory consumption for the purpose of gaining speed. They are certainly not the most memory-efficient means of storing data, but they provide very fast look-up times.

To successfully store and retrieve data from a hash table, a link is generated. That link can be mapped to one of the addressable elements of the hash table. Java™ classes implement the hashCode method to generate those links. Test execution can be reduced by a factor of 20, which is the dimension of the table that holds the links. So at best, there are 20 different entries for a value. In the Bank accounts example above, this saving would relate to 250 tests instead of 5,000 every time a new instance of Account is added.

Note:

Hashers always return a superset of the matching objects and the equality conditions still require a verification of the objects.

The following figure illustrates autohashing in the RetePlus network. It shows how the Bank accounts example would be handled in Decision Server.



The RetePlus network uses hashing to do operations on object equality tests such that a link is generated from that data. The link identifies which memory element of the hash table contains the satisfied equality test. In other words, the hash code is the link to the object matching the side of the test. The test is replaced by a hash code value and the corresponding object is stored in the hash table with that value. The hashing mechanism then uses the equality operator (==) to test whether two objects are equal according to the equals(Object) method. When the hashCode method is called on each of the two objects, it must produce the same integer result.

Join nodes are generated from these links inside the RetePlus network. A join node uses a single hash table. A hash table can process multiple equality tests.

To use autohashing, you need to create a configuration resource file so that the autohash mode is set to `true`. By default the autohash mode is turned off.

Using hashing expressions

Within a ruleset header, hashing expressions can be specified on a class. Hashing expressions and autohashers are not mutually exclusive, if both are defined a class hasher always takes precedence over an autohasher.

The following ruleset header provides an example of when to use a class hasher. The class `Customer` is assigned a category, and the possible categories are defined as `int` values.

Note:

Only methods that return an `int` can be used as a hasher.

```
ruleset CustomerRuleset
{
    hasher(Customer c) = c.getCategory();
    ...
};
```

The keyword `hasher` introduces a hasher definition. In this example, a hasher for the class `Customer` is defined. The definition says that for an object of the class `Customer` named `c`, a hasher is the `Customer`'s `Category`.

A class might have several hashers. For example, you could add this hasher to the previous header:

```
hasher(Customer c) = c.getName().length();
```

This `Customer` hasher is defined as the `length` of its `Customer.Name`.

The rule engine exploits the hashers to optimize its internal matching algorithm. Whenever the expression provided by the hasher is used in a `==` (equality operator) test, the hasher can be applied. Here are some rule conditions on which a hasher can be chosen and applied:

```
// Query customers by category
CustomerQuery(?c: getCategory());
?cus: Customer(?cus.getCategory() == c);
```

or:

```
// Query customers by the length of their names
CustomerQuery(?n: getName());
?cus: Customer(?n.length() == ?cus.getName().length());
```

For these two condition groups, the rule engine recognizes that one of the expressions on either side of the equals `==` sign matches one of the provided hashers, and it applies that hasher for optimization.

Internally, the engine classifies the instances of the class using the `int` value computed by the hashing expression.

Note:

- There could be many tests in the condition part, and a hasher is applied when a side of a condition equals `==` test matches the hashing expression.
- Hashers cannot be used with an `equals` method.

Class hashers are more flexible than the autohashing mechanism provided through the configuration file, because you can choose the most appropriate hashing expressions with the ILOG® Rule Language (IRL).

Using hasher properties

The ruleset hasher definition is extended with hasher properties, shown in the following property definition block:

```
hasher ( Customer c ) = c.age
{
    property accurate = true;
    property ordered = true;
    property final = true;
```

```
}
```

The accurate property

The accurate property activates the internal use of more efficient hashing tables. However, this might induce greater memory consumption. Use the accurate property when the domain of the hashing expression is not too large. For example, you can expect that a `c.age` attribute has a small domain included. At the opposite end, it is likely that a population attribute of a `City` class has a large domain and should not be related to accurate tables.

The ordered property

The ordered property indicates that the hasher is compatible with inequality test expressions. It requires that a domain be defined on the hashing expression in the XOM. The hash processing is then done on either equality or comparison expressions. Using a hashing comparison expression, you can expect to speed a test evaluation up to twice as much.

As an example, consider the activation of an ordered hasher on the `age > a.value` test in the rule `filterCustomer`:

```
rule filterCustomer
{
  when {
    a: Account();
    c: Customer ( age > a.value );
  }
  then {
    ...
  }
}
```

Note that the `Customer.age` attribute must have been related to a domain in the XOM, or else the ordered hasher would not be activated on the comparison test.

The final property

Declaring a hasher as `final` can speed up hash processing when the object set is constant or almost constant during ruleset execution. The Decision Server rule engine takes advantage of this constancy to further improve performance.

Declaring a changing hashing set as `final` does not cause execution errors or alter the behavior, but it might slow down the execution.

As an example, consider the following declaration:

```
hasher ( Town t ) = c.population
{
  property final = true;
}
```

The set of every town in the working memory is then supposed to be nearly constant. While any changes applied on the population of a town or insertions of new towns are possible, these operations could be more costly in execution time than if the `final` property had not been applied.

Parent topic: [Improving the performance of the RetePlus execution mode](#)

Improving the performance of the sequential execution mode

To use the sequential algorithm, make sure that your rules are homogeneous and that the rules are not chained. In the sequential algorithm, you can define the number of rules to execute per tuple and the order of execution, by using control properties.

Sequential algorithm

To guarantee the same execution with either algorithm, both the following conditions are required:

- The rules must be homogeneous. Homogeneous rules mean that the conditions of these rules must be set on the same kind and number of objects. If you try to apply the sequential algorithm to heterogeneous rules, it typically creates more tuple objects, which can result in more rules being executed than in the RetePlus mode. The common tuple signature can also cause heterogeneous rules not to fire: When there are no corresponding instances in working memory, no tuples can be created.
- The rules must not be chained, that is, the modification done in the action part of a rule must not lead to execution of another rule.

If you specify the sequential execution mode for rules that are not compliant with it, the engine raises an error.

The use of sequential execution mode is described further in the sections:

- [Tuples](#)
- [Control properties in sequential mode](#)
- [Known limitations of the sequential algorithm](#)

Tuples

A tuple is a list of objects that complies with a particular structure. A *tuple structure* is simply a sequence of class descriptors. Each location in a tuple structure is called a *slot*. The same class descriptor can appear more than once at different slots.

Take a tuple structure:

```
(Customer,Product)
```

Here is a tuple that complies with this structure:

```
(new Customer("Henry"),new DVD("Mickey"))
```

Note that the subclassing is taken into account, since a DVD is a Product. However, some tuples that do not comply with the structure:

For example, the following tuple is too large:

```
(new Customer("Henry"),new CD("Madona"),new DVD("Mickey"))
```

This tuple is not correctly ordered:

```
(new DVD("Lord of the rings"),new Customer("Henry"))
```

And this tuple is too small:

```
(new Customer("Henry"))
```

In Java™, a tuple is easily implemented as an array of objects:

```
Java Tuple = Object[]
```

The Java code to create a tuple and its objects is:

```
new Object[] {new Customer("Henry"),new DVD("Mickey")}
```

At run time, the tuples are built automatically from the content of the working memory and then passed to the rule engine. The tuples are passed one after the other to the tuple-matching method.

Control properties in sequential mode

Control properties affect the sequential mode execution and the tuple structure. See [Control properties for rule tasks in execution modes](#) for more general information.

The number of rules to be executed per tuple and the order in which they execute can be specified with control properties: **ordering**, **firing**, and **firinglimit**.

The ordering property

The ordering property is mandatory for sequential processing. It describes the order by which the rules of the rule task will be executed.

There are two values available:

- `literal`: The rules are kept in the order in which they are set into the task body (using up/down arrows).
- `sorted`: The rules are sorted according to their static priorities.

Note that another value, `dynamic`, is available only for rule tasks using the RetePlus execution mode.

The firing property

The firing property specifies whether, for each tuple, all the rules or only the first rule that applies should be executed. This property is optional.

It has two possible values:

- `allrules`
This value means that all the rules that apply should be executed before skipping to the next tuple. This is the default value.
- `rule`
This value means that only the first rule that applies should be executed on the first tuple.

The firinglimit property

The `firinglimit` property gives another level of control when the `firing` property is set to `allrules`. It is used to specify a number that represents the maximum number of rules to be executed before skipping to the next tuple. This number should be greater than zero.

Known limitations of the sequential algorithm

The following table outlines the limitations of sequential processing, in particular its use combined with certain ILOG® Rule Language (IRL) constructs.

Li mi ta ti on	Description
IL O G Rul e La ng ua ge (IR L) li mi tat io ns	<p>Not all the rule patterns that have been designed for the stateful Rete matching are available. Compile time errors occur when a rule is not compliant with the current tuple-matching implementation.</p> <p>Therefore, when sequential processing is used, the IRL does not support the following features:</p> <ul style="list-style-type: none">• <code>dynamic</code> priorities <p>Because there is no agenda in sequential processing, only the rules with static priorities are allowed.</p> <ul style="list-style-type: none">• <code>not</code>, <code>exists</code>, <code>collect</code> conditions without an enumerator <p>The <code>not</code>, <code>exists</code>, and <code>collect</code> conditions with an enumerator (<code>from</code> or <code>in</code>) are translated to Java bytecode. When an enumerator is specified, the set of objects is available as a value tied to the condition. (Note that this is not a limitation for RetePlus or Fastpath.)</p>
Co nd iti on co ns tru ct	<p>The following condition constructs are available in sequential processing:</p> <ul style="list-style-type: none">• <code>simple condition</code>—bindings and tests are available• <code>from</code>• <code>in</code>

s	
Exception handling	<p>Exception handling is not supported in rule condition parts in sequential and Fastpath modes.</p>
Refraction	<p>The data unit for an engine cycle is the tuple. The engine does not record the tuples as they are matched, it rather forgets them between cycles. Therefore, there is no built-in support for refraction.</p> <p>Using the sequential execution mode, the above rule task becomes:</p> <pre>ruletask main { algorithm = sequential; ordering = literal; body = {Person, PersonProduct} }</pre> <p>The execution trace shows that, using the sequential execution mode, the same rule are executed twice for the same parts of two different tuples.</p> <p>Although there is no built-in support for refraction, the sequential execution mode uses a particular rule application strategy when it generates the body of the tuple matching method. The strategy, however, tries to limit the use of refraction.</p>
IL OG Rule Language (IRL) facilities	<p>In contrast, practically all the script-level expressions and statements of the IRL are available in sequential processing. They serve principally to maintain the consistency of the other rule tasks that are not set to the sequential algorithm.</p> <p>Here is a list of these features:</p> <p>Functions</p> <p>Function definitions and calls are fully available.</p> <p>Script-level expressions</p> <p>All the IRL expressions are available. (Note that ?instance works in any execution mode.)</p> <p>Script-level statements</p> <p>All the IRL statements are available.</p> <p>Update</p> <p>The update construct is available since spurious updates might exist in rule tasks set to the sequential algorithm. A spurious update is an update on an object that cannot be matched by any condition part of rule tasks set to the sequential algorithm, but might be significant for other rule tasks using the RetePlus algorithm. All rule tasks are specified in the ruleset and should be kept consistent with the current engine. However, it is important to note that the usual update is not handled at all by the stateless sequential processing algorithm.</p> <p>Repeatable rules</p> <p>The repeatable rules are not handled by the sequential processing algorithm, but they might be relevant for other rule tasks that use the RetePlus algorithm.</p> <p>Insert</p> <p>Insertion of objects that could match condition parts of rule tasks set to the sequential algorithm might cause inconsistencies when the tuples of objects are extracted from working memory. This is because the objects are inserted in the first position, and the iterator building the tuples are not notified. The iterator does not, therefore, give the tuples involving the new object to the sequential processing engine. However, the insert is still be relevant for other rule tasks using the RetePlus algorithm.</p>

Parent topic: [Optimizing the decision engine](#)

Improving the performance of the Fastpath execution mode

You improve the performance of the Fastpath execution mode by modifying the condition part of rules.

To improve the performance of the Fastpath execution algorithm, you can change the order of tests, or the order of conditions that are shared between several rules.

Parent topic: [Optimizing the decision engine](#)

Publishing decision services to Decision Center

For business users and developers to work in collaboration, you publish and synchronize decision services from Rule Designer to Decision Center.

About this task

To work within the governance framework, changes to the decision service in Decision Center take place in the change activities of a release. Both releases and change activities are branches of the decision service.

In Rule Designer, you can have only one branch of the decision service in your workspace at any given time. To synchronize with the different branches in use in Decision Center, connect, disconnect, and reconnect with each branch individually. When you disconnect, keep the connection entries for that branch. These entries are stored as part of the branch, allowing you to reconnect with that branch without introducing conflicts.

You must be familiar with section [Branches and releases in synchronization](#) to avoid misuse.

Note: For publishing to work, you must run Rule Designer in the same locale as the locale that is used to persist rules in Decision Center.

To publish a decision service:

Procedure

1. Right-click the main project of the decision service in Rule Designer, and then click **Decision Center > Connect**.
2. Enter the URL and data source information that corresponds to your Decision Center session.

In the URL, you can use the /teamserver extension, or /decisioncenter extension indifferently. If no data source is entered, the wizard uses jdbc/ilogDataSource.
3. Click **Connect** to establish the connection with Decision Center.
4. Proceed with the authentication by entering your Operational Decision Manager on Cloud credentials.
5. Optional: To limit the project elements that are synchronized, select **Use the specified query to filter rule elements for synchronization** and a query. The list of queries is defined in the decision service and its dependencies.
6. Click **Next**. In **Synchronization settings**, select **Use Decision Governance Framework**.
7. In the section **Enforce project security**, select the check box to publish your project with enforced security in Decision Center. For more information, see [Security](#).
8. Click **Finish** to publish the project.

Results

If an error occurs during publishing to Decision Center, the **Problems** view displays a message, but the publishing completes. Publishing to Decision Center only copies the information. It does not notify Decision Center users of the changes.

Related concepts:

[Synchronization commands in Rule Designer](#)
[Branches and releases in synchronization](#)

Related tasks:

[Creating projects from Decision Center](#)

Related information:

[Decision services](#)
[Change management](#)
[Synchronization architecture](#)

Adding a dependent project to a decision service

To add a dependent project to a decision service, you must add it in Rule Designer and synchronize the decision service with Decision Center.

Procedure

1. From Rule Designer, connect to the branch or release of the decision service you want to modify.
2. Add the dependent project in Rule Designer:
 - a. If the project exists in Decision Center, import this project into your Rule Designer workspace (see [Creating projects from Decision Center](#)).
 - b. If this project does not exist in Decision Center, create a new project in Rule Designer and publish it to Decision Center (see [Publishing decision services to Decision Center](#)).
3. Add the dependency to this new project in the properties of the decision service.
4. Synchronize the decision service. You do not need to disconnect because you are already connected to it.
5. Publish the change to Decision Center. The `.ruleproject` file should be identified as changed.

Results

If an error occurs during publishing to Decision Center, the **Problems** view displays a message, but the publishing completes. Publishing to Decision Center only copies the information. It does not notify Decision Center users of the changes.

Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

Managing decision services

You collaborate on the development of decision services in Decision Center.

Governing rules with the Business console

You can govern and manage business rules with the Decision Center Business console.

Managing decisions with the Enterprise console

You can manage business rules in the Enterprise console, which is a web user interface for Decision Center.

Synchronizing and storing rules

You can maintain synchronized versions of rules between the business users in Decision Center and the developers in Rule Designer.

Setting up notifications from Decision Center with webhooks

You can use webhooks to receive notifications from Decision Center to the application of your choice.





Authoring and managing rules with Decision Center






Decision Center provides two web consoles that let you author, edit, and manage rules: the Business Console and the Enterprise Console.


The Business Console is the preferred environment for business users who manage and govern lifecycle of decisions. It allows business users to take advantage of change management and deployment of decision services.

The Enterprise Console allows IT users to take advantage of some advanced administrative features.

The following table summarizes which features are available in each console. If a feature exists in both consoles, it is recommended to use the Business Console.

Feature		Enterprise Console	Business Console
A u t h o r i n g	Creating rule projects	You can view and edit classic rule projects and decision services.	You can view and edit decision services, and create decision model services.
	Authoring rules	You can either use the guided editor or the Intellirule editor to build rules. Learn more	Only the Intellirule editor is available to build rules. Learn more
	Modeling decisions		You can create decision model services from scratch, or import them from IBM® Decision Composer. Learn more
	Working with ruleflows	You can preview ruleflows but cannot create or edit them. Learn more	You can create ruleflows and add different types of elements to control the execution of rules. Learn more
	Working with technical rules	You can create technical rules if you have permission to do so. Learn more	You can see and update technical rules. Learn more
	Working with functions	You can create functions if you have permission to do so. Learn more	You can see and update functions. Learn more
V a l i d a t i n g	Running tests and simulations		You can validate your rules using test suites and simulations. The classic rule engine and the decision engine both support testing and simulation. Learn more
D e p l o y i n g	Creating deployment configurations		As a permission manager, you can create and edit deployment configurations. Learn more
	Managing servers		As a permission manager, you can manage the list of servers available to Decision Center and who has access to them. Learn more
M	Managing		

a n a g i n g	changes using the decision governance framework		<p>You can use the decision governance framework, a predefined release workflow that is based on change and validation activities.</p> <p>Learn more</p>
	Managing custom permissions		<p>The Business console proposes simplified permission profiles that you can combine with the decision governance framework.</p> <p>Learn more</p>
	Managing dependencies between projects	<p>As a permission manager, you can create a project dependency to establish a reference from your project to another project.</p> <p>Learn more</p>	<p>As an administrator or configuration manager, you can set up project dependencies to make a project available in different decision services or branches.</p> <p>Learn more</p> <p>A decision service can contain several projects, which are linked together by default.</p> <p>Learn more</p>
	Managing branches	<p>You can create and merge branches to manage the evolution of projects over time.</p> <p>Learn more</p>	<p>You can create and merge branches to manage the evolution of projects over time.</p> <p>Learn more</p>
	Creating decision operations		<p>You can create decision operations. Decision operations include all the settings that are needed to define the contents of a ruleset and its parameters.</p> <p>Learn more</p>
	Working with dynamic domains	<p>You can update dynamic domains and update your project in the Enterprise console.</p> <p>Learn more</p>	<p>You can update dynamic domains and update your project in the Business console.</p> <p>Learn more</p>
	Taking snapshots	<p>You can create baselines to capture the state of a project or a branch at a specific moment in time.</p> <p>Learn more</p>	<p>You can take snapshots to capture the state of a branch at a specific moment in time.</p> <p>Learn more</p>
	Following streams and posting comments		<p>You can follow streams and post comments that other users can reply to.</p> <p>Learn more</p>
	Using smart folders	<p>You can use smart folders to navigate through your projects according to criteria that you select.</p> <p>Learn more</p>	<p></p> <p>Use the Decision Artifacts tab to navigate through your decision service. You can use the search bar or queries to find project elements according to specific criteria, and display the properties you want by selecting the columns in your user profile.</p> <p>Learn more</p>

	Generati ng reports	You can generate project reports for classic rule projects and decision services. Learn more	You can generate project reports for decision services using the Decision Center REST API. Learn more
	Creating and running queries	You can use queries to search through classic rule projects. Learn more	You can use queries to search through elements of your projects. Learn more
C o n f i g u r i n g	Setting configura tion paramet ers		You can set some configuration settings from the Settings tab. Learn more
	Using build options	You can set build options. Learn more	You can set build options for decision services. Learn more
	Running diagnosti cs	As a permission manager, you can run diagnostics on different parts of Decision Center to check the state of your system. Learn more	As a permission manager, you can run diagnostics on different parts of Decision Center to check the state of your system. Learn more

[Working with the Business console](#)

You can govern and manage business rules with the Decision Center Business console.

[Working with the Enterprise console](#)

You manage decisions with Decision Center, which you can access in a web user interface, the Enterprise Console.

Working with the Business console

You can govern and manage business rules with the Decision Center Business console.

[Introducing the Business console](#)

To adapt to evolving business needs in their company, business users need to be involved throughout the development of a decision service. The Business console offers a collaborative and secure environment for business users and rule developers to author, manage, test, and deploy rules.

[Managing decision services](#)

The Business console comes with features for managing decision services.

[Decision artifacts](#)

In the **Decision Artifacts** tab, you create and edit rule artifacts, such as action rules, decision tables, or ruleflows, which you can organize in folders. You can define decision operations, ruleset variables, and handle resources for your projects.

[Modeling decisions in the Business console](#)

Generally, IT creates the decision model on which rules are authored. For decision model services, you create both the model and the logic directly in the Business console.

[Using queries](#)

In the Business console, you use queries to search for elements of your projects. You can apply actions to the results of the query.

[Testing sets of rules in the Business console](#)

Test the rules that you create or edit to achieve the results that you expect.

[Simulating business application results](#)

Run rules on representative data to generate results that you can use to improve the application.

[Deploying from the Business console](#)

In a decision service, you can deploy a set of rules to a production environment or to a non-production environment for testing or quality assessment.

[Administering projects from the Business console](#)

A user who has the permission manager role can access the **Administration** tab in the Business console to administer security, manage servers, or run diagnostics.

Introducing the Business console

To adapt to evolving business needs in their company, business users need to be involved throughout the development of a decision service. The Business console offers a collaborative and secure environment for business users and rule developers to author, manage, test, and deploy rules.

In the Business console, you access decision services that contain all the elements necessary to author and manage business rules. These elements are contained in a rule repository, which is a database that is connected to Decision Center.

Changes to business rules are not reflected back automatically to the client applications that call them. They must first be deployed. In the Business console, users with the appropriate rights can deploy rules to a production environment. Other users can deploy rules to nonproduction servers to validate their changes.

The rules that you work on are organized as part of a greater set of rules, starting with a decision service, and then in a branching mechanism that handles changes over time. See [Identifying a set of rules](#) for more information.

[Getting familiar with the Business console](#)

The Business console provides a collaborative environment for authoring and managing rules.

[Identifying a set of rules](#)

The primary function of the Business console is to let you author business rules. A rule that you author does not exist by itself, however, and is organized as part of a greater set of rules. Identifying what set of rules you are ultimately working on is key in understanding how to manage rules.

[Searching for rules and folders](#)

You can do a text search to find action rules, decision tables, or folders. The search can look across linked projects, and for rules that support different locales.

[Following streams and posting comments](#)

In the Business console you can see what events are happening on releases or activities that you want to follow, and post comments that other users can reply to.

[Accessibility](#)

You can use keyboard shortcuts and accessibility services with the Business console.

Parent topic: [Working with the Business console](#)

Getting familiar with the Business console

The Business console provides a collaborative environment for authoring and managing rules.

If you are an authorized user, logging in gives you access to the **Home** page. Contact the administrator if you are unable to log in.

After logging in, the Business console presents three tabs:

- **Home:** Find useful links about the basic features of Decision Center in **Get started**, updates about your projects and followed items in **Recent activities**, **Followed Rules**, and **Rules Recently Worked On**.
- **Library:** See the available decision services.
- **Work:** See a list of all the ongoing activities in which you are a participant, with shortcuts to rename, cancel, or change the status of activities.

When entering your credentials, if you check the option **Keep me logged in**, you can close your browser window without logging out, and automatically log in as the same user the next time you access the Business console.

Note:

Available features in the Business console may depend on your user profile. This arrangement helps ensure the full review of rules before they are deployed to a production environment.

User profile

You can edit your user profile and select options according to your preferences. Your user options apply only to your account in the Business console. They do not affect any other users. To edit your profile, click **Profile** in the drop-down list next to the user name in the top banner of the Business console.

You can set the following options:

- Add a user photo.
- **Display:** Change your display name and select your preferred language. By default, the language is determined by a parameter in the URL that you use to access the Business console, or by the language of your browser if the URL has no language parameter. For information about how to set the browser language, refer to the help documentation for your browser.

You can change the language of the Business console to one in the list of languages, only if your development team translates the Business console environment and provides you with translated projects.

- **Grid columns:** Select and order the columns used to display the properties of rules, decision tables, ruleflows, and variable sets in:
 - The **Decision Artifacts** tab.
 - The query results in the **Queries** tab.
- **Automatically follow:** Specify what activities to follow.

Direct links

Any URL in the Business console is accessible externally. You can bookmark these URLs in your browser, or reference them from an external source, and navigate directly to a specified branch, release, activity, or project element.

Parent topic: [Introducing the Business console](#)

Related concepts:

[Accessibility](#)

[Following streams and posting comments](#)

[Managing changes with the decision governance framework](#)

[Governance principles](#)

[Rule properties](#)

[Snapshots](#)

Identifying a set of rules

The primary function of the Business console is to let you author business rules. A rule that you author does not exist by itself, however, and is organized as part of a greater set of rules. Identifying what set of rules you are ultimately working on is key in understanding how to manage rules.

The first level of identification is the **decision service**. Rules are stored within **rule projects** contained in a decision service.

Branches

The second level of identification is through **branches**. Starting from the rules contained in a rule project, Decision Center uses branches to manage rules over time. A branch of a rule project starts with the same rules as the parent, but then allows for a separate evolution of the same rules. Branches can be as follows:

- Releases and change activities of a decision service. Releases and change activities are **governed** branches because they are used within the decision governance framework, and they have their own characteristics (see [Managing changes with the decision governance framework](#)).
- A regular branch of a decision service, stemming from the main branch. This allows for work on a decision service without the decision governance framework.
- **Snapshots** of any branches can also be considered branches because they represent a read-only state of the rules of a branch at a past moment in time.

Decision operations

Finally, the third level of identification is the **decision operation**. The decision operation further identifies which rules from a given branch are deployed or used for testing. Not all the rules contained in the branch that you are working on are necessarily destined to be validated or deployed. For example, you may want to test or deploy only those rules that are **Ready to be tested** or whose status is **deployable**. The decision operation defines which rules are included in the operation.

In the Business console, you choose which decision operation to use when creating a test suite, simulation, or deployment configuration. Information relating to a decision operation is visible when you select it when creating the test suite, simulation, or deployment configuration.

Note: For decision model services, the decision operation is automatically generated and contains all the rules of the model.
--

Creating decision operations is done in Rule Designer

Parent topic: [Introducing the Business console](#)

Searching for rules and folders

You can do a text search to find action rules, decision tables, or folders. The search can look across linked projects, and for rules that support different locales.

Searching for rules

The search makes a list of matching elements, and you use display filters to control the content of the list. The order of the list depends on how closely the elements match your text.

Type the text you want to find in the search bar from a release, change activity, or ungoverned branch. The search function takes into account the language of your profile, either chosen by you or the language of your browser. If the language is not supported by the searched project, the search function uses the default language of Decision Center.

If you want to find only specific elements, you can filter the search results by:

- **Type:** Filter by action rules, decision tables, or folders. Clear the **All** check box to select individual types.
- **Status:** Filter by the status property of a rule. Clear the **All** check box to select a specific status.
- **Found in:** Filter by content or properties. The search function looks for search strings in the content of rules, or only in the properties of project elements.
- **Projects:** Filter by the projects that are linked to the current one. All the projects of a decision service are linked.
- **Last edited:** Indicate a time period for the results. The filter includes calendars for selecting the start and stop dates of the period.

The search automatically updates the list of results. The elements are listed by the number of times your search string occurs in them. For example, the search lists a rule with 10 occurrences before a rule with fewer occurrences.

The search page shows the total number of project elements that match your search string. As you scroll to the bottom of the page, the search retrieves the succeeding results and appends them to the list. The page lists up to 25 entries at a time.

Search strings and syntax

You can use words, numbers, or combinations of both. For best results, use uniquely identifiable search strings. The search ignores articles (a, an, the...), prepositions and conjunctions (and, or, of...), and words that are common to most rules (if, then, else...).

Multiple search strings

If you enter more than one string of characters, the search looks for elements that contain all the strings, and the strings individually. For example, if you enter the word loan, the search looks for only those rules that contain the word loan. If you enter loan and duration, the search looks for all the rules that contain both words, and the words individually.

Names

You can search for elements by file name or author. When you search by author, you can use the user name or display name. The user name is the name that is used to sign in to Decision Center, while the display name is the name displayed with messages and elements.

Types of text and syntax

The search lists the results that are based on how closely they match your text. It lists the project elements with all the strings before the projects with individual instances.

You can type a string that consists of letters, numbers, or both into the **Search** field. You can search for a whole string or just part of a string. You can also use special syntax to prioritize or exclude text, or to search for an exact match.

The following table lists the types of text and search syntax that you can use in search strings.

Table 1. Types of text and search syntax

Text or search syntax	Description
Letters	Individual letters or groups of letters that do not form words. For example, WYSIWYG.
Words	A complete word or words. For example, loan, miniloan, or user name.
Uppercase and lowercase	The search ignores case. You can use uppercase and lowercase letters together. For example, New Loan.

Numbers	One or more numbers. For example, 1, 2 30, or 100.
Alphanumeric combinations	Combinations of letters and numbers. For example, Y2K or Name123.
Prioritizing text with the plus (+) modifier	You can type a plus sign (+) in front of a word to have the search look only for elements with this word. The search treats any other word in the search field as optional. For example, if you type loan +duration, the search lists all the elements that contain the word duration, and any element that contains both loan and duration. It does not list elements that contain loan but do not contain duration.
Excluding text with the minus (-) modifier	You can type a minus sign (-) in front of a word to have the search ignore all the elements that contain the word. When the search finds the word, it does not list the element with the word on the search page. For example, if you type loan -duration, the search lists only the elements with the word loan, and excludes any element that contains both loan and duration.
Exact matching with quotation marks (" ")	By placing quotation marks (" ") around a search string, you have the search match the string exactly. For example, a search for "mini loan rule 3" lists elements with only the quoted text.

Parent topic: [Introducing the Business console](#)

Following streams and posting comments

In the Business console you can see what events are happening on releases or activities that you want to follow, and post comments that other users can reply to.

Streams

The Business console provides a social stream feature that you use to follow work and interact with others.

Events relating to a release or activity are listed in the **Stream** view, available next to the **Properties** tab when you display this release or activity.

An event is when someone, including you, does tasks such as:

- Create, edit, or delete a rule or folder.
- Create or change the state of a release, change activity, or validation activity.
- Create or restore a snapshot.
- Post a comment.

Another **Stream** view is available on the **Home** page. This stream includes only the events that you subscribe to follow.

Following events

The stream on the **Home** page displays events that you subscribe to follow, and any general posts. A golden star next to the name of an item (release, change activity, folder, or individual rule) means that you are subscribed to it. An empty star means that you are not.

Depending on the type of item you follow, you are notified of the following information:

Release

Important changes to the release, such as status changes, approval, cancellation, or rejection, and the main changes to the activities of the release.

Change or validation activity

Main changes to the activity, such as status changes and rule changes in the activity.

Folder

Changes to all the rules that are currently contained in the folder.

Note: To follow the entire folder, you must select all the rules of the folder, and click the star.

Individual rule

Follows events on the rule across all decision services, releases, or activities in which that rule is contained.

To reduce the number of manual steps that are required to subscribe to events, options are available to automatically follow certain events, such as rules you edit or releases or activities that you are an approver of. You can enable or disable these and more options by clicking **Profile** in the drop-down list next to your user name in the top banner.

Posting comments

You can post comments in all **Stream** views:

- **On the Home page:**

Comments posted here are visible to all Business console users by default, and can include web links and file attachments. You can select **Private Posting** to restrict who can see the post. When you click **Post** with this option enabled, you can select the release or change activity. Only users that have access to these releases or change activities see your post.

- **When displaying or editing a release, activity, or rule:**

Comments posted here are visible to users that have access to the release, activity, or rule.

You can reply to any post, from either one of these views, by adding a comment to the post.

Parent topic: [Introducing the Business console](#)

Related concepts:

[Getting familiar with the Business console](#)

Accessibility

You can use keyboard shortcuts and accessibility services with the Business console.

You use the Business console in a web browser. Mozilla Firefox, Microsoft Internet Explorer, and other browsers provide keyboard and mouse shortcuts for navigating web pages. They also provide features for the visually impaired, such as screen magnification, and color and font control. You can use these browser controls with the Business console.

You can also use the console with screen readers, which read the text aloud and can magnify it for easier viewing. Screen readers also provide navigation shortcuts.

Special keyboard shortcuts

The following tables list actions and their keyboard shortcuts in the Business console.

Table 1. Rule editor

Action	Keyboard shortcut
Open the completion menu.	Ctrl+Spacebar
Open the completion menu in a tree view.	Ctrl+Shift+Spacebar
Open the next placeholder.	Alt+Right Arrow key
Open the previous placeholder.	Alt+Left Arrow key
Open a placeholder under a caret.	Alt+Down Arrow key
Open the menu of completion options in the toolbar.	Alt+O
Insert a tab space.	Tab
Format the text.	Ctrl+Shift+F
Undo	Ctrl+Z
Redo	Ctrl+Y
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Move the focus forward from the editor to the error grid.	Ctrl+Shift+Down Arrow key When you edit a rule, press Tab to insert a tab space.
Move the focus backward from the editor to the toolbar.	Ctrl+Shift+Up Arrow key Shift+Tab also moves the focus from the text editor to the editing toolbar.
Filter the contents of the completion menu.	As you type, the contents of the completion menu are filtered by their prefixes.

Table 2. Rule editor completion controls

Action	Keyboard shortcut
Close the completion menu.	Esc
Toggle the filter mode.	Ctrl+F
Scroll through the completion menu.	Up and Down Arrow keys
Page through completion menu.	Page Up and Page Down keys
Select the next or previous entry.	Home or End keys
Collapse or expand a tree node.	Left or Right Arrow keys
Insert the selected content.	Enter or Tab

Table 3. Decision table navigation

Action	Keyboard shortcut
Move the selected cell up, down, left or right among the table cells.	Up, Down, Left or Right Arrow key
Move the selected row up or down among the table rows.	Up or Down Arrow key
Move the selected column left or right among the table columns.	Left or Right Arrow key
Extend the selected cell left or right from the currently selected cell.	Shift+Left or Right Arrow key
Extend the selected row up or down from the currently selected row.	Shift+Up or Down Arrow key

Extend the selected cell to all the cells from the currently selected cell.	Ctrl+A
Extend the selected row to all the rows from the currently selected row.	Ctrl+A
Reset a decision table to its original layout.	Ctrl+Alt+P <div>Note: Only available in automatic row ordering mode.</div>
Close the pop-up menu of the selected cell, column, or row if still open	Esc
View the definition and any errors of a selected cell, row or column.	F9
Select the column of the currently selected cell.	Ctrl+Spacebar
Select the row of the currently selected cell.	Shift+Spacebar
Toggle the column sort.	Spacebar The column must first be selected by using the Ctrl+Spacebar. <div>Note: Sorting is not available for action columns in manual row ordering mode.</div>
Make the selected column narrower by 5 pixels.	Ctrl+Shift+Left Arrow key
Make the selected column wider by 5 pixels.	Ctrl+Shift+Right Arrow key

Table 4. Decision table editing

Action	Keyboard shortcut
Edit the selected cell.	Enter
Copy a selected item.	Ctrl+C or Cmd+C
Cut a selected item.	Ctrl+X or Cmd+X
Paste a copied or cut item to another location in a table.	Ctrl+V or Cmd+V
Insert copied or cut cells.	Ctrl+Shift+V or Cmd+Shift+V
Delete the content of the selection.	Delete
Undo	Ctrl+Z or Cmd+Z
Redo	Ctrl+Y or Cmd+Shift+Z
Show the menu for a row header, column header, or cell.	Shift+F10 The table element must first be selected.
Edit the column header title.	Enter The column must first be selected.
Edit the column definition.	Ctrl+Alt+A A column must be selected.
Delete the selected column or selected row or rows.	Ctrl+Delete
Group rows.	Ctrl+Alt+P
Enable or disable an action cell or cells.	Ctrl+Q
Split	Ctrl+L Only available for condition cells in manual row ordering mode.
Merge	Ctrl+G Only available for condition cells in manual row ordering mode.

Table 5. Rule detail dialog

Action	Keyboard shortcut
--------	-------------------

Move through the active buttons.	Tab and Shift+Tab Use the Down Arrow key to activate fields and edit them.
Activate a button.	Down Arrow key The JAWS virtual PC cursor can read only the active buttons. To select a button, navigate to it by using Tab and Shift+Tab.

Table 6. Timeline navigator

Action	Keyboard shortcut
Move in the timeline navigator.	Tab and Shift+Tab
Set the focus on a timeline item in the timeline view.	Enter Sets the focus on the first item in the month.

Table 7. Timeline view

Action	Keyboard shortcut
Move through the active item links.	Tab and Shift+Tab
Jump through timeline items.	Left and Right Arrow keys
Scroll through a timeline.	Up and Down Arrow keys
Open the snapshot dialog from a selected item.	Shift+Left Arrow key
Move the focus back to the matching month.	Esc
Open a rule or snapshot in a project timeline.	Press Tab to the hotlink inside the box to open the rule or snapshot.

Table 8. Project view of rules by folder

Action	Keyboard shortcut
Move through the folders.	Up and Down Arrow keys
Move through the rules.	Up and Down Arrow keys
Collapse a folder.	Left Arrow key
Expand a folder.	Right Arrow key
Select a folder or rule.	Click the first character of the folder or rule name.
Move through folder actions.	Left and Right Arrow keys Press Enter to execute an action, or Esc to exit the drop-down menu.

Table 9. Release or Activity Properties view.

Action	Keyboard shortcut
Move into the Properties area from the top Properties tab.	Tab
Move through the editable property fields and buttons.	Left and Right Arrow keys
Move through the list sections.	Tab and Shift+Tab
Open and collapse a list section.	Spacebar
Move through the users in a list section.	Up and Down Arrow keys
Move through the editable user fields and buttons.	Left and Right Arrow keys
Open an editor in an editable field.	Enter
Leave an editor.	Enter* or Esc *With some editors, you must press Enter twice.

Table 10. Validation Activity Test Plans view

Action	Keyboard shortcut
Move into the header section.	Tab
Move through the column headers.	Right and Left Arrow keys
Sort a column.	Enter or Spacebar
Move into the grid data section.	Tab

Move through the test plans.	Up and Down Arrow keys
Edit highlighted test plan.	Enter
Delete highlighted test plan.	Delete

IBM and accessibility

For more information about the commitment that IBM® has to accessibility, see [IBM Accessibility](#).

Parent topic: [Introducing the Business console](#)

Related concepts:
[Getting familiar with the Business console](#)

Managing decision services

The Business console comes with features for managing decision services.

Decision services

A decision service contains one or more rule projects. Each rule project contains action rules and decision tables that you update directly. You can easily move among the projects to make changes, and deploy the decision service to a test or production environment.

Managing branches

Decision Center enables branching so that you can manage the evolution of projects over time.

Using governance with decision services

The decision governance framework is a methodology that provides you with tools to implement changes to your business rules in a structured, secure, and controlled way.

Linked projects

When a decision service is published to Decision Center, it can contain linked (or dependent) projects.

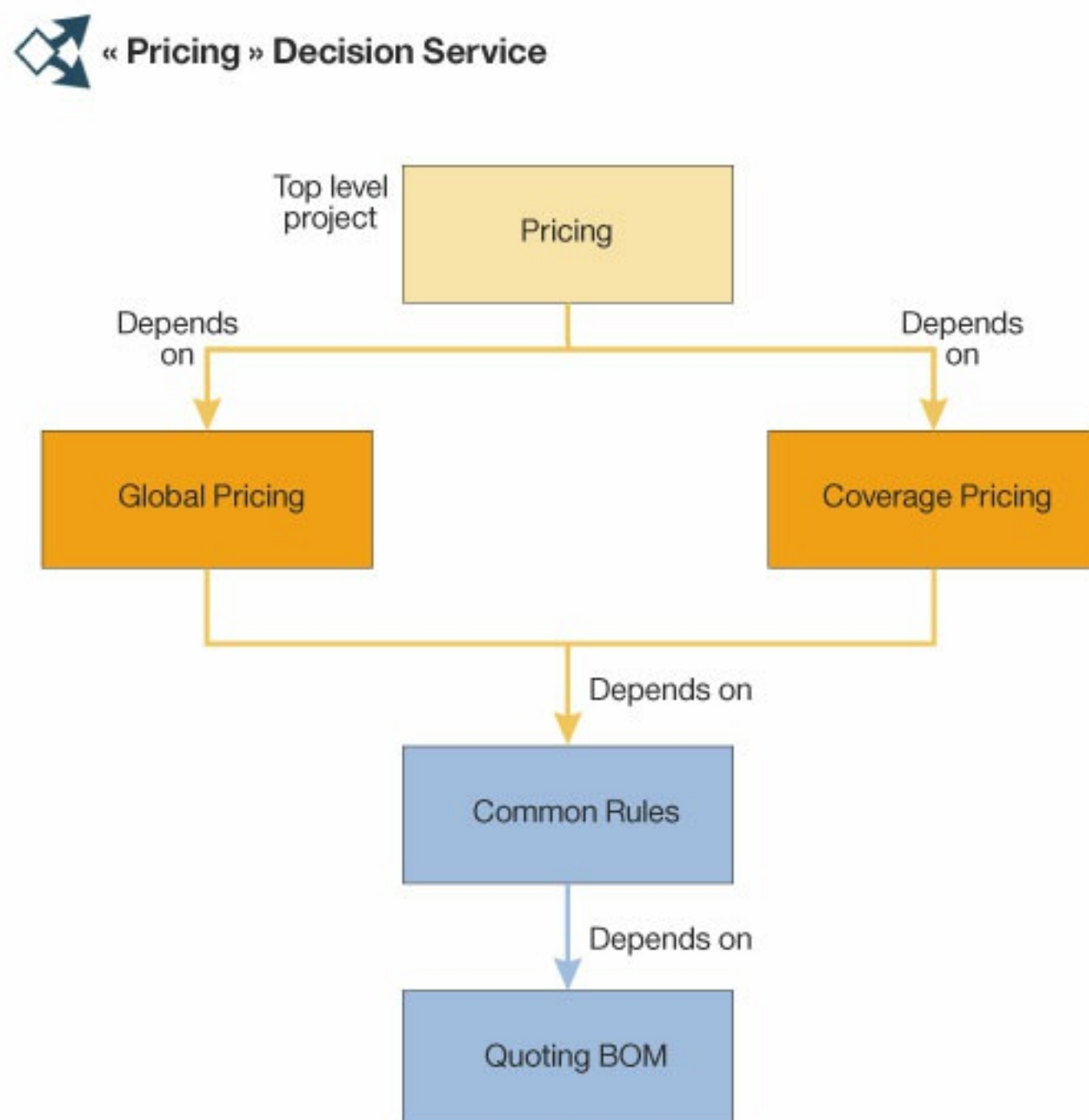
Parent topic: [Working with the Business console](#)

Decision services

A decision service contains one or more rule projects. Each rule project contains action rules and decision tables that you update directly. You can easily move among the projects to make changes, and deploy the decision service to a test or production environment.

Note: A decision model service is also available, as described in [Modeling decisions in the Business console](#).

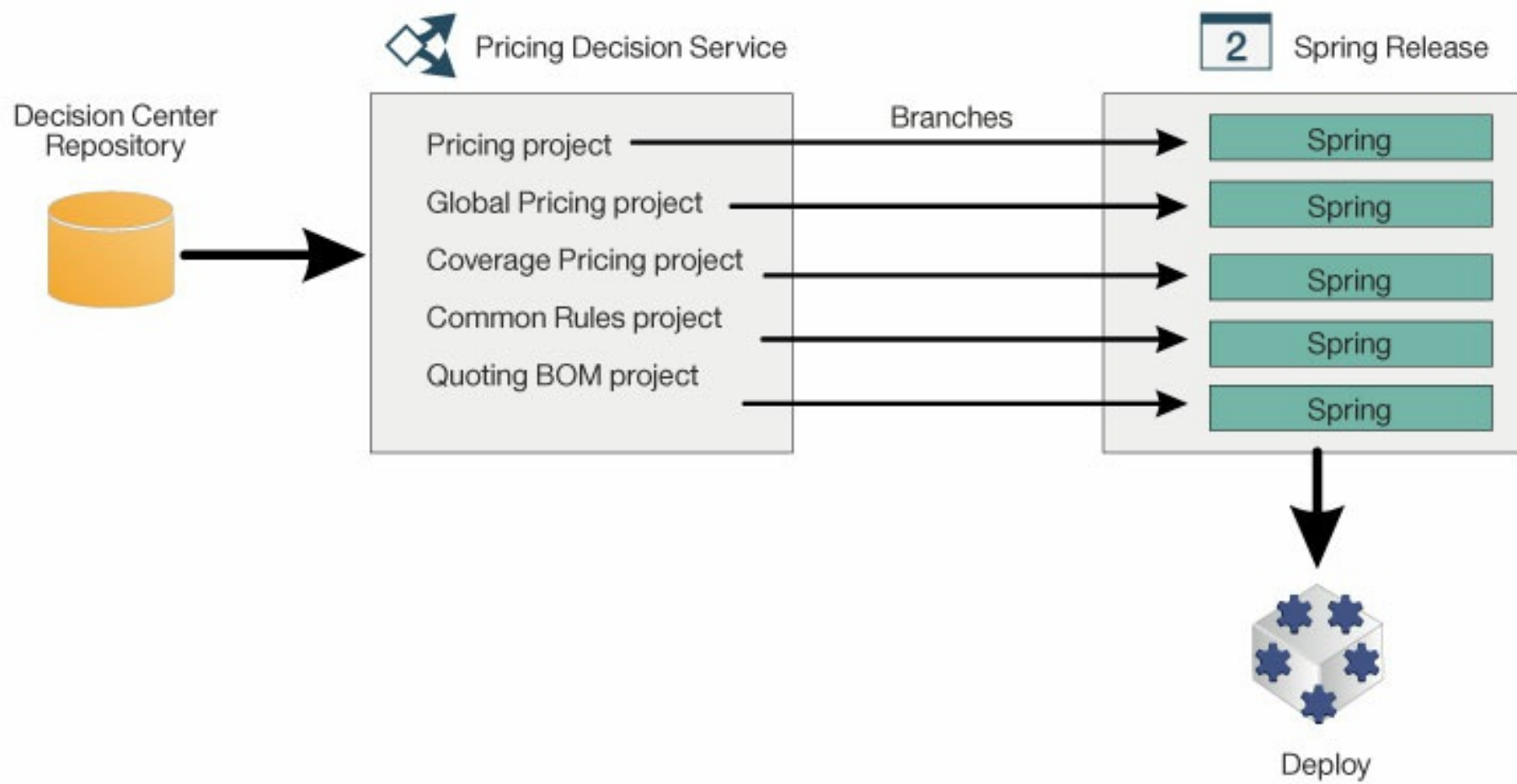
The following diagram shows an example of a decision service, with a top-level project and its dependent projects:



Lifecycle of a decision service

A decision service is published to Decision Center, and appears in the Business console library. Each decision service contains an initial release, which cannot be changed, and subsequent releases. Changes to a decision service are managed in releases and activities.

Each release is made up of a branch of each project that is contained in the decision service. For example, the following diagram shows that the Spring release of the Pricing decision service contains the Spring branch of each project that is contained in the decision service:



When a release is fully developed and approved, it can be deployed to a production environment.

A permission manager can delete a decision service from the **Library > Decision services** page in the Business console. This action permanently removes all the projects, branches, releases, and associated entries of the decision service in the database, and it no longer appears in Decision Center.

Parent topic: [Managing decision services](#)

Related concepts:

[Deploying from the Business console](#)

[Governance principles](#)

Managing branches

Decision Center enables branching so that you can manage the evolution of projects over time.

What are branches

When a branch is created, it contains an exact replica of every project element that is contained in its parent branch. You can then work on the subbranch without affecting the contents of the parent.

Snapshots

Snapshots capture the state of a branch at a specific moment in time.

Merging branches

In the Decision Center Business console, you can merge the content of branches, releases, and activities in a decision service.

Deleting and recovering elements in branches

When you delete project elements from the current state of a branch, they are put in the recycle bin of that branch. You can restore those elements to your branch.

Generating a project report

You can generate reports in HTML format based on decision services, projects, or queries. These reports show the content and properties of deployed project elements.

Parent topic: [Managing decision services](#)

What are branches

When a branch is created, it contains an exact replica of every project element that is contained in its parent branch. You can then work on the subbranch without affecting the contents of the parent.

There are several types of branches in Decision Center:

- The releases and activities of a decision service. Releases and activities are used with the decision governance framework, a prescriptive workflow to implement change management within Decision Center. They are **governed** branches, and have their own characteristics (see [Managing changes with the decision governance framework](#)).
- A regular branch of a decision service, stemming from the main branch. This branch allows for work on the decision service without the decision governance framework.
- **Snapshots** of any branches can also be considered branches because the snapshots represent a read-only state of the rules of a branch at a past moment in time.

When the initial version of a decision service is created, published, or imported, both a **main** branch and an **Initial Release** branch are created. The main branch is available if you want to work in a decision service but not use the governance framework. The Initial Release branch is published as the closed initial release of the decision service.

In the Business console, you can create new branches off the main branch, and rename them. If you use the governance framework, you can create new releases with any closed release as a starting point. The owner of a release can then create change and validation activities. You can also copy an open release and its content with any closed release as a starting point.

You can delete any branch, except for the main branch. To delete a release, you must have a Permission manager role. The release owner can also delete a release, if it is not in a *Complete* state.

Note: You might want to delete releases you do not use anymore, but be aware that all child releases, including the ones in progress, are also deleted. To keep your releases in progress, you can copy them from the **Releases** tab when you access the decision service. The copies are all created under the initial release, as child releases. That does not impact their content.

Merging of branches is done automatically within the context of the governance framework, but you might also be required to merge release and change activity branches, if changes to one need to be pushed to the other, or if the automatic merging fails due to conflicts. For these cases, you must consider the following actions:

- You can merge a change activity into any other release in the decision service when the change activity is completed and the release is not already completed.
- You can merge a release into a change activity in the decision service when the change activity is not already completed.
- You cannot merge a release with another release in that decision service.
- You can merge a change activity with another one in the decision service when the target change activity is not already completed.

Parent topic: [Managing branches](#)

Related tasks:
[Managing subbranches and baselines](#)

Snapshots

Snapshots capture the state of a branch at a specific moment in time.

You can create snapshots that are based on the current or past state of a branch, including releases and change activities.

There are different ways to take snapshots:

- You can take a snapshot of the current state of a branch by clicking **Take Snapshot** within the branch.
- Decision Center automatically creates a snapshot of a release or change activity when you create it, or of a release when a change activity merges with the release.
- You can take a snapshot that is based on what the state of a branch was at a previous moment in time. To do so, follow this procedure:
 1. Go to the branch and click **Timeline**.
 2. Make sure that you are showing **All** events and that no filters are applied.
 3. In the timeline, scroll to the time that you want to capture in the snapshot.
 4. Hover along the vertical axis of the timeline, which separates the timeline events.
 5. Click the snapshot icon that appears on the vertical axis. Enter a name and a description and click **Create**.

You can consult snapshots, and compare them with other snapshots of this branch, or with the current state of the branch. If you have the appropriate permissions, you can rename or delete existing snapshots from the **Snapshots** tab. When you consult a snapshot, you cannot edit its contents.

You can restore a snapshot so that it becomes the current state of a branch. To restore a snapshot of a release, you must be the owner of the release or an administrator. The release must be in progress, and all the activities of the release must be complete or canceled.

In addition to the snapshots that you create, Decision Center automatically creates some snapshots. For example, when you create a new change activity, Decision Center makes a snapshot of the initial state of the activity. Also, when a change activity is completed, Decision Center automatically takes a snapshot of the parent release before merging the contents of the change activity.

Finally, deployment snapshots are a special type of snapshot, which can be automatically taken at the moment of deployment, that capture the state of the rules at the moment of deployment. Deployment snapshots appear in the list of snapshots, and can be used to redeploy.

Parent topic: [Managing branches](#)

Related concepts:

[Getting familiar with the Business console](#)

Related tasks:

[Creating a snapshot](#)

[Redeploying](#)

[Viewing the timeline of a release or activity](#)

Merging branches

In the Decision Center Business console, you can merge the content of branches, releases, and activities in a decision service.

From one of the branches you want to merge, click **Merge Branches** in the toolbar, and select the branch you want to merge with. After you specify the branches to be merged, Decision Center displays a table with the projects that were modified. You can expand each project to see the name and folder of project elements that are different between the branches, and their state in both branches. This table also contains an **Actions** column, containing a proposed action to take. You can click inside the cells of this column and decide what to do with each project element that is different between the two branches:

- Update the working branch with the modifications made in the branch you selected in the **Choose Branch to Merge With** dialog
- Update the branch you selected in the **Choose Branch to Merge With** dialog with the modifications made in the working branch
- Take no action at all

The elements taken into account during a merge operation are:

- Rule Artifacts (BAL rules, technical rules, decision tables, decision trees, functions)
- Simulation artifacts (metrics, KPIs, simulation models, input data, simulations)
- Testing artifacts (test suites, test cases)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- BOM
- Vocabulary
- B2X
- Resources
- Folders

Note:

The following changes cannot be merged:

- For classic rule projects: Changes to ruleset parameters, and extractors. You must change these manually in the other branch.
- For decision services: Changes to decision service properties, categories, queries referenced in decision operations, dynamic XOM (schemas), configuration file for the classic rule engine, and the engine mode. You must change them manually in the release branch.

For both classic rule projects and decision services, project dependencies cannot be merged either.

To help Decision Center propose the correct action to take, you can specify your preferred direction for merging from the following options:

Bidirectionally

This is the default option. The proposed action is based on the assumption that you want to reflect changes made in either branch as follows:

- If a new rule exists in one branch, you want to add it to the other branch.
- If a rule has been deleted in one branch, you want to delete it from the other branch.
- If a rule has been modified in one branch but not in the other, you want to update the unmodified one.
- If a rule has been modified in both branches, no action is proposed, and you must specify what action to take.

Only to the branch you merge with

When you want all changes you made in the working branch to be pushed to the branch you are merging with, but none of the changes made in that branch to be reflected in your working branch.

Only to the working branch

When you want all changes made in the branch you are merging with to be reflected in your working branch, but none of the changes made in your working branch to be pushed to the other branch.

You can compare the different versions of a project element by hovering your cursor over it, and clicking **Show compare view**.

After you click **Apply Merge** in the upper right corner, you can add a comment, and select the option to

create a snapshot of the branches before you merge them. When the merge operations are completed, a report is displayed where you can review the changes made. If you created a snapshot of the branches before merging, you can also review these changes later, by comparing the snapshot with the current version of the branch.

Parent topic: [Managing branches](#)

Deleting and recovering elements in branches

When you delete project elements from the current state of a branch, they are put in the recycle bin of that branch. You can restore those elements to your branch.

You cannot edit an element in the recycle bin, but if you have sufficient permissions to edit in a change activity, validation activity, or ungoverned branch, you can restore it so that it returns to the current state of the branch. The recycle bin of a release is in read-only mode for all users, you must go in a change activity to restore elements.

The recycle bin also displays deleted folders. If you restore a deleted folder, you do not also restore the rules it contains. You must restore them explicitly.

Some artifact views in the recycle bin might display missing information or errors. This happens when a deleted object has a reference to an object that is not deleted. For example:

- When a deleted deployment configuration references an operation that is not deleted, the operation is not found in the recycle bin and cannot be displayed.
- When a deleted operation references a variables in a variable set that is not deleted.
- When the BOM is not found in the recycle bin, a deleted variable shows errors.

Note:

- You cannot restore elements in a closed activity.
- You can only restore elements in the recycle bin, you cannot permanently delete them.

Parent topic: [Managing branches](#)

Generating a project report

You can generate reports in HTML format based on decision services, projects, or queries. These reports show the content and properties of deployed project elements.

To generate a project report in HTML format:

1. Select your decision service, and the branch or release/change activity for which you want to view the content.
2. Click **Reports** in the top toolbar to view the **Reports** tab. You can also select **Queries > Reports**.
3. From this view, you can generate a full report of your decision service, or a report of a project in your decision service.
4. To generate a report based on a query or a project, first open the **Queries** tab and select the query or the project you want. Then go back to the **Reports** tab, where you can now generate a report based on this query or project.

If you generate a report based on a query, this query must have been saved previously. If you are editing it and want to create your report from this new version, save the query before generating the report.

The report is generated in another browser window, and you can download it. It contains the details of the following deployed artifacts:

- Action rules
- Decision tables
- Ruleflows
- Technical rules
- Functions
- Variable sets
- Operations

The report does not show resources.

On Mac OS, you need to set the Java™ VM argument `-Djava.awt.headless=false` to generate ruleflow diagrams. See the limitation "In the Enterprise console on the Mac OS, the ruleflow viewer triggers an exception" in [Operational Decision Manager Known Limitations](#).

Note: The maximum size of the report that you can generate is 100 Mb by default. If you reach this size limit, the report will be incomplete. To be able to view all your content, you can generate several reports based on your projects or queries.

Parent topic: [Managing branches](#)

Using governance with decision services

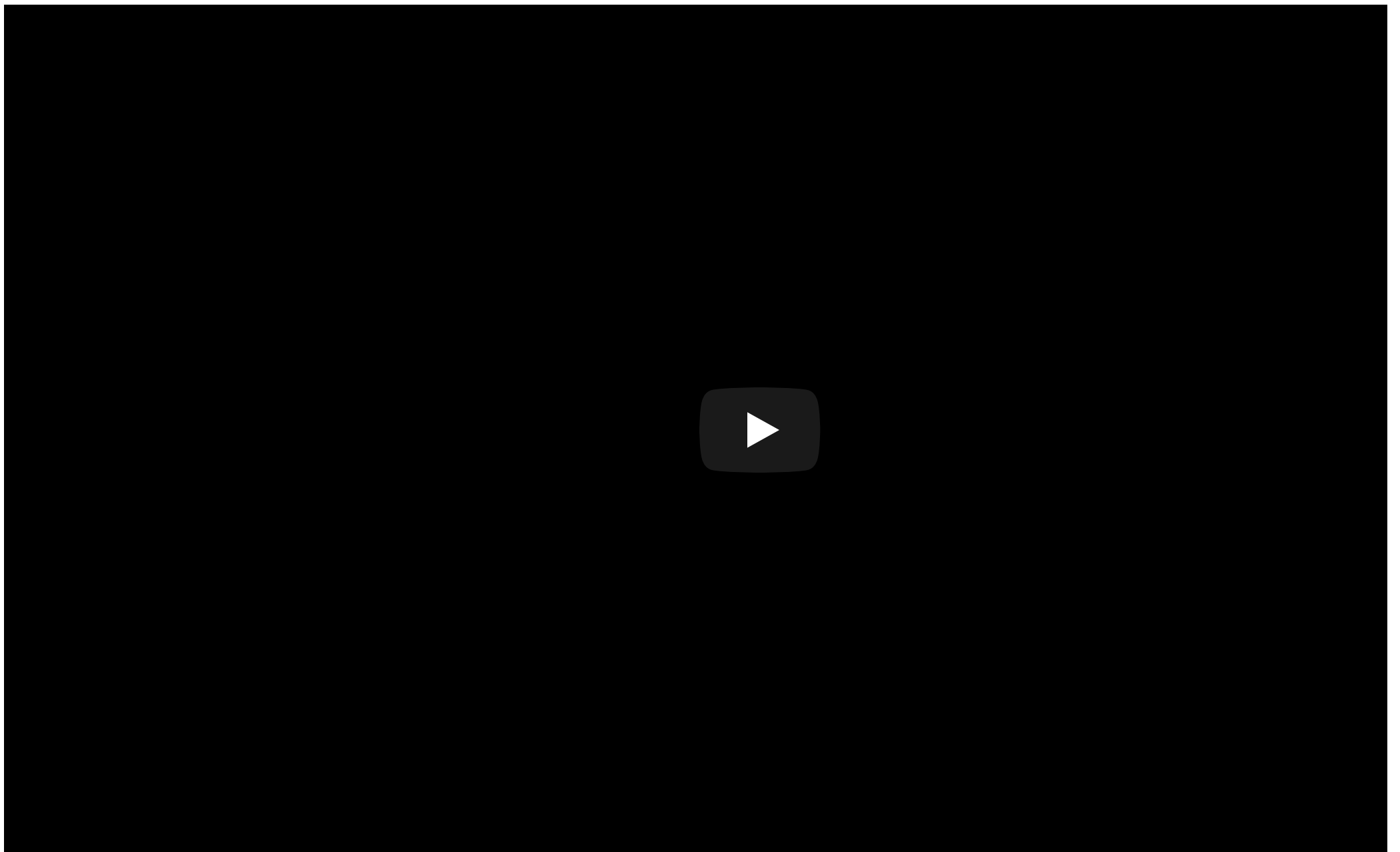
The decision governance framework is a methodology that provides you with tools to implement changes to your business rules in a structured, secure, and controlled way.

It is based on the management of releases and activities, and the distribution of tasks between users with specific governance roles. To understand the main aspects of governance, see [Governance principles](#).

Typically, the purpose of a release is to implement one or more changes to the business policy. The release is composed of change activities, and each change activity usually contains changes to a single business policy. Releases can also contain validation activities, which are used for testing and validating the change activities within the release. To know more about how to manage changes through releases and activities, see [Managing changes with the decision governance framework](#).

This structure for managing changes is enforced by a review cycle. When changes are made to an activity or release, they need to be reviewed and approved, before the activity or release can be marked as complete and deployed. This ensures that only users with the proper responsibilities can approve a release and deploy it.

Decision governance can be helpful, especially for larger teams, because it facilitates the decision-making process and makes changes easier to trace and review. You can have a look at what this feature looks like and how it is used in the following demo video.



To delve deeper into how decision governance works with Decision Center, see the [IBM® Governance Redbook: Governing Operational Decisions in an Enterprise Scalable Way](#).

[Governance principles](#)

The governance aspects of Decision Center are based on the states of releases and activities, and on the governance roles of participants who work on these releases and activities.

[Managing changes with the decision governance framework](#)

In Decision Center, you can manage changes to rules over time through the creation and management of branches. The recommended way of managing changes is by using the decision governance framework. With this approach, you work through releases and activities within a decision service.

Parent topic: [Managing decision services](#)

Governance principles

The governance aspects of Decision Center are based on the states of releases and activities, and on the governance roles of participants who work on these releases and activities.

State

The state of a release or activity can be one of *In Progress*, *Ready for Approval*, *Complete*, or *Canceled*. The state of a release can also be *Rejected*. The state of a release or activity dictates what work can be done and by whom.

Governance Role

All releases and activities have an *owner* and an *approver* role. Change activities also have an *author* role, and validation activities have a *tester* role. Users with administrator privileges can carry out the governance operations of all roles, and the owner of a release or activity can carry out some of the operations on behalf of participants with other roles.

When you create a release or a change activity, Decision Center takes a snapshot to record the starting state of the release or activity. A transition from one state to another can generate automatic snapshots or merges. Most notably, when all the approvers approve a change activity, Decision Center takes a snapshot to record the state of the rules, and then merges the changes back into the release.

Release governance

The user who creates a release:

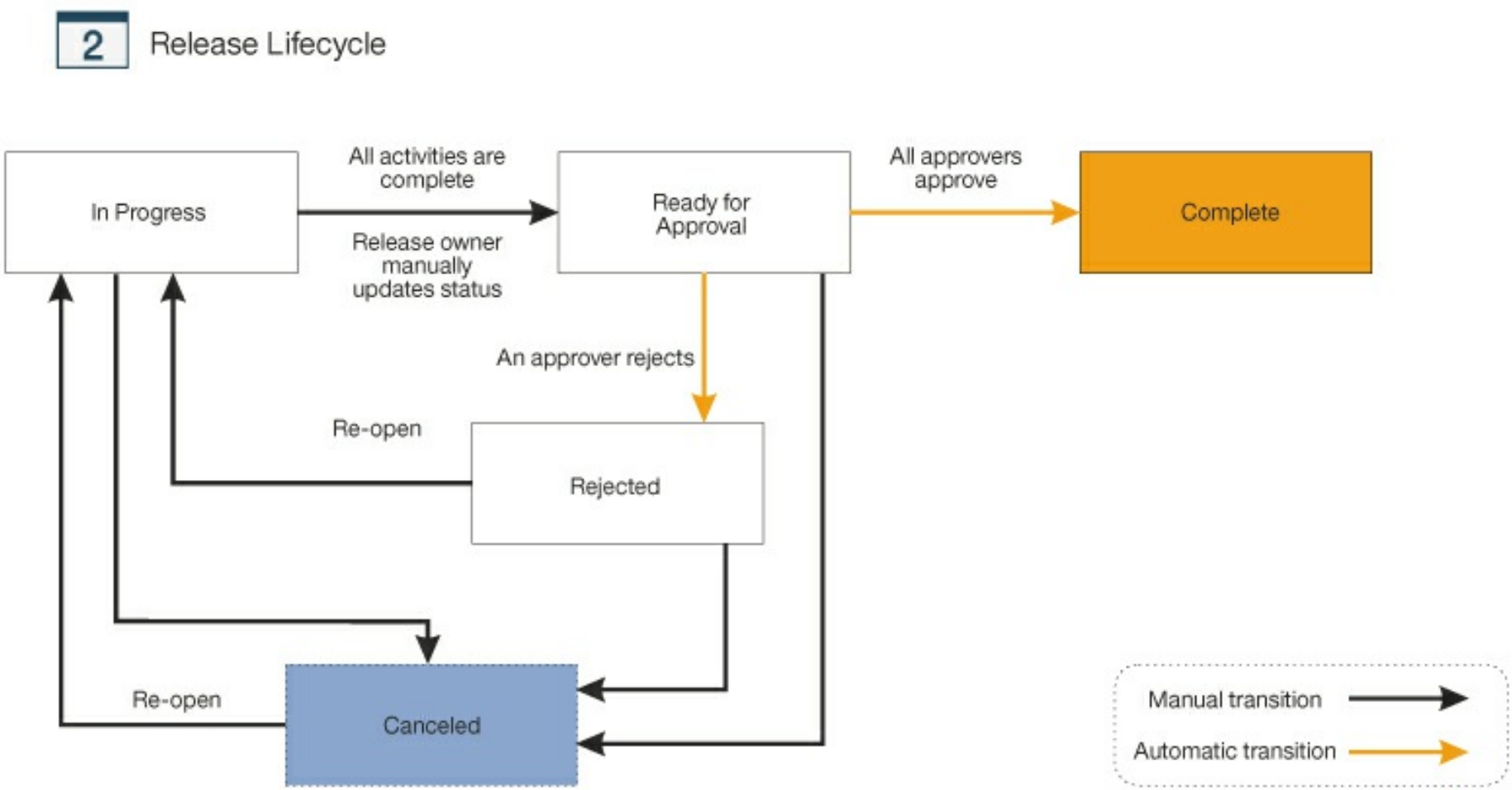
- Sets the owner of the release.
- Sets the goals of the release.
- Sets the date when the release must be completed.
- Assigns one or more participants as approver of the release.

As long as the release is in the *In Progress* state, the owner can perform the following actions:

- Change the owner of the release.
- Change the goals of the release.
- Change the due date of the release.
- Create change and validation activities.
- Delete the release

When all the activities of the release are complete, the release owner changes the state of the release to *Ready for Approval*. The approvers can then approve or reject the changes to the release.

The following diagram shows the lifecycle of a release, with manual transitions being done by the owner of the release:



The following table shows the most frequent user operations on a release, the role that is required to do the operation, and the state of the release afterward:

User operation	Role	Release state after operation
Create release	-	In Progress
Cancel	Owner	Canceled

Cancel release		Canceled
Reopen release	Owner	In Progress
Proceed to approval	Owner	Ready for Approval
Reject changes	Approver (or owner on behalf of approver)	Rejected
Approve changes	Approver (or owner on behalf of approver)	Complete
Delete a release	Owner or Permission manager <div> <p>Note: The owner can delete only releases that are not <i>Complete</i>.</p> <p>The Permission manager can delete releases in any state, except for the initial release.</p> </div>	-

Change activity governance

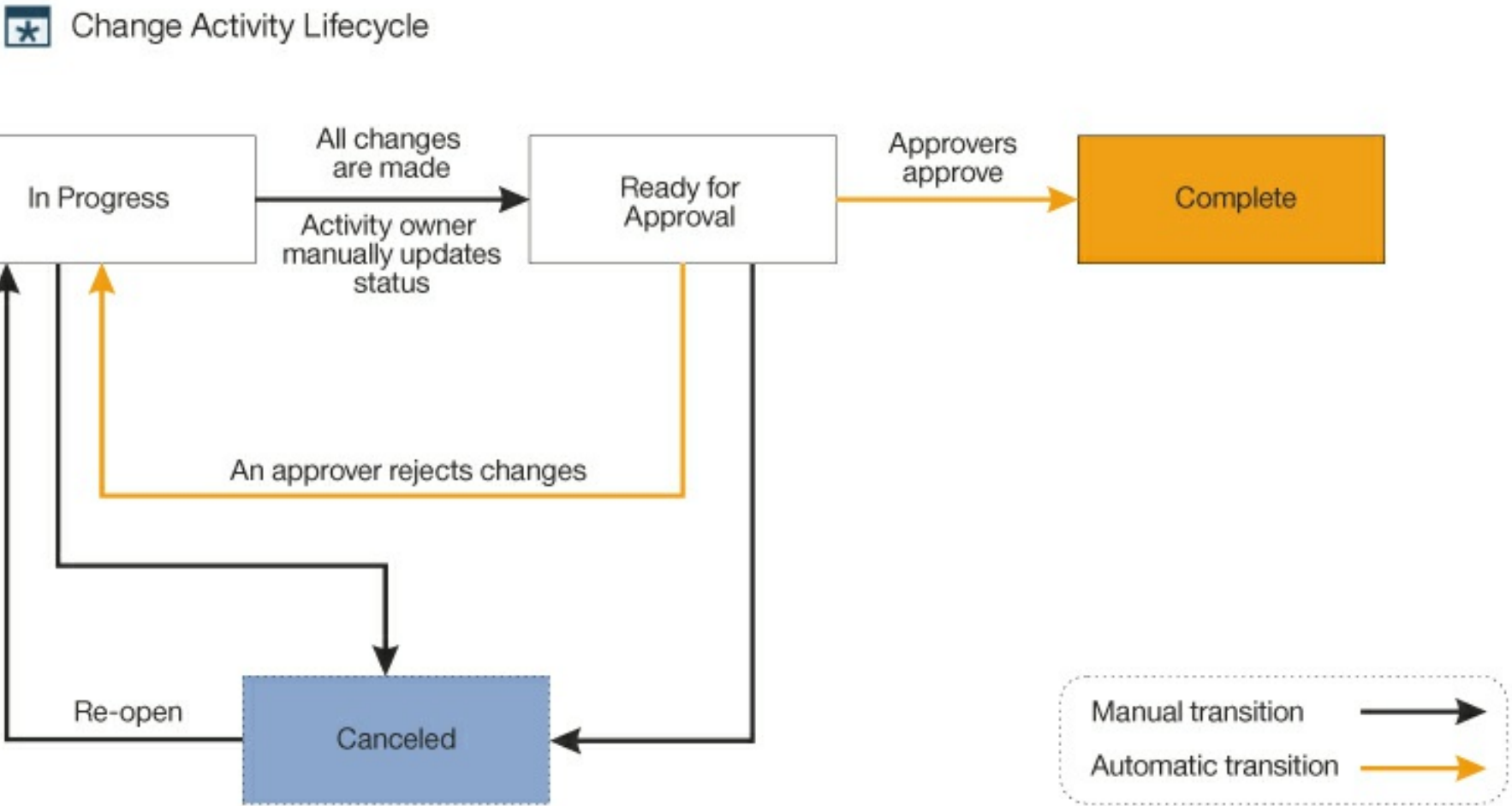
Then, as long as the change activity is not *Complete*, the owner of the activity can do the following actions:

- Change the owner of the activity.
- Change the due date of the activity.
- Change the goals of the activity.
- Assign approvers and authors to the activity.

Authors edit rules in a change activity. When they finish their work, authors change their status to **Finished**. When all the authors finish their work, the owner of the change activity acknowledges that the work is done by setting the state of the change activity to *Ready for Approval*. Then, the approvers approve or reject the changes. If there are no approvers, the change activity is automatically approved.

When all the approvers approve the change activity, Decision Center merges the changes back into the release. The change activity is then *Complete*. When a change activity is rejected, it returns to its state of *In Progress*.

The following diagram shows the lifecycle of a change activity, with manual transitions being done by the owner of the activity:



The following table shows the most frequent user operations on a change activity. It shows the role that is required to do each operation, the precondition to this operation, and the state of the change activity afterward:

User operation	Role	Precondition to the operation	State of activity after operation
Create activity	-	-	In Progress
Cancel activity	Owner	-	Canceled
Reopen activity	Owner	Activity is in Canceled state	In Progress

activity			
Proceed to approval	Owner	Authors finish work	Ready for Approval
Finish working	Author (or owner on behalf of author)	Author working	In Progress
Resume working	Author (or owner on behalf of author)	Author finishes work	In Progress
Approve changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	Complete
Reject changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	In Progress
Delete activity	Owner of the activity or the release.	Can be done only on activities that are not <i>Complete</i> or <i>Canceled</i>	-

Validation activity governance

Then, as long as the validation activity is not *Complete*, the owner of the activity can do the following actions:

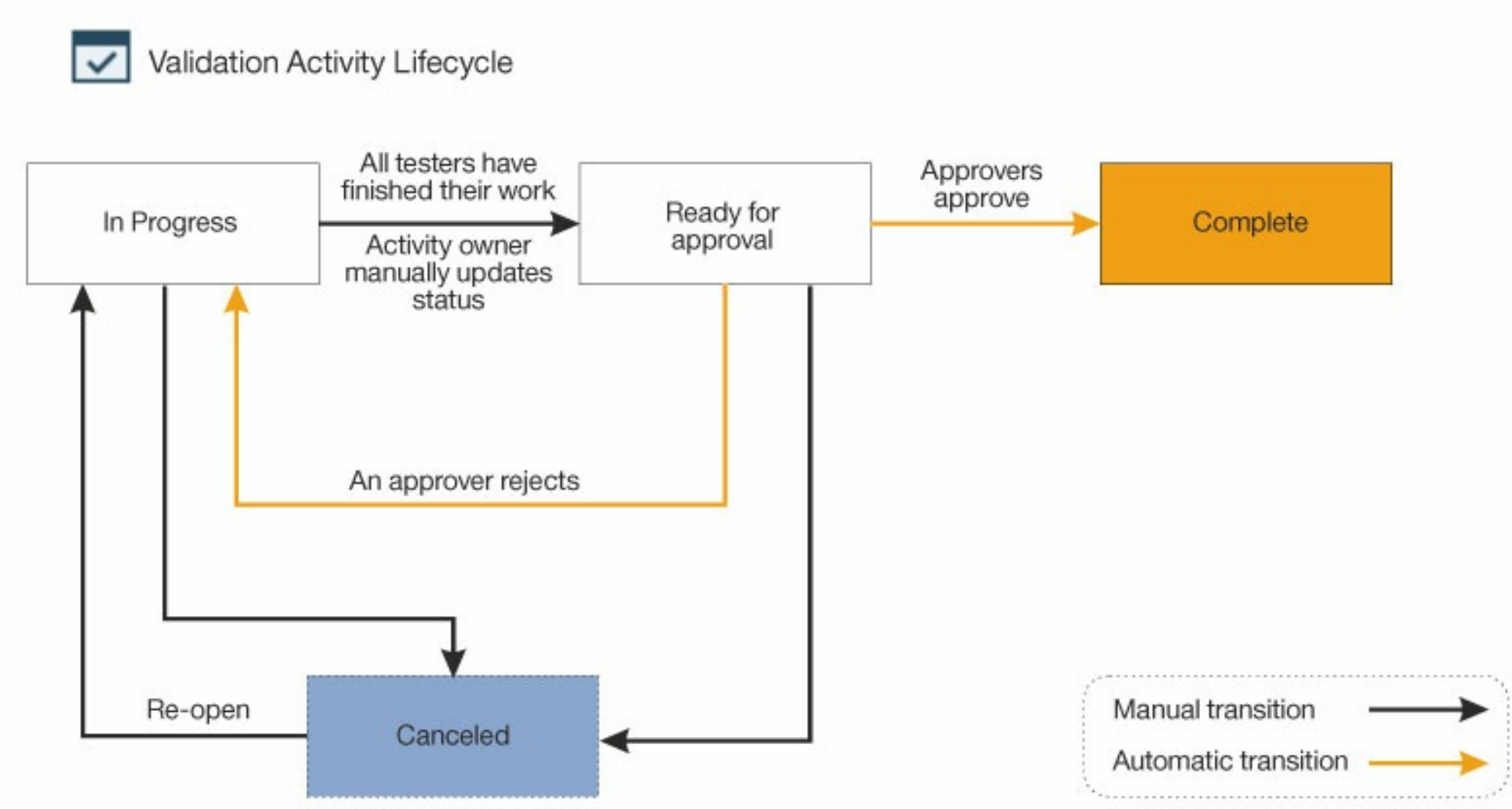
- Change the owner of the activity.
- Change the due date of the activity.
- Change the goals of the activity.
- Assign approvers and testers to the activity.

Testers run different tests that are aimed at validating a release, and note the results in the test plan. When they finish their work, and all the change activities of the release are *Complete*, testers change their status to *Finished*. When all the testers finish their work, the owner of the validation activity sets its state to *Ready for Approval*, at which point the approvers approve or reject the activity.

When all the approvers approve the validation activity, Decision Center sets the state of the validation activity to *Complete*.

Note: If a user creates a change activity in a release that contains a *Complete* validation activity, this validation activity is reopened.

The following diagram shows the lifecycle of a validation activity, with manual transitions being done by the owner of the activity:



The following table shows the most frequent user operations on a validation activity. It shows the role that is required to do each operation, the precondition to this operation, and the state of the validation activity afterward:

User operation	Role	Precondition to the operation	State of activity after operation
Create activity	-	-	In Progress
Cancel activity	Owner	-	Canceled

activity			
Reopen activity	Owner	Activity is in Canceled state	In Progress
Proceed to approval	Owner	Testers finishes work	Ready for Approval
Finish working	Tester (or owner on behalf of tester)	Tester working	In Progress
Resume working	Tester (or owner on behalf of tester)	Tester finishes work	In Progress
Approve changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	Complete
Reject changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	In Progress
Delete activity	Owner of the activity or the release.	Can be done only on activities that are not <i>Complete</i> or <i>Canceled</i>	-

Parent topic: [Using governance with decision services](#)

Related concepts:

[Getting familiar with the Business console](#)

[Decision services](#)

[Deploying from the Business console](#)

[Managing changes with the decision governance framework](#)

[Decision Center REST API](#)

Managing changes with the decision governance framework

In Decision Center, you can manage changes to rules over time through the creation and management of branches. The recommended way of managing changes is by using the decision governance framework. With this approach, you work through releases and activities within a decision service.

Releases and activities behave as branches and subbranches of your decision services, but with some special considerations.

To understand how changes are managed under this approach in the Business console, consider the following sequence, and then read the sections that follow:

1. You complete a *release*, signifying the end of that release, and then deploy its content as a *decision service*.
2. You start a new release that is based on an existing, completed release.
3. You change rules in a release by using *change activities*, in which one or more participants work.
4. When all the change activities that are related to the release are complete, you validate the release, and then complete it.

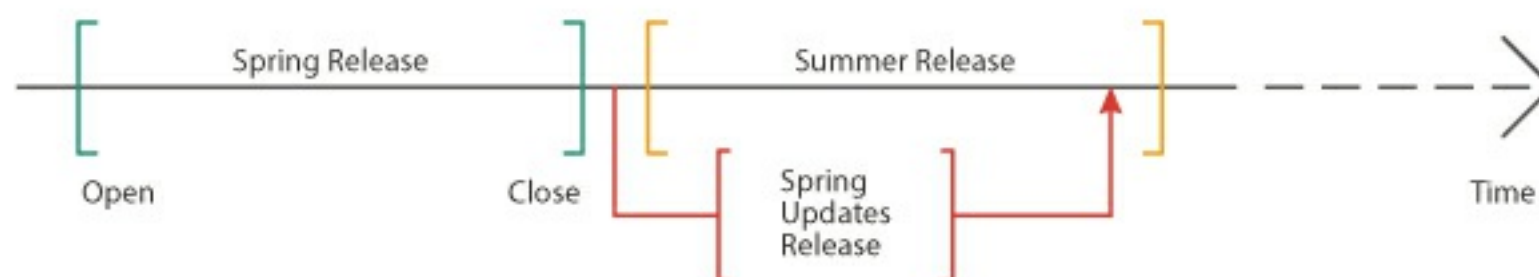
Releases

Because rules change over time, Decision Center uses *releases* to capture and trace all the changes that are related to a purpose and period in time.

The purpose is a set of business-driven goals, and the time is a beginning date and an end date. A release tracks and manages the changes that are made by its participant users and the validation of these changes (see [Governance principles](#).)

Each new release stems from an existing release. Work occurs on the release while it is open, and ends when you complete, approve, and deploy the release.

The following image shows how releases evolve over time, and that the content of a closed release can be merged into an open one:



You cannot edit a release directly. You must create and work in change activities of the release. You can make as many change activities as are needed to carry out your objectives.

Change activities

Decision Center uses *change activities* to manage the work of participants who are collaborating toward a goal, in the larger context of a release.

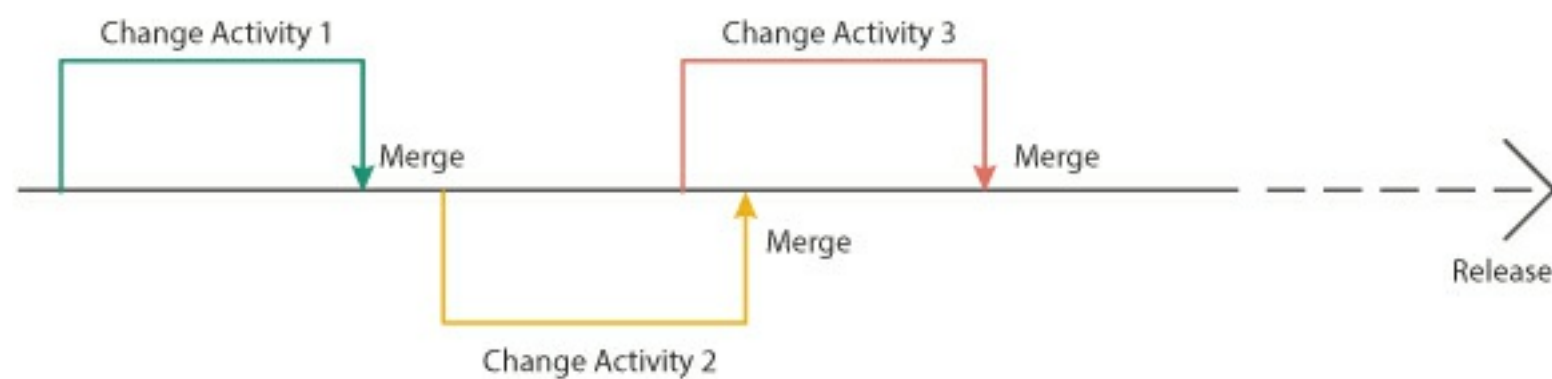
In a change activity, participants create, modify, or delete rules and get these changes approved (see [Governance principles](#).)

Each change activity stems from a release. The change activity contains the version of the rules that are found in the release at the moment when the change activity is created. When a change activity is complete, Decision Center merges it back into the release.

You can create many change activities for a release. When all the change activities are approved, a validation activity can occur on the release.

If you create a change activity before an existing one is completed and merged, the same rule might be subject to change in both activities. To avoid such conflicts, a rule that is being edited in a change activity is locked until the activity is completed and merged.

The following image shows how different change activities merge back into the release when they are completed:



Validation activities

You create and manage *validation activities* in the Business console, and use them to track and manage a validation of the content of the release. This approach can be manual tests that are entered in a test plan, automatic tests through test suites, or simulations.

It is not possible to run a test suite or simulation directly from the release. You must use a validation activity to run a test or simulation on the version of the rules that are contained in the release.

When all the validation activities are completed, the release can be approved and completed, at which point deployment can occur.

Parent topic: [Using governance with decision services](#)

Related concepts:

[Getting familiar with the Business console](#)

[Governance principles](#)

[Decision Center REST API](#)

Linked projects


When a decision service is published to Decision Center, it can contain linked (or dependent) projects.


Dependent projects are typically created when the contents of a project become too large for one project. The project is split into one or more smaller projects that are linked together. Usually, the vocabulary that is used to create rules is kept in one project, and the other projects depend on the project with the vocabulary. This arrangement facilitates maintenance of the vocabulary and the business terms that are found in the rule editors.

In the Business console, all the linked projects are displayed under the decision service.

Also, when you open a decision service, its linked projects are listed in the release properties:

ReleaseStream

Created by Paul
Nov 4, 2018

▼ Goals 

Adjustments for Scoring and Insurance

▼ Linked Projects

Loan Validation Service
Depends on: *Loan Validation Scoring, Loan Validation Determination, Loan Validation Check*

Loan Validation Check
Depends on: *Loan Validation Base*
Dependent on this project: *Loan Validation Service*

Loan Validation Determination
Depends on: *Loan Validation Base*
Dependent on this project: *Loan Validation Service*

Loan Validation Scoring
Depends on: *Loan Validation Base*
Dependent on this project: *Loan Validation Service*

Loan Validation Base
Dependent on this project: *Loan Validation Check, Loan Validation Determination, Loan Validation Scoring*

The concept of dependency among projects implies that one project depends on another and not vice versa. However, in the Business console, linked projects allow for navigation and search between dependent projects regardless of direction.

Parent topic: [Managing decision services](#)

Decision artifacts

In the **Decision Artifacts** tab, you create and edit rule artifacts, such as action rules, decision tables, or ruleflows, which you can organize in folders. You can define decision operations, ruleset variables, and handle resources for your projects.

You create business rules in two different formats: action rules and decision tables. You can create ruleflows to control the execution of rules, variable sets that you can use in the business rules of a ruleset, or decision operations to define which rules are included in a ruleset. You can also store any type of file within the Decision Center repository by uploading resources to the Business console. You find resources in the **Resources** folder, where you can create or download resources. You view and edit them offline, then refresh the existing file by re-uploading the newest version in the **Resources** folder.

Note: Some artifacts are not visible by default, and you need to activate them in the **All types** window, below the tab name.

You edit the content of a decision artifact in the editor (see [Building rules using the Intellirule editor](#) or [Editing decision tables](#)), and the properties by clicking **Details**. When you edit a business rule, the rule is locked and other users cannot edit the rule. You can copy or move any project element or folders within a project, if it is not locked by someone else and you have permission to create or modify those types of project elements. If you have permission to delete or rename project elements, you can delete or rename any project element that is not locked by someone else. To restore a deleted project, or a decision artifact, by selecting it in a timeline or snapshot, and clicking **Restore**.

Action rules

You express business policies in action rules.

Decision tables

In the Business console, you can view and edit decision tables that express sets of rules with similar conditions and actions.

Ruleflows

With ruleflows, you can manage the flow of rule execution within your ruleset. In Decision Center Business console, you create ruleflows and add different types of elements to control the execution of rules.

Decision operations

The decision operation defines which rules from a given branch are part of the ruleset. You choose which decision operation to use when creating a test suite, simulation, or deployment configuration.

Ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project, and as input and output parameters in decision operations.

Dynamic domains

A domain defines a set of values in your business terms. For example, a domain can define that the category of a customer can have one of the following values: silver, bronze, gold. You can modify these values and update your project in the Business console.

Versions and history

Decision Center creates versions of elements that you can view in a timeline and restore.

Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping. Technical rules are written using the ILOG® Rule Language (IRL). IRL is a Java™-like rule language that can be executed directly by the rule engine.

Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

Rule verification

You can perform different levels of verification on your rules.

Rule properties

Properties associate additional information with a project element, including the development status, storage location, and user group.

Tags for rule artifacts

You use tags to store data with your rule artifacts (action rules, decision tables, ruleflows).

Rule overriding

You use rule overriding to give rules precedence over other rules.

Parent topic: [Working with the Business console](#)

Action rules

You express business policies in action rules.

[How action rules work](#)

Action rules use a sentence-like structure to facilitate the creation business rules.

[Building rules using the Intellirule editor](#)

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

Parent topic: [Decision artifacts](#)

How action rules work

Action rules use a sentence-like structure to facilitate the creation business rules.

Action rules

You express business policies in rules that match actions to conditions.

Rule variables

You create a rule variable to define the scope of a rule.

Types of rule variables

You can assign different types of values to your rule variables.

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

Combinations of conditions

You can apply conditions to groups, and test nested groups.

Condition negation

You can set a rule to perform an action when a condition is not true.

Rule actions

Rule actions define what to do when the if part of the rule is true or false.

Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

Parent topic: [Action rules](#)

Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
    the credit score of 'the borrower' is less than 200
then
    in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
    set applicant to a customer
        where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

Parent topic: [How action rules work](#)

Related concepts:

[Rule variables](#)

[Rule conditions](#)

[Rule actions](#)

[Overview: Intellirule rule editor](#)

Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
    the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
    set 'the cart' to the shopping cart of customer;
if
    the value of 'the cart' is less than $100
then...
```

One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
    set Smith to a customer;
if
    the category of Smith is Gold
then
    apply 10 % discount to the shopping cart of Smith;
```

When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
()	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation

marks:

<div>R u l e d i t o r</div>	<div></div> <div>Description</div>
<div>In t e l l i r u l e e d i t o r</div>	<div>If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.</div>
<div>G u i d e d e d i t o r</div>	<div>When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.</div>

Parent topic: [How action rules work](#)

Related concepts:

[Action rules](#)

[Rule conditions](#)

[Rule actions](#)

[Types of rule variables](#)

[Dependency of rule actions on rule variables](#)

Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, customer). Once you set a variable, you can use it in any part of the rule that declares the variable.

Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and `then` parts of the rule use the same value:

```
definitions
    set maxAmount to 1000000;
if
    the amount of 'the loan' is at least maxAmount
then
    in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

Restrictions on rule variables

You can further restrict a variable in the `definitions` part of a rule by using the operator `where`.

Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
    set 'loyal customer' to a customer
        where the category of this customer is Gold;
if
    the value of the shopping cart of 'loyal customer' is more than $200
then
    apply the super discount;
```

Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
    set 'senior Gold customer' to a customer
        where all the following conditions are true:
            - the category of this customer is Gold
            - the age of this customer is at least 65;
```

Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the `definitions` part of the rule, for example:

```
definitions
    set applicant to a customer;
    set 'loyal customer' to a customer;
if
    all of the following conditions are true:
        - applicant is married to 'loyal customer'
```

```
        - 'loyal customer' is insured
then
    upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
    'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
    set 'customer 1' to a customer;
    set 'customer 2' to a customer;
if
    'customer 1' is married to 'customer 2'
    and 'customer 2' is insured
then
    upgrade 'customer 1''s rating;
```

Note: When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
    set 'gold customers' to all customers
        where the category of this customer is gold;
    set 'junior gold customer' to a customer in 'gold customers'
        where the age of this customer is at most 15;
    set 'senior gold customer' to a customer in 'gold customers'
        where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

Parent topic: [How action rules work](#)

Related concepts:

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
    the customer category is Gold
then
    redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

Parent topic: [How action rules work](#)

Related concepts:

[Action rules](#)

[Rule variables](#)

[Rule actions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
    the customer's maintenance number starts with "TX"
then
    redirect the call to call center A;
```

is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
    the purchase value of the shopping cart is more than $100
then
    apply a 10% discount to the value of the shopping cart
```

after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
    the return date of 'rented car' is after 'pickup date' and before
    'scheduled return date'
then...
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule conditions](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
  there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
  there are 10 customers where the category of each customer is Gold,
then...
```

there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
  there are at most 3 customers
    where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
  set 'silver customers' to all customers
    where the category of each customer is Silver;
  set 'silver count' to the number of 'silver customers'
if
  there are at least ('silver count' + 1) customers
    where the category of each customer is Gold,
then...
```

there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:


```
if
    there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
    set 'gold customers' to all customers
    where the category of each customer is Gold;
if
    there is at least one customer in 'gold customers'
    where the age of this customer is at least 65,
then...
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
    the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
    set 'luxury car groups' to all car groups where the daily rate of each
    car group is at least 50;
if
    'luxury car groups' contain the car group of 'the current rental
    agreement'
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Combinations of conditions](#)

[Condition negation](#)

Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting  
longer than 5 minutes
```

Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if  
    the customer is older than 60  
    or the customer is younger than 21  
    and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if  
    the customer is older than 60  
    or (the customer is younger than 21 and the category of the customer  
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if  
    (the customer is older than 60 or the customer is younger than 21)  
    and the category of the customer is Student
```

Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true`: This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true`: This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if  
    all of the following conditions are true:  
        - the category of the customer is Gold  
        - a member of the Gold team is available  
then  
    redirect the call to a member of the Gold team;
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
    any of the following conditions is true:
        - the category of the customer is Gold
        - the customer has been waiting longer than 5 minutes
then
    redirect the call to a member of the Gold team;
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Condition negation](#)

Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

it is not true that

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
    the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
    it is not true that the category of the customer is Gold
then...
```

none of the following conditions are true

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
    all the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
    none of the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
    the value of the shopping cart is more than $100
then
    apply a 15% discount on the shopping cart;
```

Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
    'the loan report' is approved
    and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
    in 'the loan report', accept the loan with the message
    "Congratulations! Your loan has been approved";
else
    in 'the loan report', refuse the loan with the message "We are sorry.
    Your loan has not been approved";
```

Parent topic: [How action rules work](#)

Related concepts:

[Action rules](#)

[Rule variables](#)

[Rule conditions](#)

[Rule actions for lists of business terms](#)

[Dependency of rule actions on rule variables](#)

Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
    for each item in expensive items:
        - apply 5% discount to this item;
        - display the message: "A 5% discount has been applied";
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule actions](#)

[Dependency of rule actions on rule variables](#)

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
    all of the following conditions are true:
        the value of the customer's shopping cart is more than $100
        the category of the customer is Gold
then
    apply a 15% discount
else
    apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
    set applicant to a customer
        where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Parent topic: [How action rules work](#)

Related concepts:

[Rule variables](#)

[Types of rule variables](#)

[Rule actions](#)

[Rule actions for lists of business terms](#)

Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

Activating and deactivating the keyword filter

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

Parent topic: [Action rules](#)

Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

When you create or open an action rule, the Intellirule Editor opens by default. If you are currently using the guided editor in a Designer, you can switch to the Intellirule editor: close the guided editor, right-click the action rule you want to edit in the Rule Explorer view, and then click **Open With > Intellirule Editor**.

You can add terms and phrases to a rule by typing or pasting in text, or by selecting them from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, value, or other phrase.

Note:

See the [Rule editor flash demo](#) on the IBM® Customer Support site for an English-only tour of the rule-editing features.

Terms and phrases

The rule editor provides an editing area for building rules. You can add terms and phrases in the following ways:

- Type directly in the editing area.
- Copy text from another editor or application, and paste it into the editing area.
- Select predefined terms and phrases from a completion menu.

Completion menu

The business vocabulary of your company defines the terms and phrases in the completion menu, and the ways you can combine them.

Placeholders

Placeholders indicate places in a term or phrase that you must complete. You must insert the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain more placeholders that you must also complete to construct a valid expression.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

[Highlighting and correcting errors](#)

Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

About this task

An action rule can comprise some or all of the following parts. The parts must be defined in the following order: definitions, if, then, and else.

Procedure

1. Click anywhere in the editing area and press Ctrl+Spacebar. The completion menu displays a list of available insertion options that are based on your current position in the rule.
2. Select an insertion option from the completion menu.
3. Press Esc or click anywhere in the editing area to hide the completion menu.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:

[Overview: Intellirule rule editor](#)

Related tasks:

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

[Highlighting and correcting errors](#)


Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Ambiguities occur when part of a rule has at least two interpretations that are semantically correct, or when there are two identical terms or phrases in the vocabulary.

For example, the following statement is ambiguous:

```
if
all of the following conditions are true :
  - the category of the customer is Gold
  - the age of the customer is at most 15
and
  the salary of the customer is more than 100
```

Rule Designer raises an ambiguity error (which appears in green ) because it cannot tell whether the salary of the customer is more than 100 is associated with the age of the customer is at most 15, or whether it is the third condition of the rule.

To remove the ambiguity, you can add a comma at the end of the the age of the customer is at most 15 condition statement:

```
if
all of the following conditions are true :
  - the category of the customer is Gold
  - the age of the customer is at most 15,
and
  the salary of the customer is more than 100
```

An alternative way to remove ambiguity is to use parentheses to group expressions.

In cases where there are two identical terms or phrases in the vocabulary, change one of the terms or phrases to remove the ambiguity.

Parent topic: [Building rules using the Intellirule editor](#)

Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

Procedure

1. In the toolbar above the editing area, click **Completion Menu Options**. A window opens.
2. Select the check boxes for the options you want to set.

Table 1. Completion menu options

Option	Description
Enable on Spacebar	<p>Select to open the completion menu whenever you press Spacebar while you are typing in the editing area.</p> <p>Clear this check box if you want the completion menu to open only when you press Ctrl+Spacebar.</p>
Enable on double-click	<p>Select to open the completion menu by double-clicking a word in the editing area.</p> <p>Clear this check box if you want double-clicking to select the current word instead.</p>
Enable auto-restart	<p>Select to automatically reopen the completion menu when a term or phrase is selected.</p> <p>Clear this check box if you want to open the completion menu by pressing Ctrl+Spacebar.</p>
Enable smart mode	<p>Select to filter the completion menu entries in a smart way to reduce the number of entries.</p>
Enable template mode	<p>Select if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable this mode if you want to insert longer phrases, and then substitute placeholders.</p> <p>Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate this mode if you prefer to build a complete rule in the same way that you might compose an email message.</p>
Use Hierarchical View	<p>Select if you want the completion menu to group terms and phrases by type, for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.</p> <p>Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.</p>
Filter unreachable phrases	<p>Select if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable.</p>
Display toolbar	<p>Select if you want the completion menu toolbar to be displayed above the list of available terms and phrases.</p>

	Clear this check box if you want to hide the completion menu toolbar.
Display documentation	<p>Select if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.</p> <p>Clear this check box if you want to hide the hover help.</p>

3. Click outside the pop-up window to save your settings.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:
[Overview: Intellirule rule editor](#)

Related tasks:
[Creating rule parts](#)
[Activating and deactivating the keyword filter](#)
[Highlighting and correcting errors](#)

Activating and deactivating the keyword filter

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

About this task

The completion menu automatically filters the list of terms and phrases as you type text to build a rule. The list becomes more restricted as you type. How the automatic filtering works depends on whether you use the keyword filter.

The keyword filter is deactivated by default. When you type with the completion menu open, the list of completion menu options shows only the terms and phrases that start with the entered text. You continue until you complete a term, or select a term or phrase from the menu.

When you activate the keyword filter, the completion menu shows a filter field above the list of options. When you type with the menu open, the filter field displays your text, and the menu displays only the terms and phrases that contain your text. The list includes all the items that contain the keyword, and not just the items that start with the keyword.

You can activate and deactivate the keyword filter as you build your rule.

Procedure

1. Click the **Toggle Keyword Filtering** button at the top of the completion menu.

The **Filter** field opens above the list of completion menu options.

2. Click the button again.

The **Filter** field closes.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:

[Overview: Intellirule rule editor](#)

Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Highlighting and correcting errors](#)

Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

About this task

Correct errors manually in rule and business model files. You can use the Problems view to view the error messages and navigate to the errors in your files.

Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
 - Hover the mouse over a highlighted error in the rule editor to display an error tip icon.
 - Double-click the text that is displayed in the error description in the Problems view to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

Parent topic: [Building rules using the Intellirule editor](#)

Related concepts:

[Overview: Intellirule rule editor](#)

Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

Decision tables

In the Business console, you can view and edit decision tables that express sets of rules with similar conditions and actions.

[How decision tables work](#)

Learn the basics for using decision tables to apply rules that have the same rule statement but different values.

[Previewing decision tables](#)

You can display and sort information about a decision table in the preview window.

[Decision table editor](#)

You use the decision table editor to edit rules in decision tables.

Parent topic: [Decision artifacts](#)

How decision tables work

Learn the basics for using decision tables to apply rules that have the same rule statement but different values.

Decision tables

A decision table groups rules that share a rule statement but have different conditions and actions.

Columns

Each column in a decision table represents a condition or an action.

Rows and cells

Each row in a decision table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

Preconditions

You can pretest data before using it with a decision table.

Parent topic: [Decision tables](#)

Decision tables

A decision table groups rules that share a rule statement but have different conditions and actions.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	0.001
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.005

To create a rule, you add a row to the decision table and enter values in the new cells. When an application calls the table, it runs the rules. If the conditions in a row are met, the rule that is formed by the row does the actions in the row. You can also define preconditions that apply to all the rules in a table, and warning tags during development show overlaps, gaps, and other errors in your rules.

In the Business console, you can use the keyboard shortcut Ctrl+F to open a search field within the decision table. The search is performed on all values of the decision table.

Parent topic: [How decision tables work](#)

Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. As shown in the following table, the top cell of each column identifies the object of a condition, or the target of an action.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	0.001
2	A	100,000	300,000	true	0.001

Each row in the table forms a rule. The rule does the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of the loan is between 100000 and 300000
then
  set the Insurance required to true
  set the Insurance rate to 0.001
```

Condition operators

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [How decision tables work](#)

Rows and cells

Each row in a decision table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:


Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

Note: If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In manual row ordering mode, you manage the grouping of conditions as you edit the table and add rows. In automatic row ordering mode, rows are automatically grouped when they share a condition value when you save and when you click **Optimize Row Order** . For more information, see [Row ordering](#).

The following table shows the rows before they are organized into partitions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	B	< 100,000		false	
3	A	100,000	300,000	true	0.001
4	B	300,000	100,000	true	0.0025

The following table shows the rows after they are organized into partitions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	
4	B	300,000	100,000	true	0.0025

In the following table, the Grade A cells of rows 1 and 2 are grouped. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1
 - if
 - all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
 - then
 - set the insurance required to true
 - set the loan rate to 0.001
- Rule 2

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row at the top of the table with the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003

The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.005
5	B	< 100,000		false	
6	B	100,000	300,000	true	0.0025
7	B	300,000	600,000	true	0.005
8	B	600,000	800,000		
9	B	≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003

3	A			1 2	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.004

Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	A	Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	A	Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```

if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 100000 and 300000
then
  - set the insurance required to true
  - set the loan rate to 0.001

```

- Rule 2

```

if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then

```

- set the insurance that is required to true
- set the loan rate to 0.002

- Rule 3

```
if
  all of the following conditions are true:
    - the loan grade is A
    - it is not true that the amount of loan is between 300000 and
600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200,000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

Parent topic: [How decision tables work](#)

Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
  set 'wealthy customer' to a customer
    where the average monthly balance of this customer
      is more than $1 000 000
if
  the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

Parent topic: [How decision tables work](#)

Previewing decision tables

You can display and sort information about a decision table in the preview window.

About this task

You can preview a decision table and sort its contents in the Business console. You can also display information about selected items.

	Grade	Amount of loan	
		Min	Max
1	A	< 100,000	
2	A	100,000	300,000
3	A	300,000	600,000
4	B	300,000	600,000

if

all of the following conditions are true :

- (the loan grade in 'the loan report' is "A")
- (the amount of 'the loan' is at least 100000 and less than 300000) ,

then

set insurance required in 'the loan report' to true ;

set the insurance rate in 'the loan report' to 0.001 ;

Procedure

The following steps take you through preview features.

1. Click your decision table in **Decision Artifacts**. A preview of the table opens. To open a different artifact, click the file in the column of decision artifacts. You can also filter the artifacts by using the **All Projects** and **All Types** buttons.
2. Select your decision table in **Decision Artifacts**, and click **Open Decision Table View**.
3. To display information about an item in a decision table, such as a column, row, or error tag:
 - a. Point at the item in the decision table. A preview window opens.
 - b. Check the preview window for information about the selected item. In the image above, the pointer is at the beginning of a row, and the preview window shows the rule that is formed by the cells in the row.
4. To view the preconditions:
 - a. Click **Preconditions** to display the preconditions. The preview shows the rule statement for the preconditions.
 - b. Click **Preconditions** again to hide the preconditions.
5. To sort or filter the contents of a column in a decision table:
 - a. Click the arrow next to the condition column header in your decision table. A window opens with sorting and filtering options.
 - b. You can select the sorting of values in ascending, or descending order.

The behavior of the sorting depends on the row ordering mode. When you preview a table for which no row ordering mode is explicitly set, the default view is in **Manual** mode, and sorting restrictions apply. For example, you cannot sort an action column. If you edit the table in **Automatic** mode and save it, then you can view it in **Automatic** mode.

- c. To filter values in a column, apply your filter in the section **Filter**. You can filter by value from an enumeration, by text, or use comparison operators and intervals for columns with numbers and dates. Filtering is semantic, which means that it takes into account all cells whose expressions include the value of your filtering. For example, if you filter with a value of > 100, results would include a cell that displays an interval from 0 to 200.
- d. You can clear the filter for each column in the **Sorting/Filter** window, or press **Backspace** to return to the original layout of the table.

Parent topic: [Decision tables](#)

Related concepts:
[Decision table editor](#)

Decision table editor

You use the decision table editor to edit rules in decision tables.

In the Business console, you can define conditions and actions, and enter values directly into table cells to form business rules. You can also use preconditions to test incoming data before the table runs a rule, and define variables for the table.

The decision table editor supports both manual and automatic row ordering. You can set the order of the rows in your decision tables, or have the decision table editor set the order for you and automatically group identical conditions.

If you want to protect your decision tables against editing by others, you can apply different types of locking mechanisms in Rule Designer (see [Using decision table locking facilities](#)). When you publish your project to Decision Center, the locks are available in the Business console.

Note: You can only create and edit decision tables locks in Rule Designer.

[Columns](#)

In the Business console, you can change the columns that define the conditions and actions for business rules in decision tables.

[Rows](#)

In the Business console, you can change the rows that form the rules in decision tables.

[Preconditions](#)

In the Business console, you can use preconditions to check incoming data, and define variables for decision tables.

[Table cells](#)

In the Business console, you can update table cells by entering changes directly or by using a rule editor.

[Errors and warnings](#)

In the Business console, you can use automatic cell checking to highlight problems in decision tables.

[Row ordering](#)

In the Business console, you can select either automatic or manual row ordering to organize the rows in decision tables.

[Transferring data to and from Excel](#)

You can export decision tables as Excel files to work offline on their content, and import these Excel files back to the Business console with your changes. You can also manually transfer data such as rows, columns, and cells between Excel files and decision tables.

Parent topic: [Decision tables](#)

Related tasks:

[Previewing decision tables](#)

Columns

In the Business console, you can change the columns that define the conditions and actions for business rules in decision tables.

A decision table contains a group of similar rules that use different conditions and actions. Each row in a table forms a rule, and each column in a decision table represents a condition or an action. With the decision table editor, you can add or remove columns through the column menu, and change the constraints that are applied by each column. When you add or remove a column, you change the rule statement that is used by all the rules in the table.

Defining a column

To define the condition or action of a column, select the column by clicking the header of the column, or selecting a cell in the column and pressing Ctrl+Space. To edit the condition or action, open the contextual menu by right-clicking the column.

You can change the condition or action expression of a column whose associated cells are already filled. The editor will try to apply the existing values in the cell to the new expression. When the number of parameters differ, the editor retains only the values that can be applied. For example, if you change the expression `score is between <min and <max> to score equals <number>`, the `<min>` value becomes the equal value. If the number of parameters increases, the editor keeps the values, but you have to complete each cell so that a complete condition or action is specified.

Editing column titles and subtitles

To edit a column title or subtitle, click the part of the title that you want to change to open an input field. You can also open the input field by selecting a column and pressing Enter. The text that you enter in the title or subtitle does not change the rule statement of the column.

Sorting and filtering columns

When you click the arrow next to the column title, a window opens with sorting and filtering options.

You can sort the rows of a column in ascending or descending order. The row sorting also depends on the row ordering mode selected for the decision table (see [Row ordering](#)). When you preview a table for which no row ordering mode is explicitly set, the default view is in **Manual** mode, and sorting restrictions apply. For example, you cannot sort an action column. If you edit the table in **Automatic** mode and save it, then you can view it in **Automatic** mode.

You can apply filtering to reduce the number of rows displayed in the table. You can filter by value from an enumeration, by text, or use comparison operators and intervals for columns with numbers and dates. Filtering is semantic, which means that it takes into account all cells whose expressions include the value of your filtering. For example, if you filter with a value of `> 100`, results would include a cell that displays an interval from 0 to 200.

In the decision table editor, you can use the **Undo** and **Redo** buttons to go quickly from a view of your filtered subset of data to your previous, unfiltered table, and vice versa.

Resizing automatically

You can set the display of a decision table to resize automatically when you change the size of your browser window. In **Details**, select **Auto-resize columns**.

Column menu

To open the column menu, right-click the top cell of the column that you want to change.

Table 1. Table of column menu commands

Co mm and	Description
Def ine Col um n	Opens a rule editor to change the condition of the column. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at bottom of the editing area.
For mat col um n	Opens an editor to set the formatting options for displaying values in a column (dates, numbers, and so on). You can select the format from the drop-down list options, and optionally edit the # symbols to personalize the format.
Che ck Ga n	Looks for gaps between values in the cells of condition columns.

r	
Check Overlap	Looks for overlapping values in the cells of condition columns.
Cut	Cuts a selected column. You can paste the column to another location.
Copy	Copies a selected column. You can paste the copy of the column to another location.
Paste	Pastes a copied or cut column to a selected location.
Insert Column	Inserts a condition or action column, depending on the current selection.
Delete	Removes the selected column and its part of the rule statement in the decision table. Deleting a column can disable a decision table.
Clear	Deletes the contents of the cells in the column.

Parent topic: [Decision table editor](#)

Rows


In the Business console, you can change the rows that form the rules in decision tables.

The rows contain the values that complete the rule statement that is defined by the condition and action columns in a table. Each row constitutes a complete rule.

Editing a row

To view the rule that is formed by a row, hover your mouse over the first cell in the row, and a window displays the rule. To edit a row value, type your changes in the cell or right-click the cell to open a menu of options.

Optimizing

When you use automatic row ordering, click **Optimize Row Order**  to group the rows that have the same conditions. The button is activated by any change to a condition in the decision table.

Row menu

To use the row menu, right-click the first cell in the row that you want to change.

Table 1. Table of row menu commands

Command	Description
Cut	Cuts a selected row. You can paste the row to another location.
Copy	Copies a selected row. You can paste the copy of the row to a selected location. This command does not work with manual row ordering.
Paste	Pastes a cut or copied row to a selected location. This command does not work with manual row ordering.
Insert copied rows	Inserts a group of copied rows. The command inserts the rows above the selected row.
Insert row	Inserts a new row above or below the selected row. Alternatively, hover your mouse over the border between the selected row and the row above or below, and click +.
Delete	Deletes the selected row.
Clear	Deletes the contents of the cells in the selected row.

Parent topic: [Decision table editor](#)

Preconditions

In the Business console, you can use preconditions to check incoming data, and define variables for decision tables.

You use preconditions to check incoming data to determine whether a decision table can process the information. You can also use the preconditions to define one or more variables for the decision table. The variables are used in only the decision table.

Setting preconditions

To change the preconditions, click **Preconditions**. One of the following dialogs opens:

- **Define Preconditions:** Opens if the table is new, or the table does not have a precondition.
- **Update Preconditions:** Opens if the table has a precondition.

Use the rule editor to create or modify the preconditions, and click **Define** or **Update** to apply your changes.

Parent topic: [Decision table editor](#)

Table cells

In the Business console, you can update table cells by entering changes directly or by using a rule editor.

The cells in the rows that form the rules in a decision table contain the values that complete the conditions and actions. The columns define the conditions and actions, which determine what you can enter in the cells.

Editing a cell

To edit a cell in a decision table, type your changes directly in the cell. You can use common keyboard commands such as Ctrl+C for copy.

Replacing Boolean values with check boxes

You can replace the Boolean values with check boxes, which can be selected or cleared to indicate actions. In **Details**, select **Use check boxes for Boolean values**.

Cell menu

To open the cell menu, right-click the cell that you want to edit. The cell menu provides the following commands, which vary between the condition and action cells.

Table 1. Table of cell editing commands

Com man d	Description
Cut	Copies and deletes the contents of the cell. The contents can then be pasted to another cell.
Copy	Copies the contents of a cell.
Past e	Pastes the copied contents of one cell to another cell.
Inser t copie d cells	Inserts copied cells and their values.
Inser t row	Inserts a row above or below the row of the selected cell.
Clear	Deletes the contents of the cell.
Chan ge oper ator	Inserts an operator to define the range of values. Condition columns only.
Disa ble/E nabl e	Disables or enables an action cell. A rule ignores the action of a disabled cell. Action columns only.
Set to other wise	Sets the value of the cell to 0therwise. Condition columns only.
Edit custo m value	Displays the contents of the cell in a rule editor. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at the bottom of the editing area.

Operators

When you edit a cell, the console can provide a list of operators based on the contents of the cell. To open the list of operators, right-click the cell and select **Change operator**. Operators are provided for numbers, strings, dates, and domains.

Table 2. Number operators

Operat or	Description
=	The input data equals the number in the cell.
≠	The input data does not equal the number in the cell.
in	The input data is in the numbers in the cell.
!in	The input data is not in the numbers in the cell.
<	The input data is less than the number in the cell.
<=	The input data is less than or equal to the number in the cell

<	The input data is less than or equal to the number in the cell.
>	The input data is more than the number in the cell.
≥	The input data is more than or equal to the number in the cell.
[..]	The input data is between the two numbers, or it is equal to one of the numbers.
]..]	The input data is more than the first number and less than or equal to the second number.
[..[The input data is more than or equal to the first number and less than the second number.
]..[The input data is more than the first number and less than the second number.

Table 3. String operators

Operator	Description
=	The input data is the same as the string in the cell.
≠	The input data is not the same as the string in the cell.
in	The input data is in the strings in the cell.
!in	The input data is not in the strings in the cell.
is empty	The input data is an empty string.
is not empty	The input data is not an empty string.
contains	The input data contains the string in the cell.
!contains	The input data does not contain the string in the cell.
starts with	The input data starts with the string in the cell.
!starts with	The input data does not start with the string in the cell.
ends with	The input data ends with the string in the cell.
!ends with	The input data does not end with the string in the cell.

Table 4. Date operators

Operator	Description
=	The input data is the same as the date in the cell.
≠	The input data is not the same as the date in the cell.
in	The input data is in the dates in the cell.
!in	The input data is not in the dates in the cell.
<	The input data is before the date in the cell.
≤	The input data is before or the same as the date in the cell.
>	The input data is after the date in the cell.
≥	The input data is after or the same as the date in the cell.
[..]	The input data is between the two dates, or it is same as one of the dates.
]..]	The input data is after the first date and before or the same as the second date.
[..[The input data is after or the same as the first date, and before the second date.
]..[The input data is after the first date and before the second date.

Table 5. Domain operators

Operator	Description
=	The input data is the same as the domain value in the cell.
≠	The input data is not the same as the domain value in the cell.
in	The input data is in the domain values in the cell.
!in	The input data is not in the domain values in the cell.

Parent topic: [Decision table editor](#)

Errors and warnings

In the Business console, you can use automatic cell checking to highlight problems in decision tables.

The decision table editor tags inconsistent content, overlaps, and gaps with triangles. The console automatically checks the contents of a decision table when you open the table. It displays error and warning tags in preview and the decision table editor.

To view the cause of an error or warning, hover your mouse over the tagged cell or the header of the column that contains the cell. A message box opens with a description of the problem. When you hover over the column header, the message box shows the rule statement for the column along with a list of any problems in the column.

Tip: You can open the error list by selecting a tagged cell or header, and pressing F9.

You can turn off the cell checking. In the decision table editor, open **Details** and clear the check box for the type of checking that you want to stop. You can completely stop the checking by clearing the boxes for both **Check gaps** and **Check overlaps**.

Errors

Errors prevent a decision table from working. The following list provides common examples of errors:

Unexpected value

Entering the wrong type of value, for example, a word where a number is expected.

Mixed characters

Entering an unexpected character, for example, adding a letter to a number.

Incorrect value

Entering a word or number that does not match any of the values that are defined for a parameter.

Wrong operator

Entering an operator that places a value out of scope for the rule statement.

When an error occurs, the console tags it as shown in the following illustration:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 10000A		false	
2	A	<div>Error The word 'I' is missing.</div>		true	0.001
3	A			true	0.003

Warnings for overlaps and gaps

The console warns you when overlaps or gaps occur between rows with shared conditions. These anomalies do not stop a decision table from working. However, they might produce incorrect results. Typically, ranges in tables are contiguous, and do not overlap.

Warnings are given for the following reasons:

Overlaps

When two cells in a column contain ranges that include the same values. In the example, the ranges [200,000; 300,000] and [200,000; 600,000] both include the numbers 200,000 to 300,000.

Gaps

When two cells in a column contain ranges that do not include values between the ranges. In the example, the ranges < 100,000 and [200,000; 600,000] do not cover the numbers between 100,000 and 200,000.

When overlaps and gaps occur in a table, the console tags them as shown in the following illustration:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<div> <div>the amount of 'the loan' is at least $\langle min \rangle$ and less than $\langle max \rangle$</div> <div> Errors Lines 5 to 8 have gaps Line 6 overlaps with line(s) 7 Line 7 overlaps with line(s) 6 </div> </div>			
2	A				
3	A				
4	A				
5	B	< 100,000		false	⊗
6	B	200,000	300,000	true	0.002
7	B	200,000	600,000	true	0.005
8	B	≥ 600,000		true	0.008

You can select overlap and gap checking for an individual column or an entire table.

For an individual column:

- 1. Right-click the header of the column.
- 2. Select or clear **Check Gap** or **Check Overlap**.


For an entire table:

- 1. Click **Details**.
- 2. Select or clear **Check Gap** or **Check Overlap**.

Updating warnings in a decision table

The decision table rule editor provides manual and automatic row ordering. How you update the warnings varies with the different row ordering modes.

Automatic row ordering

Click **Optimize Row Order**  to update the gap and overlap warnings. The warnings are displayed when the row order and partitions are updated. Any change in a partition resets the errors, for example, a change in the value of a condition cell.

Manual row ordering

All the editing commands update the errors. For example, if you change the value of a cell and press Enter, the editor checks the table for errors and updates the warnings.

Displaying errors and warnings together

If a table contains errors and warnings, both are tagged in the table. However, because errors are more severe than warnings, the table displays an error marker in the header of any column that contains both errors and warnings.

The following table shows a column with errors and warnings:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<div> <div>the amount of 'the loan' is at least $\langle min \rangle$ and less than $\langle max \rangle$</div> <div> Error Line 5 : The word '/' is missing. Line 6 overlaps with line(s) 7 Line 7 overlaps with line(s) 6 </div> </div>			
2	A				
3	A				
4	A				
5	B	< 10000A		false	⊗
6	B	200,000	300,000	true	0.002
7	B	200,000	600,000	true	0.005

Parent topic: [Decision table editor](#)

Row ordering

In the Business console, you can select either automatic or manual row ordering to organize the rows in decision tables.

Automatic row ordering groups rows that have the same condition values. You can sort any column, and you are not constrained by the partition structure. You can edit your decision table as a flat spreadsheet, and let the decision table editor combine conditions for you.

Automatic row ordering is not appropriate for a table that must have its rows in a specific order. For this type of table, you must organize the rows manually.

When you open a decision table for the first time, you are asked to select the type of row ordering. Automatic row ordering is selected by default because it is appropriate for most use cases. Manual row ordering is used in advanced cases where the order of the rows must be controlled.

You can also set the type of row ordering through **Details** in the decision table editor. You must save and reopen the decision table to implement the property change.

Automatic row ordering

Automatic row ordering organizes a decision table by grouping rows that share condition values.

Table before optimization:


	Grade	Amount of loan		Insurance	Insurance
		Min	Max		
1	A	< 100,000		false	⊗
2	B	< 100,000		false	⊗
3	A	100,000	300,000	true	0.001
4	B	100,000	300,000	true	0,0025

Table after optimization:

	Grade	Amount of loan		Insurance	Insurance
		Min	Max		
1	A	< 100,000		false	⊗
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	⊗
4	B	100,000	300,000	true	0,0025

You can add, move, and remove rows when a table is set to automatic row ordering. You can apply automatic row ordering in two ways:

Optimize Row Order button

Optimize Row Order  reorganizes the rows during an editing session. The button is activated by changes to the contents of the table.

Save

The editor automatically reorganizes the rows into groups when you save the table.

Automatic row ordering lists the groups alphabetically or numerically depending on the shared values. You can use the sort buttons to change the order of the rows. When you sort a column that does not contain shared conditions, all the rows in the table are reorganized by the values in the sorted column.

Manual row ordering

Manual row ordering organizes rows into partitioned groups. As the following example shows, a partitioned group has a shared condition value:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	
4	B	100,000	300,000	true	0.002

Rows 1 and 2 contain a partitioned group that shares Grade A, and rows 3 and 4 contain a partitioned group that shares Grade B. With manual row ordering, you work with rows in groups. You manually add rows to groups, and set the order of the groups. If you change the value of a cell in a group, you update all the other cells in the group. You cannot copy and paste rows, but you can copy and paste cells.

With manual row ordering, you can group or split rows by selecting cells, right-clicking them, and selecting **Group** or **Split**.

You can use the sort buttons to sort condition columns when you use manual row ordering, but you cannot sort action columns. When you sort a column, the grouped rows are sorted within their groups. However, independent rows can be redistributed throughout a table. Sorting does not keep split rows together. You can regroup split rows by sorting the column that contains their shared condition value.

The following images show the sorting patterns of the different row ordering modes.

Before sorting:

Amount of loan	
Min	Max
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	

Sorting with automatic row ordering:

Amount of loan	
Min	Max
< 100,000	
< 100,000	
< 100,000	
100,000	300,000
100,000	300,000
100,000	300,000
300,000	600,000
300,000	600,000
300,000	600,000
≥ 600,000	
≥ 600,000	
≥ 600,000	

Sorting with manual row ordering:

Amount of loan	
Min	Max
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	

Parent topic: [Decision table editor](#)

Transferring data to and from Excel

You can export decision tables as Excel files to work offline on their content, and import these Excel files back to the Business console with your changes. You can also manually transfer data such as rows, columns, and cells between Excel files and decision tables.

Export to Excel

To export an entire decision table from the Business console to Excel:

1. Select your decision table in the **Decision artifact** tab.
2. Click **Download this decision table as an Excel file**.

You can also click **Download as Excel** in the **Decision Table** view.

You can then share your decision table, as an Excel spreadsheet, for reviewing or reporting purposes.

If you have several decision tables in a project, you can download them as a single Excel spreadsheet. In the **Decision artifact** tab, select the check boxes next to your decision tables, then click **Download the selected decision tables...** in the toolbar. Your tables are exported in one file, with a table per sheet.

When you generate the Excel file, condition cells are merged. If you primarily want to generate the file to edit it and then paste the modifications into the Business console, you should deactivate condition cell merging. Set the `decisioncenter.web.dt.excelExport.mergeConditionCells` preference to **false**.

Import from Excel

To import your Excel file back in the Business console:

1. Open the decision table editor.
2. Click **Import Excel File**.
3. Select your Excel spreadsheet.
4. Click **Import**.

You cannot change the structure of the decision table before you import it in the Business console. For example, you cannot add columns, which would introduce a new condition or action, or edit definitions comments in column headers. You can modify only values, or add and remove rows. Importing a table with an altered structure might cause errors or unexpected behavior. You can always undo unwanted changes with the **Undo** button.

Formatting in Excel

If you are making modifications in Excel to re-import the file back into the Business console, you must write operators manually and without Excel formatting, for example:

- Operators on numbers and dates, such as:

```
[10 | 20]
>50
[01/01/2019 | 02/05/2019[
```

- Operators on strings, such as contains, starts with, ends with
- Operators on sets, such as in X, Y, !in X, Y
- Operators without parameters, such as is null, is not null, is empty, is not empty

Dates and number formatting is not supported, only values can be exported to Excel and imported into the Business console. For example, if your date value is MM/DD/YYYY and is formatted to display as MM/DD/YY, when you export or import your decision table, it displays the original value MM/DD/YYYY.

Calls to methods defined in your BOM file are supported when you export and import the decision table. The generated Excel file displays the full expression, which is underlined. When you import the file back into the Business console, underlined text is interpreted as a call to a BOM method.

Transfer data manually

You can also transfer data between Excel and decision tables by copying and pasting the elements. When copying from Excel to the Business console, the content that you select must match the content of the decision table. If the contents of a transfer does not match the contents of the decision table, the decision table editor can block the transfer or show an error.

Moving information between the two environments can alter data. You can transfer only values and not data type information. For example, the Boolean values `true` and `false` are lowercase in decision tables. However, Excel automatically displays Boolean values in uppercase: `TRUE` and `FALSE`. In this case, you must set Excel to treat the Boolean values as text to retain their lowercase format for their transfer back to the decision table.

How you transfer information depends on its role and formatting. The contents of cells can be transferred between cells, while a complete row or column must be transferred as a group of cells.

Note:

- The menu commands do not work in all browsers. You must use the keyboard shortcuts in some browsers. Also, when pasting from an external source, specifically from Excel, make sure that there is no region in the table that is marked as the paste source, which is indicated with an orange rectangle. The paste operation always selects an available source within the table first, and then checks for data in the clipboard.
- If you copy and paste content from a decision table in the Business console to Excel, you might see hyphens in cells, which represent grouped cells in the decision table. In this context, a hyphen in a cell means the cells above it must be grouped in the decision table. If you need to insert an actual hyphen, use quotation marks around it: " - "

Parent topic: [Decision table editor](#)

Ruleflows

With ruleflows, you can manage the flow of rule execution within your ruleset. In Decision Center Business console, you create ruleflows and add different types of elements to control the execution of rules.

Rules apply decisions, but they do not have a defined sequence or succession. A ruleflow organizes rules into a sequence of decisions by assembling the rules into a group of rule tasks that uses a set execution pattern.

Each rule task is evaluated to produce a result, or decision. All these results and decisions are combined to produce a single business decision, which is represented by a ruleflow. Ruleflows also specify the transitions between rule tasks. The transitions determine how, when, and under what conditions to use each rule task.

The following diagram shows the levels of refinement for selecting the rules.



The evaluation during ruleset execution selects rules in the following manner:

1. Ruleflow scope selection

Each rule task in a ruleflow has a scope that is defined by a list of rule packages and individual rules. At run time, ruleflow scope selection generates a list of the rule packages and rules that you defined for each task. When a task is run, the rule engine considers only the rules within this scope.

2. Runtime rule selection

The runtime rule selection process further determines which rules the rule engine must consider. Filtering is typically based on the value of rule properties and execution parameters.

3. Rule overriding

After the other selection mechanisms are executed, certain rules that you defined beforehand override other rules. The overridden rules are filtered out of the selection.

The rule engine evaluates the conditions of the selected rules, and runs their rule actions on the matching objects.

Overview: Ruleflows

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

Creating a ruleflow

In the Business console, you can create, and add elements to a ruleflow. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

Configuring ruleflow properties

To display the properties of ruleflow elements, select each one, and a window with properties opens. When you add a node or a task to your ruleflow, you must set their properties.

Parent topic: [Decision artifacts](#)

Overview: Ruleflows

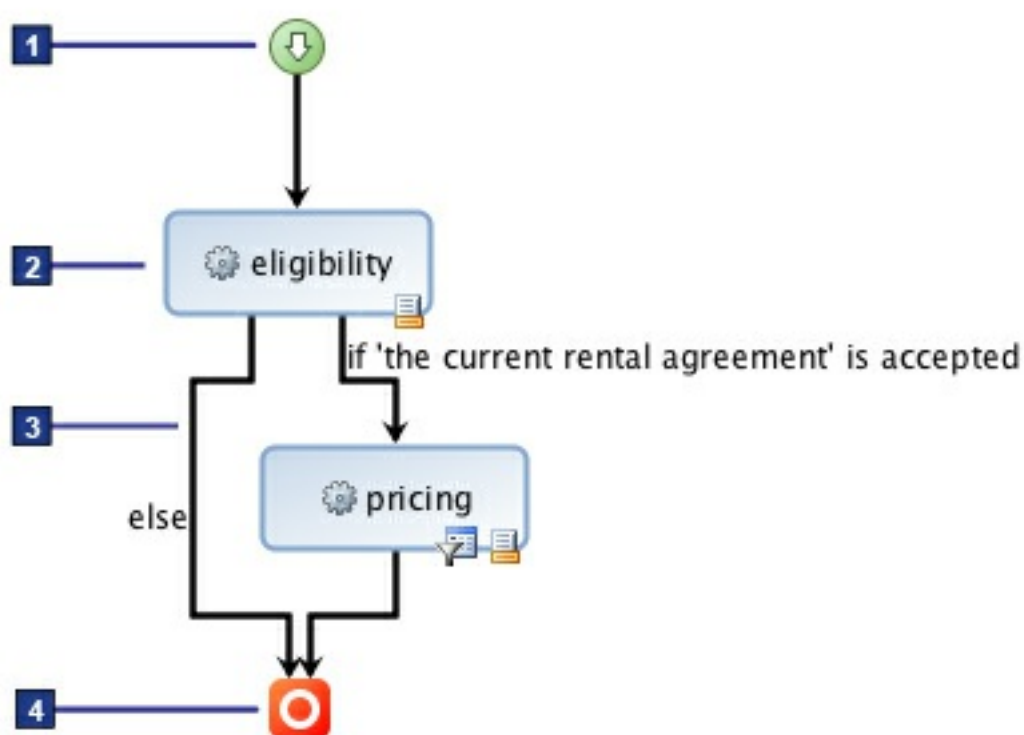
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



Note: The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter


statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

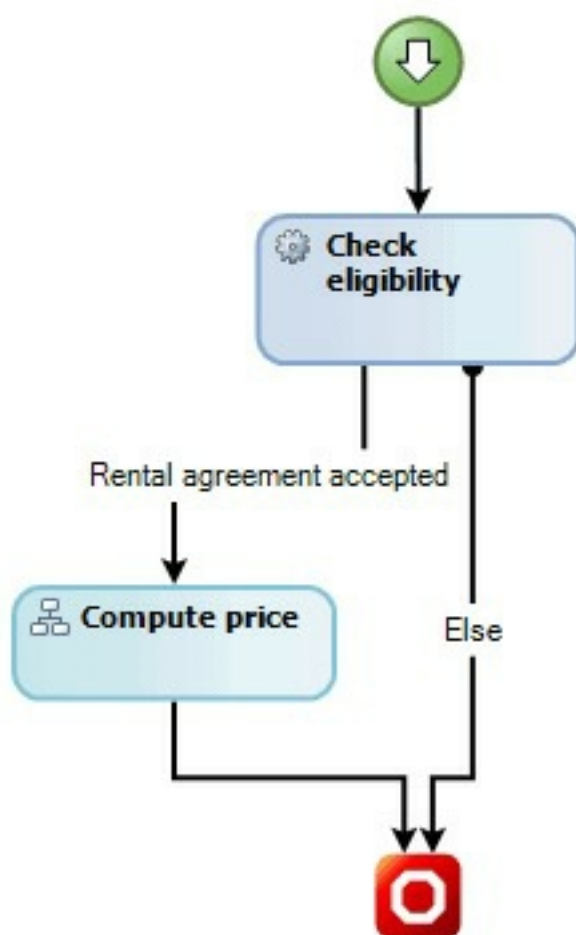
You can specify initial and final actions on subflow tasks.

Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.




A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.


Branches

A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch.

Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

Parent topic: [Ruleflows](#)

Creating a ruleflow

In the Business console, you can create, and add elements to a ruleflow. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

About this task

You can add a ruleflow to a project, or any of its packages. After adding a ruleflow to a rule project or package, you define the structure by adding the elements that you need in the ruleflow editor. To understand the use of each element, see [Ruleflows](#).

Procedure

1. In the **Decision Artifacts** tab, click **Create an artifact > New Ruleflow** to add a ruleflow in the project of your choice.

In the ruleflow editor, a start node and an end node are added by default.

2. Add elements to your ruleflow with the **+** button along the transitions to add a node between the elements, or under an element to create a node connected to this element. If you drag the new node to an existing node, a transition is created between them.

Note: You are notified of errors, or elements that are incorrectly positioned, by an icon displayed inside the node.

3. Create rule tasks, and add rules to be executed at this point in the ruleflow.
 - a. Add a rule task element in the editor, and click the rule task to open a window where you set its properties.
 - b. Next to **Uses**, click **Edit** to add rules and packages to a rule task. You can use the arrow icons to order the rules and packages. Depending on the rule execution properties of the task, this order might impact the output of ruleflow execution.
 - c. Optional: For advanced users, configure the execution of rule tasks at run time:
 - Select the rule engine algorithm, ordering, and exit criteria. For more information, see [Execution properties for rule tasks](#).
 - Use the option **Selects rules where** to select rules to be executed at run time, and **Rule selection** to define whether the runtime rule selection is static or dynamic. For more information, see [Runtime rule selection](#).

If you need to delete an element, select the node or transition that you want to delete, and press **Delete** or **Backspace** on your keyboard.

4. Optional: If you need to execute rule action statements, add action tasks in your ruleflow. Then, set the action statement, initial action, and final action.
5. Configure the properties of transitions between the tasks.
 - a. Select the transition, and provide a name for the condition in the **Label** field.
 - b. Next to **Conditions**, click **Edit**.
 - c. Select **BAL**, or **IRL**, and type a condition statement. For example, in BAL:

'the current rental agreement' is accepted

If you write the condition using IRL, make sure the text fields contain a valid Boolean expression.

In transition conditions, the variable scope is restricted to ruleset parameters and variables. It does not include access to the working memory.

6. Optional: You can create multiple, parallel paths in your ruleflow, if you need to execute rules simultaneously. For example, if you are checking the eligibility of a customer for a loan, you might want to check whether the customer meets the criteria for the loan, and also if the amount requested is valid. To do so, you use forks and joins in your ruleflow.
 - a. Add a fork node where you want your ruleflow to execute several rules in parallel. You can then add your rule tasks in the ruleflow.
 - b. Add a join where you want to combine the transitions created from the fork.

The transitions from a fork node to a join node must not have conditions, because the ruleflow follows all paths in parallel between the fork and the join.

7. Optional: You can add branches to the ruleflow to organize conditional transitions, in the same way that you could start several conditional transitions from a task.

Important:

When multiple transitions originating from a branch or a task define overlapping conditions, the path

taken to execute the ruleflow is unpredictable. Make sure the conditions you define for multiple transitions do not overlap.

- a. Add a branch node where you want your ruleflow to organize the different conditions.
 - b. To name the branch, click the branch node, and enter the name in the **Label** field.
 - c. Add transition conditions for each transition from the branch. One of the transitions must be an Else transition.
8. Optional: If you need to execute another ruleflow at some point in your main ruleflow, add a subflow task, and click it to open the properties window. Select the ruleflow you want to execute in the **Subflow** drop-down list.
 9. To align the ruleflow automatically, click **Arrange all** in the toolbar.
 10. Save the ruleflow.

Parent topic: [Ruleflows](#)

Configuring ruleflow properties

To display the properties of ruleflow elements, select each one, and a window with properties opens. When you add a node or a task to your ruleflow, you must set their properties.

Some properties are common to several ruleflow elements, and are described in the section [Common properties](#). Properties that are specific to other ruleflow elements are detailed in the following sections:

- [Rule tasks properties](#)
- [Action tasks properties](#)
- [Subflow tasks properties](#)
- [Transitions properties](#)

Common properties

Some properties are common to several ruleflow elements.

Table 1. Properties common to several ruleflow elements

Property	Pertains to	Purpose
Label	Transitions, rule tasks, action tasks, subflow tasks, branch nodes.	You can specify a label for the ruleflow element, that is displayed in the ruleflow editor.
Initial action	Start nodes, rule tasks, action tasks, subflow tasks.	The action that launches before the task starts. Select BAL or IRL to define the language to use to write the action.
Final action	End nodes, rule tasks, action tasks, subflow tasks.	The action that launches after the task ends. Select BAL or IRL to define the language to use to write the action.
Document action	All ruleflow elements.	This property displays optional information about the node, or transition. Use this property to provide notes, comments, and other useful information.

Rule task properties

Table 2. Rule task properties

Property	Purpose
Uses	List the rules and rule packages that are considered when the rule task is executed. To edit this list and reorder it, use the arrow icons.
Selects rules where	Use this property to specify runtime rule selection using a statement in BAL, or IRL code. Define a rule filter in IRL with "body ="

Table 3. Advanced rule task properties

Property	Purpose
Rule selection	After you entered a code fragment for the property Select rules where , you can specify if your runtime rule selection is dynamic, or static: <ul style="list-style-type: none">• Dynamic: The property runs each time you call the task.• Static: The property runs the first time you call the task. For more information, see Runtime rule selection .

Algorithm	<p>Use this section to specify the processing algorithm for the rule task:</p> <ul style="list-style-type: none">• RetePlus• Sequential• Fastpath <p>For more information, see Engine execution modes.</p>
Exit Criteria	<p>Use this section to specify if all rules, one rule, or one rule instance execute:</p> <ul style="list-style-type: none">• None: The default setting that all rules are executed until conditions terminate execution. The rules are executed in a particular order determined by the selected ordering.• Rule: Execution terminates after the chosen rule is executed, according to the selected algorithm.• RuleInstance: A single instance of one rule is executed. This rule is determined by the selected ordering.
Ordering	<p>Use this section to specify the order of rule execution in a rule task as follows:</p> <ul style="list-style-type: none">• Default: The algorithm determines the execution order.• Literal: The order of the rules in the rule task determines the order of rule execution• Priority: the rules are executed following a static priority in decreasing order.

Note: The properties **Algorithm**, **Ordering**, and **Exit criteria** are for advanced users. For more information, see [Execution properties for rule tasks](#). If you do not know about these properties, you can leave the default settings.

Action task properties

Table 4. Action Task properties

Property	Purpose
Action code	Set the rule action statements to be executed, in BAL or IRL.

Subflow properties

Table 5. Subflow properties

Property	Purpose
Subflow	Select a ruleflow to be executed at this point in the ruleflow that you are editing.

Transition properties

Table 6. Transition properties

Property	Purpose
Conditions	<p>In the Transition Conditions Editor, you must first select the rule language you use to write the transition: BAL or IRL. In the rule language you chose, you write an expression whose return value must be true or false, for example:</p> <ul style="list-style-type: none">• BAL: <div>'the loan' is approved</div> <ul style="list-style-type: none">• IRL: <div>(bicycle.speed > 10) && (bicycle.speed <= 30)</div> <p>If you have several transitions originating from a task, there must be one (and only one) transition with no conditions, which is considered as the default transition, also called else transition. By default, if you do not specify a label, it is displayed as "else".</p>

With several transitions, you must make sure that transitions do not overlap, which means that no more than one transition condition must return "true". Typically, you might want to use two transitions, to choose between one ruleflow path with a specific condition and one default else transition for the remaining cases.

If there is only one transition that links two tasks, you must not specify any conditions.

Parent topic: [Ruleflows](#)

Decision operations

The decision operation defines which rules from a given branch are part of the ruleset. You choose which decision operation to use when creating a test suite, simulation, or deployment configuration.

A ruleset includes rule artifacts and other elements. A decision operation includes all the settings needed to define the contents of a ruleset and its parameters. The ruleset content and parameters allow the client application to exchange information with the ruleset.

Ruleset content

The ruleset content is defined by specifying the following information:

- The **source rule project**. All the rules and variables contained in this project and any of its dependent projects become eligible to be included in the ruleset.
- Any **ruleflow** to include in the ruleset.
- A **query** or **validator** to filter the rules, so that only a subset of the rules of the source rule project and its dependent projects are included in the ruleset. Typically, this is used to include rules from specific folders of a project or according to a rule property such as status. You write queries in the **Queries** tab of the Business console. A validator can be written in Rule Designer to select the required rules. If you specify both, the query is processed first, then the validation is applied.

Ruleset signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation.

The parameters of a ruleset can have three directions:

- **INPUT**: The parameter value is provided as input to the ruleset on execution.
- **OUTPUT**: The parameter value is set by the execution of the ruleset and provided as output from the ruleset at execution completion.
- **INPUT_OUTPUT**: The parameter value is provided as input to the ruleset on execution and its value can be modified by the ruleset and provided as output at execution completion.

You create the parameters from any of the ruleset variables that are available in the projects that are part of the scope of the decision operation. Ruleset variables are internal to a ruleset and provide a way to exchange data between rules, functions, and tasks.

Parent topic: [Decision artifacts](#)

Related concepts:

[Using queries](#)

Related tasks:

[Viewing the history of a rule](#)

[Restoring a version of a rule](#)

Ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project, and as input and output parameters in decision operations.

About this task

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables. You then define one or more variables in the variable set. Ruleset variables are variables that you can use in all the business rules of the ruleset. You can also use ruleset variables to pass data between tasks in a ruleflow. Ruleset variables are accessible from business rules only if you verbalize them.

Procedure

1. In your Rules or BOM project, click the **create an artifact** button and select **New variable set**.
2. Enter a name for the variable set and click **Create**.
3. In the variable set editor, click **Add variable** to add a variable to this variable set.
4. Enter a name and a verbalization, and select a type for the variable.

Name: the name of the variable must be a valid Java™ identifier. A Java identifier can contain uppercase, lowercase, and modifier Unicode letters, Unicode digits, currency symbols, and the ASCII underscore (_). The first character of the name must be a letter, a currency symbol, or an underscore.

Type: the type must be one of the available types in the drop-down menu, or a technical Java type.

Verbalization: specifying a verbalization is optional, but it is necessary to make the variable visible from business rules.

5. Click **Save**.

Results

The ruleset variable is now defined and you can start using it in business rules or as input and output parameter in decision operations.

Parent topic: [Decision artifacts](#)

Dynamic domains

A domain defines a set of values in your business terms. For example, a domain can define that the category of a customer can have one of the following values: silver, bronze, gold. You can modify these values and update your project in the Business console.

You must first define the domain in Rule Designer before you can access it from Decision Center.

Domains are available to Decision Center through a domain provider that you manage in an Excel file. You must upload any changes to the domain to Decision Center.

Updating domains from an Excel file

You can modify the values of dynamic domains that are stored in an Excel file and update the BOM of your project with the new values.

Parent topic: [Decision artifacts](#)

Updating domains from an Excel file

You can modify the values of dynamic domains that are stored in an Excel file and update the BOM of your project with the new values.

Before you begin

You must use a new or existing change activity to update domains in projects that are using the governance framework in Decision Center.

About this task

When a project contains a dynamic domain defined in an Excel file, the Excel file is stored as a resource in the project. You can modify the Excel file and then upload the changes to Decision Center.

The Excel file has a specific structure with columns that map properties in the BOM:

- A column for the domain values
- A column for the label of each value
- A column that defines the BOM-to-XOM mapping for each value

Other optional columns for the documentation or labels in other locales might be defined in the Excel file.

You can modify the values in each column, but you must not change the columns or column headings.

Procedure

To edit the Excel file:

1. In the **Decision artifacts** tab, in the **Resources** default folder, browse to the resource that corresponds to the Excel file. If the **Resources** folder is not in the folders list, edit the artifacts filters to enable the **Resources** type.
2. Click the Excel file in the list to open the preview.
3. To download the file, click its name.
4. Open the Excel file, modify the values, and save the changes.

You can now upload the modified Excel file to Decision Center.

5. In the preview of your file in the Business console, click the edit button, then click **Choose**.
6. Select the new version of the file and click the save icon.
7. Open the **Model** tab.

This tab lists the domains of the project. You can expand a domain to see its current values and the updates that have been made in the Excel file.

8. To update the BOM with the new values, select the domains that you want to update and click **Apply changes**. This process can take several minutes to complete if the domain contains many values.

Parent topic: [Dynamic domains](#)

Versions and history

Decision Center creates versions of elements that you can view in a timeline and restore.

[Versioning in decision services](#)

Decision Center creates archived versions of elements each time you modify them, including moving or renaming them.

[Viewing the timeline of a release or activity](#)

You can view the complete history of a release or activity.

[Viewing the history of a rule](#)

Decision Center keeps a history of all the changes that you make to your rules.

[Comparing versions of a rule](#)

You can compare two versions of an action rule or decision table.

[Restoring a version of a rule](#)

You can restore a previous version of a rule to the current state.

Parent topic: [Decision artifacts](#)

Versioning in decision services

Decision Center creates archived versions of elements each time you modify them, including moving or renaming them.

You can consult the history of modifications and restore a previous version. However, you can edit or delete only the current version of an element.

The version number of an element is in the form $x.y$, where x is the major number that corresponds to a release or activity in a decision service and y is the minor number that is increased with each change:

Major number

Decision Center gives every branch of a decision service a major version number. The version number 1.0 is given to the main branch created during the initial publication from Rule Designer. Branch 2 is always the Initial Release, which is also created automatically during the initial publication. Branch 3 is the first release that you create. Then, branch 4 is the first change activity and so on.

Minor number

The initial version of an element has the minor version number 0. This minor version number is increased by 1 each time you edit the element. In a release or change activity, each element initially has the major and minor version number of its parent branch. When you edit the element in an activity for the first time, this element takes on the version numbers of the activity. For example, an element with the version number 3.0 keeps this number in the activity until you edit the element for the first time, when it becomes, for example 4.0. Subsequent changes that you make to this rule in the change activity gives it version number 4.1, 4.2, and so on.

When you complete the change activity, it is merged with the release. The major version number of any rule modification done in a change activity adopts the major version of the release. The new version is an exact copy of the version in the activity, and takes the next minor version number in the parent release.

When you restore a snapshot, Decision Center creates a version of all the elements that are found in the snapshot, and copies them to the current state of the release or activity.

When you restore a version, either from a snapshot or from the timeline of a change activity, the new version is a copy of the restored version, and takes the next version available number.

Note:

You cannot clear the history of an element.

Parent topic: [Versions and history](#)

Related tasks:

[Viewing the history of a rule](#)

[Restoring a version of a rule](#)

[Viewing the timeline of a release or activity](#)

Viewing the timeline of a release or activity

You can view the complete history of a release or activity.

About this task

A timeline shows the events that occur during the life of a decision service release or change activity, such as:

- Creation or deletion of a release or activity, or changes to its name or description.
- Creation, modification, restoration, or deletion of a rule.
- Creation, restoration, or deletion of a snapshot, or changes to the name or description of the snapshot
- Changes to the status a release or activity.

Note: Each event in the timeline contains a **Comments** field for you to enter comments to add to the stream.

Procedure

1. In the **Project** view, click **Timeline**.
2. Select an entry in the **Show** field to reduce the number of visible events to rules, activities, or snapshots.
3. Hover over the **Comments** section of an event and click the arrow to display events that relate only to that project element.
4. Click **Exit Timeline** to exit the timeline.

Parent topic: [Versions and history](#)

Related concepts:

[Versioning in decision services](#)
[Snapshots](#)

Related tasks:

[Viewing the history of a rule](#)
[Restoring a version of a rule](#)
[Creating a snapshot](#)
[Redeploying](#)

Viewing the history of a rule

Decision Center keeps a history of all the changes that you make to your rules.

About this task

The history of changes to rules is kept in a timeline, with the most recent version at the top. Versions are shown alternating on each side of the timeline down the page.

You cannot edit previous versions of a rule, but you can restore them to the current state.

Note: You cannot delete previous versions.

Procedure

1. Click the name of the rule to access its **Details** view.
2. Click **Timeline**.
3. In the timeline, click an event to display its contents in read-only mode, or enter a comment to publish it to the activity stream.
4. Click **Exit Timeline** to return to the **Details** view.

Parent topic: [Versions and history](#)

Related concepts:

[Versioning in decision services](#)

[Decision operations](#)

Related tasks:

[Restoring a version of a rule](#)

[Viewing the timeline of a release or activity](#)

Comparing versions of a rule

You can compare two versions of an action rule or decision table.

About this task

When you compare two versions of a rule, Decision Center shows a side-by-side comparison of the differences in the contents or the properties.

Procedure

1. Start in one of the following ways:
 - In the **Project** view, hover on the rule and click **Compare** in the drop-down menu.
 - Click **Compare** in the **Details** view for the rule.
 - Hover on the rule in the timeline or in a snapshot and click **Compare**.
2. Click two versions of the rule, and then click **Compare**. You can click a selected version to clear it.
3. Decision Center displays the total number of differences between the two versions. Select **Content** to see which rules or rows were added or deleted, or **Properties** to compare the properties.
4. Click **X** under the top banner to return to the current version of the rule.

Parent topic: [Versions and history](#)

Restoring a version of a rule

You can restore a previous version of a rule to the current state.

About this task

Each rule has a timeline that contains versions of the rule. You can reinstate a previous version of the rule from the timeline.

Note:

In a decision service, you cannot restore a previous version of a rule directly to a release. Changes to a release are made in change activities, and when the changes are approved, they are merged back into the release.

Procedure

1. In the timeline, click the version that you want to restore.
2. In the toolbar, click **Restore**.
3. Enter a comment and click **Restore Version**.

Results

Decision Center displays the new version in its **Details** view, and the history of the rule is updated.

Parent topic: [Versions and history](#)

Related concepts:

[Versioning in decision services](#)

[Decision operations](#)

Related tasks:

[Viewing the history of a rule](#)

[Viewing the timeline of a release or activity](#)

Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping. Technical rules are written using the ILOG® Rule Language (IRL). IRL is a Java™-like rule language that can be executed directly by the rule engine.

In the Business console, technical rules are shown when you activate the type **Technical Rules** in the **Decision Artifacts** tab.

Technical rules are an advanced subject, useful for developers. See the [Rule Designer](#) documentation for more information.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
 - `evaluate`
 - `exists`
 - `from`
 - `in`
 - `instanceof`
 - `not`
 - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

Parent topic: [Decision artifacts](#)

Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG® Rule Language (IRL) and its code is evaluated when the ruleset runs.

Functions are an advanced subject, useful for developers. See the [Rule Designer](#) documentation for more information.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)  
{  
    code  
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the `return` keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of `return` that does not return a value:

```
return;
```

Note: A function is not required to declare in its `throws` statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

Parent topic: [Decision artifacts](#)

Rule verification

You can perform different levels of verification on your rules.

Basic syntax checks

Decision Center verifies the syntax of business rules.

Decision table checks

You can check decision tables for overlaps, gaps, and symmetry.

Parent topic: [Decision artifacts](#)

Basic syntax checks

Decision Center verifies the syntax of business rules.

When you save a rule, Decision Center displays any errors or warnings that result from checking the syntax. If a rule contains a syntax error, the administrator excludes it from the ruleset that is deployed to the production system.

Decision Center checks for the following syntax errors:

- Unrecognizable or invalid rule statement
- Ambiguous rule statement with more than one correct interpretation
- Repeated or wrong variable in the definitions part
- Incomplete rule with empty placeholders
- No if part in a rule with an else part

For a full list, see the **Rule Designer** online help.

Parent topic: [Rule verification](#)

Decision table checks

You can check decision tables for overlaps, gaps, and symmetry.

Overlap checks

You can set Decision Center to check that your decision tables do not contain duplicated values.

For example, in the following condition statements, customers who are 29 or 30 years old satisfy both rules:

- Age is between 17 and 30
- Age is between 29 and 40

If these two conditions are in the same column or within a partitioned groups of cells in a decision table, Decision Center signals an overlap error.

Gap checks

You can set Decision Center to check that the values you enter in your decision tables consider all possible cases. This ensures that there are no gaps in your conditions.

For example, the following condition statements do not include customers who are 31 years old:

- Age is between 17 and 30
- Age is between 32 and 40

Decision Center reports a gap error.

Symmetry checks

You can set Decision Center to check that all the partitions in a decision table use the same set of items. You can verify the symmetry of the entire table or specified columns. This analysis helps ensure that you have covered all the possible choices in the decision table.

By default, symmetry analysis is turned off so that you can create nonsymmetrical tables.

Parent topic: [Rule verification](#)

Rule properties

Properties associate additional information with a project element, including the development status, storage location, and user group.

The properties are displayed in the **Properties** tab next to the project element.

To edit the properties, click **Details** in the edit view of the project element. If you have the correct permissions, you can change the properties.

Before you change the properties in a project, make sure that your development team implements the use of properties. Otherwise, your changes might not be used.

Note:

You might have project properties that are specific to your business domain.

The following table shows the default properties:

Prop erty	Description
Folder	Set the location of the element within the project.
Statu s	Set the status of a rule, for example: <ul style="list-style-type: none">• New• Defined• Validated• Rejected• Deployable• Inactive
Categ ories	Select a category for the project element. The default category is Any. Categories can be used to filter rules in queries.
Group	Associate the rule with a specific group. The Group property is used by the administrator to establish permissions to view, edit, create, and delete project elements.
Priorit y	Set the priority of the rule to define the order in which rules are executed.
Effecti ve Date	Specify the date on which the rule goes into effect.
Expira tion Date	Specify the date on which the rule expires.
Tags	Add tags to your action rule, decision table, or ruleflow.

Parent topic: [Decision artifacts](#)

Related concepts:
[Getting familiar with the Business console](#)

Tags for rule artifacts

You use tags to store data with your rule artifacts (action rules, decision tables, ruleflows).

For example, you can store data related to the geography of the rule or its development phase. You can retrieve this information in a query or in a report.

Tags are visible in the properties of your rule artifact, which you can access by selecting the artifact in the **Decision Artifacts** tab, then clicking **Show details**. To add or remove tags, edit your rule artifact, click **Details**, and select the **Tags** tab. When you add a tag, you declare a name and give it a value, for example:

Name	Value
Geography	New York
Phase	1

For developers, tags can be useful when you have implemented a rule model extension in Rule Designer and you synchronize with Decision Center. In this case, extended properties are converted into tags so that the synchronization remains smooth. Later, if you implement the rule model extension in both Rule Designer and Decision Center, these tags are converted back to properties.

Note: If you defined tags on rule packages in Rule Designer, you cannot synchronize them with Decision Center, because Decision Center does not support tags on rule packages.

Parent topic: [Decision artifacts](#)

Rule overriding

You use rule overriding to give rules precedence over other rules.

For example, you can set an entire decision table to override another decision table.

A rule that overrides one or more other rules is executed in place of the rules that are overridden. Normally, rule overriding operates within a hierarchy of rules, with a local or more specific rule overriding a more general rule. For example, you can set a minimum customer age for a particular state to take precedence over a minimum customer age for the country.

If an overridden rule is selected in the same rule task of a ruleflow, then this rule is filtered out and not executed.

Overridden rules are visible in the properties of your rule artifact, which you can access by selecting the artifact in the **Decision Artifacts** tab, then clicking **Show details**. To set rule overriding, edit your rule artifact, click **Details**, and select the **Overridden Rules** tab. You can then add the rules that you want the current rule to override.

Parent topic: [Decision artifacts](#)

Modeling decisions in the Business console

Generally, IT creates the decision model on which rules are authored. For decision model services, you create both the model and the logic directly in the Business console.

[Creating or importing a decision model service](#)

You can create a decision model service from scratch, or import an existing project that was created in IBM Decision Composer.

[Decision modeling basics](#)

In Decision Center, you the business expert can create and deploy a decision service that gets called by client applications when they require a specific business decision.

[More on data modeling: Custom types, default values, lists](#)

In addition to creating the data nodes, data modeling requires you to assign the appropriate type to all the nodes.

[More on decision authoring: Coverage and rule interactions](#)

Authoring the decision logic requires you to create the business rules, but also to consider coverage and rule interactions.

[Deploying a decision model service](#)

You deploy a decision model service to an execution environment using a deployment configuration, the same way as you deploy a regular decision service.

[Validating and testing decision model services](#)

With a decision model service, you can validate your decision model during development. You can also create more complete usage scenarios that you use in test suites and simulations, just like regular decision services.

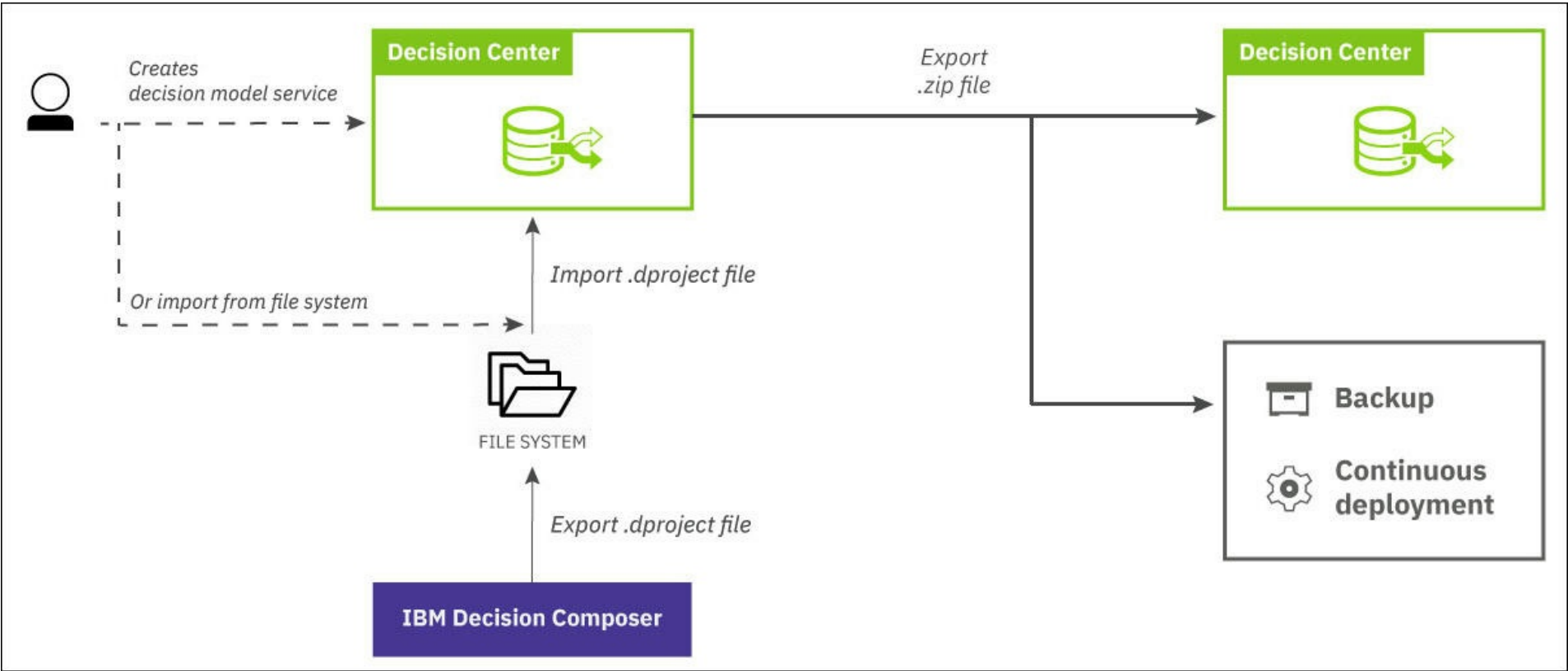
Parent topic: [Working with the Business console](#)

Creating or importing a decision model service

You can create a decision model service from scratch, or import an existing project that was created in IBM Decision Composer.


Decision projects that are exported from IBM Decision Composer as .dproject files can be imported to Decision Center, where they become integrated as decision model services. You can then export a decision model service as a .zip file to other Decision Center repositories, or to a file system for:

- Regular backup of the project.
- Use within a continuous deployment pipeline.



A decision model service makes use of a diagram in which you link decision and data nodes, and then author the decision logic (see [Decision modeling basics](#)). With a decision model service, there is no interaction with Rule Designer because the model on which you write the decision logic is part of the diagram.

A decision model service behaves the same way as a regular decision service, with the following differences:

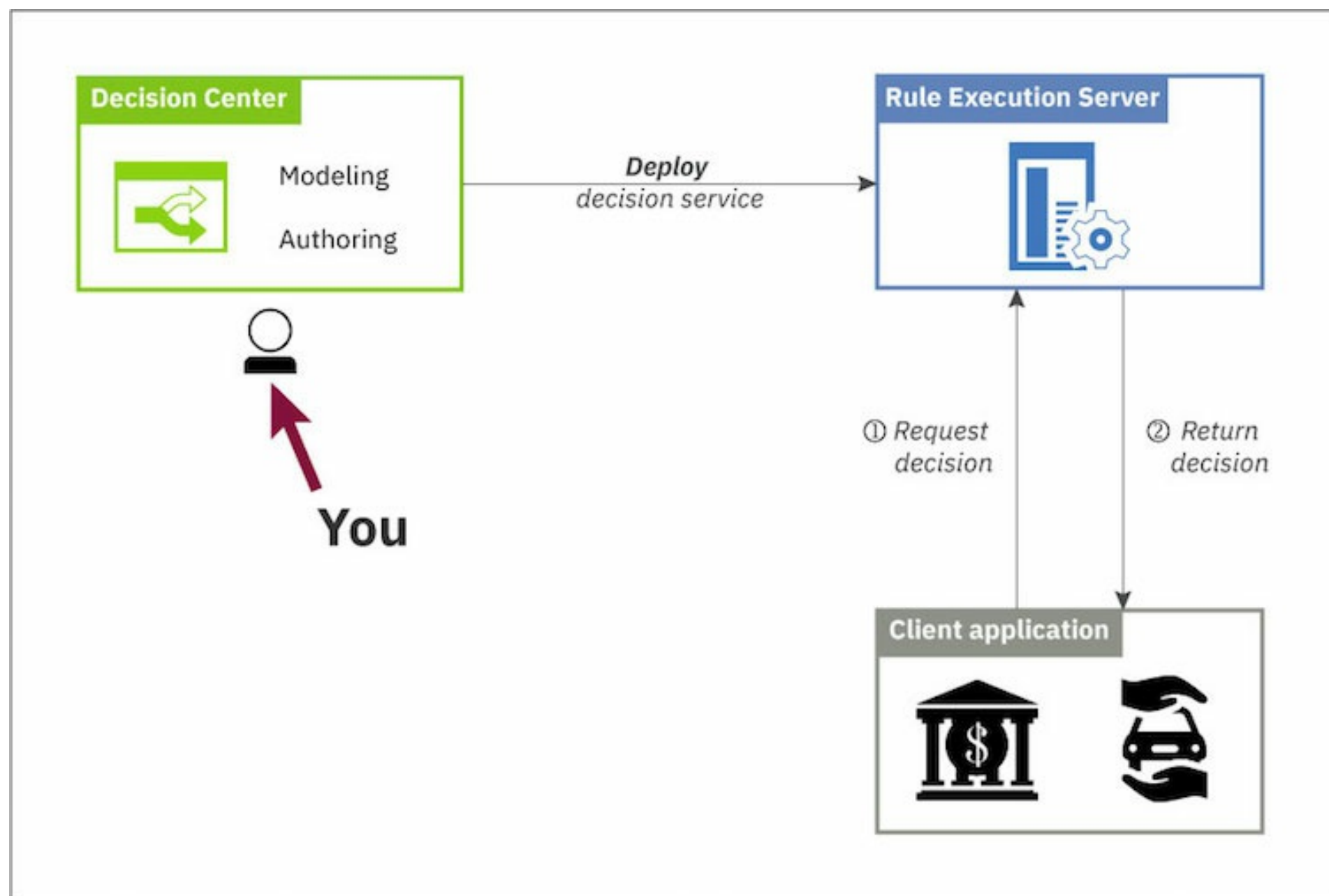
Features in decision model services	Description
A decision model service contains only one artifact, which is the decision model.	You create rules and decision tables directly in the decision nodes.
There is no ruleflow to determine rule execution.	<p>A decision model service makes use of a diagram to link decision and data nodes, which all contribute to the final decision.</p> <p>Decision Center generates an empty ruleflow for integration, but the ruleflow is not editable in the Business console.</p>
The decision model works as one versionable element.	Changes to individual business rules, decision tables, and the nodes can be consulted through the version of the decision model itself.
Cannot query or text search individual rules.	The Queries tab and the search field are not visible.
Viewing and comparing versions of a decision model.	To view, compare, and restore versions of a decision model, open the Decision model view  and use the Compare tool.
Cannot edit the decision operation.	In a decision model service, the decision operation includes the entire decision model. The decision operation is automatically regenerated every time that you save (see Deploying a decision model service .)
Using the decision governance framework.	<p>Because a decision model works as one unit, you cannot create separate change activities to work simultaneously on different parts of the decision model.</p> <p>For more information, see Governance principles.</p>

Parent topic: [Modeling decisions in the Business console](#)

Decision modeling basics

In Decision Center, you the business expert can create and deploy a decision service that gets called by client applications when they require a specific business decision.

You create this decision service in Decision Center and then deploy it to an execution environment. Client applications requesting this business decision call your decision service in the execution environment, as shown here:



In Decision Center, you can create decision services in one of two different, distinct ways:

1. Your development team creates the underlying model, and you, the business expert, author the decision logic based on this model.
2. You, the business expert, create both the model and the logic of the decision service. The term **decision model service** distinguishes this second case from the first one.

This section describes a decision model service by presenting two separate but intertwined notions:

- Modeling the node structure
- Authoring the decision logic

Note: For simplicity, the term decision service is used throughout, except when there is the need to distinguish the two approaches or when referring specifically to the modeling part of the decision service.

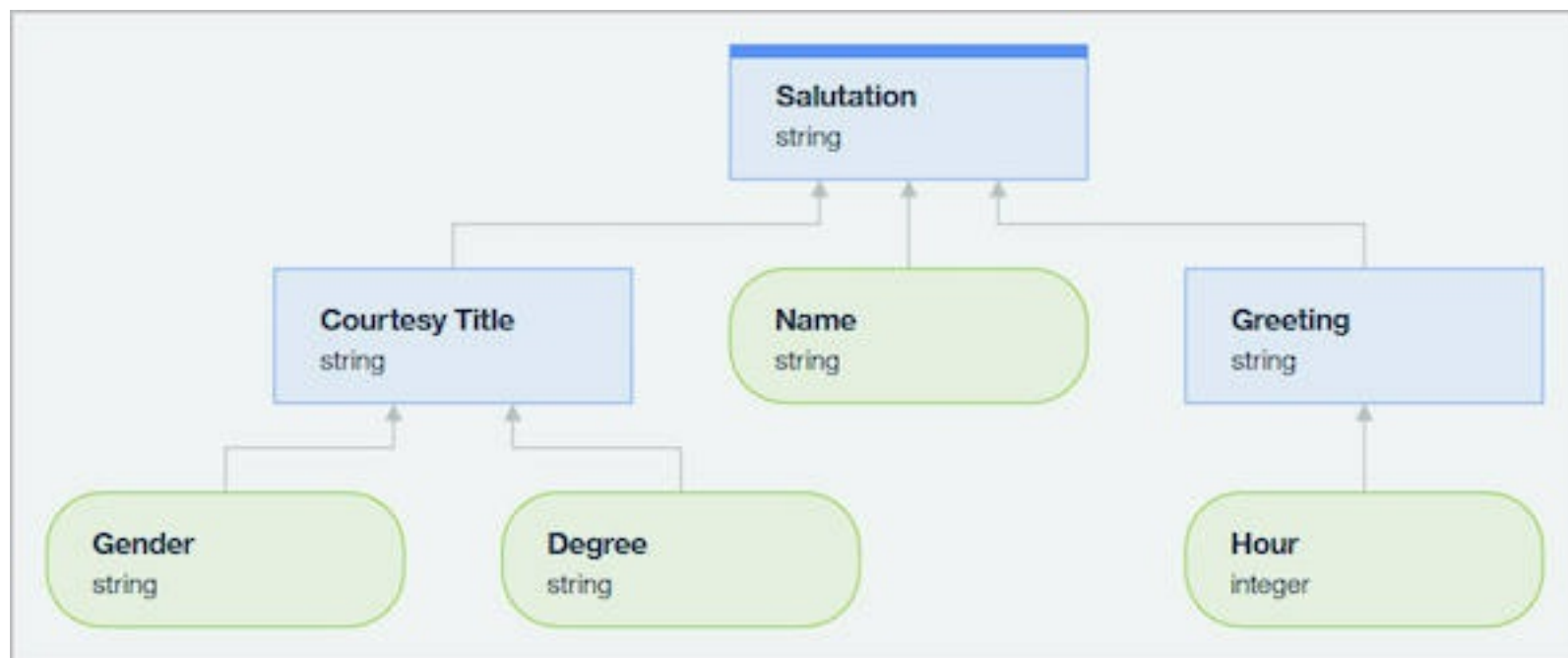
Modeling the node structure

Consider a simple decision service that greets a customer with a salutation. An example of the salutation is "Good evening Mrs. Jones".

The information required from the client application for your decision service to generate the appropriate salutation is:

- The name of the customer
- The gender and any degree (Dr, Professor, ...) that the customer has
- The hour of the day

The model of this decision service could look like this:



From a modeling perspective, you the business expert create and define the relationship between:

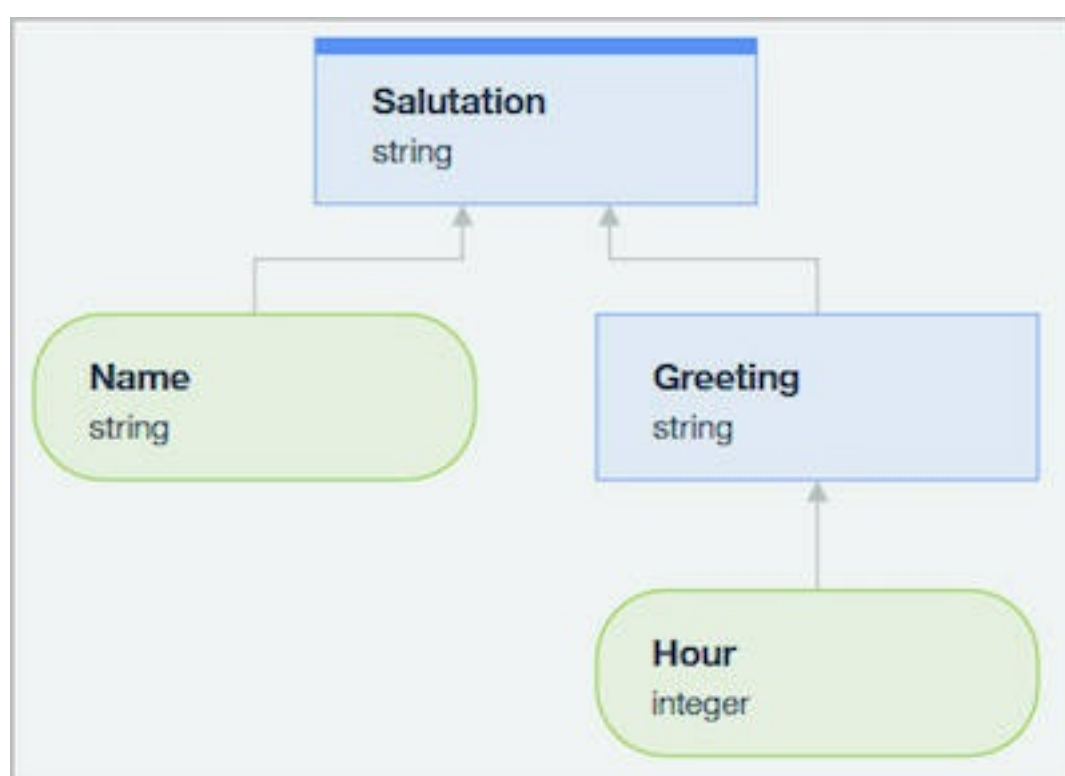
- Visual representations of the decisions required.
- Visual representations of the information (data) that must be provided to make each decision.

In the diagram, these visual representations are called **nodes**. The following shows a decision node 'Salutation' fed by a data node 'Name':



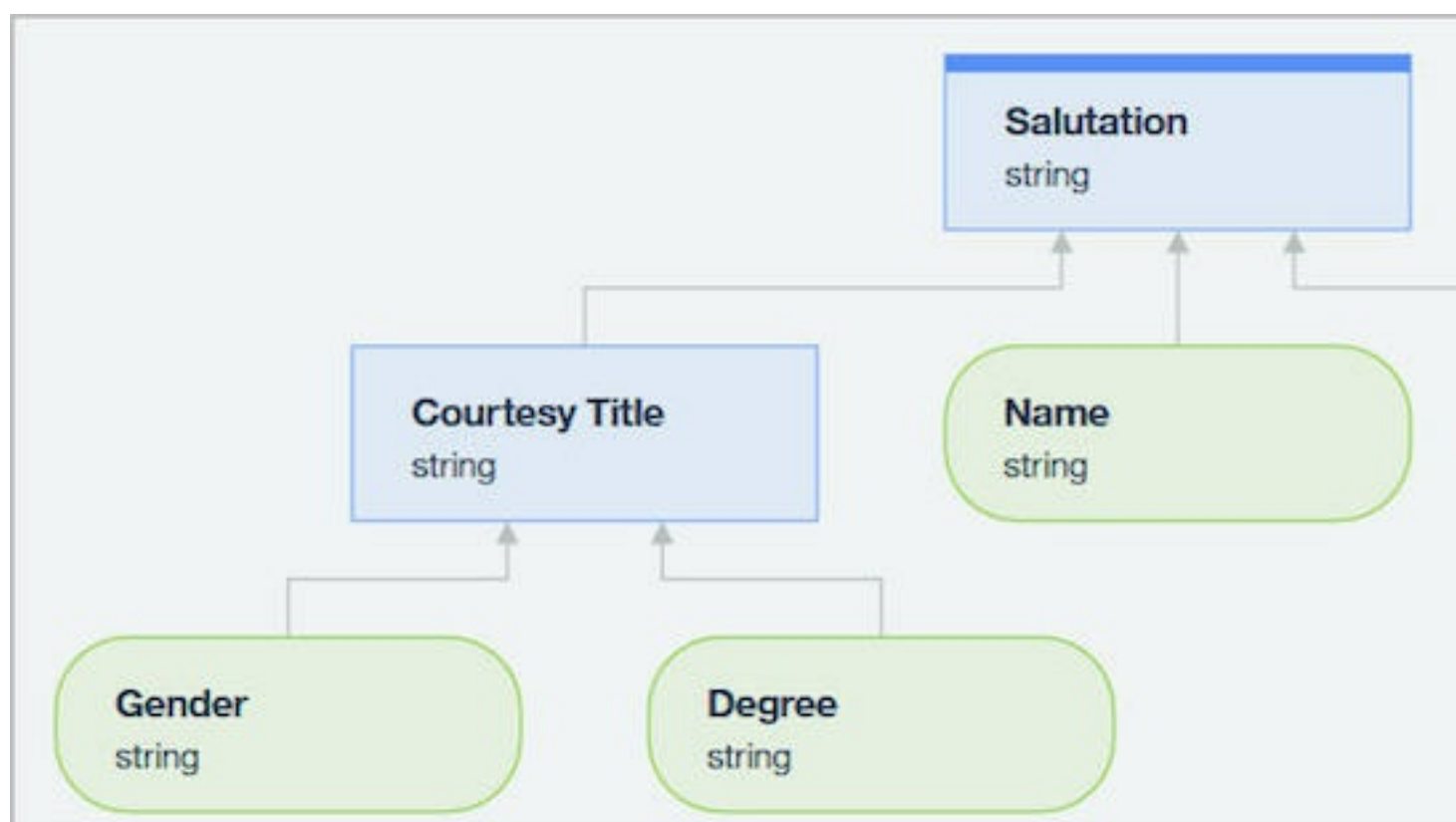
At this point, the salutation returned by your decision service when given 'Jones' for the name would be: "Hello Jones."

To make the diagram of your decision service easier to understand and maintain, you can make a decision node dependent on other decision nodes that handle a specific aspect. For example, to refine your decision service so that it returns a salutation based on the time of day, you create another decision node called 'Greeting' that uses 'Hour' as input, and link this decision node so that its result can be used by the 'Salutation' node:



At this point the salutation returned by your decision service when given the name 'Jones' at 8 p.m. would be: "Good evening Jones."

Then, to add a courtesy title (Mr, Mrs, Dr, and so on) to personalize your decision service further, you create a decision node 'Courtesy Title' fed by two data nodes, one for gender and one for degree. This node feeds the final decision node, just like the 'Greeting' node:



The salutation returned by your decision service when given 'Jones' at 8 p.m. with the gender Mrs. and no degree would be: "Good evening Mrs. Jones."

When a client application provides the information, your decision service processes this data and returns the decision. To summarize the modeling part:

- You create and link together decision and data nodes.
- You design each decision node to accept input data and output the result to the node above it or to the client application. How the result is obtained is the subject of [Authoring the decision logic](#) below.

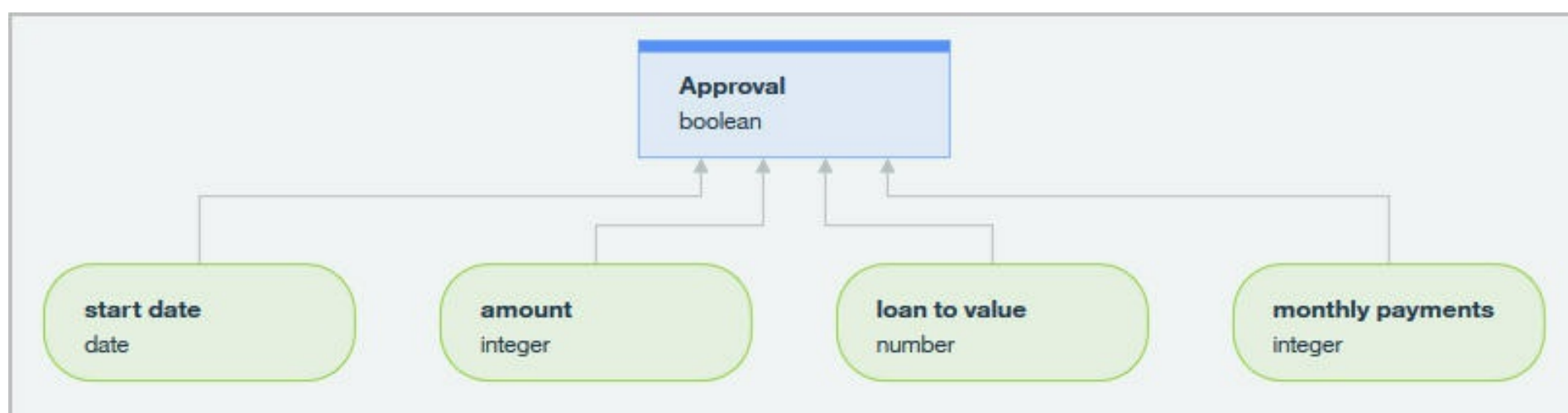
Data model

The term **data model** can be used to distinguish specifically the data aspects of the decision model.

In addition to creating the data nodes, data modeling requires you to assign the appropriate **type** to all the nodes. Assigning a type to a node specifies:

1. What information the node can accept and send, for example, a number, date, and so on.
2. The possibilities when authoring the decision logic.

In the following example, four different data nodes, of various types, feed the decision node above it:



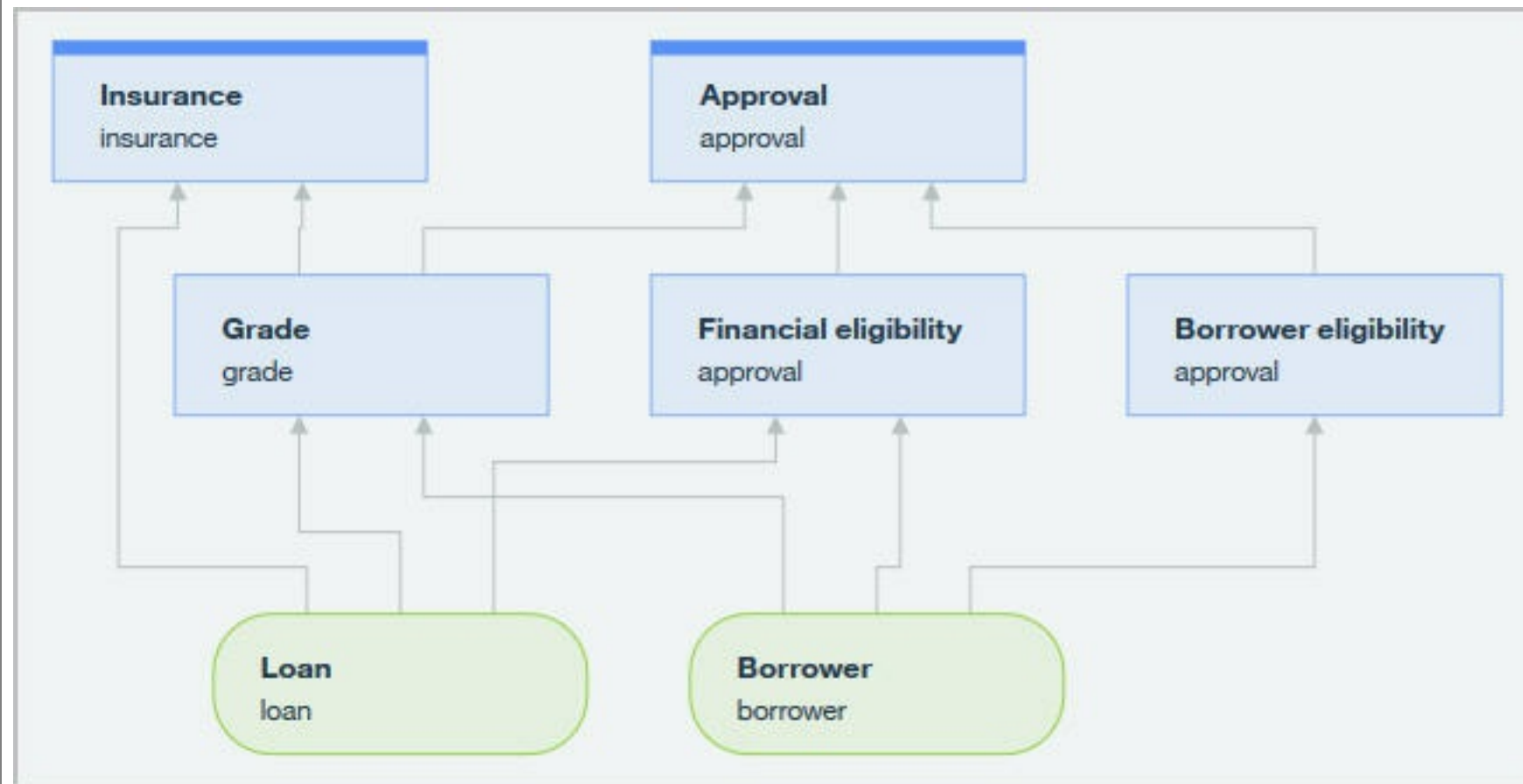
In this diagram, the four data nodes are laid-out flat. It is often more convenient for you to create **custom types**, which regroup related data under one entity. The diagram below contains the same input data as the diagram above, but all the data nodes relating to the loan have been regrouped under one node that handles data of a custom type 'loan'. This type regroups all the attributes of the loan:



Data modeling is described in more detail in [More on data modeling: Custom types, default values, lists](#), which also explains default values that you can set on the data nodes.

Note: The example used so far is simple. A more complex decision service can have more than one top-level

decision node, that is, it can return more than one decision to the client application. Also, a data node may feed more than one decision node. The following example shows both these cases:



Authoring the decision logic

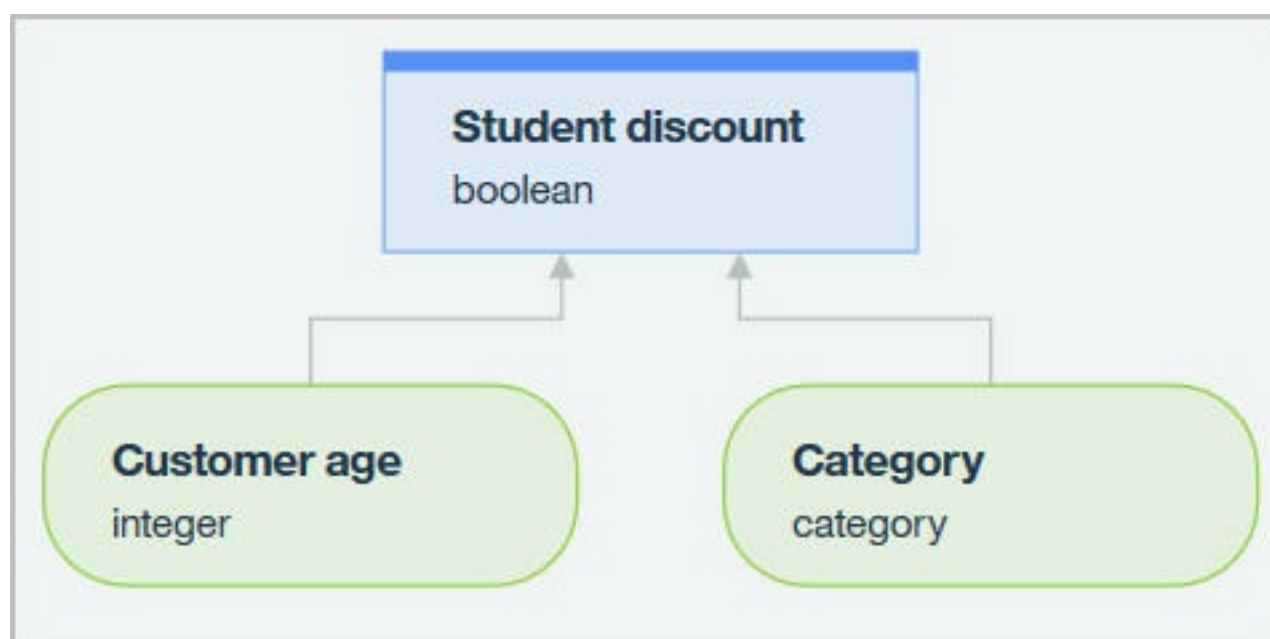
You author the logic of each decision node by creating one or more *business rules* that:

- Evaluate the input data that each decision node receives against conditions that you establish.
- Take actions to set or modify the output of the decision node, that is, the **decision**.

These business rules are simply your business policies expressed in the following form, understandable by a computer:

```
if
  <conditions>
then
  <actions>
```

Consider a decision service that decides whether a customer is eligible for a student discount, based on the customer being under 18 **and** a student. The decision model has two data nodes 'Customer age' and 'Category' that feed the decision node 'Student discount':



You implement the decision logic of the 'Student discount' node by creating the following business rule:

```
if
  'the customer age' is less than 18
  and 'the category' is Student
then
  set 'the student discount' to true,
  print "You benefit from the 20% young student discount!" ;
```

In this example, the content of the business rule is color-coded to demonstrate the different building blocks at your disposal when creating or modifying a business rule, as follows:

- The yellow parts represent the input or output data available to the decision node.
- The grey parts represent values of the data that you identify as pertinent to authoring the decision logic.
- The rest corresponds to the modeling language itself, which is used to compare, evaluate, and modify the different types of data (see [Decision Center modeling language reference](#)).

In other words, the data handled by a decision node, that is, its output and the data from the nodes that feed it directly, can be used immediately when authoring your business rules, in the form 'the <data>'. In the example, *Student discount* is usable in the rules as 'the student discount', *Customer age* is usable as 'the customer age', and *Category* is usable as 'the category'.

To further illustrate this important point, each time that you create a new rule, Decision Center evaluates the data available as input to the decision node in which you are creating the rule. Then, it proposes, as a starting point for your new rule:

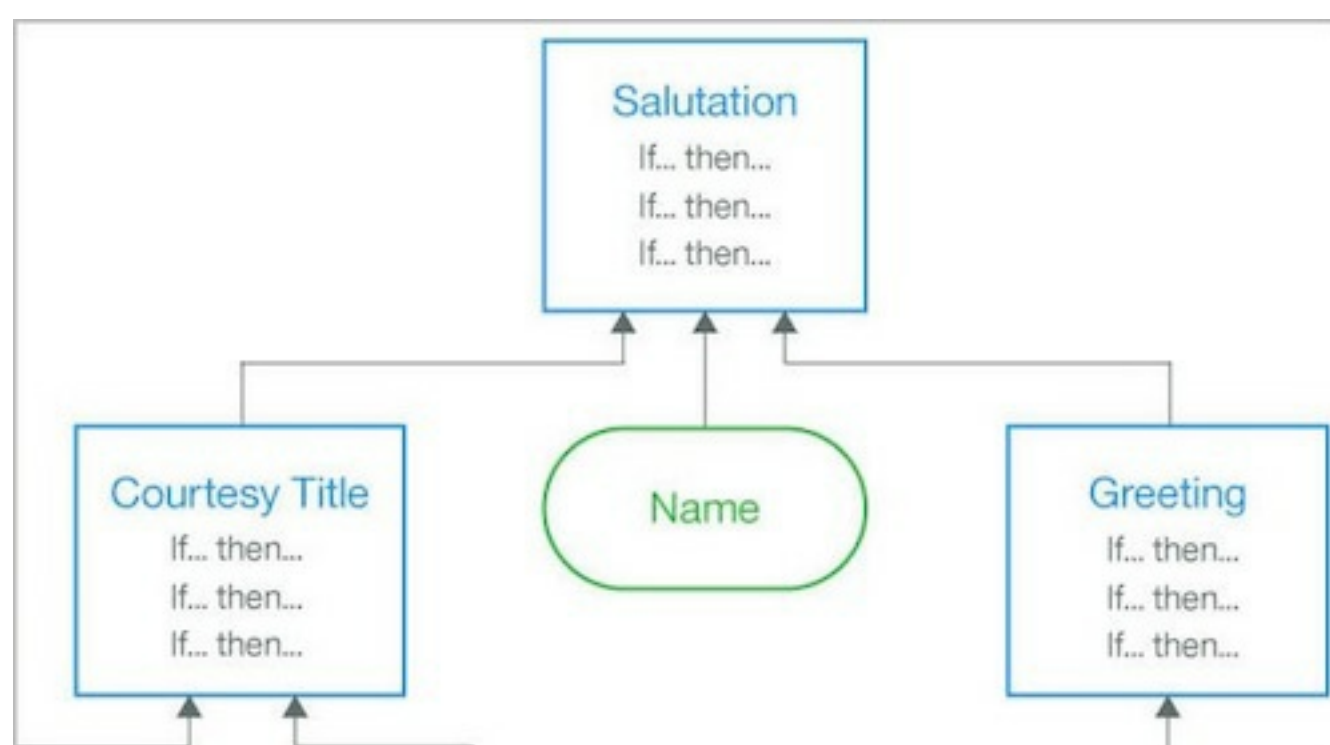
1. A typical condition statement (in the **if** part) comparing the input data according to its type. It proposes a condition statement for all the data available to the node, but you can clear them, since not all rules act on all the input data at the same time.
2. An action statement (in the **then** part) setting the value of the output of the node according to its type.

```
if
  'the customer age' is at least <min> and less than <max>
  and 'the category' is <a category>
then
  set decision to <a boolean>;
```

Note: The modeling language contains aspects that make your business rules more readable. For example, the keyword **decision** can be used interchangeably with the name of the node, which helps identify visually this important aspect of the rule, say in a report.

Coverage

When you author a decision node, you create the business rules required to reach a decision for the possible cases that the client application might provide as input data. This is referred to as coverage. Typically a decision requires several business rules to cover as many cases as possible:



To handle situations where there are many possible conditions for the input data provided, you can use decision tables, which regroup business rules that have a common structure but different values for the conditions and actions. For example, to set a salary score based on income, many business rules are required to cover different ranges of income. The following decision table handles this in a simpler way:

	Yearly income		Salary Score
	min	max	
1	< 10,000		21
2	10,000	20,000	50
3	20,000	30,000	80
4	30,000	50,000	120
5	50,000	80,000	150
6	80,000	120,000	200
7	120,000	200,000	250
8	≥ 200,000		300

Each row in a decision table corresponds to a business rule. The columns on the left represent the conditions (the **if** part of the rule) and the columns on the right specifies the decision returned for that case.

The decision logic can require one or more decision tables and some business rules to get complete coverage. Note that in these cases, the order in which the rules appear in your node is important (see [More on decision authoring: Coverage and rule interactions](#)).

Parent topic: [Modeling decisions in the Business console](#)

More on data modeling: Custom types, default values, lists

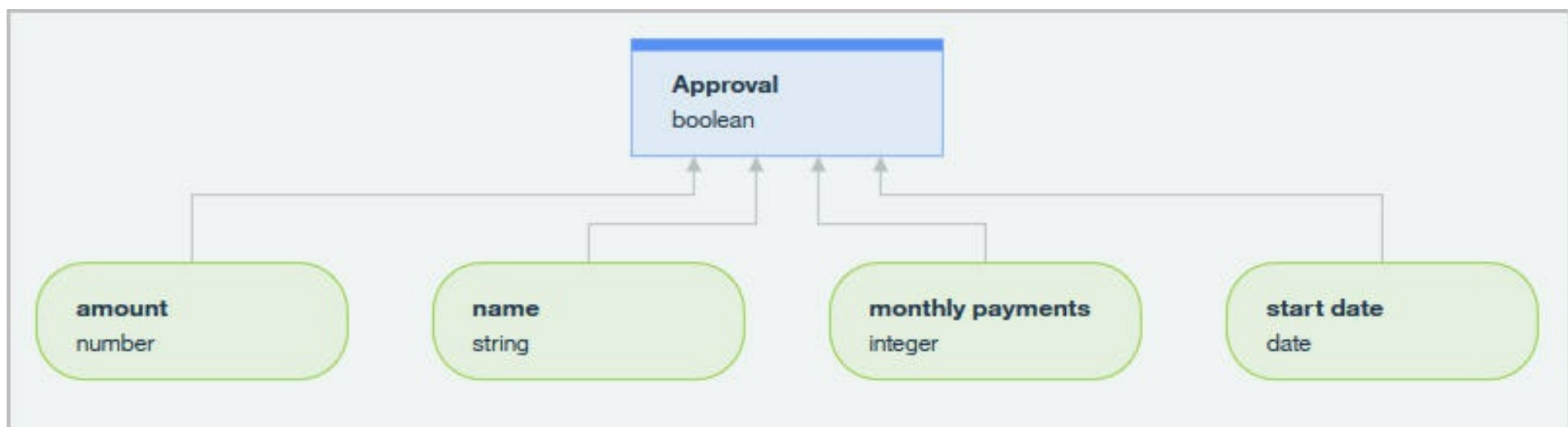
In addition to creating the data nodes, data modeling requires you to assign the appropriate type to all the nodes.

Note: This topic builds further on the information presented in [Decision modeling basics](#), and assumes that you are familiar with decision nodes, data nodes, and decision logic.

Assigning a type to a node determines:

1. What information the node can accept and send.
2. The possibilities offered in the rule and decision table editors when authoring the decision logic.

The following example shows a decision service that decides if a loan is approved. The diagram has four different data nodes feeding the decision node above it:



All the data nodes in this example handle a simple, standard **system** data type:

- **amount** handles numbers
- **name** handles text (strings)
- **monthly payments** handles integers
- **start date** handles dates

The **Approval** decision node is of type `boolean` (true/false). The node accepts data of various types, and then returns a decision of type true/false to the client application that makes a request.

Any business rule or decision table that you create in the **Approval** node will be made up of:

1. Condition statements based on the type of the data nodes that feed it directly.
2. A decision that sets the value of the output according to its type.

Each time you create a new business rule or decision table, Decision Center proposes, for your convenience, condition and action statements based on the most common formulation for the types of data handled by the decision node. In the **Approval** decision node, a business rule involving all the data nodes initially appears as follows:

```
if
  'the amount' is at least <min> and less than <max>
  and 'the monthly payments' is at least <min> and less than <max>
  and 'the name' is <a string>
  and 'the start date' is on or after <a date> and before <a date>
then
  set decision to <a boolean>;
```

From this starting point, you fill in the placeholders, for example `<min>` and `<max>`, with values pertinent to your decision logic, or change the statements with a more appropriate one from the modeling language. The modeling language contains extensive constructs to compare, evaluate, and modify data and decisions based on their type (see [Decision Center modeling language reference](#)).

Custom types

An important consideration when you create the data model is to create **custom types**, which regroup related data under one entity.

In our example, the four data nodes that feed the **Approval** decision node are laid-out flat. Because all the data nodes are not themselves the result of a previous decision, and they all feed the same decision node, these data nodes can be replaced with a single data node without any loss of information.

This makes the diagram easier to read. A second benefit is that it is not necessary to change the diagram when you add new attributes to the custom type.

To obtain a node that regroups all the data relating to the loan, you create the custom type **loan**, which sets as attributes all the data that was previously laid-out flat:

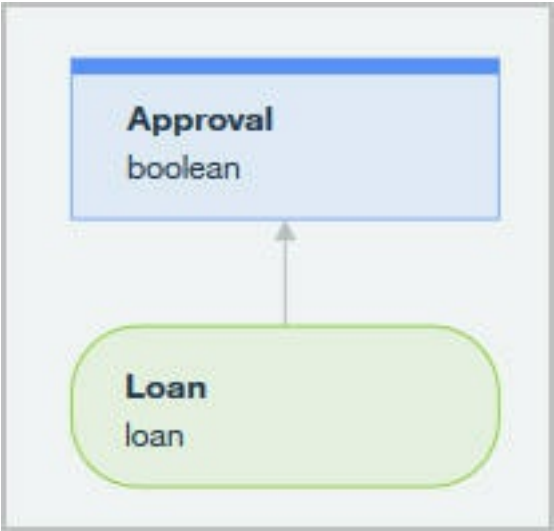
Name:

loan

Attributes:

Name	Type
amount	number
name	string
monthly payments	integer
start date	date

Then you create a single data node, here called **Loan**, and give it your new custom type loan:



The business rules and decision tables that you write now use the formulation the attribute of 'the node name', for example the amount of 'the loan'. This makes for more explicit condition statements, which is convenient on more complex business rules. The starting point proposed by the editors now looks like this:

```
if
  the amount of 'the loan' is at least <min> and less than <max>
  and the monthly payments of 'the loan' is at least <min> and less than
<max>
  and the name of 'the loan' is <a string>
  and the start date of 'the loan' is on or after <a date> and before <a
date>
then
  set decision to <a boolean>;
```

Enumerations

Another possible custom data type is an **enumeration**. An enumeration is a data type in which you establish all the possible values beforehand. In the following example, you see an enumeration called **category** with its possible values:

Name:

category

Values:

student

employed

retired

You should use enumerations over text entries for the following reasons:

- The business rule and decision table editors propose the values from the ones in your enumeration. This improves the rule editing experience, and it also helps prevent data entry errors.
- The correctness of your rules is verified immediately by comparing the rule statements with the values of your enumeration. If you use text entries, errors in the text will only be revealed during execution.
- Gaps in your decision tables are immediately revealed by the decision table editor.

Default values on data

You can set default values on decision nodes or data nodes, to provide a fall-back value if no value is available. For decision nodes, setting a default value is an important aspect of coverage, described in [More on decision authoring: Coverage and rule interactions](#).

For data nodes, you can set default values for the following purposes:

1. The data handled by the node is optional, and might not always be provided by the client application. For example, if the customer's country is USA for almost all transactions, the client application might only provide this information if the country is different.
2. For convenience, the validation feature uses the default value if no value is provided in the input data set. This can reduce your data entry in the data sets that you create.

The following examples show how to set the default value on a data node:

- On a system data type:

```
set 'the gender' to "Female";
```

- On a custom data type:

```
set 'the loan' to a new loan where  
  the amount is 100000,  
  the monthly repayment is 688;
```

Lists

You use lists when your decision acts on more than one item of something, and you need to make a distinction between the items. For example, when you check-in baggage on a plane:

- You use a list if your decision is about checking-in oversized baggage, since more than one of the baggages may be oversized.
- You do not use a list if your decision is about the fee charged, since there is only one fee for the sum of the baggages checked-in.

In the diagram, you specify whether a data node, decision node, or custom type is a list using a check box.

When checked, constructs to handle lists are then available in your rules, such as:

- there are at least <number><items> in <list>
- for each <item> in a <list>
- ...

See [Decision Center modeling language reference](#) for the available constructs.

Parent topic: [Modeling decisions in the Business console](#)

More on decision authoring: Coverage and rule interactions

Authoring the decision logic requires you to create the business rules, but also to consider coverage and rule interactions.

Note: This topic builds further on the information presented in [Decision modeling basics](#), and assumes that you are now familiar with decision nodes, data nodes, decision logic, and how these work together.

You author the logic of each decision node in your diagram by creating one or more *business rules* that:

- Evaluate the input data that each decision node receives against conditions that you establish.
- Take actions to set or modify the output of the decision node, that is, the **decision**.

Creating rules and decision tables is essentially the same for a decision model service as it is for a regular decision service:

- For rules, see [Action rules](#).
- For decision tables, see [Decision tables](#).

See [Decision Center modeling language reference](#) for the available constructs.

When creating rules or decision tables in a decision model service, the only information available for making a decision are the values of the direct predecessors of the corresponding decision node. You cannot use a rule to change these variables or those from another dependency.

When a rule tries to modify an input value, Decision Center creates a copy of the data so that the input value is never modified.

Coverage

When you author a decision node, you create the rules required to reach a decision for the possible cases that the client application might provide as input data. In practice, it is usually not possible to consider all possible cases, even for a simple decision.

An easy way to make the decision logic complete consists in specifying a default value for the decision. The default value applies if no other rule has made a decision, and is always the last rule to execute.

This default value is just a fall-back measure and should not prevent you from completing the decision logic in an informed and deliberate way.

You specify the default value as an action of the form:

```
set decision to <value>
```

If the node uses multiple values enabled by lists, use the form:

```
add <value1> to decision
add <value2> to decision
...
```

Rule interactions

For each decision node, you select a rule interaction policy that applies to all the rules and decision tables in this decision node. Interaction policies define how rules interact with each other by governing their order of execution. It is up to you to decide which of these rule interaction policies should be applied.

Int era cti on pol icy	Description
Seq uen tial	The default policy, in which rules execute in the order in which they are listed in a decision node. A rule can modify or overwrite decisions that were previously made by other rules.
Firs t rule app lies	Rules execute in the order in which they are listed in the decision node, but as soon as a rule makes a decision, this decision is final. Do not use this policy if the decision is a list that results from multiple instances of one or more rules.
Sm alle st and gre	The value of the decision is set to the minimum value available among the values that are obtained by the applicable rules. If you use the greatest value policy, it is set to the maximum value available. If no rule applies, the value of the decision is null. You can use these policies for decisions of type number and integer, and that use the set decision to construct.

ate st val ue	
Coll ect	The values of all applicable rules are collected and returned as a list. The execution order of the rules does not influence which values are collected, but it impacts the order in which the collected values appear in the list. If no rule applies, an empty list is returned. You can use the collect policy for decisions that are multi-valued, and that use the add . . . to decision construct.
Su m	The values of all applicable rules are summed up. If no rule applies, the value of the decision is 0. You can use the sum policy for decisions of type number and integer, and that use the add . . . to decision construct.

Parent topic: [Modeling decisions in the Business console](#)

Deploying a decision model service

You deploy a decision model service to an execution environment using a deployment configuration, the same way as you deploy a regular decision service.

Deploying from Decision Center is described in [Deploying from the Business console](#). You can also automate rule deployment by using the [Decision Center REST API](#).

When you deploy a decision model service, note the following:

- For your convenience, Decision Center creates a deployment configuration when you create a decision model service.

This deployment configuration deploys to a non-production server, and does not give any groups the right to deploy. Users with release manager or permission manager rights can edit the deployment configuration, for example, to enable specific groups of users to deploy with the deployment configuration.

- Decision Center creates a **decision operation** and automatically refreshes its content each time you save. You cannot edit the decision operation or create another one, because a decision model service references all the rules of the decision model, and uses the input/output parameters of the diagram.

Note: You can review the decision operation details in the deployment configuration read-only view: select the **Operations** tab, and hover your mouse over the operation name.

Parent topic: [Modeling decisions in the Business console](#)

Validating and testing decision model services

With a decision model service, you can validate your decision model during development. You can also create more complete usage scenarios that you use in test suites and simulations, just like regular decision services.

Each time that you save a new version of your decision model, Decision Center compiles the content of your decision model and displays any errors or warnings that you must fix.

Decision Center also provides a **Validate** feature for decision model services, to rapidly try out your decision model when you are defining it. At each step, you provide data to ensure that your model returns the results you expect. Decision Center runs your data on a local embedded engine. You provide the data in a friendly form or in its underlying JSON format.

Test suites and simulations

Decision model services behave the same way as regular decision services for test suites and simulations.

- [Testing sets of rules in the Business console](#) describes how to setup test suites that run rules, on an execution environment, against usage scenarios. You setup test suites after your decision model is complete, for regression testing.
- [Simulating business application results](#) describes how to create simulations to see how changes to rules or data affect the results of your decision model service, based on key performance indicators.

There are some differences to consider, because the decision model is considered as one artifact:

- You run test suites or simulations on the entire decision model, because you cannot create a decision operation that defines a subset of rules.
- When you configure a test suite or generate a scenario file, the only execution detail that can be included in the report is the duration of execution.

Parent topic: [Modeling decisions in the Business console](#)

Using queries

In the Business console, you use queries to search for elements of your projects. You can apply actions to the results of the query.

With queries, you can filter business rules or other project elements. You can perform actions to all the results of a query, or only to specific results. For example, if you have a modification to apply to a certain type of rule, you can create a query that searches for the rules corresponding to this type and modifies them. You can also create queries to find which rules have been affected by any modification you made.

You can also use queries to extract rules for generating a ruleset. To use queries as an extractor, you must define the query to use in a decision operation.

Queries are made up of two parts:

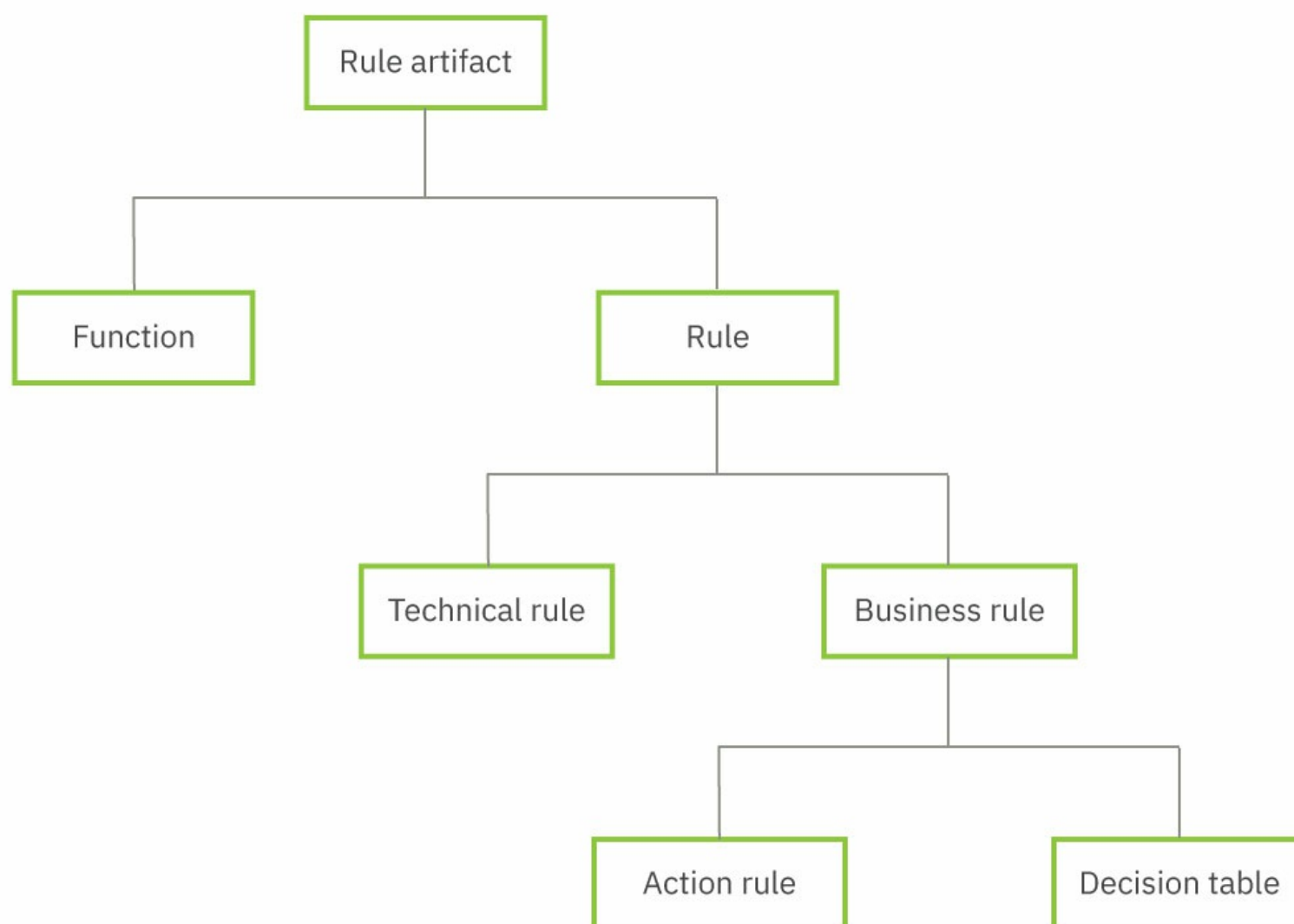
- The **query** part, for example, Find all business rules such that the creator of each business rule is me
- Optionally, an **action** part, for example Do set the status property to deployable

You construct queries in the same way that you construct rules, with a condition, and optionally an action. The default query action is to display the results, but you can also have the query carry out other actions such as moving the results of the query to another folder.

Queries are run against the current project, as well as any linked projects if you specify this option. Queries apply to the current branch, release, or change activity where you run the query.

To search for specific types of rule artifacts you must use the correct rule type in the query part, according to the following rule hierarchy:

Figure 1. Rule hierarchy



For example:

- Find all rules returns all action rules, decision tables, and technical rules.
- Find all business rules returns action rules and decision tables.

To run a query on functions, you must explicitly state it in the query part: Find all functions.

Query conditions

Query conditions are the criteria that you define for the search.

Query actions

The default action of a query is to display the query results, but you can add further actions by specifying

them under the **Do** part of the query.

Creating and running a query

Create and run a query on the **Queries** tab. You can also save your query, view its results, and apply actions to the query results.

Parent topic: [Working with the Business console](#)

Related concepts:
[Decision operations](#)

Query conditions

Query conditions are the criteria that you define for the search.

The conditions of your query appear under the Find part of the query.

By default, you run queries on the business rules but you can also query other types of project elements, such as folders (rule packages), ruleflows, variable sets, and so on.

You start by selecting the type of project element you want to query. You can then refine the query by adding a filter in the form of a `such that` statement, followed by condition statements.

Example

```
Find all business rules
  such that the creator of each business rule is me
  and the creation date of each business rule is after 3/23/2016
```

You can refine queries using a `such that` statement to filter different project items:

- [Filter by properties](#)
- [Filter by business terms](#)
- [Filter by behavior of rules during execution](#)
- [Filter by impact of rules during execution](#)

Note:

You can identify which rules have changed in the current session (by querying on `after my login time`) or which rules you have changed (by querying on `creator is me`).

Filter by properties

You can filter the query according to one of the properties of the project elements being queried (see [Rule properties](#)).

For example, the following query displays only business rules whose status is new:

```
Find all business rules
  such that the status of each business rule is new
```

Filter by business terms

You can filter the query according to the business terms used in the conditions and actions of your rules. This type of filter is useful when trying to find rules affected by a policy change, for example:

```
Find all business rules
  such that each business rule modifies the value of the insurance rate
```

The following query constructs find rules that use or modify business terms:

uses the value of

Returns rules that use the values of certain business terms.

Example

The following query returns all business rules that use loan grade values:

```
Find all business rules
  such that each business rule uses the value of the loan grade in 'a
report'
```

uses the phrase

Returns rules that call a given business term.

Example

The following query returns all business rules that use 'add to the corporate score in the loan report':

```
Find all business rules
such that each business rule uses the phrase [ add 'a number' to the
corporate
score in 'a report' ]
```

uses the phrase ... where

Returns rules that call a given business term to which a constraint is applied

Example 1

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can equal 100:

```
Find all business rules
such that each business rule uses the phrase [ set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100 ]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

Example 2

The following query returns all business rules that call the term that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a
number' to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When the elements mentioned in the constraint do not appear in a rule, the rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query would return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10
to 'the borrower';
```

modifies the value of

Returns rules that modify certain business terms.

Example

The following query returns all decision tables that modify the value of the insurance rate:

```
Find all decision tables
such that each decision table modifies the value of 'the insurance rate'
```

Filter by behavior of rules during execution

You can filter the query according to the semantics of the rules, that is, their behavior during execution. This type of filter finds rules that could be applicable when certain conditions are true or become true. This filter is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can use the following semantic query constructs:

may apply when

Returns all rules whose condition part could meet the query condition, or where there is nothing in the rule condition that would contradict the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

Example

Table 1. List of example rules

Rule	Rule content
Rule 1	If the score of the borrower is at least 10 then...
Rule 2	If the age of the borrower is at least 21 then...
Rule 3	If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

Query 1:

*Find all business rules
such that each business rule may apply when the score of the borrower is 20*

This query returns Rule 1 and Rule 2. It returns Rule 1 because if the score of the borrower is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the score of the borrower could be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

Query 2:

*Find all business rules
such that each business rule may apply when the score of the borrower is 5*

This query returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to less than 10. In Rule 2, nothing specifically stops the rule from being applicable when the score is 5. In Rule 3 at least one of the conditions could apply, and there is nothing in the other condition that negates the fact that the score could be 5.

may become applicable when

A more specific query that returns only those rules whose condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition.

Example

Table 2. List of example rules

Rule	Rule content
Rule 1	If the category of the customer is Platinum then...
Rule 2	If the category of the customer is not Platinum then...
Rule 3	If the age of the customer is at most 65 then...
Rule 4	If the age of the customer is at most 65 and the category of the customer is not Platinum...

Query 1:

*Find all business rules
such that each business rule may become applicable when the category of 'a customer' is Gold*

This query returns Rule 2 and Rule 4. It returns Rule 2 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 4 for the same reason. The additional condition relating to the age of the customer does not contradict the condition in which the category may be Gold.

The query does not return Rule 1 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 3 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer category is Gold.

Query 2:

*Find all business rules
such that each business rule may become applicable when [the age of*

'a customer' is at least 21]

This query does not return Rule 3 because it searches for rules that could “become” applicable when the customer age is over 21. Rule 3 is applicable even if the customer age is under 21. Therefore Rule 3 does not “become” applicable, but it “remains” applicable even if the customer age changes from under 21 to over 21.

may lead to a state where

Returns rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of a rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

Example

Table 3. List of example rules

Rule	Rule content
Rule 1	If the age of the borrower is at least 25 then set the credit score of the borrower to 60
Rule 2	If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

Query:

Find all business rules
such that each business rule may lead to a state where the credit score
of
the borrower is more than 50

This query returns Rule 1 but not Rule 2 because, when the age of the borrower has been checked and the credit score set, only Rule 1 shows a result of over 50.

Filter by impact of rules during execution

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule impacts the applicability of other rules, and how a rule is impacted by the execution of other rules.

may select

Returns the ruleflows and rule tasks that may select a given rule.

Example

The following query returns the ruleflows and rule tasks that may select "Rule 1".

Find all ruleflows
such that each ruleflow may select "Rule 1"

may enable

Returns rules that make a given rule applicable.

Example

Table 4. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 25 then set the category of the customer to Bronze ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;

Query:

Find all business rules
such that each business rule may enable "Rule 2"

This query returns Rule 1 because it sets the category of the customer to Bronze, and thus makes the condition of Rule 2 valid.

may disable

Returns rules that make a given rule inapplicable.

Example

Table 5. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 18 then set the category of the customer to Copper ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;
Rule 3	if the age of the customer equals 25 and the category of the customer is Bronze then set the category of the customer to Diamond ;

Query:

*Find all business rules
such that each business rule may disable "Rule 2"*

This query returns Rule 1 and Rule 3. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It returns Rule 3 because although one of the conditions is that the category of the customer is Bronze, the action is to set the category of the customer to Diamond. Therefore, if the category of the customer is Diamond, Rule 2 is not applicable.

may be enabled by

Returns rules that are made applicable by a given rule.

Example

Table 6. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 18 then set the category of the customer to Copper ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;
Rule 3	if the age of the customer equals 25 and the category of the customer is Gold then set the category of the customer to Diamond ;

Query:

*Find all business rules
such that each business rule may disable "Rule 2"*

This query returns Rule 1. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It does not return Rule 3 because the action is to set the category of the customer to Diamond, and it can only be executed from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

may be disabled by

Returns rules that are made inapplicable by a given rule.

Example

Table 7. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 25 then set the category of the customer to Bronze ;
Rule 2	if the age of the customer equals 25 and the category of the customer is Copper then set the category of the customer to Gold

Query:

*Find all business rules
such that each business rule may be disabled by "Rule 1"*

This query returns Rule 2 because it is made inapplicable by Rule 1. In Rule 1 the category of a customer whose age is 25, is set to Bronze, therefore it cannot be Copper.

Parent topic: [Using queries](#)

Query actions

The default action of a query is to display the query results, but you can add further actions by specifying them under the **Do** part of the query.

Query actions can make the following changes:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add or remove a category to the displayed elements
- Set any of the properties of the displayed elements to a value that you specify

Example

```
Find all business rules  
such that the status of each business rule is new  
Do set the status of each business rule to validated
```

By default, the actions are carried out on all the displayed elements when you click **Apply Actions** in the **Query results** tab. You can select specific elements among the results, and apply actions only to these elements.

Parent topic: [Using queries](#)

Creating and running a query

Create and run a query on the **Queries** tab. You can also save your query, view its results, and apply actions to the query results.

Procedure

1. Open your decision service, and click the **Queries** tab.
2. In the **Queries list** subtab, select the project in which you want to run your query.
3. In the field **Query expression**, type your query. You can add information in the **Description** field.

If you want to save this query, click **Save**. The query is added to the list of queries under the project. You can run a query without saving it first. If you click another project or query in the queries list, you must confirm that you want to discard your current query.

4. Click **Run**. The query results display in a subtab **Query results**.

When your query completes, you can apply actions if the query has an action part. You can review the results before you decide to apply the actions.

5. In the toolbar above your query results, click the icon **Apply the query actions to the selected elements** to perform the actions from the query. By default, all the elements retrieved by the query are selected, so the action is performed on all of them. If you select certain project elements among the results, the action is performed only on the selected project elements.

Parent topic: [Using queries](#)

Testing sets of rules in the Business console

Test the rules that you create or edit to achieve the results that you expect.

[Testing in the Business console](#)

The Business console provides capabilities to test sets of rules against usage scenarios.

[Understanding the reports](#)

When you run a test suite, it produces a large amount of information and places the relevant information in a report.

[Excel scenario files](#)

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

Parent topic: [Working with the Business console](#)

Testing in the Business console

The Business console provides capabilities to test sets of rules against usage scenarios.

During rule development, you can create and run test suites in change activities or ungoverned branches in a decision service. You can also run test suites in validation activities to test the content of a release before deployment.

To understand how testing works, consider what you can change in a decision service:

- The **set of rules** that you are testing. The set is identified by the **decision operation**, which defines a subset of rules within the release or change activity on which the testing is being done (see [Identifying a set of rules](#)).
- The **scenarios** that you are submitting. Each scenario contains all the information that is required to process a transaction. This information can be real or fictitious data. You store this scenario data in a scenario file, which you generate, complete with data, and then upload to Decision Center. You also use this scenario file to store the **expected results** for each scenario.
- The **test suite** that you create and run to provide feedback on the performance of a set of rules without changing the rules or the applications on which they act.

Running a test suite compares the results that you expect to the actual results that are obtained from applying rules to your scenarios. The test generates a report that shows the results for each scenario, and the success rate as the percentage of scenarios that generated their expected results.

Note: If you do not see the option to select an operation when you create a test suite, you can go back to the **Test** tab and select the operation from the drop-down list in the upper-right corner.

Scenarios

Scenarios represent real or fictitious use cases that you use to validate the behavior of your rules. Each scenario contains all the information that is required for your rules to run properly.

For example, a loan application might require the following information, which must be completed for each scenario:

- Borrower: Sam Adams
- Credit Score: 600
- Yearly Income: 80000
- Duration: 24 (months)
- Amount: 100000
- Yearly Interest Rate: 5 (%)

You can generate scenarios in an Excel format. Each row represents a scenario, and the columns indicate where to put the data for each scenario. For example, the following Scenarios sheet contains four scenarios that validate loan risk from a very low risk loan to a loan where the amount is too high:

		the borrower					the loan		
Scenario ID	description	first name	last name	b	S	c	y	start date	nu amount
Very low risk	Very low risk loan accepted	Sam	Adams	4	95			8/1/2009	# 100000
Low risk	Low risk loan accepted	Bob	Schwartz	4	94			8/7/2010	# 500000
Average risk	Average risk loan accepted	John	Johnson	4	85			9/1/2010	# 900000
Amount too high	Loan rejected when amount too high	Mary	Doe	4	32			8/15/2010	# 1100000

Scenarios Expected Results HELP

Test suites

You run test suites to verify that your rules are correctly designed and written. The test suites compare your expected results with the actual results that come from applying your rules against the scenarios that you defined.

You set up the expected results in a separate sheet alongside your Scenarios sheet.

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Scenarios Expected Results HELP

The example tests two different aspects of the loan request:

- Whether the loan is approved.
- Which message the loan request application generates.

The Expected Results sheet represents these two tests as columns. You specify the expected results for all the scenarios necessary to cover the validation of your rules.

The test suite returns a report that compares your expected results with the actual results of the execution. Each test in a test suite is successful if all the expected results match the actual results.

You can include an Expected Execution Details sheet for more technical tests, for example, to list the rules that are run or the duration of a run.

Note: When you import a test suite from the release in a validation activity, this test suite is considered as a reference, and is locked in all change activities and validation activities of this release.

Parent topic: [Testing sets of rules in the Business console](#)

Understanding the reports

When you run a test suite, it produces a large amount of information and places the relevant information in a report.

All test reports contain the following information:

- **Summary:** Shows the scenario success rate as a percentage of scenarios that ran successfully. It also shows information about the test, including the scenario file, decision operation, and rule source. Links are provided to rule and data sources.
- **Results:** Shows the results from running the test. It lists the scenarios in the scenario file, and the status and result of each scenario. If you ran your test on collections, you can click **Details** to obtain more information on the observed and expected results.

The results for the scenarios use the following statuses:

- **Successful:** A test is successful when the expected results match the actual results.
- **Unsuccessful:** A test is unsuccessful when the expected results are different from the actual results.
- **Error:** An error is reported when the test cannot run the scenarios, for example, when an entry in the scenario file is not correctly formatted.

Parent topic: [Testing sets of rules in the Business console](#)

Excel scenario files

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

An Excel scenario file can contain the following sheets:

- Scenarios: Contains the input data for scenarios. Both testing and simulation scenario files have this sheet.
- Data entry: Regroups information that is used in other sheets.
- Expected Results: Holds the results that you expect from tests.
- Expected Execution Details: Holds the execution details that you expect from tests.

Important:

Never modify the column structure of the sheets.

Scenarios sheet

Each row in the Scenarios sheet represents one scenario. Each scenario must contain all the information needed to process a transaction.

		the borrower					the loan		
Scenario ID	description	first name	last name	b	S	c	y	z	
Very low risk	Very low risk loan accepted	Sam	Adams	4			95	8/1/2009	# 100000
Low risk	Low risk loan accepted	Bob	Schwartz	4			94	8/7/2010	# 500000
Average risk	Average risk loan accepted	John	Johnson	4			85	9/1/2010	# 900000
Amount too high	Loan rejected when amount too high	Mary	Doe	4			32	8/15/2010	# 1100000

The columns indicate the information that you must provide for each scenario:

- **Scenario ID:** The name identifies the scenario and must be unique.
- **description:** A free text cell to describe the scenario.
- **business terms:** Values for the scenarios. Bold columns or subcolumns are mandatory.

Note:

The reduced rows between the column headers and the first scenario contain information that might be useful to developers. Never edit these rows.

The following visual aids help you complete your scenarios:

- Red triangle: When shown in a column header, it indicates the presence of a comment that explains the type of values to be entered, such as text, numbers, dates, or true or false. Hover over a triangle to display a comment as follows:

the borrower				
first name	last name	credit score	yearly income	
John	Smith	600	80000	
John	Smith	600	80000	

- Dark green triangle: When shown in a cell, it suggests a better format for the cell. For example, if you enter a numerical value in a text cell, Excel suggests that you format the cell for numerical values. To make sure that the scenario runs correctly, ignore this suggestion.
- Arrow: When shown before a column name, it indicates that the values must correspond to entries that are specified in a separate **data entry** sheet.

Data entry sheets

You create data entry sheets when you generate a scenario file. Data entry sheets regroup data to be used in other sheets.

The name of each data sheet serves as the type of data that is expected in the column of the other sheet that uses the data. For example, in the following figure, the address sheet **1** contains different addresses that are used in the addresses column **2** of the Scenarios sheet.

Entry ID	Street	City	State	Zip Code
billing address	2207 7th avenue	New York	NY	10027
shipping address	25 Elm Street	Dallas	TX	75201
personal address	110 Cactus Street	Phoenix	AZ	85003

1

	the borrower						
description	first name	last name	credit score	yearly income	amount	→ addresses	st
Loan rejecte	John	Smith	600	80000	500000	personal address	
Loan approve	John	Smith	600	80000	25000	billing address	

Enter the name of an object defined in the sheet address

Note:

A column that uses values from a data entry sheet has an arrow in its column header.

You create an entry in a data entry sheet by completing a row that includes a unique name to identify the entry in the other sheets.

Expected Results sheet

The Expected Results sheet contains the results that you expect to obtain when you run tests that use the scenarios.

You can test many aspects of a set of rules, but the Expected Results sheet contains only the tests that you specified when you generated the scenario file. For example:

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Each blue column in the Expected Results sheet corresponds to a test. If you do not enter a value in a column, the corresponding test is skipped.

Tip:

You cannot create a new column in Excel. However, you can generate another empty scenario file template, and then copy and paste a column from it.

You link the sheets that contain the scenarios and their expected results by entering names in their respective Scenario ID columns. In addition to making sure that the names match between the two sheets, it is good practice to keep the scenarios and their expected results in the same order.

When you test for a list of values, you enter each item in the corresponding cell on a separate row. You do not have to duplicate the Scenario ID for each row.

Expected Execution Details sheet

The Expected Execution Details sheet contains the execution details that you expect to obtain when you run tests that use the scenarios.

The following are examples of execution details:

- List of rules fired
- List of executed ruleflow tasks
- Duration of execution

You can test many aspects of a run, but the Expected Execution Details sheet contains only the tests that you specified when you generated the scenario file. Each green column in the Expected Execution Details sheet

corresponds to a test.

You link the sheets that contain the scenarios and expected execution details by entering names in their respective Scenario ID columns.

When you test for a list of values, you enter each item in the corresponding cell on a separate row.

4			
5		Scenario ID	the list of fired rules contains
9		Big Loan	eligibility.checkIncome
10			eligibility.approval
11			eligibility.checkCreditScore
12		Small Loan	
13			
<div>▶▶ Scenarios Expected Execution Details HELP</div>			

You do not have to duplicate the Scenario ID for each row because the last ID entered is used.

Parent topic: [Testing sets of rules in the Business console](#)

Simulating business application results

Run rules on representative data to generate results that you can use to improve the application.

Overview: Simulations in the Business console

You can determine how changes to rules or data affect the results of your business rule application before deployment.

Workflow to create and run simulations

The Business console takes you through the process for assembling and running a simulation.

Parent topic: [Working with the Business console](#)

Overview: Simulations in the Business console

You can determine how changes to rules or data affect the results of your business rule application before deployment.

You run a simulation on a decision service in a change or validation activity in a release, or an ungoverned branch. You can use fictitious data or real data from your organization, and display the results in a report defined by you. You can then use the results of the simulation to refine the behavior of your business rules.

The simulation brings together data, rules, and key performance indicators (KPIs). You select the data and rules for the simulation, and you define the KPIs by using metrics that take values from the rules.

The KPIs represent the results of the simulation. You define a simulation report by adding KPIs in a report template, and then defining their appearance. You can display KPIs as text, or in graphs to make their results easier to understand.

Versions of the simulation artifacts are saved, and the simulation runs the most recent versions of its artifacts. The Business console maintains a history of simulations, and you can rerun an old version of a simulation by restoring it.

You can change the elements in a simulation, and rerun the simulation to assess the results of the changes. You can also compare reports from different runs side by side in the console. Through this iterative approach, you can refine your rules to get the results that you want from your business rule application.

Change and validation activities

You can run simulations in the change and validation activities of a release in a decision service. In a change activity, you can use simulations to iteratively refine rules, and in a validation activity, you can use simulations to check the rules that were approved for the release.

Both types of activities use the same artifacts, that is, metrics, KPIs, scenarios, report templates, and simulation configurations. The activities within a release also share resources such as servers and decision services.

You can define artifacts in both types of activities. You can also import a simulation from a completed change activity into the validation activity of the same release. When you do so, all the artifacts in the simulation are imported, giving you a ready-to-run simulation in the validation activity. Importing a simulation saves you time in setting up a simulation in the validation activity, and you can apply a simulation that was used in developing the rules. To use a simulation from a change activity, the changes that were made in the activity must be merged into the release before the validation activity can import the simulation.

The two types of activities handle reports differently. In a change activity, reports are listed in the **Reports** tab, and a new entry is added each time that you run a simulation. The validation activity does not have a **Reports** tab. When you run a simulation in the validation activity, the report is listed with the simulation in the **Simulations** tab, and if you run the simulation again, the last report is replaced with a new report.

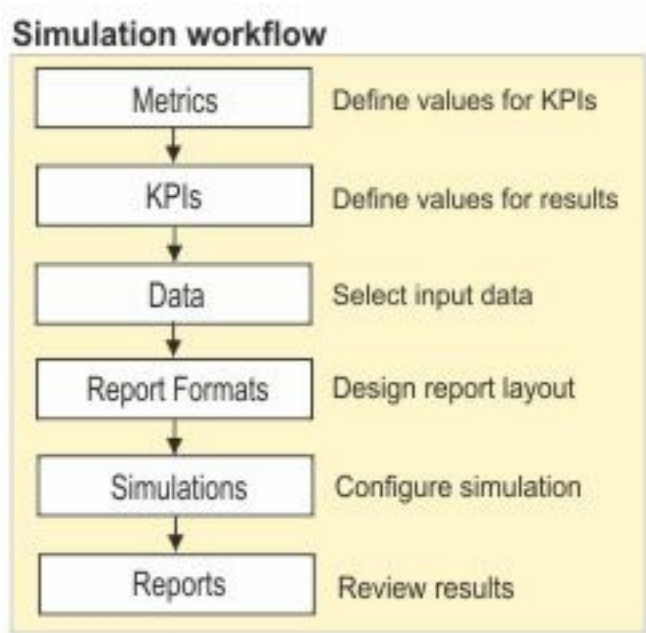
The validation activity lists the five most recent reports from different runs of a simulation in **Recent reports for this version**. You can compare simulation reports in a change activity or a validation activity, or you can compare a simulation report in a validation activity with a simulation report from a change activity.

Parent topic: [Simulating business application results](#)

Workflow to create and run simulations

The Business console takes you through the process for assembling and running a simulation.

To run a simulation, you must assemble elements that include metrics, key performance indicators (KPIs), and a scenario file. The simulation interface is organized to take you through a workflow:



Metrics

Metrics define the values that are used in the KPIs. You must create metrics before you can make KPIs.

You define a metric by using a business model language. For example, you might define a metric that is named `Loan amount` by making the metric expression the `amount` of `'the loan'`.

The simulator provides the following types of metrics:

- **Numeric:** Uses number values, for example, the `age` of `'the borrower'` or the `amount` of `'the loan'`.
- **DateTime:** Uses date and time values, for example, the `birth date` of `'the borrower'` or the `start date` of `'the loan'`.
- **Boolean:** Checks whether a value is true or false, or matches a specific group, for example, the `corporate score` of `'the loan'` is at least 10.
- **String:** Checks for text values, for example, the `zip code` of the address of `'the borrower'`.
- **Domain:** Uses predefined values in metric expressions, for example, for the expression the `category` of the customer, a domain metric might limit the category values to `Platinum`, `Gold`, and `Silver`. In this case, a simulation uses only the input data that matches the predefined values.

You must define conditions in metric expressions. You cannot create conditional KPIs. To add a condition, use `when` in your metric expression, for example, the `amount` of `'the loan'` when `'the loan'` is approved.

You can provide a default value for when a condition is not met. For example, you can set the default value of the metric `Age of Borrower` to 0. If a set of data does not meet the condition statement of the metric, a KPI that uses the metric can display 0 as its result in a simulation report.

If you do not specify a default value or select **Undefined** for Boolean types, the metric condition is not defined. Do not specify a default value when you want the KPI to determine the value. However, leaving the default value undefined is not equivalent to 0.

When you create an element for a simulation, you must select a decision operation. The **Select an operation** window displays a list of the available operations in the decision service. Each entry includes a description of the operation. The operations contain the rules that you can use in the simulation. You can select only one operation for a simulation.

Important: When you create a metric or a KPI, you cannot use a one-word verbalization such as `age`. You must use a phrase such as `age of the borrower`. A short, implicit verbalization generates an error.

KPIs

KPIs show the results from running a simulation. To define a KPI, you pair a metric with a KPI value. For example, to create a KPI that shows the total amount of the loans that are handled by a loan application, you might use the KPI expression `sum of 'Loan amount'`. When the simulation runs, it adds up the amounts of the loans and displays the total in the simulation report.

You can create two types of KPIs:

- **Scalar:** Shows a single value that was taken from a group of values. For example, in a loan validation application, you might have a KPI value for the sum of all the loan amounts.
- **Grouped:** Shows a set of values. For example, a simulation might group the average loan amounts for a set of customer categories. In this case, the report can display the set of values in a graph such as a pie chart.

A KPI can use either one or two metrics. When you use two metrics, the second metric defines a distribution of

values.

The provided KPIs cover common values. Contact your system administrator to add more KPI values.

Data

Simulations use business information that is provided as input data. To run a simulation, you must use an Excel scenario file.

You can use either real or fictitious data. The Excel format displays data clearly, and can be changed easily.

You can generate and download an Excel data file from the Business console, add data to it locally, and then upload it back to the console.

Scenarios

Scenarios represent business use cases. They can contain real or fictitious data. You use scenarios to validate the behavior of your rules. When you use a group of scenarios in a simulation, each scenario must contain all the data that is needed to run the rules.

In an Excel file, you import the scenarios in a table:

	the loan						the borrower						
description	start date	numb	amount	Loan	mon	yearly	first na	last na	birth date	SSN	credit	yearly in	zip code
Loan approved	1/1/2014	240	100000	0.2		0.04	John	Doe	1/1/1980	1234 600	70000	75001	
Loan rejected	1/2/2014	120	200000	0.2		0.04	John	Doe	1/2/1980	1234 600	70000	75001	
Loan rejected	1/3/2014	240	100000	0.2		0.04	John	Doe	1/3/1980	1234 600	70000	75001	

Scenarios

HELP

+

Each row in the table represents a complete scenario. Typically, the more scenarios, the more accurate your results.

To use the scenario data file, you must import it into the Business console.

Important:

You can use scenario data files that are made to test applications (see [Testing sets of rules in the Business console](#)). However, you cannot test business applications with data files that are made specifically for simulations. The data files for testing contain expected results. Simulations do not use expected results and ignore them in data files for testing.

Report formats

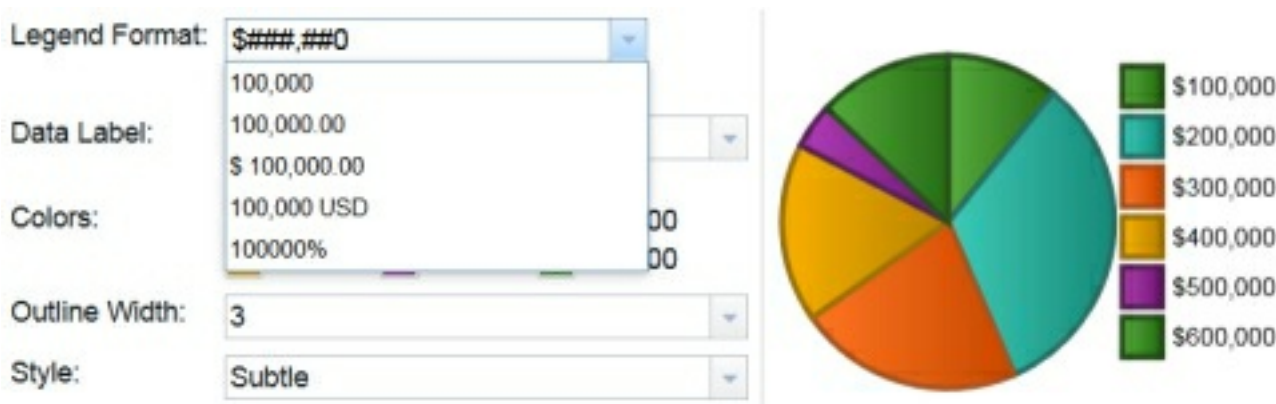
When you run a simulation, it displays its results in a report. You can define various reports for a simulation to generate different reports.

You lay out a report in a template by adding sections and KPIs. You can add as many elements as necessary.

A report can format KPIs as follows:

- Text: When a KPI produces a single value from a set of data, you display the value as text. You can configure the appearance of the text in the simulation report. The formatting options include fonts, font sizes, and colors.
- Graph: When a KPI produces a set of comparable values, you can display the values in a graph to highlight their differences. The simulation feature includes four types of graphs: bar, line, area, and pie chart. You can format a graph, including the title, colors, and axis labels.

When you format a report, you can select descriptors that match the values of the KPIs, for example:



Simulations

You configure a simulation to bring together input data, an application server, and a report format. When you run the simulation, it uses rules from the associated decision service and places the results in a report.

You can interrupt the generation of a report during a simulation. If you do, the report is still created, but it contains only partial results.

Note:

The simulations run on application servers that are defined by system administrators in the Decision Center Enterprise console.

Reports

When you run a simulation, it generates a report that you use to evaluate the results.

You can compare two reports side by side in the Business console. You can also access different parts of a simulation from a report. For example, you can go directly to the rules, change data sources, and modify the report format. When you change aspects of a simulation from a report and run the simulation again from the **Simulations** tab, a new report is generated.

Note: In validation activities, the simulation reports are listed in the **Simulations** tab.

Parent topic: [Simulating business application results](#)

Deploying from the Business console

In a decision service, you can deploy a set of rules to a production environment or to a non-production environment for testing or quality assessment.

You deploy from a decision service by using a deployment configuration. Any user who can access a decision service in the Business console can deploy its branches (release, change activity, regular branch) from any deployment configuration available to that user in that branch. However, the following conditions apply:

- In a change activity, you can deploy only with a deployment configuration whose type is non-production.
- In a release, you can deploy with a deployment configuration whose type is production only if the release is completed.

Note: For these two cases you can deploy to a RuleApp archive.

Only a user with configuration manager or administrator rights can create, edit, or delete deployment configurations.

Only a user with Permission Manager rights can create, edit, or delete deployment configurations.

This ensures a basic level of security on deployment. These users define deployment configurations with information that includes the target servers, the decision operations that define the business rules, and settings for versioning of rulesets and RuleApps. The following aspects of a deployment configuration relate to making sure that deployment is secure:

- The configuration type as production or non-production. This has the effects noted above.
- Which of the existing target servers this deployment configuration can deploy to. When deploying, the list of possible servers is then presented as a selectable option.
- Users belonging to which groups can deploy with the configuration.

When working within the governance framework, a deployment configuration can only be created or edited within a change activity.

The deployment configuration contains a setting to establish if a deployment snapshot is required when deploying. Deployment snapshots can be found in the list of snapshots. You can then redeploy from the deployment snapshot or from the report of the deployment.

[Deployment configurations](#)

You use deployment configuration in a decision service branch (release, change activity, or regular branch) to deploy sets of rules.

[Redeploying](#)

You can redeploy a decision service branch from a deployment snapshot or from the original report of the deployment.

[Version policies](#)

Version policies determine how client applications identify deployed rulesets.

Parent topic: [Working with the Business console](#)

Related concepts:

[Decision services](#)

[Governance principles](#)

Deployment configurations

You use deployment configuration in a decision service branch (release, change activity, or regular branch) to deploy sets of rules.

Each deployment configuration contains all the information required to deploy rulesets, including the target server. When working within the decision governance framework, creating and editing a deployment configuration must be done within an open change activity.

Note: You need Permission Manager rights to create and edit deployment configurations.

A deployment configuration is organized in five sections:

General

Contains the name and description of the deployment configuration, and indicates the project in which it is stored. Also contains the name and the base version number of the deployed RuleApp. Also contains properties for the RuleApp.

You set the configuration type to *Nonproduction* or *Production*. Typically, a nonproduction configuration is used in developing and testing a decision service. A production configuration is used to deploy a finished release to a production environment.

Operations

Lists the decision operations that are available in the decision service, and those selected for deployment.

With an initial deployment, you typically deploy the available decision operations. In updates, you might limit deployment to specific decision operations.

Note:

When selecting decision operations, be careful to avoid conflicts during deployment. Make sure that every ruleset name in the deployment configuration is unique.

Targets

Lists the Rule Execution Server instances that are available in Decision Center, and those selectable when deploying. The deployment process packages the rulesets in the decision service into a RuleApp archive that is sent to each selected server.

If no servers are available, you can still use the deployment configuration to create a RuleApp archive that can be saved locally.

Ruleset Properties

Contains the base version number to be used in the ruleset path. You also set the version policy for calculating the version numbers that are used on successive deployments.

You use the Advanced properties section to configure individual rulesets. You select a ruleset and choose options for enabling, debugging, and tracing the ruleset. You can also select and define properties that control in detail how the ruleset executes in Rule Execution Server.

Groups

You select which user groups can see this deployment configuration.

Parent topic: [Deploying from the Business console](#)

Related concepts:

[Administering projects from the Business console](#)

Redeploying

You can redeploy a decision service branch from a deployment snapshot or from the original report of the deployment.

About this task

When you deploy a decision service, settings are available in the deployment configuration to create a deployment snapshot. Deployment snapshots are visible in the **Snapshots** tab. You can use the deployment snapshot to redeploy at a later time.

You can also redeploy from the original report of the deployment, available in the **Reports** section of the **Deployment** tab.

To repeat a decision service deployment, you use its deployment snapshot.

Procedure

1. In the **Snapshots** tab of your release, change activity, or branch, locate the deployment snapshot.
2. In the drop-down list next to the name of the deployment snapshot, click the **Redeploy** icon. The deployment dialog opens.
3. Click **Deploy** to deploy the deployment snapshot.

Note: If you originally used the deployment to generate an archive file, you can use the deployment snapshot to generate only another archive file.

Parent topic: [Deploying from the Business console](#)

Related concepts:
[Snapshots](#)

Related tasks:
[Creating a snapshot](#)
[Viewing the timeline of a release or activity](#)

Version policies

Version policies determine how client applications identify deployed rulesets.

In a deployment configuration for a decision service, you establish the version policies to define the paths of the rulesets in the RuleApp deployment. The ruleset path determines how an application calls a ruleset through Rule Execution Server.

The ruleset path can have the following structure:

DeploymentConfiguration/RAVmajor.RAVminor/Operation/RSVmajor.RSVminor

DeploymentConfiguration identifies the decision service. RAVmajor.RAVminor identifies the version of the decision service interface that is being used. Client applications use the version number to ensure that they are compatible with the decision service interface. Operation is the name of the ruleset. The major version number of a ruleset (Vmajor) often represents a specific release of the ruleset, and the minor version number (Vminor) represents a deployment version of the ruleset. Typically, client applications call the latest enabled ruleset, but what they actually call depends on the ruleset path that is used in the client application.

The version numbers use base numbers that you define in the deployment configuration. You manually define a base number for the RuleApp and each ruleset. Deployment increments the minor ruleset version number, or replaces or creates a ruleset version number (Vmajor.Vminor).

Note: The version policies that are set in a Rule Designer deployment configuration are synchronized with Decision Center.

Select a policy by the type of deployment

The version policy that you use depends on the type of deployment:

Type of deployment	Version policy
Successive deployments of a release to a specific server	Increment minor version numbers
Development of a decision service	Use the base version numbers
Test fix to correct a deployed solution	Use the base version numbers
Test fixes or updates to an earlier release	Define the version numbers

Increment minor version numbers

This policy serves as the default setting. It deploys the decision operations of the release, changing the minor version number of each ruleset. It looks for the latest version number to increment on the server to which it has access, and the same deployed version number is used.

A client application uses a ruleset based on the ruleset path that is defined in the client application., but previous deployments remain available on Rule Execution Server. You can still use a previous deployment by specifying its full ruleset path. You use this policy with successive deployments of a release to a specific server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/2.1</div>

Use the base version numbers

This policy deploys the decision operations and replaces the base version on Rule Execution Server. The replacement ensures that the deployment configuration path corresponds exactly to what is deployed on the server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 (replaced)</div>

Define the version numbers

With this policy, you enter a specific ruleset version number at deployment time. You use this policy with test fixes or updates to an earlier release. Upon deployment, Rule Execution Server uses the specified ruleset version, and replaces the last version of the ruleset.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 (replaced) /ruleset/2.0</div>

Parent topic: [Deploying from the Business console](#)

Administering projects from the Business console

A user who has the permission manager role can access the **Administration** tab in the Business console to administer security, manage servers, or run diagnostics.

[Administering security](#)

You can manage groups of users, and set project security to control access to decision service branches.

[Managing servers](#)

As a permission manager, you use the Business console to manage the list of servers available to Decision Center for deployment, testing, and simulations. You can also restrict access to specific groups.

[Diagnostics](#)

To get information about your system, you can run diagnostics in the **Diagnostics** subtab of the **Administration** tab.

[Configuration settings](#)

You can set some configuration options to customize Decision Center behavior, or display in the Business console. You can also set custom parameters.

[Exporting or importing the current project](#)

As a permission manager, you can export the working branch of the current project as a .zip file, and import the project previously exported back into Decision Center.

[Setting up project dependencies](#)

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in different decision services or branches.

Parent topic: [Working with the Business console](#)

Administering security

You can manage groups of users, and set project security to control access to decision service branches.

[Security](#)

Decision Center provides a security feature to control access to branches of decision services, and set permission profiles.

[Managing groups of users](#)

From the **Administration** tab of the Business console, you can create, delete, and edit groups, and assign users to these groups.

[Setting project security](#)

You can control access to decision service branches by enforcing security for projects in releases, activities, or ungoverned branches. You can also control access to these projects for specific groups of users.

Parent topic: [Administering projects from the Business console](#)

Security

Decision Center provides a security feature to control access to branches of decision services, and set permission profiles.

You can enable security so that branches of decision services are only visible to certain groups of users. You can enable security in the Business console (see [Setting project security](#)), but security applies to both consoles.

When you have enabled security, you must specify, for each group that can access the branch, what permissions they have on artifacts: read only, full authoring, or none. You can set permission profiles from the Business console **Administration** tab, when you edit a group in the **Groups** subtab.

Groups are managed by a permission manager in Decision Center, and users are managed by the administrator in the Operational Decision Manager on Cloud portal. For more information, see [User roles](#).

Branch security and decision governance

To use the decision governance framework (see [Governance principles](#)) with the Decision Center security feature, you must understand how both work and how they interact.

Releases and activities are types of branches, and as such are subject to the Decision Center security and permissions feature. You can enable security on individual branches to define which groups of users have access to them. When security is enabled on a branch, you specify the permissions of groups of users to read only, have full authoring on the artifacts that are contained in the branch, or have no access to these artifacts.

Branches in the decision governance framework inherit the security state of their parent branches. If you enable security on the initial release of a decision service, all the releases and activities that stem from the initial release have security that is enabled by default. Similarly, you can enforce security on a validation activity by setting security on its parent release.

Releases, change activities, and validation activities inherit artifacts from their parent branch, and are subject to permissions. You can, for example, give a group of users security access to a release but restrict their permission to view the change activities of the release.

You can also give or restrict the permissions of a group of users to update the following properties of releases and activities:

Property	Used to give or restrict permission to...
Owner	Change the owner
Due date	Change the due date
Goals	Change the goals
Status	Proceed to approval, cancel, and reopen
Approvers	Add or remove approvers and change their working status
Authors / testers	Add or remove authors in change activities or testers in validation activities, and change their working status

The decision governance framework provides governance aspects that are based on states and governance roles:

- The states of releases and activities (for example, *In Progress* and *Complete*).
- The current user's governance role as a participant in the release or activity (owner, author, tester, approver).

A user who has administrator privileges in Decision Center can do the operations of all the different governance roles. Similarly, users with these privileges have access to all the branches and have all the permissions on all artifacts. However, administrators cannot force state transitions that are not allowed by the decision governance framework.

When you use the decision governance framework, some conditions are imposed, and you must take into account any conditions that are imposed by the security feature. For example, you can rename a release only under the following conditions:

- The state of the release is not *Complete*.
- You are the owner of the release.
- You have the Update/Release/Name permission on the release, or security on the release is not enforced.

Parent topic: [Administering security](#)

Managing groups of users

From the **Administration** tab of the Business console, you can create, delete, and edit groups, and assign users to these groups.

In the **Users** tab, you see a list of users who have access to the Business console. Users are created and managed by the administrator in the Operational Decision Manager on Cloud portal. From the **Users** tab of the Business console, you can assign users to one or more groups, which you define in the **Groups** tab.

Decision Center uses groups for security access to the different branches, and permissions on the artifacts (see [Security](#)). To make use of this security and permissions feature, you must create groups in the Business console, and assign all users to one or more of these groups. You create in Decision Center as many groups as you need to organize your users functionally.

Note: Permissions are computed at login time, and are kept during the entire session. If permissions for a group change, or if a permission manager adds a member in a group, users need to log out and log in again to view the artifacts with the appropriate permissions.

Permission profiles

In Decision Center, security defines which groups have access to the different branches of a decision service. By default, no group is allowed to create, update, view, or delete anything until these permissions are explicitly granted, unless the group has administrator privileges.

In the Business console, you can assign one of the following permission profiles to each group:

None

Groups assigned this permission profile have access to the branch, but cannot see its content.

Read Only

Groups assigned this permission profile can view the contents of the branch, but cannot create, update, or delete content.

Full Authoring

Groups assigned this permission profile can view, create, update, or delete all content in the branch.

Note: Permissions that you define for a group apply to all Decision Center branches that enforce security.

Parent topic: [Administering security](#)

Setting project security

You can control access to decision service branches by enforcing security for projects in releases, activities, or ungoverned branches. You can also control access to these projects for specific groups of users.

About this task

When you enforce security within a decision service, you can control which groups of users have access to projects in the different branches. For example, you can enforce security for an entire decision service, and select groups that can access this decision service. You can then restrict access for a specific project, so that among these groups, only some groups can access this project.

You define which users are members of a group in the tab **Groups**.

Procedure

1. Log in to the Business console as a permission manager.
2. In the **Administration** tab, click **Project security**. In this tab, you have a tree grid view of the decision services. You can expand each decision service and their branches, to see the projects in each branch.
3. Hover your cursor over the branch for which you want to enforce security, and click the icon to edit it.
4. In the window that opens, a status under **Security** indicates whether the security is enforced or not. Click this status to modify the state of the security.

Note: If the parent branch already has security enforced, the security settings are inherited from this branch. You can click the message **Security settings are inherited from *Branch*** to override the security settings.

5. For branches on which security is enforced, select the groups of users who can access the branch in the **Groups** section.
6. Optional: After you enforce security in a branch, you can restrict access to some projects in this branch for specific groups of users. Click the Edit icon for your project, and select the groups that can access this project.
7. Click **Done** to save your changes.

Parent topic: [Administering security](#)

Managing servers

As a permission manager, you use the Business console to manage the list of servers available to Decision Center for deployment, testing, and simulations. You can also restrict access to specific groups.

Procedure

To manage the available servers:

1. On the **Administration** tab, click the **Servers** subtab.

The release manager can access this subtab in read-only mode. Only the permission manager can manage servers.

2. Click **New Server** to add an external server. To edit it, click **Edit Server**.

Note: You cannot edit the servers that are accessible by default.

3. Enter the following server details:

- **Server name:** Name displayed when selecting the server from the list of servers.
- **Server URL:** Web address of the Rule Execution Server or Decision Runner server.

Note: A Rule Execution Server is used for deployment, and a Decision Runner server is used for testing and simulation purposes. By default, the URLs are `http://<hostname>:<port>/RES` and `http://<hostname>:<port>/DecisionRunner`.

You can add or manage HTTPS servers if the certificate that you use is a CA-signed certificate.

- **User name/Password:** Access credentials to the server. You can leave these credentials blank if you do not want them to be stored.

For servers used to deploy RuleApps, you are asked to enter them when deploying. For running test suites and simulations, you are asked to enter them when clicking the **Test** button. In both cases, your credentials are not stored in Decision Center.

- **Description:** Text to help you identify the server in the table of servers.

4. Specify how each server is used and who has access to it:

- **Usage:** Select whether you want to use the server for deploying RuleApps, or for running test suites and simulations.
- **Groups:** Control which Decision Center groups can access this server when configuring deployment or running test suites and simulations. By default, all groups can access a server. To reduce this access, clear **All groups** and select the required groups in the list.

5. Test that the connection is working correctly by clicking **Test**.

6. Click **OK**.

Parent topic: [Administering projects from the Business console](#)

Diagnostics

To get information about your system, you can run diagnostics in the **Diagnostics** subtab of the **Administration** tab.

The permission manager can run diagnostics on different parts of Decision Center to check the state of the system. It shows results on versioning, data source, extensions, verbalizers, and database, which can be used to troubleshoot issues.

Diagnostics for artifact counts or projects give you information about how many artifacts you have, which type, and the organization of your decision services. This helps you get precise metrics about your projects and decision services.

You can also find the results of the diagnostics that you run in the Business console in your server logs, or download the results as a JSON file.

Parent topic: [Administering projects from the Business console](#)

Configuration settings

You can set some configuration options to customize Decision Center behavior, or display in the Business console. You can also set custom parameters.

Table 1 shows the configuration options that you can set from the **Administration > Settings** tab.

Note: You must have the Permission manager role to edit configuration settings.

Table 1. Business console settings

Setting	Use
Show number of artifacts in Decision Artifacts tree (may impact performance)	Displays the number of artifacts in each project from the Decision Artifacts tab.
Default row ordering for decision tables	Select whether you want to order rows in decision tables with automatic row ordering, which organizes a decision table by grouping rows that share condition values, or manual row ordering, which organizes rows into partitioned groups.
Do not show customer survey	Activate this setting to disable the customer survey that pops up in the Business console and solicits feedback from users.

Build options for a decision service

You can set build options for a decision service. Open the decision service, and set the options in the **Decision Service** panel.

Choose when to cancel archive generation

Allow ruleset archive generation to be cancelled on warnings, or errors, or never cancelled.

Build automatically

Decision Center generates the compiled version of a rule as soon as you save it. Otherwise, this generation occurs when generating the ruleset. You might find enabling this option useful to accelerate ruleset generation.

Parent topic: [Administering projects from the Business console](#)

Exporting or importing the current project

As a permission manager, you can export the working branch of the current project as a .zip file, and import the project previously exported back into Decision Center.

Exporting

You can export the current state of your working branch from the Business console. Select the branch, release, or activity that you want to export, and click **Export** in the toolbar. In the dialog that opens, generate your .zip file, then click the link to download it. Some artifacts are not included in the .zip file, or included with limitations (see the note at the end of this page).

Importing

You can import decision services or branches from projects that you previously exported as .zip files. For example, you might want to import the .zip file in another Decision Center, or after recreating the database.

Importing a decision service

From the **Decision Services** page, click the **Import Decision Service** icon to import an entire decision service into Decision Center. You can select **Use Decision Governance Framework** if you want your decision service to use the decision governance framework. In this case, the decision service contains a main branch and an initial release. If you do not select this option, your imported decision service contains only a main branch.

If the decision service you are importing already exists in Decision Center, an error message indicates that you must import the elements from a branch.

Importing projects

To import decision artifacts into an existing decision service, select your release, change activity, or branch, then click **Import** in the toolbar. During the import, Decision Center synchronizes the existing project with the imported .zip file.

During this synchronization, Decision Center handles differences the following way:

- New elements found in the .zip file are added to the Decision Center version of the project.
- Deleted elements are not synchronized. If you deleted an element in Decision Center and not in the .zip version, the element is added to the Decision Center version. Similarly, if an element is deleted in the .zip version, it is not removed from the Decision Center version.
- Changes made to the elements in the .zip version override elements in Decision Center if you check the option **Replace existing elements in Decision Center**. If this option is not checked, the Decision Center version remains intact.

Note:

Limitations:

- The history of your branch is not exported.
- Test suites and simulations are exported, with the following limitations:
 - Reports for tests and simulations are not exported.
 - Test suites based on snapshots are not exported.
 - When you import a project into Rule Designer that was previously exported from Decision Center, test suites and simulation artifacts are converted into text files and put in a folder named .validation, hidden by default in Rule Designer. They are available for your information, but **should not be modified**. These files cannot be synchronized with Decision Center.

Parent topic: [Administering projects from the Business console](#)

Setting up project dependencies

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in different decision services or branches.

A project in a decision service can be referenced by other decision services, or in other branches of the same decision service. All items in the referenced project are then available to you for use in your current decision service or branch. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that you need to maintain only one BOM and one vocabulary, and changes made to the BOM and vocabulary are automatically propagated to all the projects that use them.


Before you create project dependencies, make sure that you have a clear idea of how your decision services and branches will be organized to share one or several projects. Take some time to assess how any modification to a rule artifact in a shared project can impact the different branches and decision services that share this project.

When you share a project between branches of a decision service, or in other decision services, you must create a project dependency. You create a project dependency in the Dependencies editor of the Business console where you specify which project and which branch of that project to reference.

Note: You cannot share a project if you use the Decision Governance Framework.

Creating a project dependency

To create a project dependency:


1. Open the **main** branch of your decision service.
2. In the **Branch** pane, expand the **Linked Projects** section, and click **Edit dependencies**. The Dependencies editor opens.
3. Select your project in the **Project** list to see the list of its dependencies (if any). Click the **+** icon to add a new dependency.
4. Double-click the blank fields of the new dependency to select the decision service, project, and branch you want to reference in your current project.
5. Click **Save**.
6. Go back to your branch with the breadcrumbs. Your shared project is indicated by the  shared icon in the **Decision Artifacts** tab, and a tooltip when you hover over the project name. The tooltip is limited to five dependencies at most, but you can review the full list in the **Linked Projects** section.

Note: Some operations on versioned elements have no impact on the dependencies. For example, if you restore a snapshot that was taken before creating or editing a project dependency, or if you merge branches that share projects, the dependencies will not be modified.

Working with branches

When you create a branch in a decision service, it is populated with a new version of each project from the parent branch. If there were shared projects available in the parent branch, those shared projects *are no longer shared* in the new branch. If you need to continue sharing these projects, you must manually re-establish the dependencies.

To re-establish the project dependencies in a new branch:

1. Open your new branch.
2. In the **Linked Projects** pane, click **Edit Dependencies**.
3. In the dependencies list, you see your decision service and project. Double-click the branch name and select **main**.
4. Click **Save**.
5. Go back to your branch with the breadcrumbs. You can see that your common project is shared again, as shown by the  shared icon, and the tooltip where you can see the decision services this project is dependent on.

Note: When you re-establish the dependency, the shared project *replaces* the copy of this project that was made when you created a branch. Make sure that you have not modified your business rules or BOM in this copy because in this case you might lose your changes.

Setting project security for shared projects

Administrators can set permissions for groups of users to access a project from the **Administration > Project Security tab** (see [Setting project security](#)).

When you enforce security on a decision service, the security settings do not apply to shared projects in this decision service. You must set the permissions for these shared projects manually.

The security settings that you apply for a shared project are available to all decision services that references this project.

Parent topic: [Administering projects from the Business console](#)

Working with the Enterprise console

You manage decisions with Decision Center, which you can access in a web user interface, the Enterprise console.

[Introducing the Decision Center Enterprise console](#)

You use the Decision Center Enterprise console to author, edit, organize, and search for business rules.

[Decision Center basics](#)

Find out about rule projects, and the basic components: smart folders, versions, branching, baselines, and the different types of project elements that you can use in Decision Center.

[Explore: Navigate your projects](#)

The Explore tab provides a number of features that you can use to navigate within your rule projects.

[Compose: Create project elements](#)

You use the Compose tab in the Enterprise console to create project elements.

[Query: Search your projects](#)

You use queries to search through your rule projects to display the business rules or other project elements that correspond to criteria of your choice.

[Analyze: Check your projects](#)

You can check your projects for consistency and completeness and generate rule project reports.

[Project: Manage your project](#)

You use the Project tab to manage advanced features of your projects.

[Working with the editors](#)

You can choose which editor to use when editing business rules.

[Configure: Manage your project configuration](#)

You use the Configure tab to carry out some administrative tasks.

Introducing the Decision Center Enterprise console

You use the Decision Center Enterprise console to author, edit, organize, and search for business rules.

Overview: Collaborative working

Decision Center provides a designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules.

The Enterprise console environment

Learn how to access all the features available in the Decision Center Enterprise console.

User options

You can change user options to make them match your preferences or specific requirements. These options only apply when you use Decision Center, and do not affect any other users.

The online help

You can obtain help when using the Enterprise console, by clicking **Help** in the top banner.

Parent topic: [Working with the Enterprise console](#)

Overview: Collaborative working

Decision Center provides a designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules.

The corporate policies that govern your company have been extracted from its various computer applications and now exist as business rules within a decision management system. Business rules are policy statements expressed in a way that a computer system can interpret. Decision Center serves as a designated workspace where business users can work collaboratively within the decision management system.

The business rules that you author or modify within Decision Center are not immediately reflected back to the rule-based computer applications on which they act. You must deploy them. Deployment involves sending sets of rules to a Rule Execution Server that handles the execution of the rules and its interaction with the rule-based applications.

Parent topic: [Introducing the Decision Center Enterprise console](#)

Related information:

[The Enterprise console environment](#)

[Decision Center basics](#)

The Enterprise console environment

Learn how to access all the features available in the Decision Center Enterprise console.

Overview of the Enterprise console environment

The Enterprise console is designed with options in the top banner, tabs for various operations, navigation for selecting projects and branches, and explanations to help you work.

The Home tab

You use the **Home** page to select the decision service to work on. You choose a branch or a release activity, and an action to do.

The Explore tab

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

The Compose tab

Use the **Compose** tab to create or edit project elements.

The Query tab

Use the **Query** tab to create, edit, and run queries.

The Analyze tab

Use the **Analyze** tab to generate reports on your projects and their project elements.

The Project tab

Use the **Project** tab to carry out advanced project-related tasks.

Parent topic: [Introducing the Decision Center Enterprise console](#)

Overview of the Enterprise console environment

The Enterprise console is designed with options in the top banner, tabs for various operations, navigation for selecting projects and branches, and explanations to help you work.

The Enterprise console provides a group of tabs for you to access information and features:

Home

To select a rule project or decision service, a branch or release and activity, and an action.

Explore

To move through the current project.

Compose

To create or edit project elements.

Query

To do specific searches on your projects.

Analyze

To generate reports on your projects and project elements.

Project

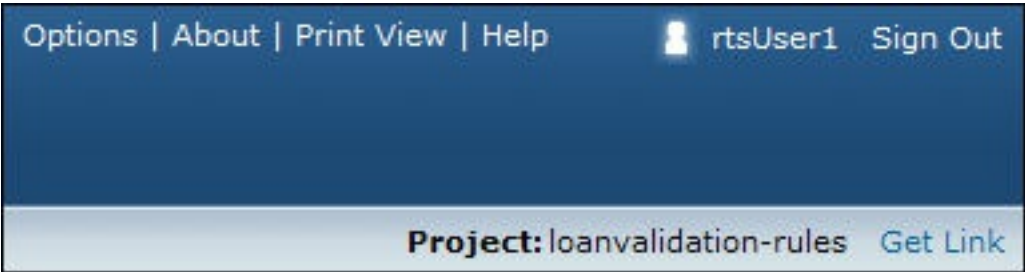
To carry out project-related tasks.

Note:

Some types of users have extra privileges that give access to other tabs used for specific tasks. For information about the extra privileges, refer to [Configure: Manage your project configuration](#).

The top banner

The top banner is available regardless of which tab you select.



You use the top banner to sign out of the Enterprise console or to access the following buttons:

Options

To set user options such as language, theme, columns to include in tables, order of smart folders, and choice of editor.

Help

To display online help.

Print View

To display the current view in a format for printing. Click **Normal View** to return to a view with the top banner.

The name of the current project and branch is visible under these buttons.

Note:

The Details or Version pages provide a **Get Link** hyperlink next to the project name or next to the name of project elements. Right-click this hyperlink to copy the exact URL leading to the project or project element.

Interaction between tabs

When you do certain actions, you might move from one tab to another. For example, clicking **New** or **Edit** in the **Explore** tab takes you to the **Compose** tab.

A tab can contain more than one level of pages. For example, you consult previous versions of a project element ([Exploring the history of an element](#)) on the **History** page of the **Explore** tab:

Home

Explore

Compose

Query

Analyze

Details > History

Explore Version Details

Compare 2 Versions

Restore Version

Copy

Display by

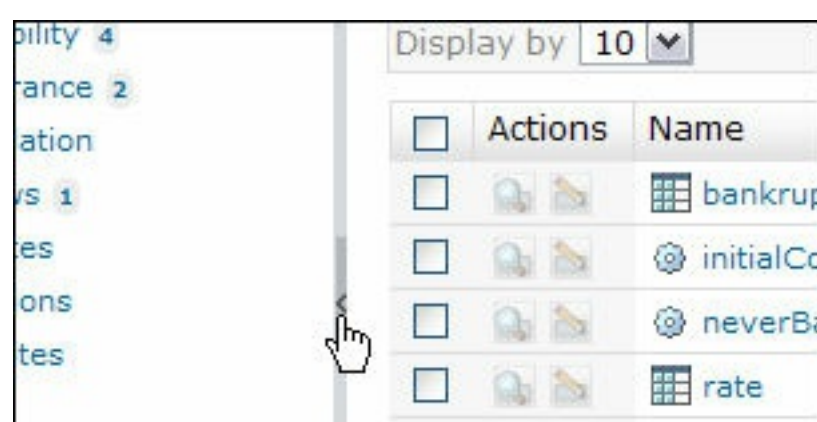
10

<input type="checkbox"/>	Version	Changed By	Comment
<input type="checkbox"/>	1.0	rtsAdmin	
<input type="checkbox"/>	1.1	rtsUser1	

The breadcrumbs, which are located just under the tab names, indicate which page of a tab you are currently on. You also use them to move back to previous pages. Use the breadcrumbs to move back to a page, not the **Back** button.



You can adjust the size of your work area in the **Explore**, **Compose**, and **Query** tabs by sliding the division area. The button on this division area hides or shows the area completely.



Parent topic: [The Enterprise console environment](#)

Related information:

[Overview: Collaborative working](#)
[Decision Center basics](#)

The Home tab

You use the **Home** page to select the decision service to work on. You choose a branch or a release activity, and an action to do.

When the Enterprise console opens, it displays the **Home** page.

The **Home** page contains the following sections:

- **Work on a decision service:** You work on the rules in a decision service without the decision governance framework.
- **Work on a decision service within the decision governance framework:** You use the decision governance framework that is used in the Business console. You work in change activities within releases.

Note:

When you select a decision service, it is displayed in the top banner.

After you select a decision service and a release activity, you choose an action:

Work on branch/release

This action is selected by default to carry out work on your decision service.

View a baseline/deployment baseline

These actions are available when baselines exist for the selected decision service.

View deleted items

This action activates the recycle bin. You can restore items from the bin.

Create subbranch

This action creates a subbranch of the current branch. You can also create a subbranch from the **Project** tab (see [Managing subbranches](#)).

Parent topic: [The Enterprise console environment](#)

Related concepts:

[Rule projects](#)

Related tasks:

[Signing in to the Enterprise console](#)

[Baselines](#)

[Branches](#)

[Recycle bin](#)


The Explore tab

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

In the **Explore** tab, you can select project elements and then edit, copy, delete, and lock them. For more information about what you can do on the **Explore** tab, refer to [Explore: Navigate your projects](#).

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed.

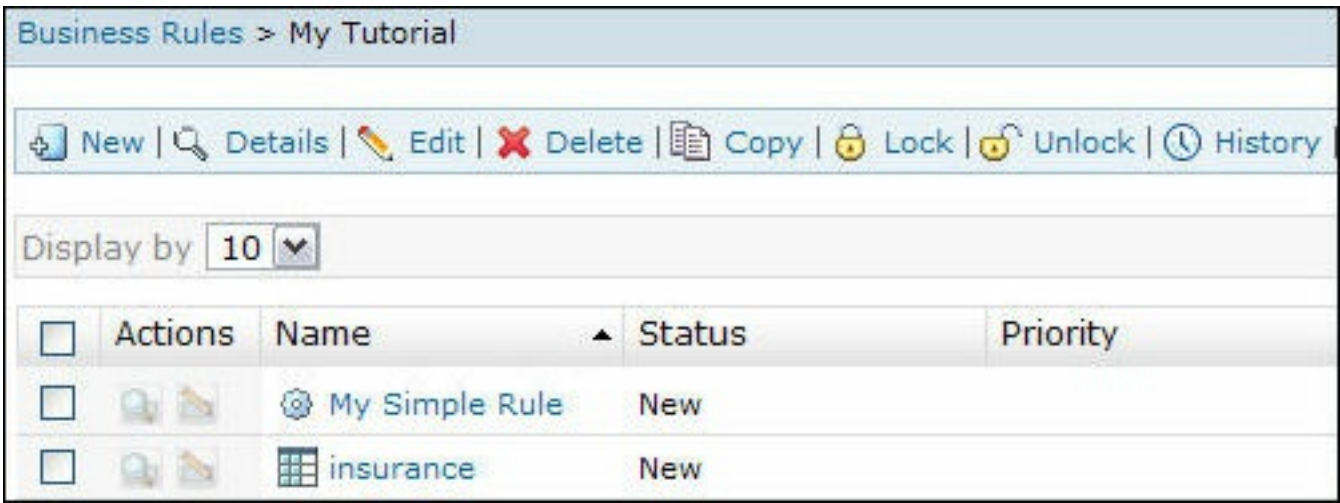
The default smart folder for a rule project is the **Business Rules** smart folder, which displays all business rules (action rules, decision tables, and decision trees) contained in the project, organized in the folders to which they belong.

You can add new smart folders or customize existing ones (see [Creating a smart folder](#)). Use the refresh icon  on the toolbar if you want your current session of Decision Center to display immediately changes done to the project by concurrent users.

The default smart folders display most of the project elements you need to work on. However, some more technical project elements, such as resources, variable sets, technical rules, and functions are not displayed in any of the default smart folders. You can create new smart folder for these types of project elements.

Tables


Click a folder to display its contents in a table:



Tables consists of rows and columns. The first row displays the headings, and each subsequent row corresponds to an item in the folder or view. Click a heading name to sort the items alphabetically or by date, depending on the nature of that column.








Note:

You can choose what information the tables display by clicking **Options** in the top banner.

The first column contains check boxes. To work on one of the items in the table, select it by clicking its check box , and then use the toolbar to perform specific tasks on it.


The icon next to the name of the project element indicates the type of element represented in that row. The following table describes the different icons.

Table 1. Icons for types of elements


Icon	Type of element
	Action rule.
	Decision table.
	Ruleflow.
	Smart folder.
	Variable set.
	Function.
	Resource.


The **Locked** icon  indicates that an item is locked.


Use the following commands to edit an element:

- Click **Edit** on the toolbar to access the Compose wizard for that element.
- Click the **Edit this element** icon  to access the editor area under the table on the Explore tab.

Click the name of the project element to display its Details page.

Click the **Preview** icon  to display a quick preview of the selected project element just under the table:

 **Rule Preview**

 Edit

Name

My Simple Rule

Status

New

if

it is not true that the spouse of **'the borrower'** has filed a bankruptcy

then

set the credit score of **'the borrower'** to the credit score of **'the borrower'** + 20 ;

Parent topic: [The Enterprise console environment](#)

Related concepts:

- [Smart folders](#)
- [Folders](#)

Related tasks:

- [Locking project elements](#)
- [Displaying project element details](#)

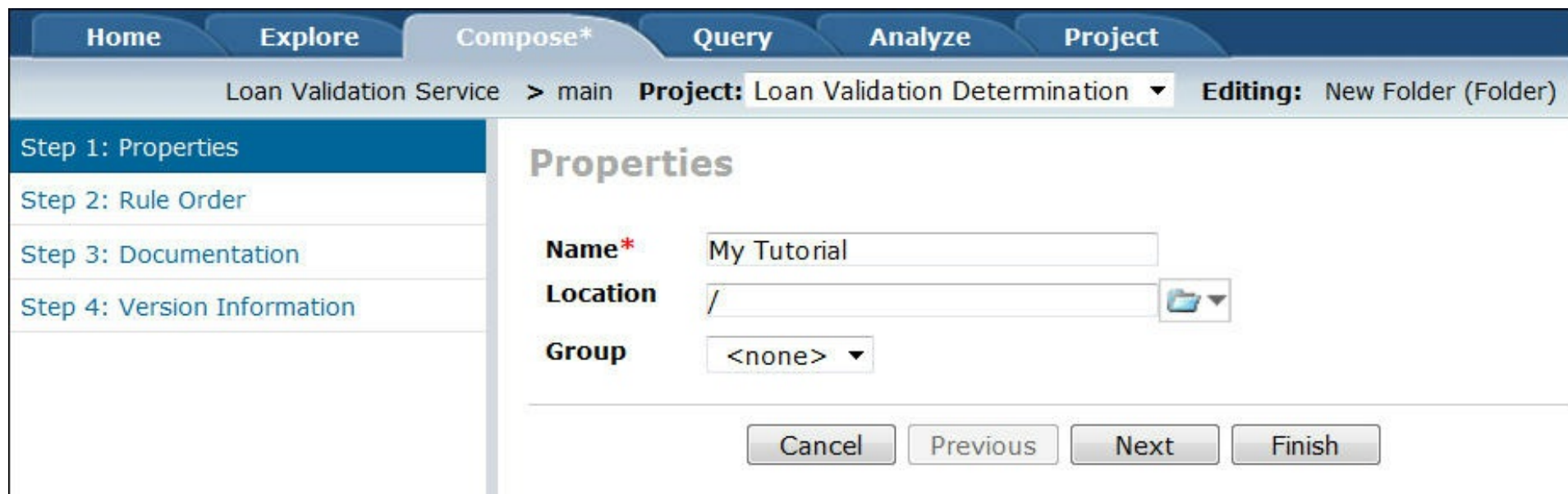
Related information:

- [Editing a project element](#)

The Compose tab

Use the **Compose** tab to create or edit project elements.

To create a new element, select the type of element from the available types or templates, and then use the Compose wizard to work through the steps required to create or edit your element:



The screenshot shows the 'Compose*' tab selected in a navigation bar with other tabs: Home, Explore, Query, Analyze, and Project. Below the navigation bar, a breadcrumb trail reads 'Loan Validation Service > main'. To the right, it says 'Project: Loan Validation Determination' with a dropdown arrow, and 'Editing: New Folder (Folder)'. On the left, a sidebar lists four steps: 'Step 1: Properties' (highlighted), 'Step 2: Rule Order', 'Step 3: Documentation', and 'Step 4: Version Information'. The main area is titled 'Properties' and contains three fields: 'Name*' with the value 'My Tutorial', 'Location' with a slash '/' and a folder icon, and 'Group' with a dropdown menu showing '<none>'. At the bottom right of the main area are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

An asterisk (*) next to the name of the **Compose** tab indicates that you are in the process of creating or editing a project element. This asterisk disappears when you click **Finish** or **Cancel**. If you move to another tab while editing a project element, the asterisk remains and serves as a reminder that you are currently editing a project element.

Note:

The rules that you create or edit are verified at different levels before they are deployed (see [Rule verification](#)).

Parent topic: [The Enterprise console environment](#)

Related information:

[Decision Center basics](#)

[Compose: Create project elements](#)

The Query tab

Use the **Query** tab to create, edit, and run queries.

You use queries to search through your projects for business rules or other project elements that correspond to criteria of your choice. For example:

```
Find
  all business rules
  such that the status of each business rule is deployable
```

On the results of a query you can perform actions, generate a report, or carry out rule analysis.

Parent topic: [The Enterprise console environment](#)

Related concepts:
[Introducing queries](#)

Related information:
[Query: Search your projects](#)

The Analyze tab

Use the **Analyze** tab to generate reports on your projects and their project elements.

You use the **Analyze** tab to perform the following actions:

Check Project Consistency

Checks the consistency of the rules in your project.

Generate Project Report

Provides detailed information on the ruleset parameters, project elements, and ruleflows in your project.

Generate Task / Rule Report

Generates an Excel report with detailed information on the rule project elements and rule tasks in your project.

Parent topic: [The Enterprise console environment](#)

Related information:

[Analyze: Check your projects](#)

[Decision Center basics](#)

The Project tab

Use the **Project** tab to carry out advanced project-related tasks.

You use the **Project** tab to perform the following actions:

Manage Subbranches and Baselines

To create and manage subbranches and baselines.

Merge Branches

To merge the contents of different branches.

Edit Project Options

To enable or disable rule analysis.

Reload Dynamic Domains

To update the values of certain business terms.

Parent topic: [The Enterprise console environment](#)

Related information:

[Project: Manage your project](#)

User options

You can change user options to make them match your preferences or specific requirements. These options only apply when you use Decision Center, and do not affect any other users.

To change the following options, click **Options** in the top banner of the Enterprise console to open the **User Options** page.

Change language to

To select the language in which the Enterprise console is displayed. The language in which you display the Enterprise console is determined by a parameter in the URL you use to access Decision Center, or by the language of your browser if the URL has no parameter. For information about how to set the browser language, refer to the documentation for your browser.

You can also change the language in which you display the Enterprise console to one in the list of languages in **User Options**. Regardless of whether you set the language through the URL, browser, or options, the Enterprise console displays the language only if your development team has translated the Enterprise console environment and provided you with translated projects.

Note:

If you select a language and the Enterprise console does not display it, contact your development team.

Select a theme

To change the general appearance of the Enterprise console. The themes for the user interface include classic, gray, and blue.

Select a calendar type

To select a calendar for the Enterprise console. Calendars include Gregorian, Hebrew, Islamic and Buddhist.

Select the columns

To select the columns displayed in the rule tables.

Number of characters for values

To set the number of characters displayed for values in tables. The minimum number of characters is 10, and the maximum number of characters is 100. This is useful when the names of your project elements are long and get truncated in the tables.

Display order of smart folders

To set the order in which smart folders are displayed in the **Explore** tab.

Show the stack

To show the stack when an exception occurs.

Show a warning for display performance

To receive a warning when heavy smart folders might slow down the display of a page.

Select the rule editor

To set the editor used by the Enterprise console:

- **Default rule editor**, the editor already associated with a project (see [Project options](#)).
- **Guided editor**
- **Intellirule editor**

Mirror GUI

To flip the interface horizontally, from right to left. This supports bidirectional (bidi) languages such as Arabic and Hebrew.

Input Text Orientation

To set the direction in which text is entered.

The options include:

- **Contextual**: matches the direction of the input text (right to left for bidi, and left to right for non-bidi).
- **UI Based**: matches the **Mirror GUI** option (user input language).

- **Right to Left:** enforces right-to-left text direction, usually to support languages such as Arabic and Hebrew.
- **Left to Right:** enforces left-to-right text direction, usually to support languages such as English and French.

Use keyboard shortcuts

In the Enterprise console, you can navigate through the different tabs and features by pressing the **Tab** key until the feature is highlighted, and then pressing **Enter** to select it. You can also use the **Up** and **Down** arrows to navigate through the smart folders.

In addition, if you select the **Use keyboard shortcuts** option, you can press **Ctrl+Alt** with the following numbers or letters to access different features:

- 1–7: To select a tab, starting at 1 for the **Home** tab.
- 0: To select **OK** on the first page of the **Compose** wizard.
- N: To select **Next** in the steps of the **Compose** wizard.
- P: To select **Previous** in the steps of the **Compose** wizard.
- C: To select **Cancel** in the steps of the **Compose** wizard.
- F: To select **Finish** in the steps of the **Compose** wizard.

Parent topic: [Introducing the Decision Center Enterprise console](#)

Related tasks:

[Signing in to the Enterprise console](#)

Related information:

[The Enterprise console environment](#)
[Decision Center basics](#)

The online help

You can obtain help when using the Enterprise console, by clicking **Help** in the top banner.

You can navigate the online help from the table of contents, or using the search tab. In addition, clicking on the **Help** icons located throughout the Enterprise console opens the online help with the appropriate topic displayed.

If you are unable to find the information you are looking for, contact IBM® Customer Support.

Parent topic: [Introducing the Decision Center Enterprise console](#)

Related information:

[The Enterprise console environment](#)

[Decision Center basics](#)

Decision Center basics

Find out about rule projects, and the basic components: smart folders, versions, branching, baselines, and the different types of project elements that you can use in Decision Center.

[Rule projects](#)

A rule project corresponds to a specific organization of rules and other project elements.

[Branches](#)

Branches facilitate work done on parallel releases of a project.

[Smart folders](#)

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed in the Enterprise console.

[Folders](#)

Folders in Decision Center help you organize your project elements.

[Versioning](#)

Decision Center creates archived versions of elements contained in your projects each time you modify them.

[Baselines](#)

Baselines capture the state of a project or branch at a specific moment in time.

[Recycle bin](#)

The **recycle bin** contains project elements that have been deleted from the current state of a branch.

[Action rules](#)

You express business policies in rules that match actions to conditions.

[Decision tables](#)

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

[\(Deprecated\) Decision tree overview](#)

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

[\(Deprecated\) Templates](#)

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

[Project element properties](#)

A property provides additional information about a project element.

[Overview: Ruleflows](#)

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

[Rule verification](#)

You can perform different levels of verification on your rules before you deploy them to the rule-based computer applications on which they act.

[Variable sets](#)

Variable sets contain variables that you can use across the different elements of a rule project. You can also use these variables to pass data between tasks in a ruleflow.

[Functions](#)

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

[Technical rules](#)

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

Parent topic: [Working with the Enterprise console](#)

Rule projects

A rule project corresponds to a specific organization of rules and other project elements.

The Decision Center repository can contain many projects, but projects are displayed one at a time in the console.

A project is made of a main line, sometimes called the main branch. To support the development cycle of projects, you can create subbranches of the main line or of any branch, and then merge changes between branches.

You can also create baselines of a project or branch. Baselines are a snapshot of the state of a project or branch at a given point in time.

Note:

You can see what the current project, branch, and baseline is, either in the top banner or on the **Home** tab.

Parent topic: [Decision Center basics](#)

Related tasks:

[Baselines](#)

[Branches](#)

[Recycle bin](#)

[Erasing the current project](#)

Related information:

[Explore: Navigate your projects](#)

[Project: Manage your project](#)

Branches

Branches facilitate work done on parallel releases of a project.

About this task

You select the branch you want to work on in the **Home** tab. You can also create subbranches from the **Home** tab, and manage them from the **Project** tab.

Initially, a project contains only a main line in which you work. From the main, sometimes referred to as the main branch, you can create one or more subbranches. From any new branch you can create other subbranches, as many as you need to manage your releases.

Note:

The name of the branch you are currently working on appears in the top banner next to the project name.

When you create a subbranch, it contains an exact replica of every project element contained in the parent branch. You can then work on the subbranch without affecting the content of its parent, and eventually merge different branches together ([Merging branches](#)).

Note:

To understand how different versions of project elements are handled, see [Versioning](#).

Procedure

To work on a branch:

1. Go to the **Home** tab.
2. On the **Home** tab, select the required branch from the drop-down list next to **Branch in use**.

Parent topic: [Decision Center basics](#)

Related concepts:

[The Home tab](#)

Related tasks:

[Managing subbranches](#)

[Versioning](#)

[Recycle bin](#)

Related information:



[Merging branches](#)

Smart folders

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed in the Enterprise console.

For example, you might want to display only the rules that are new to the project so that you can validate that they correctly implement company business policies.

The default smart folder for a rule project is the **Business Rules** smart folder, which displays all business rules contained in the project, organized in the folders to which they belong.

To access the smart folders for a given rule project, go to the **Explore** tab. Each smart folder has a **smart folder** icon  next to it. Click the expand  icon next to a smart folder to see its contents.

To change the order in which your smart folders are displayed on the **Explore** tab, click **Options** in the top banner. You cannot lock smart folders manually, but a view locks automatically when you edit it.

Note:

If project security is enforced, you must at least have View permission on smart folders to see other project elements on the **Explore** tab.

Parent topic: [Decision Center basics](#)

Related concepts:

[Rule projects](#)
[Folders](#)

Related tasks:

[Creating a smart folder](#)
[Locking project elements](#)

Folders

Folders in Decision Center help you organize your project elements.

Folders in Decision Center are similar to folders in a file system, but are closely linked to smart folders (see [Smart folders](#)) in the Enterprise console. Existing folders are only visible under smart folders that are set up to display their project elements by folders.

You assign a project element to a given folder through the project element's Folder property (see [Project element properties](#)). Changing the folder property of a project element is the same as moving it from one folder to another (see [Moving a folder or rule](#)).

Note:

A project element does not have to be assigned to a folder, but it is good working practice to reduce the number of project elements that are not assigned.

To ensure that folders meant to display certain types of project elements appear under the appropriate smart folders, the Enterprise console proposes different folder hierarchies:

Folder type:	Under smart folders that display:
Business rules	Business rules (action rules and decision tables), technical rules, ruleflows, functions, or variable sets.
Resource	Resources
Templates	Templates (Templates are deprecated)

When you create a project element, the Enterprise console only proposes folders of the correct type for that project element.

Note:

An empty folder does not synchronize with Rule Designer.

Parent topic: [Decision Center basics](#)

Related concepts:
[Smart folders](#)

Related tasks:
[Creating a folder](#)

Related information:
[Explore: Navigate your projects](#)

Versioning

Decision Center creates archived versions of elements contained in your projects each time you modify them.

About this task

You can consult the history of modifications made over time, and restore a previous version if required. You can edit or delete only the current version of an element.

The version number of a project element is made of a major number and a minor number:

Major number

Found before the decimal point, the major version number corresponds to the branch. Decision Center gives the main line the major version number 1, and attributes the next available number (2, 3, and so on) for each branch of a project that you create.

Minor number

Found after the decimal point, the initial version of a project element has the minor version number 0. This minor version number increments by 1 each time you edit the project element.

When you create a subbranch, each project element initially keeps the version number of the parent branch. Then, if you edit the element in the subbranch, it takes on the versioning policy of the subbranch. For example, a project element with the version number 1.5 keeps this number in the subbranch until you edit it, at which point it becomes x.0.

When you merge a subbranch with its parent branch, Decision Center creates a new version of the project element in the parent branch. This new version is an exact copy of the version in the subbranch, and takes the next version number available in the parent branch.

Similarly, when you restore a baseline, Decision Center creates a new version of all the elements found in that baseline, and makes them the current state.

When you restore a version, either from a baseline or from the history of the project or branch, the new version is a copy of the version restored, with the next version number available. For example, if the current version is 1.4 and you restore it with version 1.2, both versions 1.4 and 1.2 remain in the history, but the new working version is 1.5, and 1.5 is an exact copy of version 1.2.

Note:

You cannot clear the history of a project element.

Procedure

To restore a previous version:

1. Access the **History** page for the project element, as described in [Exploring the history of an element](#).
2. On the **History** page, select the version you want to restore by clicking the check box in the corresponding row.
3. Click **Restore Version** in the toolbar.

A new version of the element is created and added to the table.

Note:

To copy a previous version, click **Copy** instead and specify the location.

Parent topic: [Decision Center basics](#)

Related concepts:
[Version information](#)

Related tasks:
[Exploring the history of an element](#)
[Branches](#)

Related information:
[Editing a project element](#)

Baselines

Baselines capture the state of a project or branch at a specific moment in time.

About this task

For example, if you create monthly baselines of a project, you will be able to consult what the state of the project was in past months, and revert back to a previous baseline if required. See [Managing baselines](#) for information on creating baselines.

Note:

Note also the existence of deployment baselines, which can be created when deploying RuleApps.

Each branch of a project can contain many baselines, all of which you can consult. When you consult a baseline you cannot edit any of the project elements; you must return to the current state of the project or branch to do so. However, when you consult a baseline you can access the history of the project elements, and restore individual elements to the current state.

Note:

When you consult a baseline, the name of the baseline appears in the top banner.

Procedure

To consult a baseline:

1. On the **Home** tab, from the **Current action** drop-down list, select **View a baseline** and the required baseline.
2. To return to the current state, select **Work upon branch** from the **Current action** drop-down list.

Parent topic: [Decision Center basics](#)

Related concepts:

[Rule projects](#)

Related tasks:

[Managing baselines](#)

[Managing deployment baselines](#)

[Recycle bin](#)

Recycle bin

The **recycle bin** contains project elements that have been deleted from the current state of a branch.

About this task

You cannot edit an element in the recycle bin but you can restore it so that it returns to the current state of the branch. You can only restore elements in the recycle bin. You cannot permanently delete them.

Note:

The recycle bin does not display project elements that you restore to the current state.

The recycle bin contains the two standard smart folders (Business Rules and Ruleflows) to help you to organize the content, as well as any other smart folders that you have deleted.

The recycle bin also displays deleted folders. If you restore a deleted folder, you do not also restore the rules it contains. You must restore them explicitly.

Procedure

To consult the recycle bin:

1. On the **Home** tab, select **View deleted items** from the drop-down list next to **Current action**.
Decision Center displays all deleted project elements that have not been restored.
2. To return to the current state of the branch in the project, select **Work upon branch** from the drop-down list next to **Current action**.

Parent topic: [Decision Center basics](#)

Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
    the credit score of 'the borrower' is less than 200
then
    in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
    set applicant to a customer
    where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

Rule variables

You create a rule variable to define the scope of a rule.

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

Rule actions

Rule actions define what to do when the if part of the rule is true or false.

Parent topic: [Decision Center basics](#)

Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
    the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
    set 'the cart' to the shopping cart of customer;
if
    the value of 'the cart' is less than $100
then...
```

One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
    set Smith to a customer;
if
    the category of Smith is Gold
then
    apply 10 % discount to the shopping cart of Smith;
```

When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
()	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation

marks:

R u l e d i t o r	
	Description
In t e l l i r u l e e d i t o r	If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.
G u i d e d e d i t o r	When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.

Types of rule variables
You can assign different types of values to your rule variables.

Parent topic: [Action rules](#)

Related concepts:
[Types of rule variables](#)
[Dependency of rule actions on rule variables](#)

Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, customer). Once you set a variable, you can use it in any part of the rule that declares the variable.

Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and then parts of the rule use the same value:

```
definitions
    set maxAmount to 1000000;
if
    the amount of 'the loan' is at least maxAmount
then
    in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

Restrictions on rule variables

You can further restrict a variable in the definitions part of a rule by using the operator `where`.

Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
    set 'loyal customer' to a customer
        where the category of this customer is Gold;
if
    the value of the shopping cart of 'loyal customer' is more than $200
then
    apply the super discount;
```

Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
    set 'senior Gold customer' to a customer
        where all the following conditions are true:
            - the category of this customer is Gold
            - the age of this customer is at least 65;
```

Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the definitions part of the rule, for example:

```
definitions
    set applicant to a customer;
    set 'loyal customer' to a customer;
if
    all of the following conditions are true:
        - applicant is married to 'loyal customer'
```

```
        - 'loyal customer' is insured
then
    upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
    'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
    set 'customer 1' to a customer;
    set 'customer 2' to a customer;
if
    'customer 1' is married to 'customer 2'
    and 'customer 2' is insured
then
    upgrade 'customer 1''s rating;
```

Note: When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
    set 'gold customers' to all customers
        where the category of this customer is gold;
    set 'junior gold customer' to a customer in 'gold customers'
        where the age of this customer is at most 15;
    set 'senior gold customer' to a customer in 'gold customers'
        where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

Parent topic: [Rule variables](#)

Related concepts:

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
    the customer category is Gold
then
    redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

Combinations of conditions

You can apply conditions to groups, and test nested groups.

Condition negation

You can set a rule to perform an action when a condition is not true.

Parent topic: [Action rules](#)

Related concepts:

[Conditions that test for existence](#)

[Combinations of conditions](#)

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
    the customer's maintenance number starts with "TX"
then
    redirect the call to call center A;
```

is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
    the purchase value of the shopping cart is more than $100
then
    apply a 10% discount to the value of the shopping cart
```

after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
    the return date of 'rented car' is after 'pickup date' and before
    'scheduled return date'
then...
```

Parent topic: [Rule conditions](#)

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
  there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
  there are 10 customers where the category of each customer is Gold,
then...
```

there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
  there are at most 3 customers
    where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
  set 'silver customers' to all customers
    where the category of each customer is Silver;
  set 'silver count' to the number of 'silver customers'
if
  there are at least ('silver count' + 1) customers
    where the category of each customer is Gold,
then...
```

there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:

```
if
    there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
    set 'gold customers' to all customers
        where the category of each customer is Gold;
if
    there is at least one customer in 'gold customers'
        where the age of this customer is at least 65,
then...
```

Parent topic: [Rule conditions](#)

Related concepts:

[Rule conditions](#)

[Combinations of conditions](#)

Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
    the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
    set 'luxury car groups' to all car groups where the daily rate of each
    car group is at least 50;
if
    'luxury car groups' contain the car group of 'the current rental
    agreement'
```

Parent topic: [Rule conditions](#)

Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting  
longer than 5 minutes
```

Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if  
    the customer is older than 60  
    or the customer is younger than 21  
    and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if  
    the customer is older than 60  
    or (the customer is younger than 21 and the category of the customer  
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if  
    (the customer is older than 60 or the customer is younger than 21)  
    and the category of the customer is Student
```

Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true`: This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true`: This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if  
    all of the following conditions are true:  
        - the category of the customer is Gold  
        - a member of the Gold team is available  
then  
    redirect the call to a member of the Gold team;
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
    any of the following conditions is true:
        - the category of the customer is Gold
        - the customer has been waiting longer than 5 minutes
then
    redirect the call to a member of the Gold team;
```

Parent topic: [Rule conditions](#)

Related concepts:

[Rule conditions](#)

[Conditions that test for existence](#)

Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

it is not true that

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
    the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
    it is not true that the category of the customer is Gold
then...
```

none of the following conditions are true

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
    all the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
    none of the following conditions are true:
        - the category of the customer is Gold
        - the age of the customer is at most 15
then...
```

Parent topic: [Rule conditions](#)

Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
    the value of the shopping cart is more than $100
then
    apply a 15% discount on the shopping cart;
```

Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
    'the loan report' is approved
    and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
    in 'the loan report', accept the loan with the message
    "Congratulations! Your loan has been approved";
else
    in 'the loan report', refuse the loan with the message "We are sorry.
    Your loan has not been approved";
```

Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

Parent topic: [Action rules](#)

Related concepts:

[Rule actions for lists of business terms](#)

[Dependency of rule actions on rule variables](#)

Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
    for each item in expensive items:
        - apply 5% discount to this item;
        - display the message: "A 5% discount has been applied";
```

Parent topic: [Rule actions](#)

Related concepts:

[Rule actions](#)

[Dependency of rule actions on rule variables](#)

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
    all of the following conditions are true:
        the value of the customer's shopping cart is more than $100
        the category of the customer is Gold
then
    apply a 15% discount
else
    apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
    set applicant to a customer
        where the category of this customer is Gold
if
    the value of the applicant's shopping cart is more than $100
then
    apply a 15% discount
else
    apply a 5% discount
```

Parent topic: [Rule actions](#)

Related concepts:

[Rule variables](#)

[Types of rule variables](#)

[Rule actions](#)

[Rule actions for lists of business terms](#)

Decision tables

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥600,000		true	0.005
5	B	<100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		≥600,000		true	0.0075

When an application calls a decision table, the action rules are executed. If the conditions in a row are met, the rule that is formed by the row performs the actions in the row.

You can add rows to the decision table and enter values in their cells to create new rules. You can also define preconditions that apply to all the rules in a table. Error markers help you find overlaps and gaps in your rules.

Columns

Each column in a decision table represents a condition or an action.

Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

Preconditions

You can pretest data before using it with a decision table.

Parent topic: [Decision Center basics](#)

Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. The top cell of each column identifies the object of a condition or the target of an action.

Row	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001

Each numbered row in the table forms a rule. The rule performs the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of the loan is between 100000 and 300000
then
  set the Insurance required to true
  set the Insurance rate to 0.001
```

Condition operators

You can split a condition across columns in a decision table when a rule statement contains more than one value. For example, the following condition requires you to specify values for <min> and <max>:

```
if
  the age of the customer is between <min> and <max>
```

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [Decision tables](#)

Related concepts:

- [Rows and cells](#)
- [Preconditions](#)

Related tasks:

- [Adding, removing, and editing columns](#)
- [Creating business rules](#)
- [Adding, removing, and editing rows](#)
- [Defining preconditions](#)

Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

Note: If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In Rule Designer, you group cells by merging them.

In the following table, the Grade A cells of rows 1 and 2 are grouped into one cell. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 100000 and 300000
then
  - set the insurance required to true
  - set the loan rate to 0.001
```

- Rule 2

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row that has the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3		300,000	600,000	true	0.003

The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.005
5	B	< 100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		600,000	800,000		
9		≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003
3	A			12	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.004

Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003
3		Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3		Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 100000 and 300000
then
  - set the insurance required to true
  - set the loan rate to 0.001
```

- Rule 2

```
if
  all of the following conditions are true:
    - the loan grade is A
    - the amount of loan is between 300000 and 600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.002
```

- Rule 3

```
if
  all of the following conditions are true:
    - the loan grade is A
    - it is not true that the amount of loan is between 300000 and
600000
then
  - set the insurance that is required to true
  - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

Parent topic: [Decision tables](#)

Related concepts:[Columns](#)[Preconditions](#)**Related tasks:**[Adding, removing, and editing columns](#)[Creating business rules](#)[Adding, removing, and editing rows](#)[Defining preconditions](#)

Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
  set 'wealthy customer' to a customer
    where the average monthly balance of this customer
      is more than $1 000 000
if
  the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

Parent topic: [Decision tables](#)

Related concepts:

[Columns](#)

[Rows and cells](#)

Related tasks:

[Adding, removing, and editing columns](#)

[Creating business rules](#)

[Adding, removing, and editing rows](#)

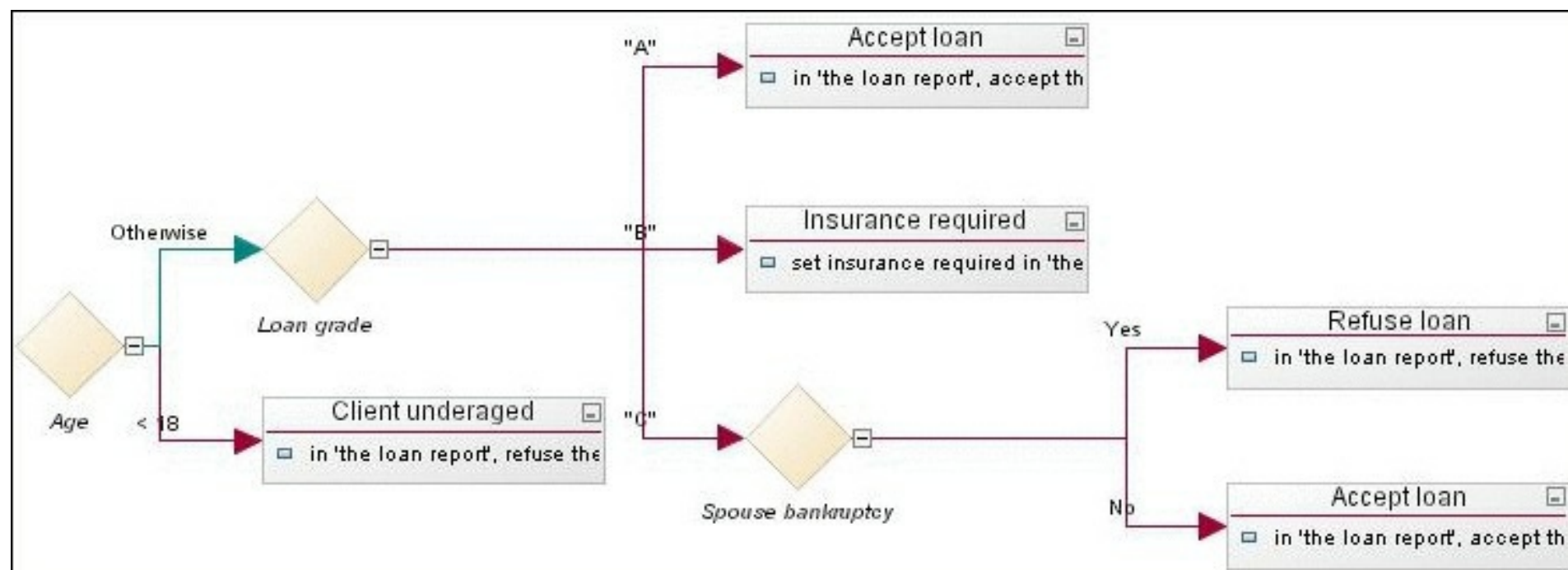
[Defining preconditions](#)

(Deprecated) Decision tree overview

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

Decision trees provide a way to view and manage large sets of business rules in diagrams.

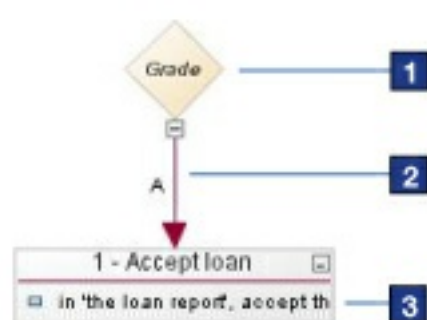
Decision trees make the interaction of nonsymmetrical rules easier to understand. The path from the first condition to the end of the actions along any branch represents one rule.



Looking at the following figure, you can see why decision trees are easy to understand:

- A condition is declared in its diamond-shaped node **1**.
- The possible values for the condition are represented by branches **2**.
- The actions are declared at the end of each branch **3**.

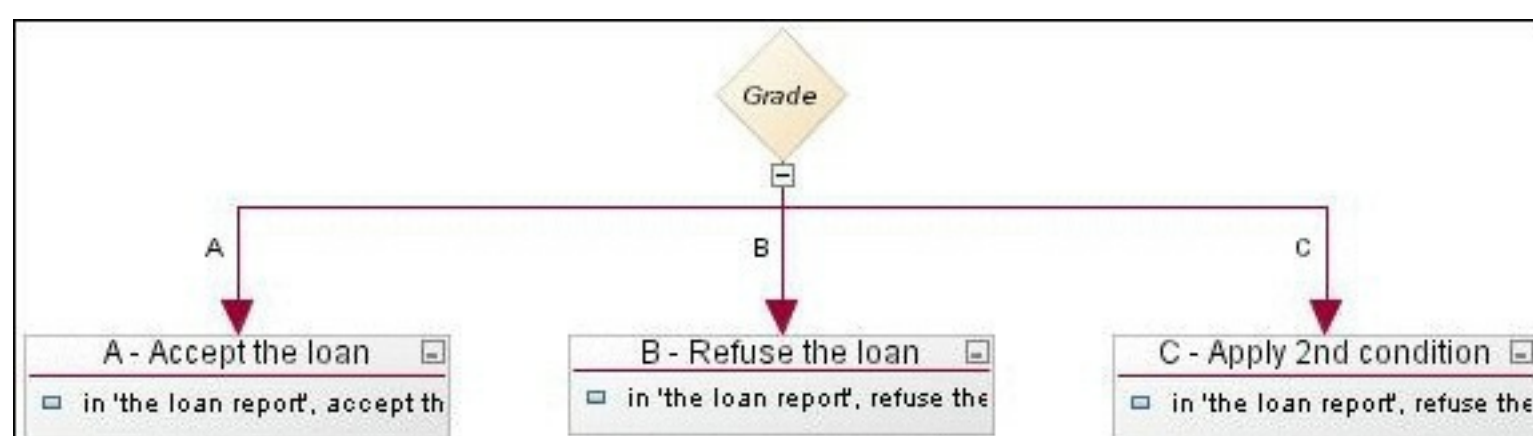
This simple decision tree corresponds to the following rule:



if
 the grade in the loan report is 'A'
then
 in the loan report, accept the loan with the message "Loan accepted"

By adding branches, you add new rules that have different values for the condition.

For example, the following decision tree forms three rules for a loan application (A, B, and C):



You can put as many actions as you want at the end of a branch, and add another condition to a branch.

For example, rules 1 and 2 in the following decision tree have only one condition (Grade), while rules 3 and 4


```

graph TD
    Grade{Grade} -- A --> A1[1 - Accept loan  
in 'the loan report', accept th]
    Grade -- B --> A2[2 - Refuse loan  
in 'the loan report', refuse the]
    Grade -- C --> BK{Spouse bankruptcy}
    BK -- Yes --> A3[3 - Refuse loan  
in 'the loan report', refuse the]
    BK -- No --> A4[4 - Accept loan  
in 'the loan report', refuse the]
  
```

```

graph TD
    Grade{Grade} -- A --> A1[1 - Accept loan]
    Grade -- B --> A2[2 - Refuse loan]
    Grade -- C --> BK{Spouse bankruptcy}
    Grade -- Otherwise --> CS[Customer support]
    BK -- Yes --> A1
    BK -- No --> CS
  
```

The flowchart for the 'Loan' use case is as follows:

- Grade** (Decision):
 - If **A**, proceed to **1 - Accept loan**.
 - If **B**, proceed to **2 - Refuse loan**.
 - If **C**, proceed to **Spouse bankruptcy** (Decision).
 - If **Otherwise**, proceed to **Customer support**.
- Spouse bankruptcy** (Decision):
 - If **Yes**, proceed to **1 - Accept loan**.
 - If **No**, proceed to **Customer support**.

Each process box contains a note: "in 'the loan report', accept th" for '1 - Accept loan', "in 'the loan report', refuse the" for '2 - Refuse loan', and "in <a report>, add the messa" for 'Customer support'.

Preconditions

- Variables that can be used in the decision tree.
- Condition that is applied to an entire decision tree.

For example, you can apply the following precondition to a decision tree:

```
set 'wealthy customer' to a customer
    where the average monthly balance of this customer
        is more than $1,000,000
```

Preconditions are tested before each rule in a tree is run.

Parent topic: [Decision Center basics](#)

(Deprecated) Templates

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

By defining it as such in the template, parts of rules that are created from a template can be frozen so that they cannot be edited.

Templates simplify the task of writing rules and decision tables, and are useful for:

- Creating action rules and decision tables that are already partially written.
- Restricting users from editing certain parts of partially completed rules.

A template applies only at the moment you create a rule or a decision table. If you subsequently modify the template, this will have no impact on rules or decision tables previously created from this template. You would need to update these rules or decision tables manually.

Parent topic: [Decision Center basics](#)

Project element properties

A property provides additional information about a project element.

Most properties are useful to help you manage your business rules. For example, the query mechanism searches for project elements that match given criteria based on properties. You can also select which properties you want to display in your tables: click **Options** in the top banner and then select **Edit Display Options**.

The properties of a project element appear on the **Details** page. You can edit some properties in the **Properties** step of the Compose wizard (see [Compose: Create project elements](#)). You cannot edit properties that Decision Center automatically generates, such as the **Created By** property.

Although rule projects all contain certain properties, you can customize (extend) rule projects to capture information specific to the business domain of your company. The tables in the following sections describe properties marked as *extended properties*. These extended properties might not appear in your specific rule projects.

Management properties

The following table describes properties that are useful for managing and authoring your project elements. You can normally edit them.

Property	Used to...
Categories	<p>Choose the categories of business terms that the rule displays in its drop-down lists when editing their content in the rule editor.</p> <p>By default, rules belong to the Any category, which means that they have all the available business terms of the project available in the drop-down lists.</p> <p>You can replace the Any category with another one to limit the number of entries in the drop-down lists. For example, you can limit the list to include only those entries that concern the Sales department. In this case, the Sales category must already be available next to the Categories property.</p>
Effective Date (<i>extended property</i>)	<p>Specify the date on which the rule comes into effect. This property is normally used by a smart folder or a query to identify rules that come into effect.</p> <div>Note: This is an extended property that might not be implemented in your application. Contact your development team for more information.</div>
Expiration Date (<i>extended property</i>)	<p>Specify the date on which the rule expires. This property is normally used by a smart folder or a query to identify rules that become out of date.</p> <div>Note: This is an extended property that might not be implemented in your application. Contact your development team for more information.</div>
Folder (Location)	<p>Set the location of the project element. Decision Center proposes separate folder hierarchies for business rules, resources, and templates (see Folders).</p> <div>Note: Folders correspond to rule packages in the developer environments.</div>
Group	<p>Associate the rule with the user’s group. If the user belongs to more than one group, this property will be set to the first group of the user unless you specify otherwise.</p> <p>The Group property is used by the Administrator to set permissions.</p>
Name	Give a name to the project element.
Locale	Specify the language of the text displayed in the rules. The default language, en_US , corresponds to English.
Status (<i>extended</i>)	Set the status of a rule, for example:

<i>property</i>)	<ul style="list-style-type: none">• new• defined• validated• rejected• deployable
----------------------	---

The following table describes properties that are defined by Decision Center. You can display them but you cannot edit them.

Property	Used to...
Created By	See who created the project element.
Created On	See when the project element was created.
Last Changed By	See who last modified the project element.
Last Changed On	See when the project element was last modified.
Project	See to which project the element belongs. This property is useful when dealing with dependent projects.
Template	See from which template the rule was created. (Templates are deprecated.)
Type	See the type of project element: for example, folder, action rule, decision table.

Execution properties

The following table describes the properties related to executing rules.

Property	Used to...
Active (extended property)	<p>Set whether or not the rule is active. You can deactivate a rule by deselecting this property.</p> <p>CAUTION: When using this property, make sure that the extractor being used to generate the ruleset has been set up to extract active rules.</p>
Main Subflow Task	Identify the main ruleflow of the project. A project can contain many ruleflows but only one main ruleflow. You identify a main ruleflow by setting its Main Subflow Task property to true.
Priority (extended property)	<p>Set the priority of the rule. You use the Priority property to establish precedence between different rules.</p> <p>By default, the Priority property is set to 0. If you want a rule to be considered before another, set the priority of the first rule to a higher number than the second. To allow for any future rules that may be created, leave a gap between entries. For example: -100, 0, 100, 200.</p> <p>Rules with the same priority are processed according to the order in which they are added to the rule engine agenda. Rules more recently inserted into the agenda are chosen before those less recently inserted. For rules inserted at the same time, the order in which the rules appear in the agenda corresponds to the order in which the rules appear in the ruleset source file.</p> <div><p>Attention:</p><p>To enable accurate synchronization with Rule Designer, enter the priority value as an integer. If you do not, synchronization sets the priority to the default priority value of 0.</p></div>

Technical properties

The following table describes the properties specific to technical rules, functions, and ruleflows:

Property	Used in...
Imports	

	Functions, technical rules, and ruleflows. Imports correspond to the classes used in the rule.
Return Type	Functions.
Initial Actions	Ruleflows.
Final Actions	Ruleflows.

Parent topic: [Decision Center basics](#)

Related concepts:
[Overview: Ruleflows](#)
[Functions](#)
[Technical rules](#)

Overview: Ruleflows

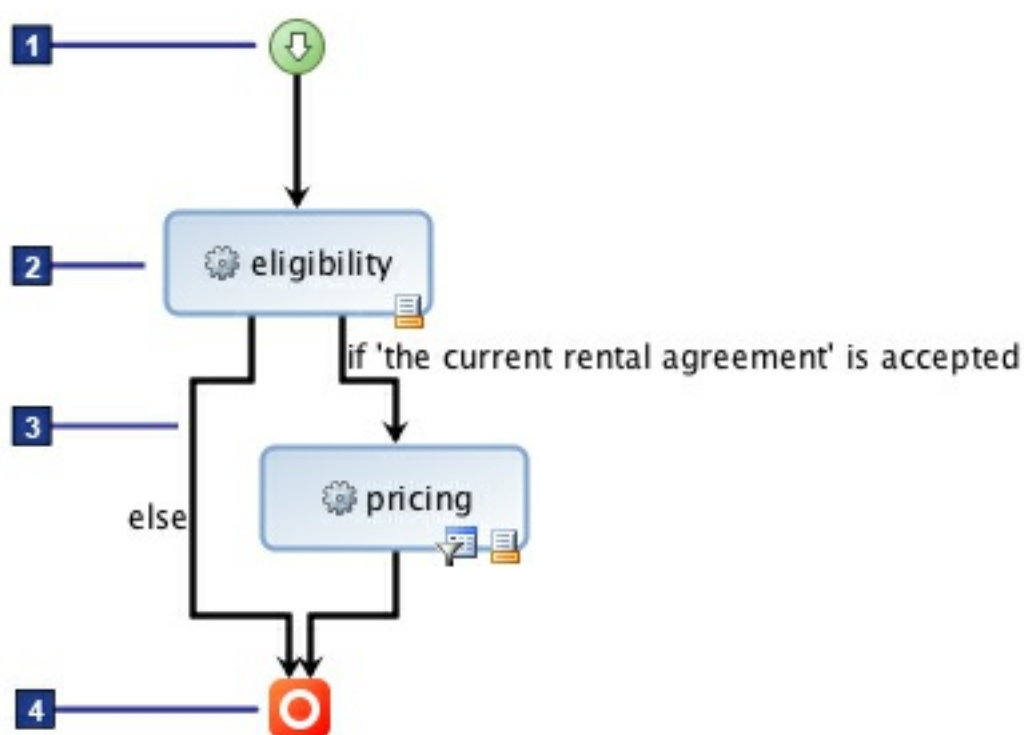
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



Note: The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter


statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

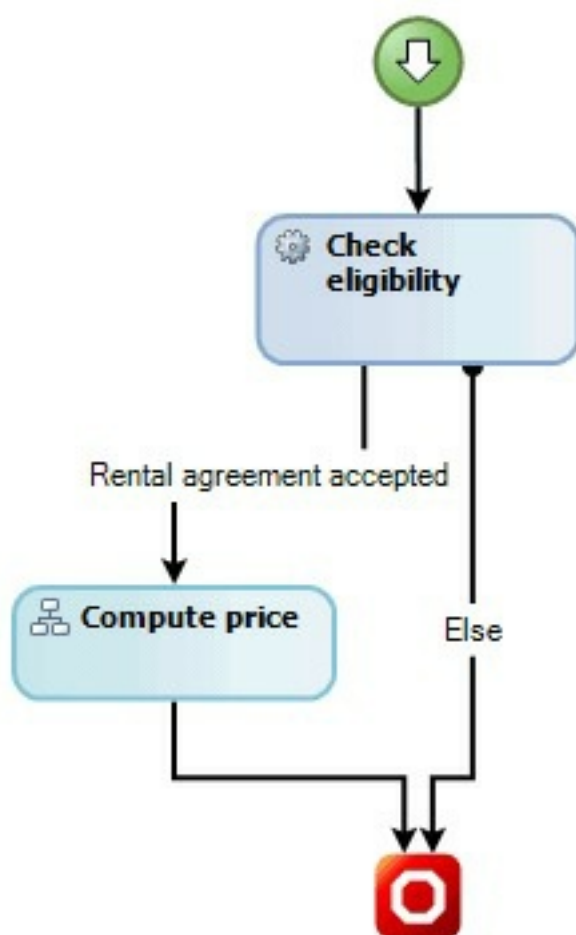
You can specify initial and final actions on subflow tasks.

Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.




A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.


Branches

A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch.

Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

Parent topic: [Decision Center basics](#)

Rule verification

You can perform different levels of verification on your rules before you deploy them to the rule-based computer applications on which they act.

Basic syntax checks

Decision Center verifies the syntax of any business rule that you edit.

Decision table and decision tree checks

You can check decision tables and decision trees for gaps and overlap errors. You can also test decision tables for symmetry.

Parent topic: [Decision Center basics](#)

Basic syntax checks

Decision Center verifies the syntax of any business rule that you edit.

When you save the rule, the **Details** page displays any errors or warnings resulting from the syntax checks. Any rule that contains any syntax error is excluded when the Configuration Manager generates a ruleset.

Decision Center checks for the following syntax errors:

- A rule statement uses a business term that is not recognized or no longer valid.
- A rule statement is ambiguous, that is, more than one correct interpretation exists. In this case, use parentheses to clarify the syntax.
- A variable in the definitions part is already declared or it is of the wrong type.
- A rule is incomplete, that is, some of the placeholders have not been filled in.
- A rule has an else part with no if part.

For a more exhaustive list of the types of checks available, see the **Rule Designer** online help.

Parent topic: [Rule verification](#)

Related tasks:

[Editing on the Explore tab](#)

Decision table and decision tree checks

You can check decision tables and decision trees for gaps and overlap errors. You can also test decision tables for symmetry.

Overlap checks

You can set Decision Center to check that the values you enter in your decision tables or decision trees do not overlap or are not identical.

For example, in the following condition statements, customers who are 29 years old satisfy the condition for both rules:

- Age is between 17 and 30
- Age is between 29 and 40

If these two conditions stem from the same condition node of a decision tree, or are part of the same column or within a partitioned groups of cells in a decision table, Decision Center signals it as an overlap error.

Gap checks

You can set Decision Center to check that the values you enter in your decision tables or decision trees consider all possible cases. This ensures that there are no gaps in your conditions.

For example, in the following condition statements, customers who are 31 years old are never taken into account:

- Age is between 17 and 30
- Age is between 32 and 40

Decision Center reports a gap error.

In decision tables, Decision Center evaluates gaps for all the entries in a given column, or within partitioned groups of cells. In decision trees, Decision Center evaluates them for the branches of a condition node or for the entire tree.

Symmetry checks

You can set Decision Center to check that all partitions of a decision table use the same set of items. You can verify symmetry for the whole table or for specified columns. Such analysis is useful to enforce that you have covered all choices in a decision table.

By default, symmetry analysis is off, allowing you to create nonsymmetrical tables.

Parent topic: [Rule verification](#)

Related concepts:

[\(Deprecated\) Decision tree overview](#)

Related information:

[Consistency checks in decision tables](#)

[Consistency checking in decision trees](#)

[Decision tables](#)

Variable sets

Variable sets contain variables that you can use across the different elements of a rule project. You can also use these variables to pass data between tasks in a ruleflow.

Variable sets have a name and folder that you use to manage them.

Each variable in a variable set has the following components:

- A name to identify it
- An initial value
- A verbalization
- A BOM type

You can access variables from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.

Note:

Variable sets are an advanced subject normally reserved for developers. For more information, refer to the Rule Designer documentation.

Parent topic: [Decision Center basics](#)

Related concepts:

[Rule projects](#)

[Overview: Ruleflows](#)

Related tasks:

[Creating a variable set](#)

Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG Rule Language (IRL) and its code is evaluated when the ruleset runs.

Note: Functions are an advanced subject that is normally reserved for developers. See the Rule Designer documentation for more information.

Note: Rule Designer does not refactor functions. If you rename, move, or delete a function, the references in other rule project artifacts are not updated.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)  
{  
    code  
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the return keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of return that does not return a value:

```
return;
```

Note: A function is not required to declare in its `throws` statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

Parent topic: [Decision Center basics](#)

Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

Note: Technical rules are an advanced subject normally reserved for developers. See the Rule Designer documentation for more information.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
 - `evaluate`
 - `exists`
 - `from`
 - `in`
 - `instanceof`
 - `not`
 - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

Parent topic: [Decision Center basics](#)

Explore: Navigate your projects

The Explore tab provides a number of features that you can use to navigate within your rule projects.

[Displaying project elements](#)

The Explore tab displays the smart folders and the project elements corresponding to these smart folders.

[Displaying project element details](#)

You can display the details of any project element provided you have the relevant user permissions to do so.

[Editing a project element](#)

You can edit a project element in a number of ways. Editing automatically locks the element so that others cannot edit it at the same time. You can also explicitly lock elements.

[Deleting a project element](#)

You can delete an element from a project.

[Copying a project element](#)

You can create copies of project elements.

[Moving a folder or rule](#)

You can move project elements.

[Locking project elements](#)

When you edit a project element, Decision Center automatically locks it so that others cannot edit it at the same time.

[Exploring the history of an element](#)

Decision Center keeps a history of all the changes you make to any project element.

Parent topic: [Working with the Enterprise console](#)

Displaying project elements

The Explore tab displays the smart folders and the project elements corresponding to these smart folders.

About this task

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

Note:

The default smart folders display most of the project elements you need to work on. However, some more technical project elements, such as resources, variable sets, technical rules, and functions are not displayed in any of the default smart folders. You must create a new smart folder to display these types of project elements.

Procedure

To display a project element:

1. Click the **Explore** tab.
2. Select the appropriate smart folder to display its elements in the table.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[The Explore tab](#)
[Smart folders](#)

Displaying project element details

You can display the details of any project element provided you have the relevant user permissions to do so.

About this task

The **Details** page for a project element contains the following information:

- Contents
- Properties
- Documentation
- Any attached items, such as tags, requirements, initial values, and overridden rules

Procedure

To display project element details:

1. On the **Explore** tab, select the project element in the table.
2. Click **Details** in the toolbar or click the project element name in the table.

Results

Important:

For smart folders, hover your mouse over the smart folder and click the Details icon  to access the Details page.

The **Details** page displays the information on the chosen element.

You can return to the main page of the **Explore** tab by clicking on the tab name.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[Rule projects](#)

[Project element properties](#)

Related information:

[The Enterprise console environment](#)

Editing a project element

You can edit a project element in a number of ways. Editing automatically locks the element so that others cannot edit it at the same time. You can also explicitly lock elements.

Editing on the Compose tab

Use the Compose tab for editing if you want to access all the properties and documentation of an element or make minor changes to its contents.

Editing on the Explore tab

Use the Explore tab for editing if you want to change the name or status of an element or make minor changes to its contents.

Parent topic: [Explore: Navigate your projects](#)

Editing on the Compose tab

Use the Compose tab for editing if you want to access all the properties and documentation of an element or make minor changes to its contents.

About this task

You can use this method to edit any project element as long as you have permission to do so.

When you edit a project element, the **Compose** tab displays with the steps of the wizard filled in with the current properties, content, and version information of the element.

Procedure

To edit a project element on the Compose tab:

1. Select the check box for the element in the table on the **Explore** tab and click **Edit** in the toolbar above the table.

Note:

For smart folders, display its details first (see [Displaying project element details](#)) and then click **Edit** in the **Details** page toolbar.

The **Compose** tab displays the existing contents and properties for the element.

2. Click **Next** or **Previous** to step forwards or backwards through the wizard and edit the different steps as required.
3. Click **Finish** to save your changes.

If you click **Cancel**, none of your changes are saved.

Parent topic: [Editing a project element](#)

Related tasks:

[Selecting your preferred rule editor](#)
[Locking project elements](#)

Related information:

[Compose: Create project elements](#)

Editing on the Explore tab


Use the Explore tab for editing if you want to change the name or status of an element or make minor changes to its contents.

About this task

You can edit any type of element that is listed in the table on the **Explore** tab, except for ruleflows.

Procedure

To edit a project element on the Explore tab:

1. Click **Edit this element**  next to the name of the project element in the table.
The element contents and some properties display in the editing area under the table.
2. Edit the properties or contents of the element in the editing area.

Note:

Use the **Add a comment to this version** field to summarize the changes you have made.

3. Click **Save** to save your changes.

Parent topic: [Editing a project element](#)

Related tasks:

[Selecting your preferred rule editor](#)

Deleting a project element

You can delete an element from a project.

About this task

You can delete any project element as long as it is not locked by someone else and you have permission to delete project elements.

Smart folders

Deleting a smart folder does not delete any of the folders or rules it displays. You can delete a smart folder even if it displays locked elements.


Folders

Deleting a folder also deletes all the rules and any subfolders it contains. However, you cannot delete a folder if another user has locked anything contained in it.

Before you delete a folder, verify that the smart folder that you are using to display the folder has been set up to display all its content. This is to make sure that you do not delete any rules not visible in that view.

Procedure

To delete a project element:

1. On the **Explore** tab, select the project element in the table, or access the **Details** page for smart folders (see [Displaying project element details](#)).
2. Click the **Delete** icon .

Results

Project elements that you delete go to the recycle bin. You can restore deleted project elements from the recycle bin, or from any baseline that contains that project element.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[Smart folders](#)
[Folders](#)

Related tasks:

[Locking project elements](#)
[Baselines](#)

Copying a project element

You can create copies of project elements.

About this task

You can copy any project element as long as you have permission to create those types of element. You can copy a project element to the current project or to another project, and to a folder of the same type (see [Folders](#)).

Note:

Smart folders can only be copied to the root of the current project.

If you choose the current folder location, your copy keeps the same name but prefixed with **Copy of**. If you choose another location, the copy keeps the same name as the original.


Procedure

To copy a project element:

1. On the **Explore** tab, select the project element you want to copy.

Note:

For smart folders, display its details instead (see [Displaying project element details](#)).

2. Click the **Copy** icon  in the toolbar.
3. Select the project and folder to which you want to copy the project element.
4. Click **OK**.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[Rule projects](#)
[Folders](#)

Related tasks:

[Locking project elements](#)

Related information:

[The Enterprise console environment](#)

Moving a folder or rule

You can move project elements.

About this task

You can move a project element to another folder of the same type, or move a folder to another location.

Procedure

To move a folder or project element:

1. In the **Explore** tab, select the project element or folder you want to move and then click **Edit**.
2. In the **Properties** step of the wizard, click the folder icon next to one of the following fields and then use the drop-down list to set the destination folder:
 - Click the **Location** field, to move a folder.
 - Click the **Folder** field, to move a project element.
3. Enter the version information and then click **Finish** to save the change.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[Rule projects](#)

[Folders](#)

[Project element properties](#)


Related tasks:

[Editing on the Compose tab](#)

Locking project elements



When you edit a project element, Decision Center automatically locks it so that others cannot edit it at the same time.

About this task

When people work concurrently on the same project, it is important to make sure that only one person at a time edits a given project element. When an element is locked, other users can open the element and view it, but they cannot edit it until you have saved your changes. A locked element displays the locked icon .

Note:

Hold the mouse over the locked icon to find out who has locked the element.

You can also manually lock most project element types. For example, if you want to work on all the rules contained in a folder, you can lock the folder manually. When you lock a folder, you also lock all the project elements it contains so that only you can edit them. A manually locked element displays the key icon  to you and the locked icon  to others.

Note:

The Administrator can release any locks by clicking **Release Lock** in the toolbar.

When you lock an element manually, you can also specify the following options to subfolders and sessions:

O p t i o n	Description
S u b f o l d e r s	<p>When you edit a folder, it locks automatically, along with all the project elements it contains. However, other users can still edit any subfolders unless, when you lock the folder, you select the option Apply on subfolders.</p> <p>To release all the locks set with this option, you only need to release the lock on the folder itself, and the locks on the subfolders will also be released.</p>
S e s s i o n	<p>By default, any lock (automatic or manual) is released when you end your working session (see Signing in to the Enterprise console). To retain the lock, select the Keep lock after the session closes option when locking.</p> <p>If you select this option, the element remains locked until you unlock it, regardless of how many times you sign in and out.</p>

Procedure

To lock an element manually:

1. On the **Explore** tab, select the project element. For folders, access the **Details** page (see [Displaying project element details](#)).
2. Click **Lock** in the toolbar.
3. On the **Lock** window, select the locking options for subfolders and session.
4. Click **OK**.

Results

The locked element now displays the key icon, showing you have locked the resource, and other users see a lock icon with the element.

When you complete your work, you can unlock the folder manually by proceeding as above and clicking **Unlock**. There is no confirmation message.

Parent topic: [Explore: Navigate your projects](#)

Related concepts:

[Rule projects](#)
[Folders](#)

Related information:

[Editing a project element](#)

Exploring the history of an element

Decision Center keeps a history of all the changes you make to any project element.

About this task

The change history displays a table listing all the versions of the element, including the current one, and shows the following details:

- Who created or changed the element, and when.
- Any comment entered as version information (see [Version information](#)).
- Which baselines, if any, contain the element.

Note:

The history retains a complete history of all elements. You cannot clean out previous versions, nor edit them.

From the history page you can explore the details of a version, restore a version, or copy a previous version. You can also compare the changes between two versions by displaying the **Compare 2 Versions** page:

Differences between versions 1.0 and 1.1			
Type	Property	Old Value	New Value
update	Name	insurance	My Decision Table
update	Content

The table displays the changes between the two versions you selected. Each row represents a change made to an element, and displays the following columns:

Type of change	Description
Type	The type of change made: create, update, delete.
Property	The property that changed.
Old Value	The value that was changed.
New Value	The replacement value.

If the change was made on the definition of the project element itself, the **Old Value** and **New Value** fields are empty. To display the changes in table form, click the hyperlink in the row.

The following image shows the differences between two versions of a decision table:

8		≥ 600,000		true	0.0075	9		< 100,000		true	0.0035
9		< 100,000		true	0.0035	10		100,000	300,000	true	0.006
10		100,000	300,000	true	0.006	11	C	300,000	600,000	true	0.0085
11		300,000	600,000	true	0.0085	12		600,000	800,000	true	0.012
12		≥ 600,000		true	0.0145	13		≥ 800,000		true	0.0145

Procedure

To explore the history of a project element:

1. Select the project element in the **Explore** tab. For smart folders, access the **Details** page instead (see [Displaying project element details](#)).
2. Click **History** in the toolbar.

Results

The **History** page opens and displays all previous versions of the selected project element.

From the history page you can explore the details of a version by selecting it and clicking **Explore Version Details** in the toolbar.

Note:

Alternatively, click the version number in the history table.

You can also select the two versions you want to compare by clicking the check boxes in the rows corresponding to the required versions and clicking **Compare 2 Versions** in the toolbar.

Parent topic: [Explore: Navigate your projects](#)

Related tasks:

[Versioning](#)
[Baselines](#)
[Displaying project element details](#)

Compose: Create project elements

You use the Compose tab in the Enterprise console to create project elements.

[Creating a smart folder](#)

You create smart folders in the **Compose** tab.

[Creating a folder](#)

You create folders to organize your project elements.

[Creating business rules](#)

You can create business rules, that is, action rules, decision tables, or decision trees.

[\(Deprecated\) Creating a rule template](#)

You use a rule template as a starting point for creating a rule.

[\(Deprecated\) Creating a decision table template](#)

You use a decision table template as a starting point for creating a decision table.

[Creating a variable set](#)

Variable sets contain variables that you can use across the different elements of a branch.

[Creating a function](#)

You create functions from the Compose tab.

[Creating a technical rule](#)

You create technical rules from the Compose tab.

[Creating a resource](#)

You create resources from the Compose tab.

[Tags](#)

You use tags to store data with your project elements.

[Rule overriding](#)

You use rule overriding to give rules precedence over other rules.

[Project element documentation](#)

You can add project element documentation when you create or edit project elements.

[Version information](#)

Version information is associated with the version to which you add it.

Parent topic: [Working with the Enterprise console](#)

Creating a smart folder

You create smart folders in the **Compose** tab.

About this task

You use smart folders to navigate through your projects according to criteria that you select.

You can change the order in which your smart folders are displayed on the **Explore** tab by clicking **Options** in the top banner.

Procedure

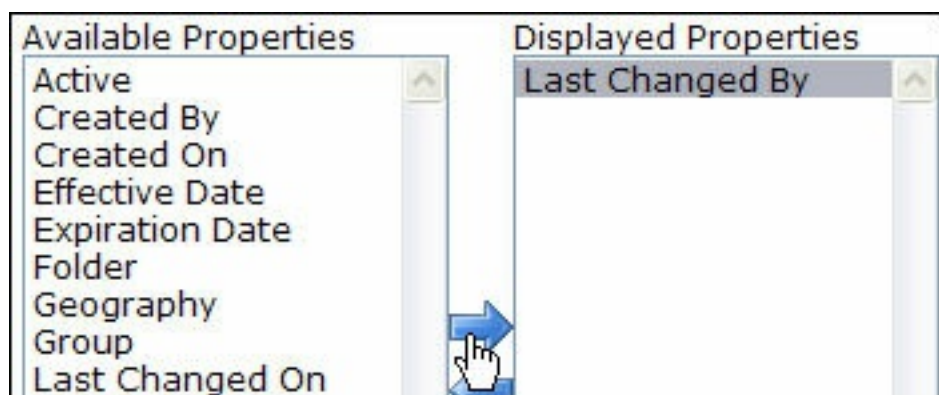
To create a smart folder:

1. On the **Compose** tab, select **Smart Folder** as the type of project element to create and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name of your smart folder and then click **Next**.
3. In **Step 2: Query**, specify how the smart folder searches through the project.

For information about queries, refer to [Query: Search your projects](#).

4. In **Step 3: Displayed Properties**, specify how you want the smart folder to display the elements that it finds in its query.

To select properties, in the **Available Properties** window, select the properties that you want displayed and then, using the arrow between the two properties windows, move the required properties to the **Displayed Properties** window.



Note:

Each property you move to the **Displayed Properties** window adds a level to the display.

5. Use the up and down arrows next to the **Displayed Properties** window to change the display order.
6. In **Step 4: Documentation**, document your smart folder.
7. In **Step 5: Version information**, enter the version information.
8. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

Creating a folder

You create folders to organize your project elements.

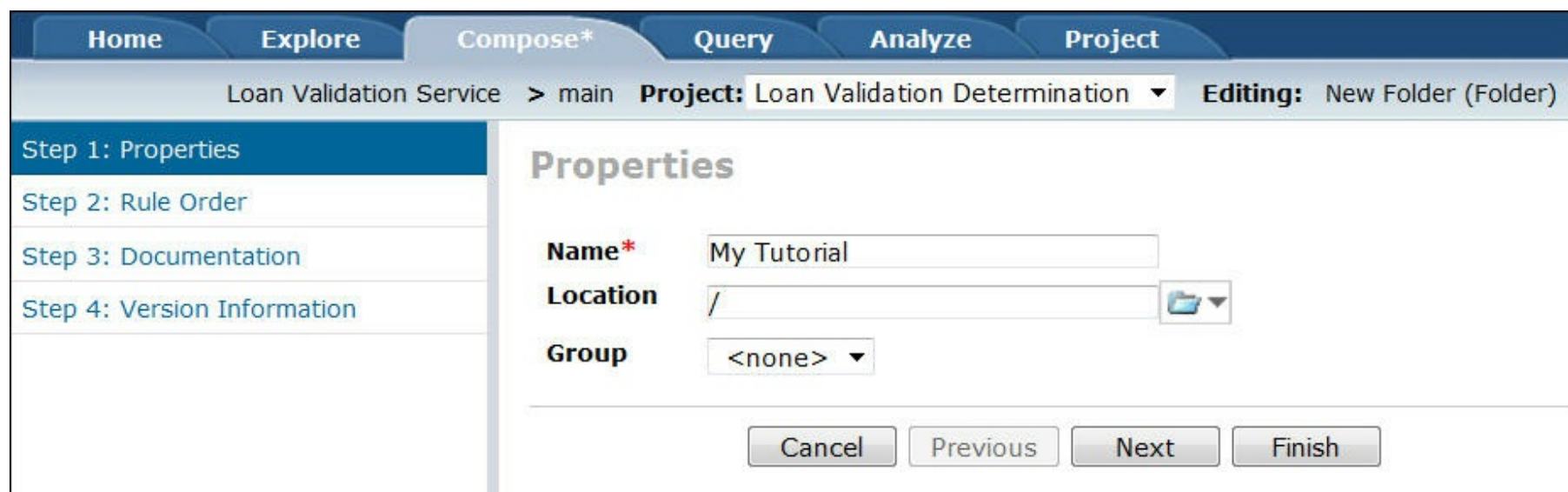
About this task

You can create folders in the **Compose** tab.

Procedure

To create a folder:

1. On the **Compose** tab, select **Folder** as the type of project element to create, choose the correct folder hierarchy, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name, location, and other properties of your folder and then click **Next**.

The screenshot shows the 'Compose*' tab in a software interface. The breadcrumb trail is 'Loan Validation Service > main'. The 'Project' dropdown is set to 'Loan Validation Determination' and the 'Editing' status is 'New Folder (Folder)'. On the left, a sidebar lists four steps: 'Step 1: Properties' (selected), 'Step 2: Rule Order', 'Step 3: Documentation', and 'Step 4: Version Information'. The main area is titled 'Properties' and contains three input fields: 'Name*' with the value 'My Tutorial', 'Location' with the value '/', and 'Group' with a dropdown menu showing '<none>'. At the bottom right, there are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

3. In **Step 2: Rule Order**, specify the order in which the business rules in the folder will be treated.

When you first create a new folder it contains no rules, so you can skip this step. When you have added some rules to the folder you can edit the folder to set the rule order. Refer to [Editing on the Compose tab](#).

The rule order is effective only in the literal ordering. In literal ordering mode, the rule order at folder level only has an impact if the folder itself is added to the task, and not when the rules are added.

Ordering applies only between rules in the same folder. If you add several folders to the same rule task, you must set their relative order in the ruleflow.

Note:

This step is only pertinent when creating folders of type Business Rule (see [Folders](#))

4. In the **Documentation** step, document your folder.
5. In the **Version information** step, enter the version information.
6. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Folders](#)

[Project element properties](#)

[Overview: Ruleflows](#)

[Version information](#)

[Project element documentation](#)

Creating business rules

You can create business rules, that is, action rules, decision tables, or decision trees.

About this task

You create business rules in the **Compose** tab using the wizard.

Procedure

To create a business rule:

1. On the **Compose** tab, select the type of project element to create and then click **OK**.

You can select action rule, decision table, or decision tree. For action rules and decision tables, you can also start from an existing template under **Start from a template**, if some are available.

2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your business rule and then click **Next**.
3. In **Step 2: Contents**, write the contents of the action rule, decision table, or decision tree.

To write rules, you must be familiar with both the structure of action rules (see [Action rules](#)) and how to use the different rule editors (see [Working with the editors](#)).

4. In **Step 3: Tags**, add any tags you want to associate to the rule.
5. In **Step 4: Override Rules**, declare any rules to override.
6. In **Step 5: Documentation**, document your business rule.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

[Tags](#)

[Rule overriding](#)

[Columns](#)

[Rows and cells](#)

[Preconditions](#)

(Deprecated) Creating a rule template

You use a rule template as a starting point for creating a rule.

About this task

Existing templates are available on the **Explore** tab in the **Templates** smart folder.

To use a template as a starting point for creating a rule, select the required template under **Start from a template** on the **Compose** tab.

You can create new rule templates if you have user permissions to do so. You create a new rule template in a similar way to creating a new rule, except for the following:

- You do not have to complete the rule statements. Instead, because the template is the starting point for creating similar rules, build the rule statements as you want them to appear.
- You can freeze any part of the rule statements so that they cannot be modified when someone uses the template to create their rules.
- You can set the properties that you want rules created using your template to have.

Any modifications to an existing template have no impact on rules previously created from this template. The modifications apply only when you use the template to create a new rule.

Procedure

To create a rule template:

1. On the **Compose** tab, select the template as the type of project element to create, specify **Action Rule** as the rule type for this template in the drop-down list, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your template and then click **Next**.
3. In **Step 2: Content**, write the contents of an action rule as you want it to appear when using your template.

To freeze parts of a rule, right-click the part of the rule statement that you want to freeze and then select **Freeze** from the drop-down list.

To unfreeze a frozen part of a rule, right-click the frozen statement and then select **Unfreeze**.

4. In **Step 3: Initial Values**, set the properties that you want the rules created using your template to have. You set properties by building the **Initial Values** table. For example:

Name	Value
priority	high
status	defined

Note:

You cannot freeze these properties.

5. In **Step 4: Documentation**, document your template.
6. In **Step 5: Version information**, enter the version information.
7. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[\(Deprecated\) Templates](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

(Deprecated) Creating a decision table template

You use a decision table template as a starting point for creating a decision table.

About this task

Existing templates are available on the **Explore** tab in the **Templates** smart folder.

To use a template, select the required template under **Start from a template** on the **Compose** tab.

You can create new decision table templates if you have user permissions to do so. You create a new decision table template in a similar way to creating a new decision table, except for the following:

- You do not have to complete the condition and action columns. Because the template is the starting point for creating similar decision tables, you build the templates only with the content that is common to all decision tables to be created using this template.
- You can set the properties that you want the decision tables created using your template to have.

Any modifications to an existing template have no impact on decision tables previously created from this template. The modifications apply only when you use the template to create a new decision table.

Procedure

To create a decision table template:

1. On the **Compose** tab, select the template as the type of project element to create, specify **Decision Table** as the type for this template in the drop-down list, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your template and then click **Next**.
3. In **Step 2: Content**, write the contents of a decision table as you want it to appear when using your template.
4. In **Step 3: Initial Values**, set the properties that decision tables created using your template will have.
5. In **Step 4: Documentation**, document your template.
6. In **Step 5: Version information**, enter the version information.
7. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[\(Deprecated\) Templates](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

Creating a variable set

Variable sets contain variables that you can use across the different elements of a branch.

About this task

Each variable in a variable set consists of a name to identify it, an initial value, a verbalization, and a BOM type. Variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors. You can create new variable sets only if you have user permissions to do so.

Note:

You must create a smart folder that displays variable sets to see them in Decision Center.

Procedure

To create a variable set:

1. On the **Compose** tab, select **Variable Set** as the type of project element to create and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your variable set and then click **Next**.
3. In **Step 2: Variables**, create the set of variables.
4. In **Step 3: Documentation**, document your variable set.
5. In **Step 4: Version information**, enter the version information.
6. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Project element properties](#)

[Variable sets](#)

[Smart folders](#)

[Project element documentation](#)

[Version information](#)

Creating a function

You create functions from the Compose tab.

About this task

You can create new functions if you have permission to do so.

Note:

You must create a smart folder that displays functions to see them in the Enterprise console.

Procedure

To create a function:

1. On the **Compose** tab, select **Function** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your function and then click **Next**.
3. In **Step 2: Contents**, write the function.
4. In **Step 3: Tags**, add any tags you want to associate with the function (see [Tags](#)).
5. In **Step 4: Arguments**, declare the function arguments: the type, name, and order.
6. In **Step 5: Documentation**, document your function.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Functions](#)

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

Creating a technical rule

You create technical rules from the Compose tab.

About this task

You can create new technical rules if you have user permissions to do so.

Note:

You must create a smart folder that displays technical rules to see them in the Enterprise console.

Procedure

To create a technical rule:

1. On the **Compose** tab, select **Technical Rule** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your technical rule and then click **Next**.
3. In **Step 2: Content**, write the technical rule.
4. In **Step 3: Tags**, add any tags you want to associate with the rule (see [Tags](#)).
5. In **Step 4: Override Rules**, declare any rules to override (see [Rule overriding](#)).
6. In **Step 5: Documentation**, document your technical rule.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Technical rules](#)

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

Creating a resource

You create resources from the Compose tab.

About this task

Resource elements let you store any type of file within the Decision Center repository. You can create new resources if you have user permissions to do so.

Note:

You must create a smart folder that displays resources to see them in the Enterprise console.

Procedure

To create a resource:

1. On the **Compose** tab, select **Resource** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, browse to select your resource. The resource takes the name of the file you selected.

Enter the folder, group, and documentation of your resource, and then click **Next**.

3. In **Step 2: Version information**, enter the version information.
4. Click **Finish**.

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Smart folders](#)

[Project element properties](#)

[Version information](#)

Related tasks:

[Storing domains in Excel](#)

[Updating dynamic domains](#)

Tags

You use tags to store data with your project elements.

For example, you can store data related to the geography of the rule or its development phase. You can retrieve this information in a query or in a report.

Tags are presented as a table of names that you declare and values that you give them. You create them in the **Tags** step of the Compose wizard, where you build the table of tags for that project element. For example:

Name	Value
Geography	New York
Phase	1

For developers, tags can be useful when you have implemented a rule model extension in Rule Designer and you synchronize with Decision Center. In this case, extended properties are converted into tags so that the synchronization remains smooth. Later, if you implement the rule model extension in both Rule Designer and Decision Center, these tags are converted back to properties.

Note: If you defined tags on rule packages in Rule Designer, you cannot synchronize them with Decision Center, because Decision Center does not support tags on rule packages.

Parent topic: [Compose: Create project elements](#)

Related tasks:

- [Creating business rules](#)
- [Creating a function](#)
- [Creating a technical rule](#)

Rule overriding

You use rule overriding to give rules precedence over other rules.

For example, you can set an entire decision table to override another decision table or decision tree.

A rule that overrides one or more other rules is executed in place of the rules that are overridden. Normally, rule overriding operates within a hierarchy of rules, with a local or more specific rule overriding a more general rule. For example, you can set a minimum customer age for a particular state to take precedence over a minimum customer age for the country.

The rule overriding hierarchy means that rules that are at the top of the hierarchy override rules at the bottom if they are found in the same rule task of a ruleflow.

You set rule overriding in the **Override Rules** step of the Compose wizard for the rule, by building the table of rules that you want the current rule to override. For example:

Rule	Type
bankruptcyScore	Overridden rule
defaultInsurance	Overridden rule

Parent topic: [Compose: Create project elements](#)

Related concepts:

[Overview: Ruleflows](#)

Related tasks:

[Creating business rules](#)

[Creating a technical rule](#)

Project element documentation

You can add project element documentation when you create or edit project elements.

You add any information that you consider useful or pertinent to that project element.

You write documentation in a free text field. This field can remain empty, although it is not good working practice to do so.

Note:

Do not confuse element documentation with version information. You display version information when you consult the history of an element.

Parent topic: [Compose: Create project elements](#)

Related concepts:
[Version information](#)

Version information

Version information is associated with the version to which you add it.

You enter version information when creating or editing a project element.

Decision Center creates a new version of all project elements when they are saved, so it is good working practice to enter a comment in the text area of the **Version Information**. This comment is associated with the version and is displayed in the table when you consult the history of the element.

Note:

For information about version numbers, refer to [Versioning](#).

Parent topic: [Compose: Create project elements](#)

Query: Search your projects

You use queries to search through your rule projects to display the business rules or other project elements that correspond to criteria of your choice.

Note: Queries in the Enterprise console only pertain to classic rule projects. To use this feature with decision services, you must use the Decision Center Business console.

[Introducing queries](#)

You can use queries to search through your rule projects.

[Query conditions](#)

Query conditions are the criteria that you define for the search.

[Query actions](#)

The default action of a query is to display the query results, but you can specify additional actions.

Parent topic: [Working with the Enterprise console](#)

Introducing queries

You can use queries to search through your rule projects.

Queries display the business rules or other project elements that correspond to criteria of your choice.

You can use queries to evaluate the impact of changes to your rules. When you obtain the query results, you can perform actions on them, generate a report, or perform checks.

Queries are made up of two parts:

- The **query** part, for example, Find all business rules such that the creator of each business rule is me
- The **action** part, for example Do set the status property to deployable

You can use queries throughout the Enterprise console: in smart folders, reports, when generating rulesets. You create, run, and save them on the **Query** tab.

Note:

Click the refresh icon  to check for updated queries run by concurrent users.

You construct queries in the same way that you construct rules. The query condition contains the criteria for the query action to be executed. The default query action is to just display the results, but you can also have the query carry out other actions such as moving the results of the query to another folder.

Queries run independently of what is currently selected in the **Explore** tab. They run against the current rule project, as well as any dependent projects if you specify this option.

To run a query, click the **Run Query** button. The query results display in the same tables as those on the **Explore** tab, which give you access to the same exploration features such as view details and history.

When your query completes, you can carry out the following tasks:

Apply Actions

If your query has an action part, the actions are performed only when you click the link. The action is performed on all the rules retrieved by the query.

Generate Report on Query Results

The report runs in another browser window and displays the content and properties of the results of the query, ready to print.

Parent topic: [Query: Search your projects](#)

Query conditions

Query conditions are the criteria that you define for the search.

The conditions of your query appear under the Find part of the query.

By default, you run queries on the business rules but you can also search for specific types of business rules, such as action rules, decision tables, or another type of rule specific to your company.

You can also search for folders (rule packages), ruleflows, smart folders, functions, or variable sets.

Note:

You can query only the working version of project elements, that is, the version referenced in the baseline currently selected in Decision Center.

You start by selecting the project element you want to query. You can then refine the query by adding a filter in the form of a `such that` statement, followed by condition statements.

Example

```
Find all business rules
such that the effective date of each business rule is after 31/1/2016
and the effective date of each business rule is before 28/2/2016
```

You can refine queries using a `such that` statement to filter different project items:

- Properties: see [Filter by properties](#)
- Business terms: see [Filter by business terms](#)
- Rule behavior: see [Filter by behavior of rules during execution](#)
- Rule impact: see [Filter by impact of rules during execution](#)

Note:

You can identify which rules have changed in the current session (by querying on `after my login time`) or which rules you have changed (by querying on `creator is me`).

Filter by properties

You can filter the query according to one of the properties of the project elements being queried (see [Project element properties](#)).

For example, the following query displays only business rules whose status is new:

```
Find all business rules
such that the status of each business rule is new
```

Filter by business terms

You can filter the query according to the business terms used in the conditions and actions of your rules. This type of filter is useful when trying to find rules affected by a policy change, for example:

```
Find all business rules
such that each business rule modifies the value of the insurance rate
```

The following query constructs find rules that use or modify business terms:

uses the value of

Returns rules that use the values of certain business terms.

Example

The following query returns all business rules that use loan grade values:

```
Find all business rules
such that each business rule uses the value of the loan grade in 'a report'
```

uses the phrase

Returns rules that call a given business term.

Example

The following query returns all business rules that use 'add to the corporate score in the loan report':

```
Find all business rules
such that each business rule uses the phrase [ add 'a number' to the
corporate
score in 'a report' ]
```

uses the phrase ... where

Returns rules that call a given business term to which a constraint is applied

Example 1

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can equal 100:

```
Find all business rules
such that each business rule uses the phrase [ set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100 ]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

Example 2

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can be at least 100:

```
Find all business rules
such that each business rule uses the phrase [ set the credit score of 'a
borrower'
to 'a number' , where 'a number' is at least 100 ]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 200;
```

Example 3

The following query returns all business rules that call the term that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a
number' to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When the elements mentioned in the constraint do not appear in a rule, the rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query would return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10
```

to 'the borrower';

modifies the value of

Returns rules that modify certain business terms.

Example

The following query returns all decision tables that modify the value of the insurance rate:

*Find all decision tables
such that each decision table modifies the value of 'the insurance rate'*

Filter by behavior of rules during execution

You can filter the query according to the semantics of the rules, that is, their behavior during execution. This type of filter finds rules that could be applicable when certain conditions are true or become true. This filter is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can use the following semantic query constructs:

may apply when

Returns all rules whose condition part could meet the query condition, or where there is nothing in the rule condition that would contradict the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

Example

Rule 1: If the score of the borrower is at least 10 then...

Rule 2: If the age of the borrower is at least 21 then...

Rule 3: If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

Query 1:

*Find all business rules
such that each business rule may apply when the score of the borrower is
20*

Query 2:

*Find all business rules
such that each business rule may apply when the score of the borrower is
5*

Query 1 returns Rule 1 and Rule 2. It returns Rule 1 because if the score of the borrower is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the score of the borrower could be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

Query 2 returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to less than 10. In Rule 2, nothing specifically stops the rule from being applicable when the score is 5. In Rule 3 at least one of the conditions could apply, and there is nothing in the other condition that negates the fact that the score could be 5.

may become applicable when

A more specific query that returns only those rules whose condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition.

Example

Rule 3: If the category of the customer is Platinum then...

Rule 4: If the category of the customer is not Platinum then...

Rule 5: If the age of the customer is at most 65 then...

Rule 6: If the age of the customer is at most 65 and the category of the customer is not Platinum...

Query 1:

*Find all business rules
such that each business rule may become applicable when the category of
'a customer' is Gold*

This query returns Rule 4 and Rule 6. It returns Rule 4 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 6 for the same reason. The additional condition relating to the age of the customer does not contradict the condition in which the category may be Gold.

The query does not return Rule 3 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 5 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer category is Gold.

Query 2:

*Find all business rules
such that each business rule may become applicable when [the age of
'a customer' is at least 21]*

This query does not return Rule 5 because it searches for rules that could “become” applicable when the customer age is over 21. Rule 5 is applicable even if the customer age is under 21. Therefore Rule 5 does not “become” applicable, but it “remains” applicable even if the customer age changes from under 21 to over 21.

may lead to a state where

Returns rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of a rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

Example

Rule 7: If the age of the borrower is at least 25 then set the credit score of the borrower to 60

Rule 8: If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

Query:

*Find all business rules
such that each business rule may lead to a state where the credit score
of
the borrower is more than 50*

This query returns Rule 7 but not Rule 8 because, when the age of the borrower has been checked and the credit score set, only Rule 7 shows a result of over 50.

Filter by impact of rules during execution

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule impacts the applicability of other rules, and how a rule is impacted by the execution of other rules.

may select

Returns the ruleflows and rule tasks that may select a given rule.

Example

The following query returns the ruleflows and rule tasks that may select "Rule 1".

*Find all ruleflows
such that each ruleflow may select "Rule 1"*

may enable rule

Returns rules that make a given rule applicable.

Example

Rule 1:

if the age of the customer is 25

then set the category of the customer to Bronze ;

Rule 2:

if the category of the customer is Bronze
then give Champagne to the shopping cart of the customer with message:
"Congratulations" ;

Query:

Find all business rules
such that each business rule **may enable** "Rule 2"

This query returns Rule 1 because it sets the category of the customer to Bronze, and thus makes the condition of Rule 2 valid.

may disable

Returns rules that make a given rule inapplicable

Example

Rule 1:

if the age of the customer is 18
then set the category of the customer to Copper ;

Rule 2:

if the category of the customer is Bronze
then give Champagne to the shopping cart of the customer with message:
"Congratulations" ;

Rule 3:

if the age of the customer equals 25 and the category of the customer is
Bronze
then set the category of the customer to Diamond ;

Query:

Find all business rules
such that each business rule **may disable** "Rule 2"

This query returns Rule 1 and Rule 3. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It returns Rule 3 because although one of the conditions is that the category of the customer is Bronze, the action is to set the category of the customer to Diamond. Therefore, if the category of the customer is Diamond, Rule 2 is not applicable.

may be enabled by

Returns rules that are made applicable by a given rule

Example

Rule 1:

if the age of the customer is 18
then set the category of the customer to Copper ;

Rule 2:

if the category of the customer is Bronze
then give Champagne to the shopping cart of the customer with message:
"Congratulations" ;

Rule 3:

```
if the age of the customer equals 25 and the category of the customer is
Gold
then set the category of the customer to Diamond ;
```

Query:

```
Find all business rules
  such that each business rule may disable "Rule 2"
```

This query returns Rule 1. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It does not return Rule 3 because the action is to set the category of the customer to Diamond, and it can only be executed from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

may be disabled by

Returns rules that are made inapplicable by a given rule

Example

Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

Rule 2:

```
if the age of the customer equals 25 and the category of the customer is
Copper
then set the category of the customer to Gold
```

Query:

```
Find all business rules
  such that each business rule may be disabled by "Rule 1"
```

This query returns Rule 2 because it is made inapplicable by Rule 1. In Rule 1 the category of a customer whose age is 25, is set to Bronze, therefore it cannot be Copper.

Parent topic: [Query: Search your projects](#)

Query actions

The default action of a query is to display the query results, but you can specify additional actions.

You can add further actions by specifying them under the **Do** part of the query.

Query actions can make the following changes:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add or remove a category to the displayed elements
- Set any of the properties of the displayed elements to a value that you specify

Example

```
Find all business rules  
  such that the status of each business rule is new  
Do set the status of each business rule to validated
```

The actions are carried out on all the displayed elements when you click **Apply Actions** in the **Query** tab.

Parent topic: [Query: Search your projects](#)

Related concepts:
[Project element properties](#)

Analyze: Check your projects

You can check your projects for consistency and completeness and generate rule project reports.

Note: The topic in this section about [Managing merge reports](#) only pertains to classic rule projects.

[Generating a project report](#)

You use HTML reports that show the content and properties of project elements.

[Managing merge reports](#)

You can access and manage the reports created when you merge branches.

Parent topic: [Working with the Enterprise console](#)

Generating a project report

You use HTML reports that show the content and properties of project elements.

If you are viewing a branch or baseline when you generate the report, the report describes the elements of the branch or baseline.

To generate a project report in HTML format:

- Select the **Analyze** tab and then click **Generate Project Report**.

You can base your report on any of the existing queries available on the **Query** tab. If the rule project has no queries, the report uses all the rules in the rule project.

The report is generated in another browser window.

Note:

You can also generate the report directly from the **Query** tab.

Parent topic: [Analyze: Check your projects](#)

Related information:

[Query: Search your projects](#)

Managing merge reports

You can access and manage the reports created when you merge branches.

About this task

Decision Center keeps an Excel version of the report when you merge branches. You can download these reports and manage them from the **Analyze** tab.

Procedure

To manage the reports of merged branches:

1. On the **Analyze** tab, click **Manage Merge Reports**.
2. Select the branch for which you want to see the reports.

Decision Center lists the reports for that branch.

3. Click the name of the report in the table to download it, or select it in the table and click **Rename** to rename it or **Delete** to delete it.

Parent topic: [Analyze: Check your projects](#)

Related tasks:

[Branches](#)

[Managing subbranches](#)

Related information:

[Merging branches](#)

Project: Manage your project

You use the Project tab to manage advanced features of your projects.

Note: The topics in this section about [Managing subbranches and baselines](#), [Merging branches](#), [Dynamic domains](#) only pertain to classic rule projects. To merge branches or use dynamic domains with decision services, you must use the Decision Center Business console.

[Managing subbranches and baselines](#)

From any branch, including the main, you can manage subbranches, baselines, and deployment baselines.

[Merging branches](#)

You can merge branches together.

[Project dependencies](#)

When you set up a project to access the contents of other projects, you create a project dependency.

[Project options](#)

You can set options on the project.

[BOM path](#)

A BOM path entry defines a set of business elements in the business object model.

[Dynamic domains](#)

Domains are visible as sets of values in your business terms.

Parent topic: [Working with the Enterprise console](#)

Managing subbranches and baselines

From any branch, including the main, you can manage subbranches, baselines, and deployment baselines.

Managing subbranches

You can create subbranches, and then manage them by renaming or deleting them.

Managing baselines

You can create baselines, and then manage them by renaming, deleting, cloning them to a branch, or restoring them.

Managing deployment baselines

You create deployment baselines when deploying RuleApps, and then manage them by renaming, deleting, or cloning them to a branch.

Parent topic: [Project: Manage your project](#)

Related tasks:

[Baselines](#)

[Branches](#)

[Versioning](#)

Related information:

[Merging branches](#)

Managing subbranches

You can create subbranches, and then manage them by renaming or deleting them.

About this task

Branches facilitate work done on parallel releases of a project.

Initially, a project contains only a main line in which you work. From the main, sometimes referred to as the main branch, you can create one or more subbranches. From any new branch you can create other subbranches, as many as you need to manage your releases.

The way you manage branches depends on how many releases you need to manage at a given time:

- If you only have **one release in production** at any given time, use the main line as the production branch. Then, to work in parallel on an upcoming release, create a branch for the new release. Changes to production continue in the main, from which you deploy, and changes for the upcoming release are made in the new branch. Periodically, you can merge the production changes into the upcoming release branch so that the upcoming release always has the most up-to-date rules. When you are ready for the upcoming release to go into production, merge the changes from the subbranch to the main, so the main continues to represent the production release. At this point, the branch receives no further edits and exists only for the historical record.
- If you have **multiple releases in production** at a given time, you need multiple active subbranches, each representing a production release. The main represents work in progress on an upcoming release. You may also be working in parallel on multiple upcoming releases, so some subbranches will also represent upcoming releases. By the time any given release goes to production, it must have its own branch. Production changes are made in each of these subbranches, and merged into other branches as needed. As you retire releases from production, these branches receive no further edits and exist for the historical record.

Note:

When keeping a branch for the historical record, you should baseline the branch and have the Administrator remove permissions to edit it.

Procedure

To manage subbranches:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose subbranches you want to manage. By default, Decision Center selects the branch you are currently working in and displays its subbranches in the table.
3. In the **Subbranches** section of the page, click **New** to create a subbranch.

Note:

You can also create a subbranch directly from the **Home** tab.

If the subbranch already exists, select it and click one of the following:

- **Rename:** To rename the subbranch.
- **Delete:** To delete the subbranch. You cannot delete a branch if you are currently in it or one of its subbranches. You have to be viewing a parent branch.

Parent topic: [Managing subbranches and baselines](#)

Related concepts:

[The Home tab](#)

Related tasks:

[Branches](#)

[Versioning](#)

Related information:

[Merging branches](#)

Managing baselines

You can create baselines, and then manage them by renaming, deleting, cloning them to a branch, or restoring them.

About this task

Baselines are designed to capture the state of a project or branch at a given moment in time. Each branch of a project can contain many baselines.

You cannot edit a baseline; they are for consultation only. You consult them by selecting them in the **Home** tab.

Note:

Note also the existence of deployment baselines, which you can create when deploying RuleApps (see [Managing deployment baselines](#)).

Baselines that you create appear in the table of baselines for that branch.

When you create a baseline, it includes any project dependencies.

In addition to creating baselines, you can do the following as part of managing baselines:

Delete a baseline

When you delete a baseline you do not delete any project elements, only the snapshot itself.

Clone to branch

You can create a branch from a baseline. You can then continue editing the content as part of a branch, which can then be merged with another branch. The original baseline remains intact when you clone it to a branch.

Restore a baseline

You can restore a baseline so that it becomes the current state of the project or branch.

Project elements created after the baseline was created are overwritten when that baseline is restored. Consequently, before restoring a baseline, you might want to create a temporary baseline to keep a trace of any new project elements that have not been captured in the initial baseline.

Note:

If you lose new project elements when restoring a baseline, you can retrieve them from the Decision Center recycle bin.

Procedure

To manage baselines:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose baselines you want to manage. By default, Decision Center selects the branch you are currently working and displays its baselines in the table.
3. In the **Baselines** section of the page, click **New** to create a baseline.

If the baseline already exists, select it and click one of the following:

- **Rename:** To rename the baseline.
- **Delete:** To delete the baseline.
- **Clone to branch:** To create a new branch from the contents of the baseline.
- **Restore:** To restore the baseline.

Parent topic: [Managing subbranches and baselines](#)

Related concepts:
[The Home tab](#)

Related tasks:
[Baselines](#)
[Versioning](#)
[Recycle bin](#)

Managing deployment baselines

You create deployment baselines when deploying RuleApps, and then manage them by renaming, deleting, or cloning them to a branch.

About this task

Deployment baselines capture the version of the elements contained in the RuleApp at the time they were deployed. Then, to redeploy the RuleApp at some later time, you can choose whether the version of the elements corresponds to the initial deployment or to a subsequent redeployment.

Deployment baseline are similar to standard baselines, but have the following differences:

- You cannot create them manually; you create them as part of RuleApp deployment.
- Since they are created from the ruleset extraction, there is one deployment baseline created per ruleset contained in the RuleApp. Decision Center names deployment baselines **Name Given / RuleApp version / ruleset version**.
- Because RuleApps are independent of projects, deployment baselines created in a project are visible in any dependent project.

You can consult existing deployment baselines by selecting them in the **Home** tab. You can do the following as part of managing deployment baselines:

Delete a deployment baseline

When you delete a deployment baseline you do not delete any project elements, only the snapshot itself.

Clone to branch

You can create a branch from a deployment baseline. You can then continue editing the content as part of a branch. The original deployment baseline remains intact.

Procedure

To manage deployment baselines:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose deployment baselines you want to manage. By default, Decision Center selects the branch you are currently working and displays its deployment baselines in the table.
3. In the **Deployment Baselines** section of the page, select the entry in the table and click one of the following:
 - **Rename**: To rename the deployment baseline.
 - **Delete**: To delete the deployment baseline.
 - **Clone to branch**: To create a new branch from the contents of the deployment baseline.

Parent topic: [Managing subbranches and baselines](#)

Related tasks:

[Baselines](#)

[Versioning](#)

Merging branches

You can merge branches together.

[Using branch merging](#)

You can merge branches together.

[Using Diff and Merge](#)

Use Diff and Merge when there are several changes to a rule and you want to specify which of these changes to merge.

Parent topic: [Project: Manage your project](#)

Related tasks:

[Branches](#)

[Managing subbranches](#)

[Versioning](#)

[Managing merge reports](#)

Using branch merging

You can merge branches together.

About this task

When you merge branches, you decide what to do with each project element that is different between the two branches:

- Replace the version in the target branch
- Update the source branch with the version found in the target branch
- Take no action at all

After you specify the two branches you want to merge, Decision Center looks at both branches and displays a table containing all the differences. This table contains the name and folder of project elements that are different between the branches, and their state in both branches. This table also contains an **Actions** column, containing a proposed action to take. You can click inside the cells of this column and change individual actions as you see fit.

Filter:					
Name ▲	Folder	main	Spring Updates	Action	
approval	eligibility	Unmodified	Deleted	Delete from 'main'	
<i>checkIncome</i>	<i>eligibility</i>	<i>In Conflict</i>	<i>In Conflict</i>	No action to perform	
Rule 1	eligibility	-	New	Add in 'main'	

Click next to the **Action** column when there are several changes to the rule and you want to specify which of these changes to merge (see [Using Diff and Merge](#)). This option is only available with action rules or decision tables.

The elements taken into account during a merge operation are:

- Rule Artifacts (BAL rules, technical rules, decision tables, decision trees, functions)
- Simulation artifacts (metrics, KPIs, simulation models, input data, simulations)
- Testing artifacts (test suites, test cases)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- BOM
- Vocabulary
- B2X
- Resources
- Folders
- DVS artifacts (scenario suites for testing and simulation)
- Templates

Note:

Changes to ruleset parameters, extractors, project dependencies, categories and dynamic XOM (schemas), cannot be merged. You must change these manually in the other branch.

To help Decision Center propose the correct action to take, you can specify your preferred direction for merging from the following options:

Bidirectionally

This is the default option. The proposed action is based on the assumption that you want to reflect changes made in either branch as follows:

- If a new rule exists in one branch, you want to add it to the other branch.
- If a rule has been deleted in one branch, you want to delete it from the other branch.
- If a rule has been modified in one branch but not in the other, you want to update the unmodified one.
- If a rule has been modified in both branches, no action is proposed, and you must specify what action to take.

Only to target branch

When you want all changes you made in the working branch to be pushed to the target branch, but none of the changes made in the target branch to be reflected in your working branch.

Only to source branch

When you want all changes made in the target branch to be reflected in your working branch, but none of the changes made in your working branch to be pushed to the target branch.

Procedure

To merge a branch:

1. From one of the branches to merge, go to the **Project** tab and click **Merge Branches**.
2. Select the target branch to merge with and click **Next**.
3. Change the direction in which to merge changes if different from **Bidirectionnally**.
4. Specify the actions to take and click **Apply Merge**.

Results

Decision Center shows the details of the merge. You can access an Excel version of the report from the **Analyze** tab.

Parent topic: [Merging branches](#)

Related tasks:

[Branches](#)

[Managing subbranches](#)

[Using Diff and Merge](#)

[Versioning](#)

[Managing merge reports](#)

Using Diff and Merge

Use Diff and Merge when there are several changes to a rule and you want to specify which of these changes to merge.

Before you begin

When merging decision tables in which columns have been added or removed, you must resolve column differences before you resolve row differences. If you try to copy a row before you have copied columns, a message prompts you to apply the requested change to the columns first.

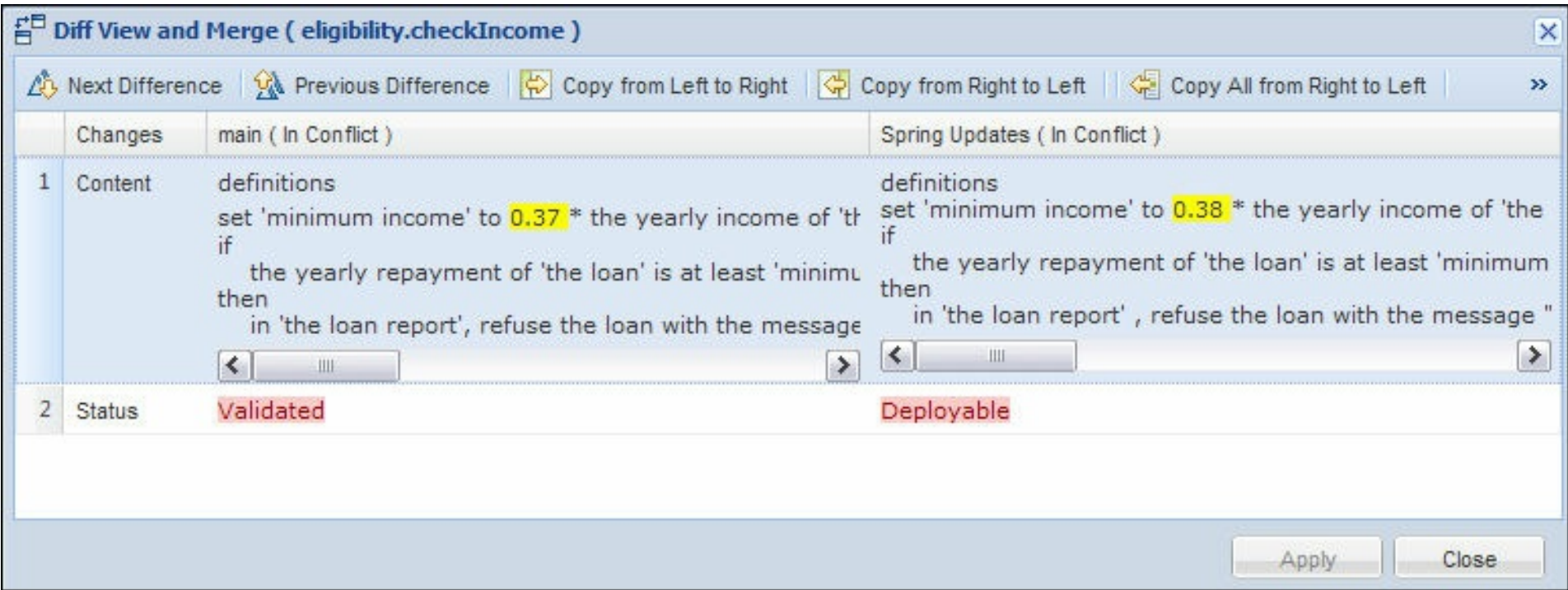
When merging decision table differences, the following differences are detected and collected together in a single row labeled Decision Table Properties in the Diff and Merge table. This means that the only option available to you is to copy them as a whole from one side to the other:

- Settings in the Table Properties editor, such as table view settings, locks, table checks, and preconditions.
- Column options such as column width, locks, and default value.

For example, you cannot copy table view settings from left to right and preconditions from right to left. This restriction does not apply to differences in properties that are defined in Step 1: Properties of the Compose wizard, such as name and status: these differences are listed in separate rows in the Diff and Merge table.

About this task

Use Diff and Merge when there are several changes to the rule and you want to specify which of these changes to merge. Diff and Merge displays the contents and properties of each branch of the project element side by side. You can step through each difference in turn, visually compare the values, decide which value you want to retain, and copy it from one side to the other. You can request a partial merge by resolving some differences and retaining others.



Procedure

To use Diff and Merge:

1. In the Diff and Merge menu bar, click **Next Difference**. Diff and Merge highlights the next available difference in the project element.
2. Take one of the following actions:
 - To merge the difference, click **Copy from Right to Left** or **Copy from Left to Right** depending on which value you want to retain. No changes are made to the project element at this stage.
 - To retain this particular difference, click **Next Difference**.
3. Repeat the previous steps to work through each instance of a difference that you intend to resolve. You do not need to resolve every difference.

Note:

To undo the most recent individual merge, click **Revert**.

4. When you have finished specifying how you want to resolve each difference, click **Apply**.

Results

Diff and Merge closes and you return to the table displaying the list of project elements that contain differences in the two selected branches. At this point, the changes you have specified are not committed. The project element you have worked on displays the action **Apply Merge**, indicating that this project element is now ready for you to apply the merge. For information on how to apply the merge to commit the changes, see [Using](#)

[branch merging.](#)

Parent topic: [Merging branches](#)

Related tasks:

[Branches](#)

[Managing subbranches](#)

[Using branch merging](#)

[Versioning](#)

Project dependencies

When you set up a project to access the contents of other projects, you create a project dependency.

You can set up a project or branch to access the contents of other projects or branches of that project. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that several rule projects can access the same BOM and vocabulary without duplication.

You set up project dependencies on the **Project** tab, under **Edit Project Dependencies**.

Note:

You must sign in to the Enterprise console with **Configuration Manager** permissions to be able to set up project dependencies.

When you create a project dependency, you establish a reference from your project to another project. When you have established a dependency, you can specify whether or not the smart folders and queries that you use in your projects include the dependent projects.

When you create a project dependency, you specify which project to reference and whether to reference the main of that other project or one of its branches or baselines. Keep in mind that when you create a baseline or branch of a project, the baseline or branch also keeps any dependencies. It stores the reference to the other project and not the actual contents of the other project.

Parent topic: [Project: Manage your project](#)

Project options

You can set options on the project.

You can set the following project options on the **Project** tab, under **Edit Project Options**.

Build options

Allow ruleset archive generation to be cancelled on warnings or errors, or not at all. When you select **Build automatically**, Decision Center generates the IRL for a rule as soon as you save it. Otherwise, this generation occurs when generating the ruleset. You might find enabling this option useful to accelerate ruleset generation.

Rule editor

Select the default rule editor to be used for editing elements that belong to this project. Options: **Guided editor** and **Intellirule editor**. Users can override the default rule editor on the User Options page, which you access from the top banner of the Enterprise console.

Show number of elements in the smart folders

Smart folders show the number of elements they contain.

Refresh children on expand in the smart folders

Folders are refreshed automatically when you move from one to another in the smart folders.

Parent topic: [Project: Manage your project](#)

Related tasks:

[Selecting your preferred rule editor](#)

BOM path

A BOM path entry defines a set of business elements in the business object model.

In the Enterprise console you can display the BOM path by clicking **View BOM Path** on the **Project** tab.

Note:

You can view the BOM path only if you sign in with **Configuration Manager** permissions.

The business object model (BOM) makes business rule editing user-friendly by providing tools that you can use to set up a natural-language vocabulary. Policy managers can then use this vocabulary to describe their business logic in a business rule language.

A BOM path comprises one or more BOM path entries. BOM entries can target BOM files or BOMs from other projects.

You can order BOM entries so that if you have two business elements with the same name in two BOM entries, the one in the first BOM entry in the path overrides the other.

Note:

For more information, see the Rule Designer user documentation.

Parent topic: [Project: Manage your project](#)

Related concepts:
[Project dependencies](#)

Dynamic domains

Domains are visible as sets of values in your business terms.

With a dynamic domain, the set of values is stored and managed outside the business object model. Changes to the set of values are then reflected in the BOM. When you write business rules that use domains, the list of values that you choose from is created dynamically, so you never have to change the BOM manually.

Domains are available to Decision Center through a domain provider. The domain provider can be managed externally or by you in an Excel file. In either case, you must upload any changes to the domain to Decision Center.

Storing domains in Excel

You can modify the values of dynamic domains that are stored in an Excel file.

Updating dynamic domains

When your domain values change, you must update them in Decision Center.

Parent topic: [Project: Manage your project](#)

Storing domains in Excel

You can modify the values of dynamic domains that are stored in an Excel file.

About this task

When a project contains a dynamic domain defined in an Excel file, the Excel file is stored as a resource in the project. You can modify the Excel file and then upload the changes to Decision Center.

The Excel file has a specific structure with columns that map properties in the BOM:


- A column for the domain values
- A column for the label of each value
- A column that defines the BOM-to-XOM mapping for each value

Other optional columns for the documentation or labels in other locales might be defined in the Excel file.

You can modify the values in each column, but you must not change the columns or column headings.

Procedure

To edit the Excel file:

1. In the smart folder dedicated to resources, browse to the resource corresponding to the Excel file. If the smart folder does not exist, create it. In **Step 1: Properties**, select **Include dependencies** if the Excel file is in a dependent project, and in **Step 2: Query**, use the query Find all resources to populate the smart folder.
2. Select the Excel file in the table, and click  **Download this element**.
3. Save the file on your local drive.
4. Open the Excel file, modify the values, and save the changes.

You can now upload the modified Excel file to Decision Center.

5. In the **Explore** tab, make sure that the Excel file is selected, and click **Edit**.
6. Click **Browse** to select the modified Excel file.
7. Click **Finish**.
8. To update the BOM with the new values, click **Update Dynamic Domains** in the **Project** tab.

Parent topic: [Dynamic domains](#)

Related tasks:

[Creating a resource](#)

[Updating dynamic domains](#)

Updating dynamic domains

When your domain values change, you must update them in Decision Center.

About this task

Your domains can come from an outside provider or from an Excel file that you store as a resource. If your domain values change, you update Decision Center as follows:

Procedure

1. Click **Update Dynamic Domains** in the **Project** tab. All domains available in the current project are displayed.
2. Select the domains that you want to update. Click **Ready to be updated** to see only the domains that changed.
3. Click **Update**.

Parent topic: [Dynamic domains](#)

Related tasks:

[Creating a resource](#)

[Storing domains in Excel](#)

Working with the editors

You can choose which editor to use when editing business rules.

Using the rule editors

You can use the guided editor or the Intellirule editor to edit action rules in Decision Center.

Editing decision tables

When you have created your decision table and its properties, you can use the decision table editor.

(Deprecated) Editing decision trees

You can use the decision tree editor when the decision tree and its properties have been created at the project level, that is, in the Tree part of the Compose wizard.

Previewing ruleflows

You cannot edit ruleflows in the Enterprise console.

Parent topic: [Working with the Enterprise console](#)

Using the rule editors

You can use the guided editor or the Intellirule editor to edit action rules in Decision Center.

Selecting your preferred rule editor

You can select the editor you use for editing action rules in Decision Center.

Building rules using the guided editor

You use the guided editor to construct rules by following the predefined rule structure and adding rule fragments to it.

Building rules using the Intellirule editor

You use the Intellirule editor to construct rules by inserting terms and phrases.

Parent topic: [Working with the editors](#)

Selecting your preferred rule editor

You can select the editor you use for editing action rules in Decision Center.

Procedure

To select your preferred rule editor:

1. In the top banner, click **Options**.
2. In the “Select the rule editor” area, do one of the following procedures:
 - Click **Default rule editor defined for the current project** to use whichever editor is selected for the current project.
 - Click **Guided editor** to use the Guided Editor every time you edit an action rule.
 - Click **Intellirule editor** to use the Intellirule editor every time you edit an action rule.

Parent topic: [Using the rule editors](#)

Related information:

[Building rules using the guided editor](#)

[Building rules using the Intellirule editor](#)

Building rules using the guided editor

You use the guided editor to construct rules by following the predefined rule structure and adding rule fragments to it.

[Overview of the guided editor](#)

The guided editor presents you with a rule outline showing the different rule parts.

[Adding or removing rule statements using the guided editor](#)

When using the guided editor, you can refine your rules by adding or removing rule statements.

[Controlling how condition statements are combined](#)

You can control multiple condition statements in a rule.

[Changing the grouping order of rule statements](#)

You can use parentheses to group rule statements.

[Negating a condition statement](#)

You can negate a condition statement in the guided editor.

[Inserting an arithmetic expression](#)

You can build complete arithmetic expressions to refine your rule statements.

[Creating a new variable](#)

You can use variables to identify and subsequently reference an occurrence of something by a convenient name.

[Opening an empty part of a rule](#)

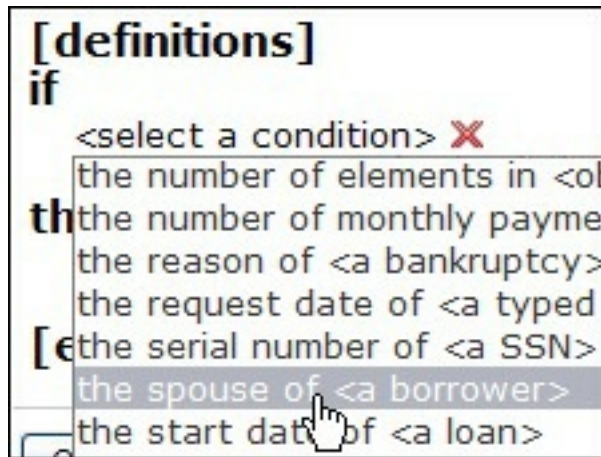
You can open an empty part of a rule and insert a statement in it.

Parent topic: [Using the rule editors](#)

Overview of the guided editor

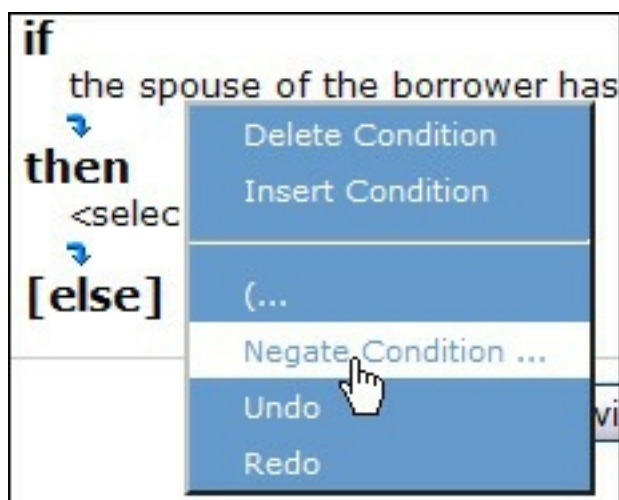
The guided editor presents you with a rule outline showing the different rule parts.

You construct rules from the predefined rule fragments by entering text or numbers in fields, or by selecting items in drop-down lists:



A business term can be part of a longer phrase. For example, if you select an item such as the spouse of <a borrower> from the drop-down list, you must subsequently complete the business term <a borrower> in that phrase.

You can right-click any part of a rule, including fields. Right-clicking provides a drop-down list appropriate to that part of the rule, known as a contextual menu.



Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:



[Creating business rules](#)

Adding or removing rule statements using the guided editor

When using the guided editor, you can refine your rules by adding or removing rule statements.

Procedure

To add or remove a rule statement after the last existing statement:

1. To add a rule statement after the last existing statement:
 - a. Click the add icon under any rule statement  .
2. To add a rule statement:
 - a. Right-click the first business term of the existing statement
 - b. Select **Insert Condition** from the drop-down list.
3. To delete a rule statement:
 - a. Click the red cross found next to the rule statement 
 - b. Alternatively, right-click the first business term and then select **Delete Condition** from the drop-down list.

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Controlling how condition statements are combined

You can control multiple condition statements in a rule.

About this task

When the condition part of a rule contains more than one statement, you can control whether the condition part of the rule is met if *all* (represented by the keyword and) or *any* (represented by the keyword or) of its rule statements are valid.

Procedure

To change how condition statements are combined:

1. Click the **and** or **or** keyword.
2. Choose the appropriate keyword.



Results

For more information about how and and or are interpreted, see [Condition negation](#).

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Changing the grouping order of rule statements

You can use parentheses to group rule statements.

About this task

By default, the and keyword takes precedence over the or keyword when you have several rule statements. Parentheses let you group rule statements to change the order in which they are processed.

Procedure

To add or remove parenthesis to a rule statement:

1. To add an opening parenthesis to a rule statement:
 - a. Right-click the operator or business term.
 - b. Select **...(** from the drop-down list.
2. To add a closing parenthesis to a rule statement:
 - a. Right-click the last value of the statement (or right-click the keyword of the next statement).
 - b. Select **)...** from the drop-down list.
3. To remove a parenthesis:
 - a. Right-click the parenthesis.
 - b. Select **Delete** from the drop-down list.

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Negating a condition statement

You can negate a condition statement in the guided editor.

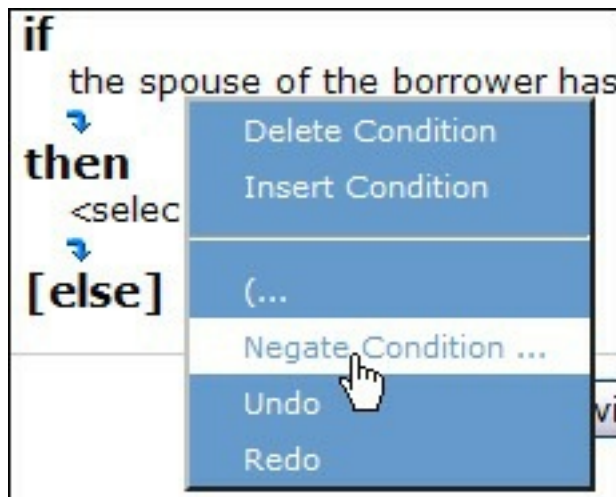
About this task

You do so by adding `it is not true` at the beginning of the statement. For more information about negating condition statements, refer to [Condition negation](#).)

Procedure

To insert or remove 'it is not true' in a condition statement:

1. To insert 'it is not true' in a condition statement:
 - a. Right-click the start of the statement (either the operator or the business term).
 - b. Select **Negate Condition** from the drop-down list.



2. To remove 'it is not true' from the statement:
 - a. Right-click **it is not true**.
 - b. Select **Delete** from the menu.

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Inserting an arithmetic expression

You can build complete arithmetic expressions to refine your rule statements.

Procedure

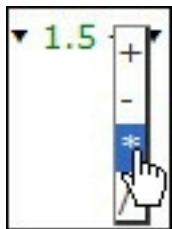
To insert the arithmetic operator:

1. Select \pm in a rule statement.



The multiplication operator (*) displays.

2. Click the multiplication operator to change it to another operator.



Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Creating a new variable

You can use variables to identify and subsequently reference an occurrence of something by a convenient name.

Procedure

To create a variable:

1. Click **definitions** to expand it if it is empty.
2. Click variable1 in the definitions part and then enter the name of your variable.
3. Click <select a choice> and then select how you want to set your variable.

For information on setting variables in the definitions part of a rule, see [Action rules](#).

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Opening an empty part of a rule

You can open an empty part of a rule and insert a statement in it.

About this task

Brackets [] around the title of a part of a rule means that it is currently empty. For example, the definitions and else parts are empty by default when you create a new rule.

Procedure

To open an empty part of a rule:

1. Click the title.
2. Insert a statement in that part.

Parent topic: [Building rules using the guided editor](#)

Related concepts:

[Action rules](#)

Related tasks:

[Creating business rules](#)

Building rules using the Intellirule editor

You use the Intellirule editor to construct rules by inserting terms and phrases.

Overview of the Intellirule editor

When you use the Intellirule editor to build a rule, you add the appropriate terms and phrases to the editing area. You can add terms and phrases by typing or pasting text or by selecting terms and phrases from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, a value, or some other phrase.

Displaying the completion menu

You can display the completion menu as you build your rule to view the list of terms and phrases that are valid in the current context of the rule, and to select a valid term or phrase.

Setting completion menu options

You can set options for the way the Intellirule editor presents terms and phrases in the completion menu and for the way rules are parsed.

Activating and deactivating keyword filtering

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

Highlighting and correcting errors using the Problem List

The Intellirule editor highlights errors in red in the editing area and lists them in the Problem List under the editing area.

Parent topic: [Using the rule editors](#)

Overview of the Intellirule editor

When you use the Intellirule editor to build a rule, you add the appropriate terms and phrases to the editing area. You can add terms and phrases by typing or pasting text or by selecting terms and phrases from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, a value, or some other phrase.

Note:
Look for the **Rule editor flash demo** on the support site for an English only tour of the Intellirule editor features: [IBM® Customer Support site](#)

Terms and phrases

The Intellirule Editor provides an editing area into which you add terms and phrases to build rules. You can add terms and phrases in the following ways:

- Type directly in the editing area
- Copy text from another editor or application, and paste it into the editing area
- Select predefined terms and phrases from a completion menu

Completion menu

The terms and phrases available to you in the completion menu and the different ways in which you can combine them are defined in the business vocabulary of your company.

The following table displays the icons that the completion menu uses to identify the different types of elements that are available for selection.

I c o n	Description
	Term. A business term defined in the vocabulary of your business domain.
	Constant or text string.
	Predicate phrase. A phrase with a value of either true or false. See Rule conditions .
	Navigation phrase. A phrase that refers to an attribute of a term. See Rule conditions .
	Action phrase. See Rule actions .
	Arithmetic operator.
	Automatic variable. A variable that is automatically declared for each term in the vocabulary of your business domain.
	Explicit variable or ruleset parameter. A variable that is declared in the definitions part of a rule, or a parameter that is declared by the rule application.
	Rule language construct.

Placeholders

Placeholders indicate places in a term or phrase that you must complete by inserting the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain additional placeholders that you must also complete to construct a valid expression.

Parent topic: [Building rules using the Intellirule editor](#)

Related tasks:
[Displaying the completion menu](#)
[Setting completion menu options](#)

Displaying the completion menu

You can display the completion menu as you build your rule to view the list of terms and phrases that are valid in the current context of the rule, and to select a valid term or phrase.

About this task

If you do not want the completion menu to open automatically as you type, deactivate the **Enable on Spacebar** option.

Procedure


To display the completion menu:

Choose one of the following methods:

- If you have selected the **Enable on Spacebar** option, click anywhere in the business rule and start typing. The completion menu opens when you press **Spacebar** or **Ctrl+Spacebar**.
- If you have not selected the **Enable on Spacebar** option, click anywhere in the business rule and then press **Ctrl+Spacebar**.
- Click a placeholder.
- If you have selected the **Enable on double-click** option, double-click a word in the editing area.

Results

The completion menu displays a list of available insertion options based on your current position in the rule.

You can hide the completion menu by clicking **Close**  in the top corner of the completion menu or by pressing **Esc**.

Parent topic: [Building rules using the Intellirule editor](#)

Related tasks:


[Setting completion menu options](#)

Setting completion menu options

You can set options for the way the Intellirule editor presents terms and phrases in the completion menu and for the way rules are parsed.

Procedure

To set completion menu options:

1. In the toolbar above the editing area, click **Completion Menu Options** . A pop-up window opens.
2. Select the check boxes for the options you want to set:
 - **Enable on Spacebar:**
 - Select this check box if you want to open the completion menu whenever you press the Spacebar while typing in the editing area.
 - Clear this check box if you want the completion menu to open only when you press **Ctrl+Spacebar**.
 - **Enable on double-click:**
 - Select this check box if you want to open the completion menu by double-clicking a word in the editing area.
 - Clear this check box if you want to suppress opening of the completion menu by double-clicking a word in the editing area. Double-clicking selects the current word instead.
 - **Enable auto-restart:**
 - Select this option to automatically reopen the completion menu when a term or phrase is selected.
 - Clear this check box if you want to suppress opening of the completion menu when a term or phrase is selected. You open the completion menu by pressing **Ctrl+Spacebar**.
 - **Enable smart mode:**
 - Select this check box to filter the completion menu entries in a smart way to reduce the number of entries.
 - Clear this check box to disable smart mode.
 - **Enable template mode:**
 - Select this check box if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable completion menu template mode if you prefer to insert longer phrases first and then substitute placeholders.
 - Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate completion menu template mode if you prefer to build the rule systematically from beginning to end in the same way that you might compose an email message.
 - **Use Hierarchical View:**
 - Select this check box if you want the completion menu to group terms and phrases together by type; for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.
 - Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.
 - **Filter unreachable phrases:**
 - Select this check box if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable is defined.
 - Clear this check box if you want the completion menu to display phrases even though they cannot be used because no suitable ruleset parameter or rule variable is defined.
 - **Display toolbar:**

- Select this check box if you want the completion menu toolbar to be displayed above the list of available terms and phrases.
- Clear this check box if you want to hide the completion menu toolbar.

- **Display documentation:**

- Select this check box if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.
- Clear this check box if you want to hide the hover help.

3. Click outside the pop-up window to save your settings.

Parent topic: [Building rules using the Intellirule editor](#)

Related tasks:

[Displaying the completion menu](#)

Activating and deactivating keyword filtering

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

About this task

The completion menu automatically filters the list of terms and phrases as you type text and build your rule. The list becomes progressively more restricted as you continue to type. The way in which automatic filtering works depends on whether you activate or deactivate keyword filtering.

Keyword filtering is deactivated by default. Whenever you type while the completion menu is displayed, the text is entered directly in the editing area, and the list of completion menu options is restricted to terms and phrases that start with the text that you type until you have completed a term or have selected a term or phrase from the completion menu.

When you activate keyword filtering, the completion menu includes a filter field above the list of options. Whenever you type while the completion menu is displayed, the text is entered in the filter field instead of in the editing area and the list of completion menu options is restricted to terms and phrases that contain the filter text. The list includes all items that contain the keyword, not just those that start with the keyword.

You can toggle between activating and deactivating keyword filtering as you build your rule.

Procedure

To toggle between activating and deactivating keyword filtering:

Click the **Toggle Keyword Filtering** button  at the top of the completion menu.

Results

When you activate keyword filtering, the **Filter** field is displayed above the list of completion menu options. When you deactivate keyword filtering, the **Filter** field is hidden from the completion menu.

Parent topic: [Building rules using the Intellirule editor](#)

Highlighting and correcting errors using the Problem List

The Intellirule editor highlights errors in red in the editing area and lists them in the Problem List under the editing area.

About this task

If the Problem List is too small or not visible at all, you can increase its size by dragging the horizontal dividing line of the editing area.

Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
 - Hover the mouse over a highlighted error in the Intellirule Editor to display an error tip icon.
 - Double-click the text displayed in the Message column on an error line in the Problem List to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

Parent topic: [Building rules using the Intellirule editor](#)

Editing decision tables

When you have created your decision table and its properties, you can use the decision table editor.

Decision table editor

You edit a decision table by clicking on the row, header, or cell you want to edit, which displays the editor with the appropriate content.

Adding, removing, and editing columns

You can modify the column structure of your decision tables to define the conditions and actions of all the rules in the table.

Adding, removing, and editing rows

You can add rows to and remove rows from a decision table, and edit the contents of a row using the decision table editor.

Editing cell values

You can edit decision table cells directly in the table, or using the decision table editor.

Disabling action cells

You can disable action cells so that the action is not carried out for that particular row.

Applying cell format

You can implement special formatting on cells or columns in the Rule Designer developer environment.

Defining preconditions

You can define preconditions in your decision tables.

Consistency checks in decision tables

As you submit values into decision table cells, Decision Center automatically analyzes the rules for consistency.

Parent topic: [Working with the editors](#)

Decision table editor

You edit a decision table by clicking on the row, header, or cell you want to edit, which displays the editor with the appropriate content.


Condition editor

✕ ✓ the amount of the loan [±] is at least ▼ 300000 [±] and less than ▼ 600000 [±]

📘 Use this editor to edit the selected cell.

	Grade	Amount of loan		Insurance required
		Min	Max	
1	A	< 100,000		false
2		100,000	300,000	true
3		300,000	600,000	true
4		≥ 600,000		true
5	B	< 100,000		false
6		100,000	300,000	true
7		300,000	600,000	true
8		≥ 600,000		true
9	C	< 100,000		true
10		100,000	300,000	true
11		300000	600000	true
12		≥ 600,000		true

The editor displays different buttons in its toolbar and different entries to modify, depending on whether you select a cell in a condition column, an action column, a row, or column header.

If you have many values to modify, you can edit the cells in “edit all” mode. To activate edit all mode, click  in the editor.

To commit all values that you modify, whether in the editor or in the table in edit all mode, click the **SUBMIT** button.

You can also use in-place editing, as described in [Editing cell values](#).

Note:

When a decision table opens, it shows only the first rows (1 - 16). Click **16 - 31** at the top of the table to display rows sixteen to thirty one, and so on. Click **All** to add a scroll bar so that you can scroll through all the rows.

Locks

A lock on a column indicates restrictions on what can be done in this column. For example, a lock might be set to prevent editing values or adding rows. You set or release locks through the options panel for each column, or through table properties for global locks.

Parent topic: [Editing decision tables](#)

Adding, removing, and editing columns

You can modify the column structure of your decision tables to define the conditions and actions of all the rules in the table.





About this task

You can have any number of columns a decision table. However, tables with too many columns are difficult to read. As best practice, use several smaller tables instead of one very large table.

You start all changes by clicking on a column header to display the column information in the editor. You can then make various changes in the editor, such as modifying rule statements, adding and removing columns, and changing column headings. You commit changes globally by clicking **SUBMIT**.



Procedure

To modify a column:

1. To modify the rule statement corresponding to the column:
 - a. Click the column header.
 - b. Click the rule statement in the Condition Column editor and modify the entries.
 - c. Click the **SUBMIT** button  to validate the change.
2. To change the header title:
 - a. Change the name that displays as the column title by typing in the title cell for that column.
For columns with subtitles, change the subtitles by typing in the subtitle cell.
 - b. Click anywhere outside the cell to validate the change.
3. To add a new column:
 - a. Decide where you want to insert the new column and then click the header in the adjoining column.
 - b. In the toolbar, click the required icon:
 - Click  icon to add a column after the current one.
 - Click  to add a column before the current one.
 - c. Click the header of the new column and use the editor to build the statement.
4. To delete a column:
 - a. Click the header of the column you want to delete.
 - b. In the toolbar, click .



The column is immediately deleted.
5. To sort columns:

The editor sorts the entire column or within groups of related cells.

- a. Click the header of the column you want to sort.
 - b. In the toolbar, click the required sort icon:
 - Click  to sort the cells alphabetically in descending order.
 - Click  to sort the cells alphabetically in ascending order.
6. To move an action column:
 - a. Click the header of the column to move.

Note:

You can move only action columns.

- b. In the toolbar, click the move icons  and  to move the action column sideways.
7. To set consistency checking options:
 - a. Click the header of the column.

Note:

You set consistency checking at individual column level.


- b. Click **Show options** > > to display the option panel.

- c. Select the check boxes corresponding to the consistency checking you want: overlap, symmetry, and gaps.

For information about the checks you can set up, refer to [Consistency checks in decision tables](#).

- d. Click **SUBMIT** to validate.

8. To resize columns:

- a. In the editor toolbar, click  to access the table properties.
- b. Clear the **Auto resize table** check box and then click **SUBMIT**.
- c. Click the column headings.
- d. Click **Show options** > > to display the option panel.
- e. Enter the size in the **Width** field and then click **SUBMIT**.

Parent topic: [Editing decision tables](#)

Related concepts:

[Columns](#)

[Rows and cells](#)

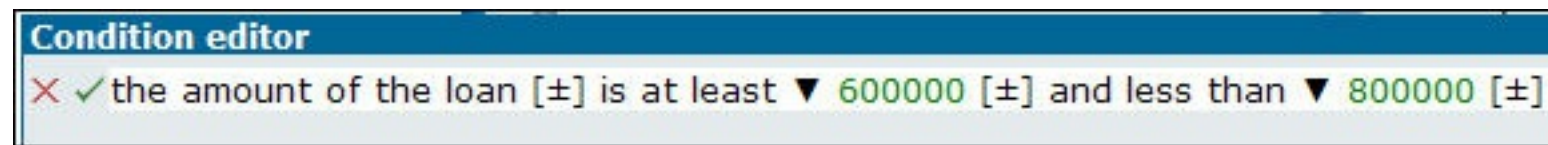
[Preconditions](#)

Adding, removing, and editing rows

You can add rows to and remove rows from a decision table, and edit the contents of a row using the decision table editor.

About this task

To edit a row, click in the decision table on the appropriate cell in the row. The editor opens, showing the information for the cell you clicked on, and the toolbar displays the tasks you can carry out:



Note:

You cannot edit rows if the column is locked. You set or release locks through the options panel for each column, or through table properties for global locks.

Click any row number to display the rule in the editor in its usual if-then form.

Procedure





The editor toolbar provides various options for adding, deleting and editing rows in decision tables.

1. To add a row:

- a. Click the appropriate cell in the row before or after the position in which you want to add the new row.

Clicking the correct cell is particularly important when you add a row from a grouped cell.

- b. In the editor toolbar, select how to add the row:


-  - Inserts the row above the cell you select.
-  - Inserts the row under the cell you select.
-  - Inserts an Otherwise row under the cell you select.
-  - Inserts as many rows as there are remaining possible values for that column. This button is available only when the column contains grouped cells.

- c. Complete the cells of the new row.

2. To delete a row:



- a. Click the appropriate cell in the row you want to delete.

Clicking the correct cell is particularly important when you delete a row from a grouped (partitioned) cell.

- b. In the editor toolbar, click .

3. To move a row up or down:


- a. Click a cell in the row you want to move. The chosen cell must be in a condition column.

- b. In the editor toolbar, click  to move the row up or  to move the row down.

If you click a grouped cell, the row can be moved only within its partition.

4. To split a row:



- a. Click a cell in the row you want to split. The chosen cell must be in a condition column that has subcolumns.

- b. In the editor toolbar, click  to split the row.

- c. Complete the empty values.

5. To merge rows:

- a. Click a cell in the row you want to merge. The chosen cell must be in a condition column, and the row itself must be completed.

- b. In the editor toolbar, click  to merge the row with the one above it, or  to merge the row with the one under it.

When you merge rows, the values in the condition part merge, but the merged row retains the values of the row that was above for the action parts of the row.

Parent topic: [Editing decision tables](#)

Related concepts:

[Columns](#)

[Rows and cells](#)


[Preconditions](#)

Editing cell values


You can edit decision table cells directly in the table, or using the decision table editor.

About this task

You can edit cells in a number of different ways:

- Clicking the cell, changing the value in the editor (located above the decision table), and then clicking **SUBMIT**.
- Clicking the edit all table icon  in the editor, changing values in the table, and then clicking **SUBMIT**.
- Changing the value directly in the table. If you enter an erroneous value, the error displays in the Edit bar as you type.

Note:
You can lock a column to prevent users from editing or disabling it.

To delete the contents of a cell, click  in the toolbar.

Procedure

To edit a cell:

1. Enter the value in the cell:

You can carry out the following types of edit:

- If the cell contains a text or number field, type in the new value in the field displayed.
- Select predefined values from a drop-down list.
- Change the operator that acts on a given cell.

You can use any of the following formats for values in a decision table:

- Value
- Numeric value with the following operators: =, !=, >, >=, <, <=
- Set of numeric or non-numeric values: {10, 15, 18} {New York, Rhode Island, Massachusetts}

Note:
You cannot select predefined values from a set directly in a cell. To select values, click the cell, and then use the editor (located above the decision table) to select the required values.

- Range of numeric or non-numeric values: [12..15] [A..F]

2. To commit the modified value, click **Next** or **Finish** in the Compose wizard. Click **Cancel** if you want to cancel all your current edits.

Parent topic: [Editing decision tables](#)

Related tasks:
[Creating business rules](#)

Related information:
[Decision tables](#)

Disabling action cells


You can disable action cells so that the action is not carried out for that particular row.

About this task

This is equivalent to clearing the contents of the cell, except that you keep the information the row contains in case you need to re-enable it later.

Procedure

To disable an action cell:

1. Click in the action cell you want to disable.
2. In the editor toolbar, click  to disable the action cell.

This is a toggle key, so if you click it again you re-enable the action cell.

Results

The action cell is disabled and the disabled icon displays in that cell.


Parent topic: [Editing decision tables](#)

Applying cell format

You can implement special formatting on cells or columns in the Rule Designer developer environment.

Procedure

To display existing cell or column formats:

1. In the editor toolbar, click  to access the table properties:
2. In the editor, check **Apply cell formatting**.
3. Click **SUBMIT** to validate.

Parent topic: [Editing decision tables](#)

Defining preconditions

You can define preconditions in your decision tables.


About this task

Decision tables contain preconditions that you can use to set variables and conditions:

- You can set variables that you use in the decision table.
- You can set a condition that applies to the entire decision table. If the precondition is not satisfied, none of the rules in the decision table are evaluated.

Procedure

To define preconditions:

1. In the editor toolbar, click  to access the table properties:
2. In the editor, build your preconditions in the same way that you build them in the guided editor.
3. Click **SUBMIT** to validate.

Parent topic: [Editing decision tables](#)

Related concepts:


[Columns](#)

[Rows and cells](#)

[Preconditions](#)

Consistency checks in decision tables

As you submit values into decision table cells, Decision Center automatically analyzes the rules for consistency.

The affected cells display a squiggly red line. In addition, the column header displays the error icon  and a message appears in the option panel of the editor.

The following figure shows errors on some cells in a column:

300,000	600,000	true	0.008
600,000	800,000	true	0.012
≥ 600,000		true	0.014

You can specify which types of analysis to perform on a column by clicking the column header and selecting the appropriate check boxes displayed in the editor. You can check columns for overlap, gaps, or symmetry, as described in [Rule verification](#). You set consistency checking capabilities on a per-column basis, and only for condition columns.

A decision table that has a consistency error does not execute unless you disable consistency checking.

Parent topic: [Editing decision tables](#)

Related tasks:
[Creating business rules](#)

Related information:
[Decision tables](#)

(Deprecated) Editing decision trees

You can use the decision tree editor when the decision tree and its properties have been created at the project level, that is, in the Tree part of the Compose wizard.

Decision tree editor

You edit a decision tree by clicking on the part of the decision tree you want to edit.

Declaring condition nodes

You must declare the initial condition of all your rules before you can add other conditions to a branch.

Working with branches

You set up branches by setting the value of the condition for the branch, adding other conditions nodes if required, and setting actions.

Defining preconditions

You can define preconditions that apply to all the rules of the decision tree.

Decision tree display

Decision trees are designed to make it easier for you to view and manage large sets of business rules.

Consistency checking in decision trees

You can set consistency checking on a decision tree or its individual conditions.

Parent topic: [Working with the editors](#)

Decision tree editor

You edit a decision tree by clicking on the part of the decision tree you want to edit.

Clicking the tree highlights the part in blue and the editor displays the appropriate content.

The editor displays different buttons in its toolbar and different entries to modify, depending on whether you select a condition node, a branch, or the actions.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:
[\(Deprecated\) Decision tree overview](#)

Related tasks:
[Creating business rules](#)


Declaring condition nodes

You must declare the initial condition of all your rules before you can add other conditions to a branch.

About this task

Initially, a decision tree contains an empty condition node that you must declare so that it becomes the first condition of the rules in your tree.

When you have a condition node declared you can add branches to the node, and add and delete other condition nodes:

Action	Description
Add branches to the node	You can add different types of branches, as described in Working with branches .
Add a condition node before an existing node	<p>You do this by selecting the node and clicking . You can access this icon on the toolbar or directly in the decision tree.</p> <p>When you add a condition node from an existing condition, you insert the new node before the existing one. The condition thus applies to all the branches of the existing node. To add a condition node after a node, so that it applies only to that branch, add it from the branch itself.</p>
Delete a condition node	<p>To delete a condition node, delete all the branches stemming from that node. When you delete the last branch, you also remove the condition node.</p>

Procedure

To declare a condition node:

1. Click the condition node (anywhere in the yellow diamond).
2. In the editor, use the drop-down lists to declare the condition.

Only the value of the condition remains empty, as this is done in the different branches.
3. In the **Label** field, enter the name you want to be displayed in the tree for the node.

If you leave this field blank, the label adopts the entire name of the condition.
4. Set any consistency checking you want to have carried out on the branches of that node.

These checks can be different from the checks on the rest of the tree.
5. Click **SUBMIT** to commit your changes.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:
[\(Deprecated\) Decision tree overview](#)

Related tasks:
[Creating business rules](#)

Related information:
[Consistency checking in decision trees](#)



Working with branches

You set up branches by setting the value of the condition for the branch, adding other conditions nodes if required, and setting actions.

When you create a condition node (see [Declaring condition nodes](#)), it has an empty branch stemming from it. You must set up this branch and any other branches that you add. For each branch, you must set the value of the condition for the branch and set actions. If required, you can also add other condition nodes.





Set up a branch

You set up a branch as follows:


Action	Procedure
Set the value of the condition for the branch	<div><div>1. Select the branch by clicking on the arrow or the arrowhead.</div><div>2. In the editor, in the Label field, enter the name you want to appear on the arrow in the tree.</div><div>3. Set the value of the condition for that branch and then click SUBMIT to commit your changes.</div></div>
Set the actions of the branch	<div><div>1. Click the actions located at the end of the branch.</div><div>2. In the editor, in the Label field, enter the name you want to appear for the set of actions in the tree.</div><div>3. Create the actions and then click SUBMIT to commit your changes.</div></div> <div>When you have set the first action, you can add other actions to the branch by clicking . You can access this icon from the toolbar or the tree.</div> <div>You can remove individual actions from the set of actions by selecting the action in the editor and clicking the X next to it.</div>
Add other condition nodes you want to include in the branch	To add a condition node to the branch, select the branch (either the arrow or the actions) and click  . You can access this icon from the toolbar or the tree.

Add branches

You must set up the branches for every branch that you add to your tree. You can add branches in various ways:

Branch	Description
Single branch	Select the condition node and then click  . You can access this icon from the toolbar or the tree.
Otherwise branch	<div>Select the condition node and then click  in the toolbar. The new branch displays the Otherwise label.</div> <div>An ‘Otherwise’ branch caters for the case where none of the other possibilities for that condition are correct. You can remove the ‘Otherwise’ status by clicking the same icon that you used to set it.</div>
Multiple branches	<div>Select the condition node and then click  in the toolbar. The new branches display the appropriate labels and empty actions are added to the end of each branch.</div> <div>You can create multiple branches only if the selected condition node allows it, that is, if a predefined list of items already exists. For example, the loan grade is ‘A’, ‘B’, ‘C’, or ‘D’.</div>
Duplicate a branch	<div>You can add a branch by duplicating an existing one and then modifying the new one. To duplicate an existing branch, select the branch and then click  in the toolbar.</div> <div>If the overlap check is active for the condition node, you get an error for both branches until you modify one of them.</div>

Delete branches

You delete a branch by selecting the branch (the arrow or the actions) and then clicking  in the toolbar.

When you delete a branch, it immediately deletes any condition nodes and any subbranches that stem from them.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:
[\(Deprecated\) Decision tree overview](#)

Related tasks:
[Creating business rules](#)

Defining preconditions

You can define preconditions that apply to all the rules of the decision tree.


About this task

Decision trees can contain preconditions for you to perform the following actions:

- Set variables that can be used in the decision tree. For information about setting variables, see [Rule variables](#).
- Set a condition that applies to an entire decision tree. If the precondition is not satisfied, none of the rules in the decision tree can be evaluated.

Procedure

To define preconditions:

1. Click the **Edit Properties** button  in the toolbar.
2. In the editor, build your preconditions as you would in the guided editor.
3. Click **SUBMIT** to validate.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:
[\(Deprecated\) Decision tree overview](#)





Related tasks:
[Creating business rules](#)

Related information:
[Using the rule editors](#)



Decision tree display

Decision trees are designed to make it easier for you to view and manage large sets of business rules.

Decision Center provides a number of display options to make it easier for you to view your decision trees:

- To display a decision tree horizontally, click the  icon on the toolbar.
- To display it vertically, click .
- You can also make viewing easier by minimizing the branches or actions: click the  symbol on a condition node. To restore the branch to normal, click the .

Similarly, you can minimize or restore actions by clicking  or  next to the rule heading.

- To change the position of a branch with respect to the other branches, click  or  to move the branch in the direction indicated.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:

[\(Deprecated\) Decision tree overview](#)

Related tasks:

[Creating business rules](#)

Consistency checking in decision trees

You can set consistency checking on a decision tree or its individual conditions.

You can analyze the rules that make up your decision tree to make sure that there are no gaps and overlaps. You can do this at the tree level and for the branches of a given condition node.


Note:

You cannot check dates for gaps and overlaps.

If a decision tree has inconsistencies, for example, two branches that use the same value for a condition, Decision Center displays the 🚫 symbol to identify the affected condition node and its incorrect branches. To display the corresponding error message in the editor, click the condition node or branch.

You cannot execute a decision tree that has an inconsistency error unless you disable consistency checking.

Tree and node verifications

You control consistency checking for the decision tree in the editing part, which displays when you click the Edit Properties button  in the toolbar. You select the condition node and set the checks. Decision Center then checks for gaps and overlaps on the selected condition nodes.

If you do not select consistency checking at tree level, no analysis is performed on any condition node, regardless of any checks you set for the individual condition nodes.

Parent topic: [\(Deprecated\) Editing decision trees](#)

Related concepts:

[\(Deprecated\) Decision tree overview](#)

Related tasks:

[Creating business rules](#)

Previewing ruleflows

You cannot edit ruleflows in the Enterprise console.

To create and edit a ruleflow in Decision Center, you must use the Business console. In the Enterprise console, you can only view the ruleflow and its properties, by accessing the **Details** page of your ruleflow. The **Detail** page contains the following details:

- Ruleflow diagram: You can use the toolbar to zoom in and out of the diagram.
- Ruleflow properties
- Ruleflow task information, which is displayed at the bottom of the **Details** page.

Parent topic: [Working with the editors](#)

Related concepts:

[Overview: Ruleflows](#)

[Project element properties](#)

Related information:

[Editing ruleflows](#)

Configure: Manage your project configuration

You use the Configure tab to carry out some administrative tasks.

The Permission manager can access the **Configure** tab to run diagnostics, clean the cache, export and erase current projects.

Running diagnostics

In case of unexpected errors, the error message can include a link to the **Diagnostics** page. You can also run the diagnostics manually.

Cleaning the cache

You can clean out the files cached on the server.

Exporting the current project

You can export the working branch of the current project to a .zip file.

Erasing the current project

You can erase the current project from Decision Center.

Parent topic: [Working with the Enterprise console](#)

Running diagnostics

In case of unexpected errors, the error message can include a link to the **Diagnostics** page. You can also run the diagnostics manually.

About this task

The diagnostics feature checks the configuration of Decision Center for Manager Bean access and connection to the data source. It shows your rule model extensions and verbalizers.

Procedure

To run diagnostics:

On the **Configure** tab, click **Diagnostics**.

Parent topic: [Configure: Manage your project configuration](#)

Cleaning the cache

You can clean out the files cached on the server.

About this task

IRL files generated when you create rulesets are cached on the server in the location specified in `teamserver.war/WEB-INF/lib/teamserver-model-XXX.jar/ilog/rules/teamserver/preferences.properties`.

Over time these files can become obsolete, or you might want to modify the way the IRL is generated. In these cases, you can clean out the cache.

Procedure

To clean the cache:

On the **Configure** tab, click **Clean Decision Center Cache**.

Results

This action deletes the files cached on the server for the current project.

Parent topic: [Configure: Manage your project configuration](#)

Exporting the current project

You can export the working branch of the current project to a .zip file.

Before you begin

You can then import these files into your Eclipse workspace or import them into another Decision Center.

Procedure

To export the current project:

1. On the **Configure** tab, click **Export Current Project State**.
2. Enter the location to save the file and click **OK**.

Parent topic: [Configure: Manage your project configuration](#)

Erasing the current project

You can erase the current project from Decision Center.

About this task

You can erase a rule project so that all its entries in the database are permanently removed and it no longer appears in Decision Center.

Procedure

To erase a project:

1. On the **Configure** tab, click **Erase Current Project**.
2. Click **Yes** to confirm.

Results

The project is deleted and you are returned to the **Home** page with the next available project displayed.

Parent topic: [Configure: Manage your project configuration](#)

Storing and synchronizing rules

You can store and maintain rules in two different repositories, a Source Control System (SCC) and a database. Therefore, you must synchronize the two rules repositories periodically to bring them at the same level.

Why synchronize?

As rules can be stored and maintained in two repositories in parallel, you must synchronize these repositories from time to time to keep them consistent.

Sharing repositories between user profiles

To avoid conflicts when rules are shared between business users and developers, you must clearly establish which is the master repository: source code control (SCC) or the Decision Center database.

Synchronization architecture

The synchronization process compares your local Rule Designer workspace, the remote Decision Center database, and a reference that computes the state of the synchronization.

Branches and releases in synchronization

Synchronization between Rule Designer and Decision Center is done one branch at a time.

Guidelines for synchronizing BOM changes

You can avoid synchronization errors by following these guidelines for synchronizing BOM changes in rule projects.

Synchronizing from Rule Designer

With the synchronization commands in Rule Designer, you can create a project from an existing Decision Center project or publish an existing project to Decision Center.

Synchronization and source code control

Use source code control (SCC) to share Rule Designer artifacts, and commit rule project resources.

Rule project items

A rule project is a container for organizing rule artifacts and setting up the business object model (BOM) and rule authoring vocabulary. Each rule project item is associated with a folder.

Why synchronize?

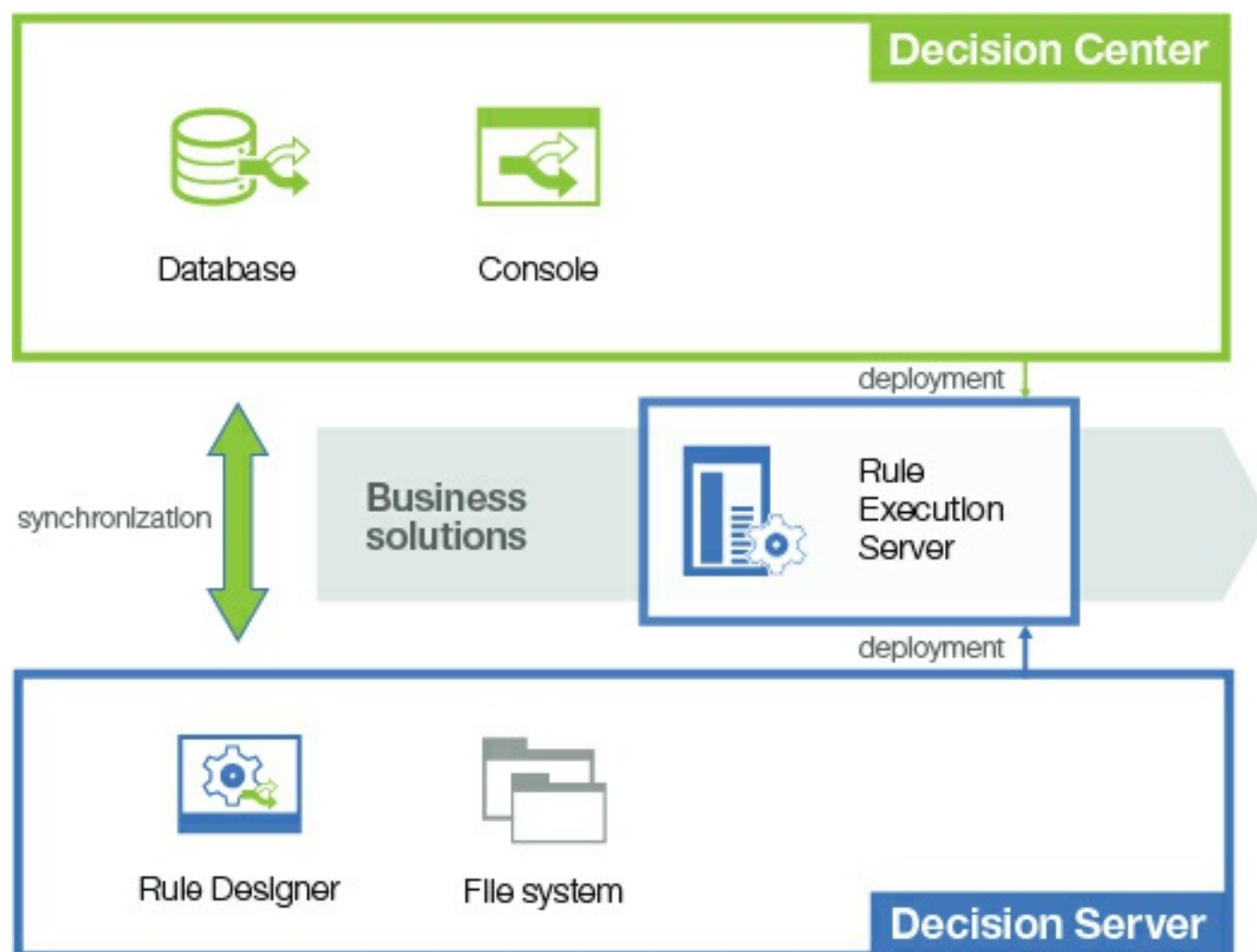
As rules can be stored and maintained in two repositories in parallel, you must synchronize these repositories from time to time to keep them consistent.

At the beginning of a decision rule development project, developers create rules using the Eclipse-based Rule Designer application, and store them in a **source control system** (SCS) to share files, handle versions, and resolve potential conflicts that can arise when several users are committing changes to the same repository .

At a later point in time, developers must push rule projects or decision services developed in Rule Designer to the Decision Center environment to make them available to business users (rule authors and policy managers). In Decision Center, rule projects or decision services are stored in a **database**. Decision Center handles concurrent accesses to the database and versioning.

When a rule project or decision service is both maintained in Rule Designer and Decision Center, it is necessary to synchronize them periodically to bring them at the same level. Synchronization is a process that can be done either manually or automatically with ant tasks, and is always initiated from Rule Designer.

The following illustration shows how rule projects or decision services are synchronized between Rule Designer and Decision Center.



Parent topic: [Storing and synchronizing rules](#)

Related information:

[Sharing repositories between user profiles](#)

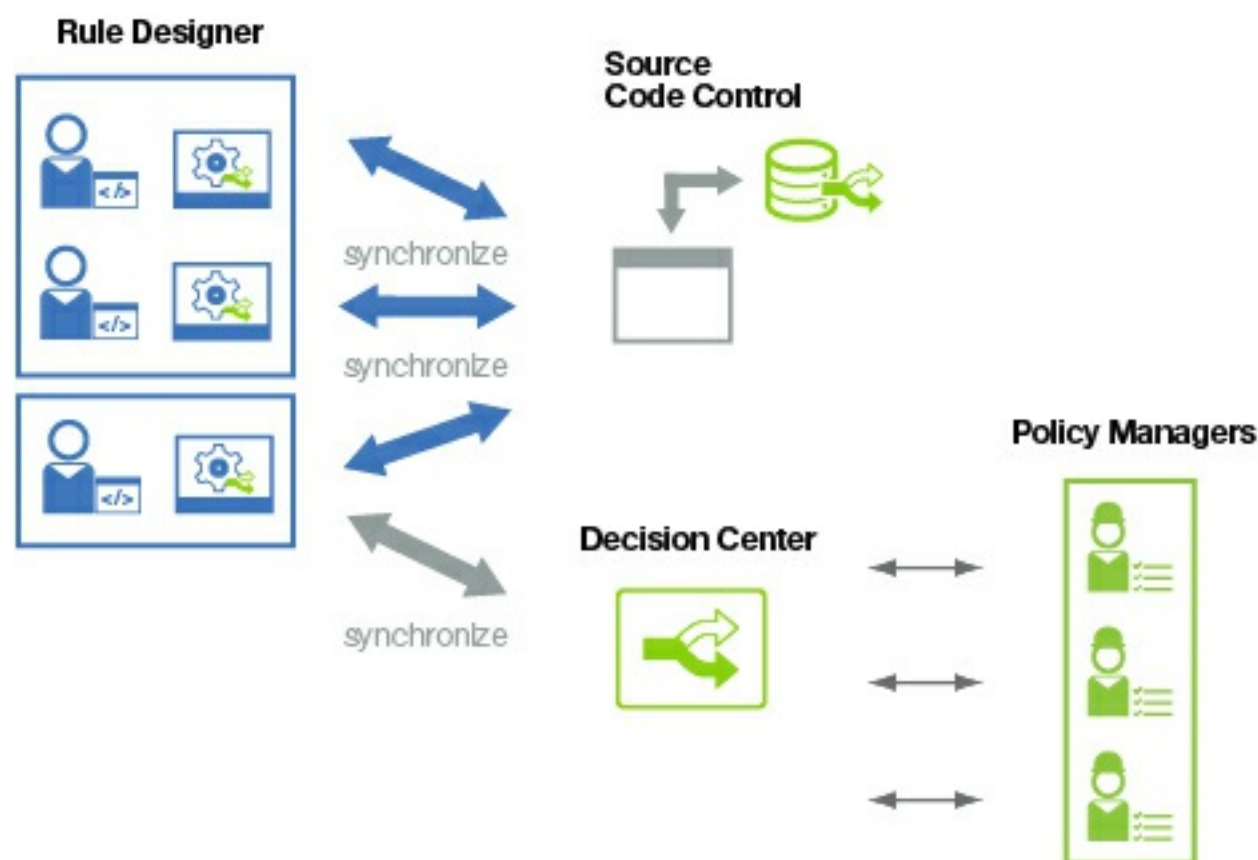
[Synchronizing by using Ant tasks](#)

[Synchronizing from Rule Designer](#)

Sharing repositories between user profiles

To avoid conflicts when rules are shared between business users and developers, you must clearly establish which is the master repository: source code control (SCC) or the Decision Center database.

The interaction between the two user profiles is shown in the following figure. However, the choice of master repository depends on whether you are in the initial or later stages of project development:

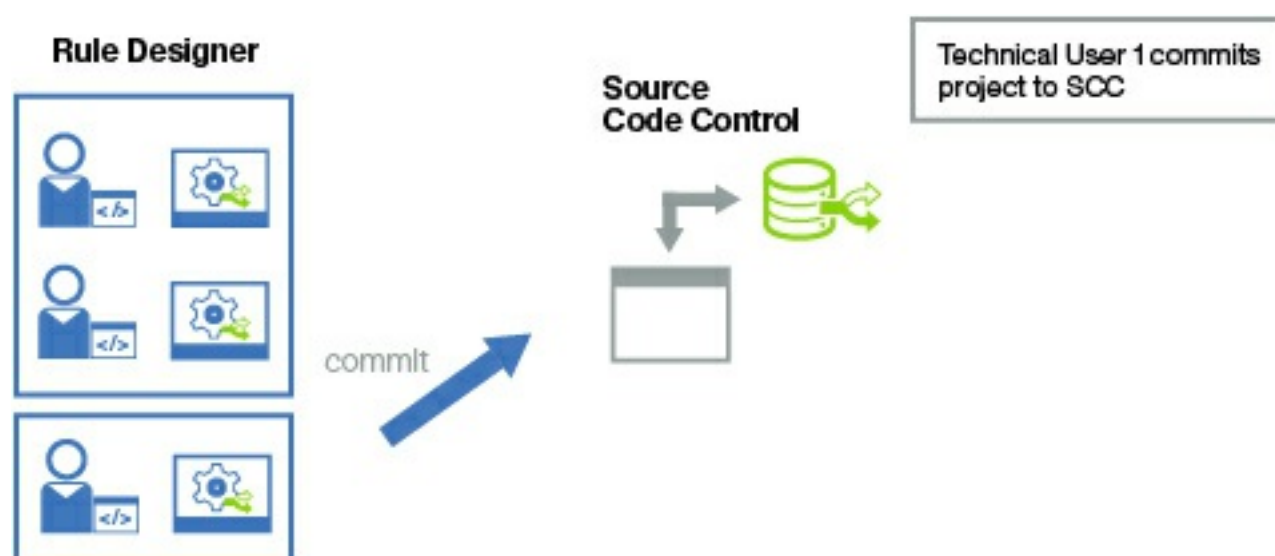


- A **developer-centric approach** is better adapted for an initial development situation. Technical users synchronize their rules through SCC. One dedicated technical user publishes to Decision Center for testing, and synchronizes any changes between the two repositories (see [Rule project items](#)).
- A **business user-centric approach** is better adapted for the later stages of project development, when the business object model stabilizes. You consider the Decision Center repository to be the source of truth and any Rule Designer developer copy in SCC as a temporary copy of the project.

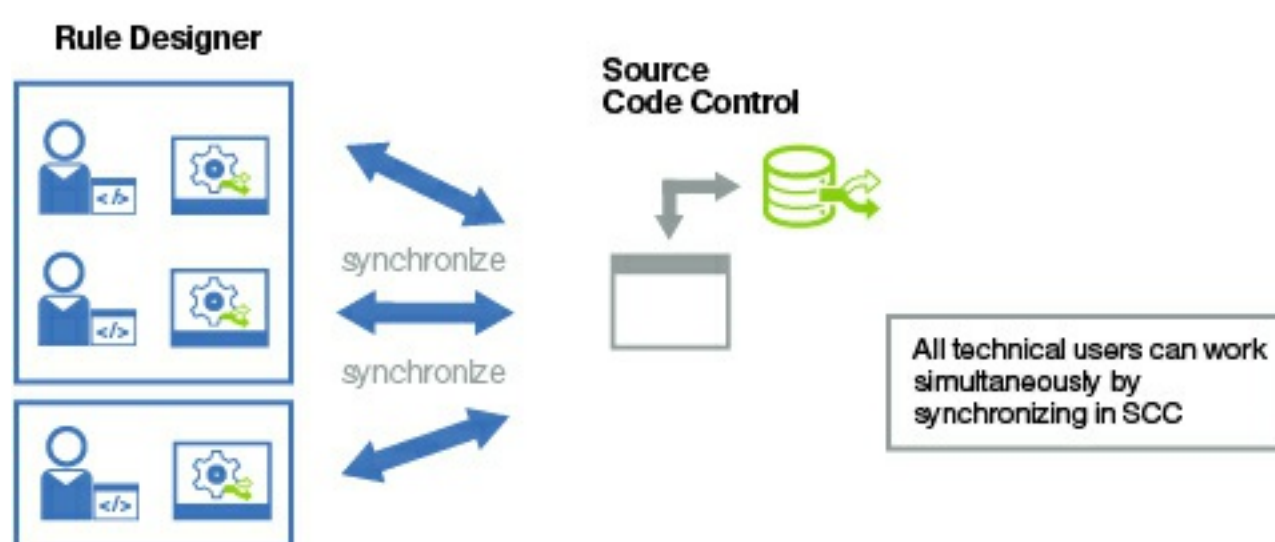
Project lifecycle and the choice of master repository

In the early phases of a project, developers use Rule Designer to develop a business object model and other project artifacts. During this phase, you can maintain the project entirely in Rule Designer by using SCC to control multiuser updates as follows:

- A technical user (Technical User 1) creates the project in Rule Designer and then commits it to SCC.



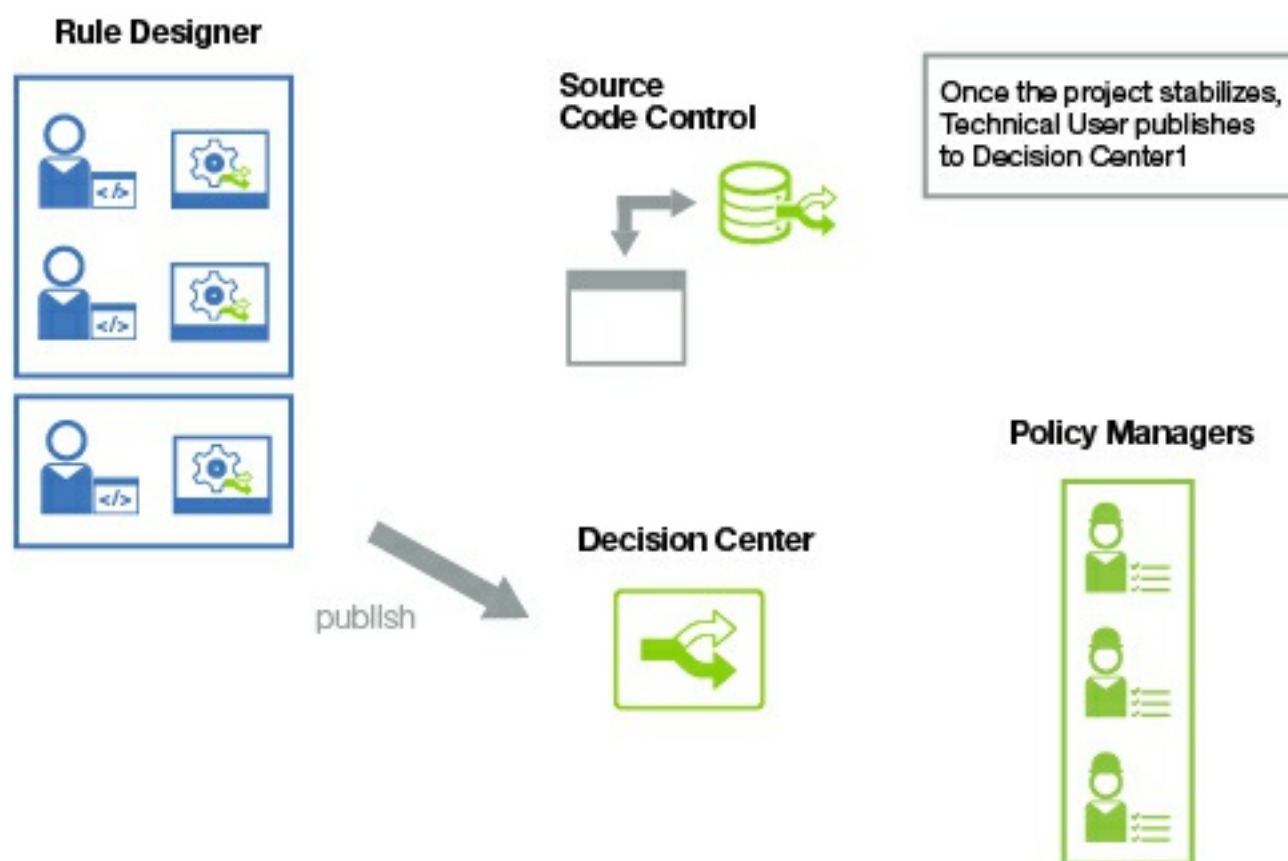
- From that point, other developers can retrieve the project from SCC and start to work. All technical users work simultaneously by synchronizing through SCC.



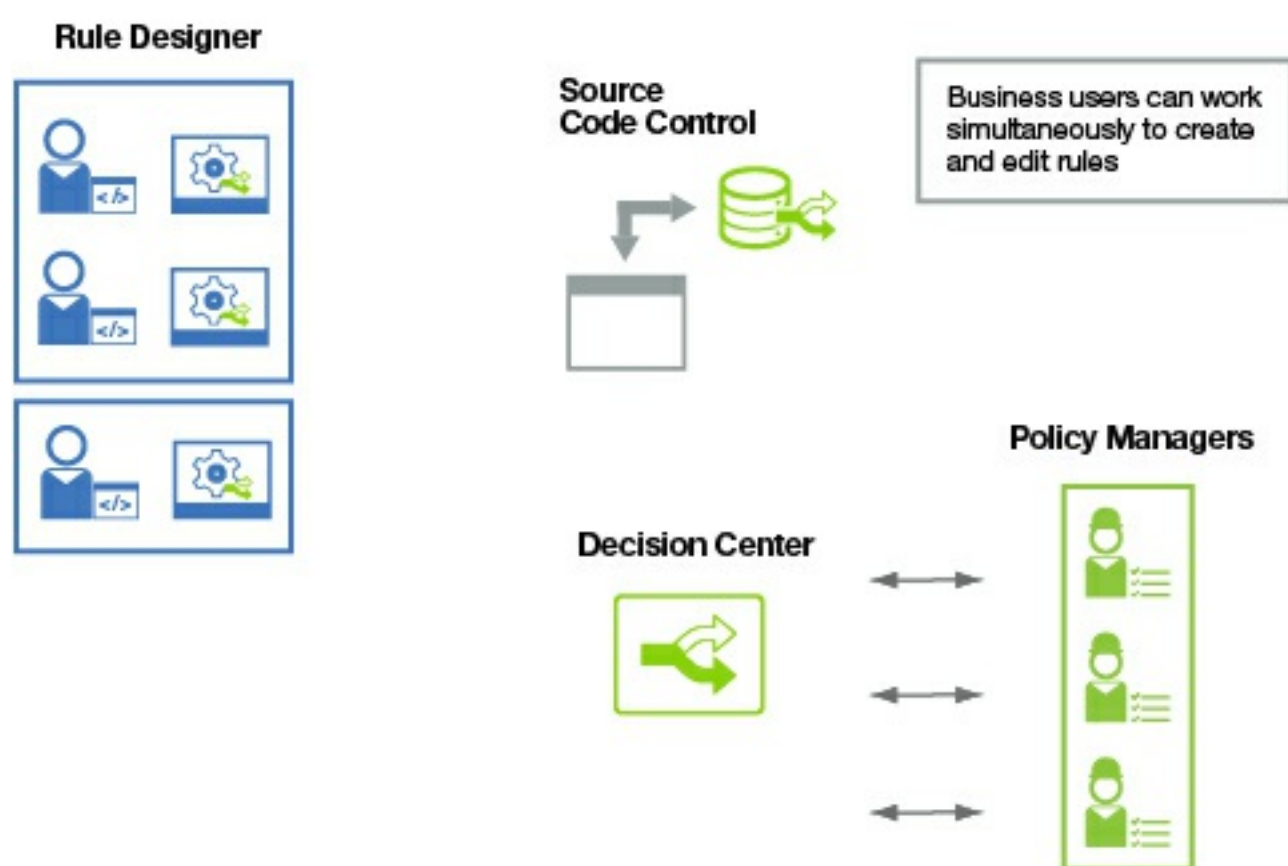
As the business object model begins to stabilize, business users start authoring rules.

To support the business user, Technical User 1 publishes the project to Decision Center, which becomes the copy of record for rules:

- Technical User 1 publishes to Decision Center.



- Business users can then work with the project in Decision Center to create and edit rules.



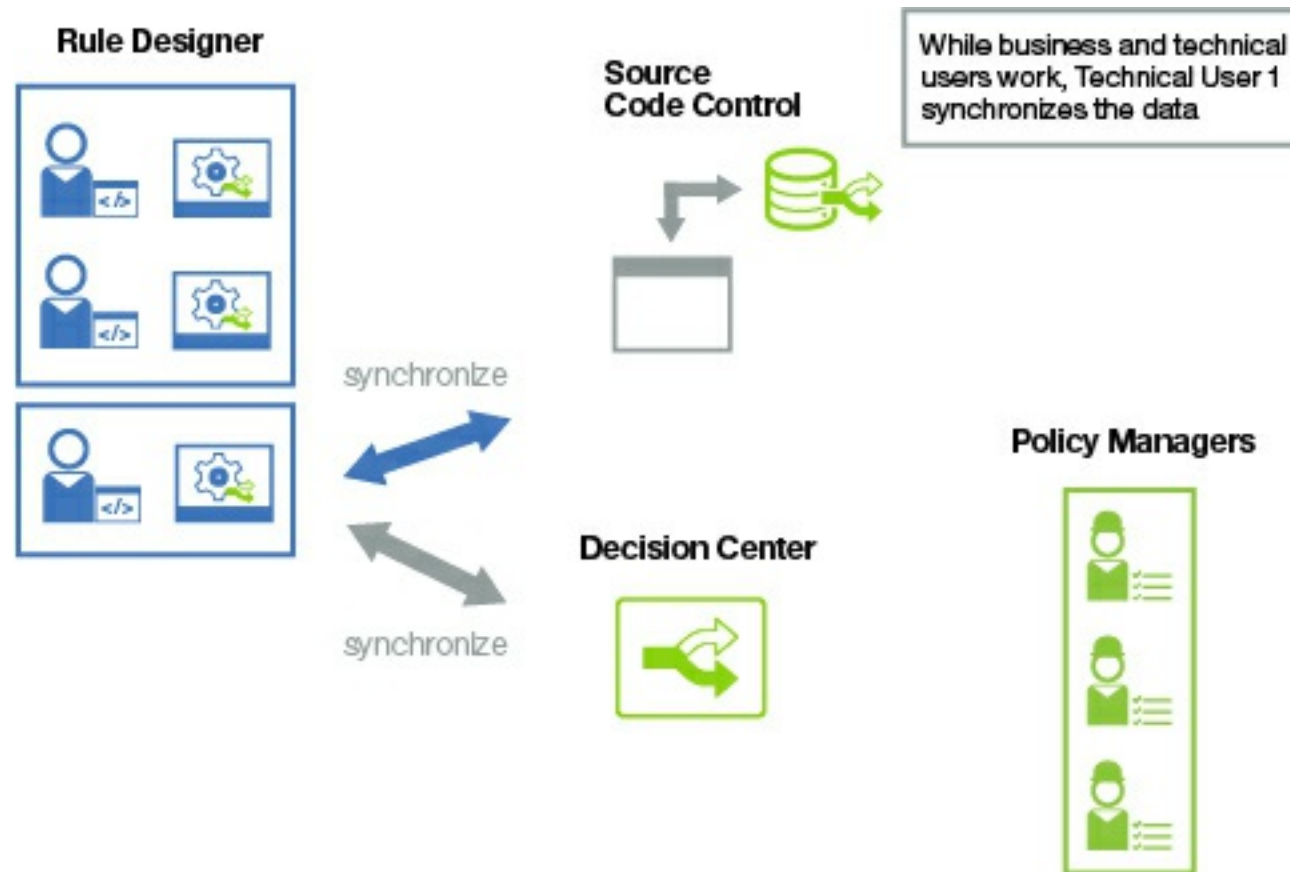
Control of the business object model now remains with the rule project in Rule Designer and developers must maintain it. To keep the developer view of the project in Rule Designer synchronized with the work of the business users, developers periodically update the rule project from Decision Center. Changes to the business object model are likewise published to Decision Center so that business users can work with the latest vocabulary.

Note:

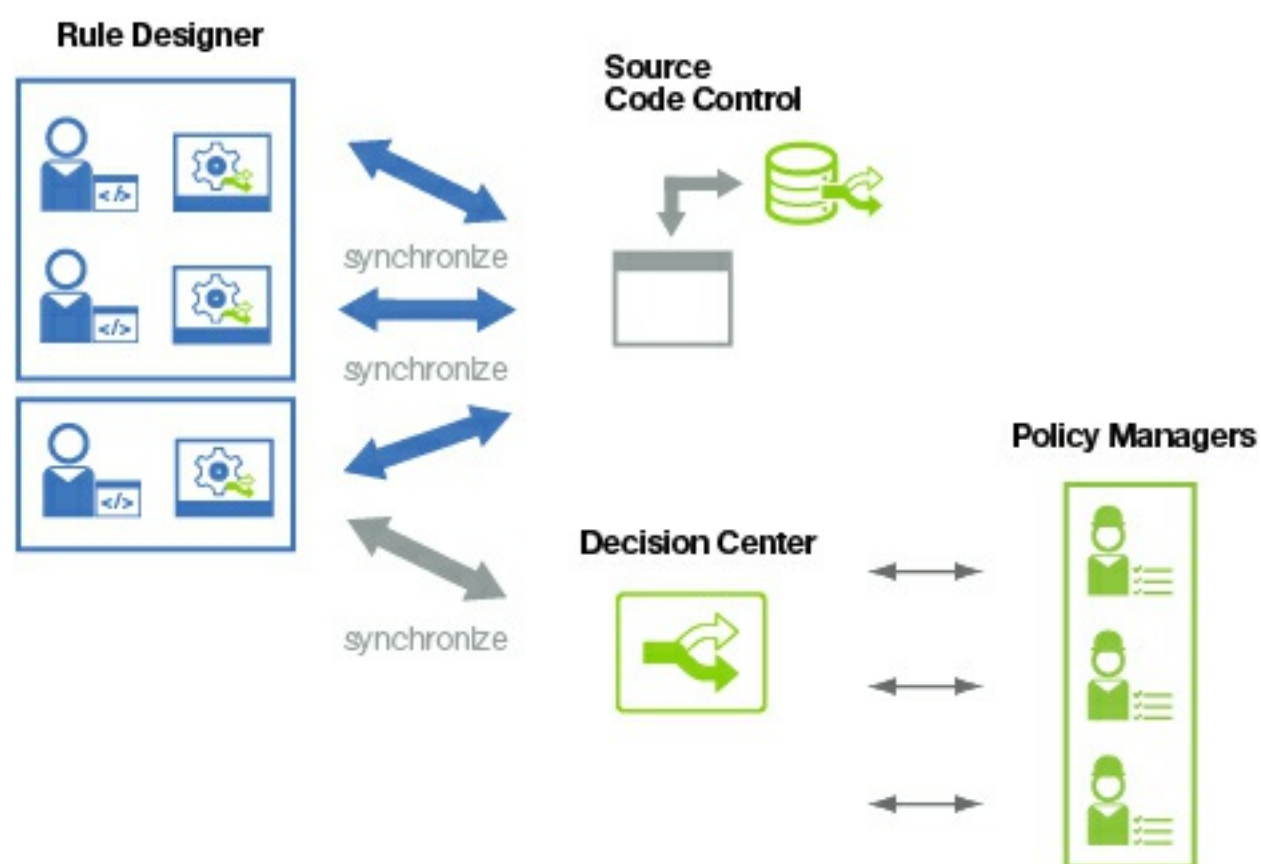
When BOM changes affect the vocabulary that is used in existing rules, you must bring these rules into Rule Designer for refactoring (see [Guidelines for synchronizing BOM changes](#)).

You maintain synchronization as follows:

- Technical User 1 periodically synchronizes the data for all the users.



- All the business and technical users work simultaneously, but Technical User 1 has the added role of synchronizing between the two user profiles.



When you send the project to production, the Rule Designer copy of the project is essentially in an archive state, and not required for day-to-day management of the rules. Afterward, business users continue to author and modify rules, and to deploy them to production as the business cycle dictates. The Decision Center repository remains the source for auditing all the policy changes that are implemented by rules.

When you want to develop the application further, you create a new branch in the development SCC system. You then populate it with the rule portion of the project by creating a new copy of the production rule project from Decision Center. From this point, development starts again.

Parent topic: [Storing and synchronizing rules](#)

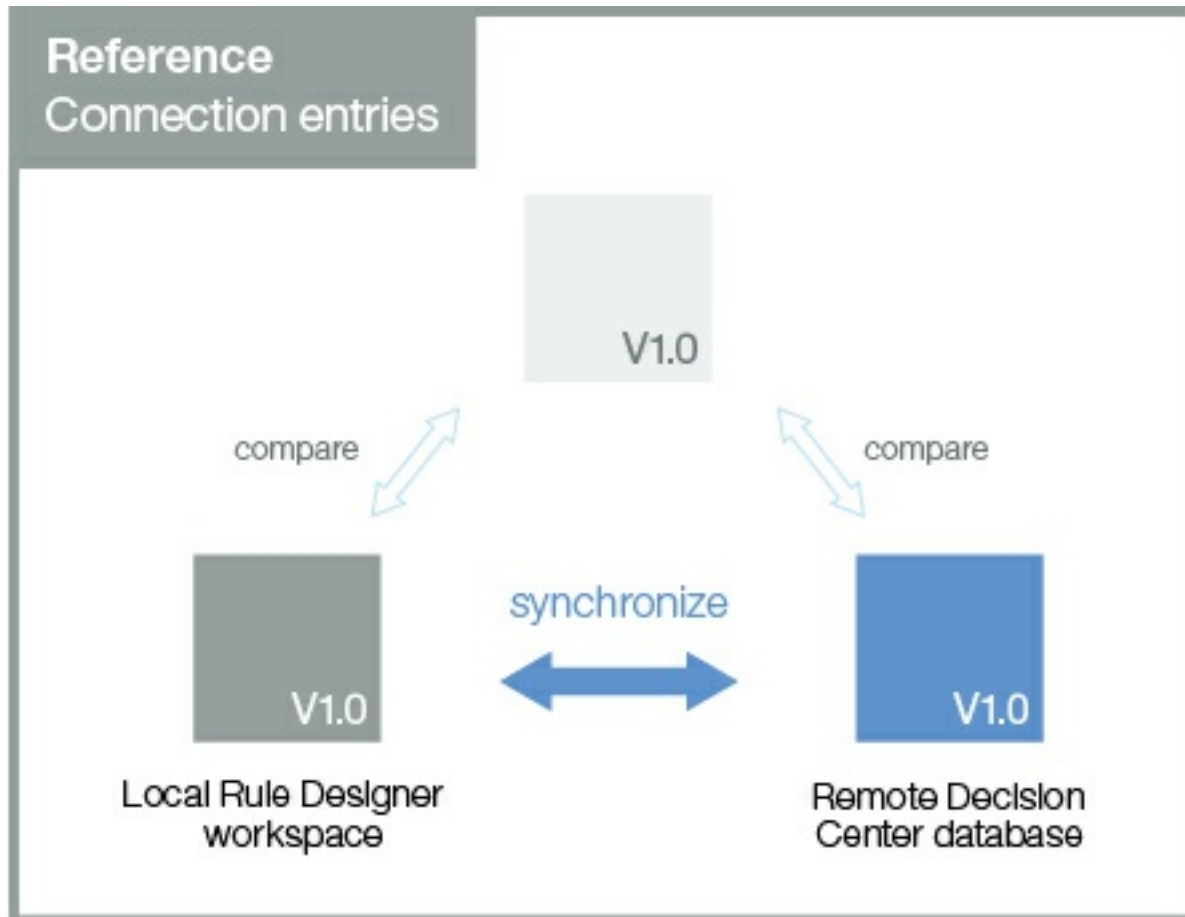
Related concepts:
[Branches and releases in synchronization](#)

Related information:
[Synchronization architecture](#)

Synchronization architecture

The synchronization process compares your local Rule Designer workspace, the remote Decision Center database, and a reference that computes the state of the synchronization.

This reference state is created as a connection entry in your workspace when you connect to Decision Center.



The Synchronization tool in Rule Designer queries the Decision Center database remotely. To improve performance, the three-way comparison uses a checksum on both the remote and the local rules, and then compares them to the reference state.

Three-way comparison indicates the direction of changes:

Incoming

Changes in Decision Center that you must update in Rule Designer.

Outgoing

Changes that are made locally in Rule Designer that you must publish to the remote Decision Center.

Conflict

Changes made to the local and remote versions.

If the Synchronization view advises any changes, you update or publish the changes as indicated, or override the proposed direction.

Note:

- If you rename a folder or subfolder in Rule Designer and synchronize, you obtain a new version in Decision Center for every artifact in that folder.
- The synchronization process is based on a user and a server. For example, if you synchronize between Rule Designer and Decision Center, then disconnect your project from Decision Center, and reconnect with another user credentials or connected to another server, then the synchronization can display conflicts even if no changes were made to your decision service or project.

Connection entries in synchronization

When you connect to Decision Center, a connection file (.syncEntries) is created in your workspace for the project.

The connection file tracks the state of the synchronization with one or more Decision Centers if you connect to more than one.

Important:

You must keep this entry to retain the three-way comparison that gives you the state of the synchronization.

When you close your Eclipse session, the connection file remains in your workspace, so you do not need to reconnect the next time you open Rule Designer.

When you disconnect from one Decision Center to connect to another one, you must keep the connection file of the first, unless you plan never to reconnect.

If you delete the connection entry and later reconnect and synchronize with the same project, the Synchronize view displays conflicts, even though the rules are the same on both sides, because you lost the checksum information when you deleted the entry. In this case, you can delete the project in Rule Designer and reimport it from Decision Center.

Only delete the connection entry if you do not need to reconnect, or if you want to clean the connection files, such as when you send a rule project (not the workspace) to another user.

UUIDs in synchronization

Synchronization uses Universal Unique Identifiers (UUIDs) to determine whether rules in Rule Designer and Decision Center are synchronized.

If you manually change the UUID of a rule in Decision Center, you get conflicts when you synchronize. For example, developers might change a UUID when they share rule projects through source code control (SCC) and want to work on different branches of a project in the same workspace. This approach breaks the synchronization. You must import different branches into separate workspaces to keep the synchronization.

When you copy a rule in Rule Designer as a starting point for another rule that you rename, copy it from the Rule Perspective to automatically change the UUID of the copied rule. Synchronization then detects the new rule on the Rule Designer side.

If you copy a rule in the Resource view or Windows Explorer, the copied rule has the same UUID as the original rule, and the copy process raises an error. To correct this error, right-click the copied rule in the Rule Explorer and click **Update UUIDs**.

Note:

If you want to check the UUID of a rule, go to the `.brl` view in Rule Designer. To show the UUID in the Decision Center rule tables, click **Options** in the top banner of the Enterprise console.

Parent topic: [Storing and synchronizing rules](#)

Related concepts:

[Synchronization commands in Rule Designer](#)

Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

Branches and releases in synchronization

Synchronization between Rule Designer and Decision Center is done one branch at a time.

In Rule Designer, you can have only one branch in your workspace at any given time. To synchronize with the different branches in use in Decision Center, connect, disconnect, and reconnect with each branch individually. When you disconnect, keep the connection entries for that branch. These entries are stored as part of the branch, allowing you to reconnect with that branch without introducing conflicts.

Releases and change activities

In Decision Center, releases and change activities are used within the decision governance framework, a ready-to-use approach to change management and governance. The first publishing of a decision service creates an initial, closed release in Decision Center. Subsequent changes to the decision service in Decision Center take place in releases that are created from this initial release. In the Business console, changes to a release can only be done in change activities. Releases and change activities are branches of the decision service.

Note: If you do not want to use the decision governance framework with your decision services, you can publish to a branch called main.

After the initial publishing you publish changes to open change activities, and not directly to the release branch or to a closed changed activity. Of course you can update changes done in any branch in Decision Center back to your Rule Designer workspace so that all versions are synchronized.

A decision service is designed for several rule projects to be grouped as one entity under a main project. The decision service is assigned the name of the main project. All other projects contained in the decision service contain a direct or indirect dependency to the main project. When you publish from the main project, all dependent projects are also published to Decision Center.

If you need to add a project to the decision service, consider this as a redesign of the decision service. Consequently, you can only add a project to an open release, and not to a change activity. To publish a new project to the decision service, add the project in Rule Designer and set its dependency to the main project. Then connect to the release and publish the project. After the publish is completed, synchronize the decision service and publish the changes relating to the new dependencies.

Administrator privileges

You can publish changes to a release branch if you connect with administrator rights. This privilege is offered for flexibility, but use it with restraint, because it circumvents the governance framework. Synchronizing to a release is the only way dependencies of a decision service can be changed.

Parent topic: [Storing and synchronizing rules](#)

Related concepts:

[Synchronization commands in Rule Designer](#)

Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

Related information:

[Change management](#)

Guidelines for synchronizing BOM changes

You can avoid synchronization errors by following these guidelines for synchronizing BOM changes in rule projects.

Changes to the BOM in Rule Designer can cause corresponding changes in rules through the Rule Designer refactoring mechanism. You must handle these changes cautiously to maintain synchronization with Decision Center.

For example, consider a project that is worked on and managed simultaneously in Rule Designer and Decision Center:

- The project has a BOM class Customer verbalized as the `customer` with an integer member `age` verbalized as the `age of the {this}`
- The project has the following rule named `Legal Age`:

```
if
    the age of the customer is less than 21
then
    ...
```

If a Rule Designer user changes the verbalization of the class `Customer` to the `client`, Rule Designer refactors the rule `Legal Age` to read:

```
if
    the age of the client is less than 21
then
    ...
```

Suppose that, in the meantime, a Decision Center user creates a second rule named `Senior`:

```
if
    the age of the customer is more than 65
then
    ...
```

If the Rule Designer user updates from Decision Center, the new rule is added to the Rule Designer project, but is in error because “the age of the customer” is no longer part of the vocabulary in the Rule Designer project.

Similarly, if the Rule Designer user publishes the changes in the verbalization, all the rules in Decision Center referring to “the customer” are in error.

These occurrences are a natural consequence of simultaneous development of dependent artifacts; the rules depend on the BOM vocabulary. You can avoid these errors by following the development practices that are described here for synchronous and asynchronous changes.

Simplest case: synchronous vocabulary change

The simplest approach to managing vocabulary change during simultaneous development is to synchronize changes to the BOM and vocabulary. This approach might not be practical for large projects, but works with smaller groups and small changes after the initial setup of the BOM.

In this case, you make all the changes to the BOM or vocabulary on a fully up-to-date copy of the rule project. Perform the following sequence of events:

1. Before you make the changes, all the developers commit their projects to SCC.
2. A single developer updates a copy of the project from SCC to make it current.
3. The developer updates from Decision Center to get a current copy of all the rule changes.
4. The developer makes the BOM and vocabulary changes, and refactors rules as required.
5. The developer commits the modified project to both SCC and Decision Center.
6. Simultaneous rule development in Rule Designer and Decision Center can then resume.

Managing asynchronous changes

For large teams, or when you must manage multiple versions of a project for extended periods, implement the following practices:

- Developers who need to make BOM changes must not delete BOM classes or members. Deprecate obsolete classes and members (use the deprecated property in the BOM editor) during ordinary development. Remove them only when a fully synchronized copy of the project is available and all the rules are refactored.
- Developers must avoid changing argument numbers and types in the BOM. If you need a new argument number or a type change, create a new method with the “corrected” arguments and deprecate the old member.
- You do not need to place restrictions when you add classes or members to the BOM. However, you must establish processes to make sure that such additions are made only one time to a single copy of the project, and then propagated by synchronization to avoid having to merge incompatible additions.
- As much as possible, defer vocabulary changes (other than additions) to occasions when you can use a fully synchronized copy of the project so that all the rules are correctly refactored.

Change the vocabulary:

- Deprecate the member for which the vocabulary is to be changed (the “original” member)
- Add a new virtual member on the same class, with the required verbalization (the “new” member)
- Use BOM-to-XOM mapping to map the new virtual member to the appropriate real method in the XOM
- To resolve duplicate verbalizations when you have a fully synchronized version of the project:
 - Remove the new member and save the BOM entry that contains this member. All the rules that use the new verbalization will be temporarily in error.
 - Change the verbalization on the old member to the required new verbalization, and save the BOM entry. Examine all aspects of the verbalization carefully to make sure that it exactly matches the new verbalization that you want. Rule Designer refactors the rules to use the new verbalization and rebuilds the project, clearing the temporary errors from the first step.

Parent topic: [Storing and synchronizing rules](#)

Related information:
[Synchronization architecture](#)

Synchronizing from Rule Designer

With the synchronization commands in Rule Designer, you can create a project from an existing Decision Center project or publish an existing project to Decision Center.

Note: For synchronization to work correctly, you must use Rule Designer and Decision Center in the same persistence locale.

[Synchronization commands in Rule Designer](#)

The Synchronize view in Rule Designer shows where changes in Rule Designer and Decision Center are not synchronized. You can publish, update, or override changes to achieve synchronization.

[Creating projects from Decision Center](#)

You can create a project or in Rule Designer from an existing Decision Center project.

[Use of Text Compare in synchronization](#)

Display conflicting rules and their differences in the Eclipse Text Compare Editor.

Parent topic: [Storing and synchronizing rules](#)

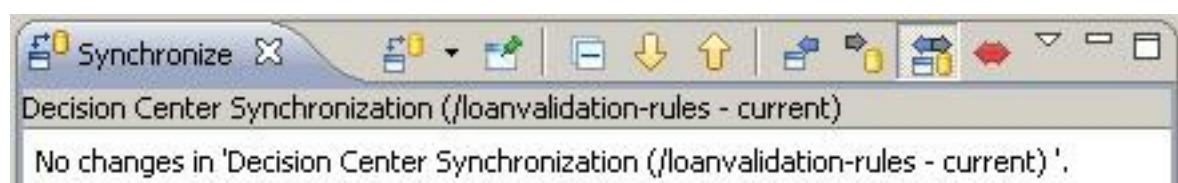
Synchronization commands in Rule Designer

The Synchronize view in Rule Designer shows where changes in Rule Designer and Decision Center are not synchronized. You can publish, update, or override changes to achieve synchronization.

You synchronize rule projects from Rule Designer. Synchronization starts when you publish a Rule Designer project or decision service to Decision Center, or create a decision service or project from Decision Center.

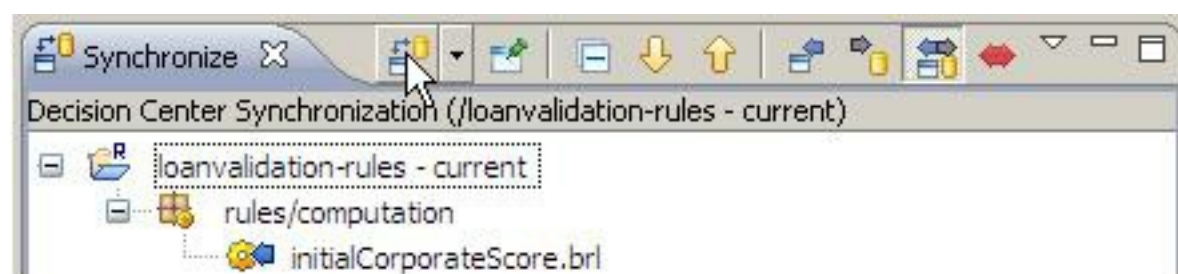
Projects and decision services are synchronized one branch at a time. You should be familiar with section [Branches and releases in synchronization](#) to avoid misuse.

The Synchronize view displays a rule project or decision service branch when you connect to Decision Center:



Note: When you upgrade to a new version of Operational Decision Manager, data structure discrepancies might occur when synchronizing for the first time. When a discrepancy occurs, override the older version.

The Synchronize view displays any modifications that are made to either the local Rule Designer or the remote Decision Center versions of the branch. Changes that are made in Rule Designer display automatically, and changes that are made in Decision Center display when you click the **Synchronize** button:



To open the Synchronize view:

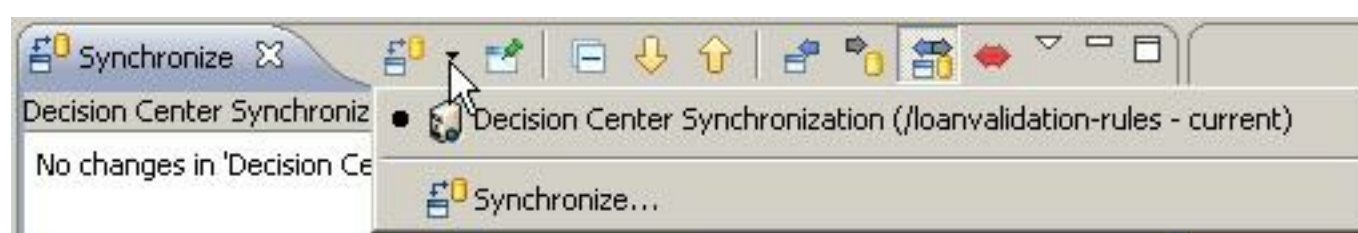
1. In Rule Designer, open **Window > Show View > Other > Team**.
2. Click **Synchronize**.
3. Click **OK** to open the view.

Selecting a project for synchronization

In the Synchronization view, you can change the project and branch to be synchronized with Decision Center.

To select a project for synchronization:

1. In the Synchronization view, click the down arrow next to the **Synchronize** button to open the synchronization list:



2. In the list, click **Synchronize**.

The Synchronize - Rule Model Synchronization window opens with a list of available types of synchronization.

3. Select **Synchronization**, and click **Next**.

The synchronization window opens with a list of the projects that are currently in the workspace.

4. Select the project branch that you want to synchronize with Decision Center, and click **Finish**.

The Synchronizing message appears, and the selected project and a list of any changes that were made with Decision Center are displayed in the Synchronization view.

Publish, Update, and Override


Entries in the Synchronize view can indicate that the local and remote versions of your project branch are no longer synchronized. You must publish or update the changes to synchronize the versions.

The Synchronize view includes the project branch and package of each changed artifact. You can publish one or more entries at a time, or all the changes that are detected in a package or project branch together.


Because project branches can be modified locally, remotely, or both concurrently, entries in the Synchronize

view show the direction of the change and an action:


Publish

Entries with a black arrow ( repayment.brl (1.0)) indicate that a change occurred in Rule Designer. Publish this change to Decision Center by right-clicking the entry in the Synchronize view and clicking **Publish**.

Update

Entries with a blue arrow ( repayment.brl (1.1)) indicate that a change occurred in Decision Center. Update this change back to Rule Designer by right-clicking the entry in the Synchronize view and clicking **Update**.

Override

Red entries with double arrows ( repayment.brl (1.1)) indicate that changes occurred in both Rule Designer and in Decision Center. This change corresponds to a conflict, and you must decide how to proceed. You cannot do automatic merging.

To resolve a conflict, you must **override** the proposed direction of a change by right-clicking the entry in the Synchronize view and clicking one of these options:










- **Override and Update** to keep the changes from Decision Center and update them back to Rule Designer.
- **Override and Publish** to keep the changes from Rule Designer and publish them to Decision Center.

Note:

If a change made on the BOM affects a rule, the Synchronization view displays both the changed BOM and the affected rule. You can decide whether to publish or update these changes.

In addition to its direction, the decoration of an entry also indicates the nature of the change, such as whether artifacts are added, deleted, or modified. The following table describes the options.

Table shows synchronization view decorations.

De cor ati on	Description
	An incoming (+) means that Decision Center contains a new artifact. Updating transfers the artifact to your Rule Designer workspace.
	An incoming change means that Decision Center contains a changed artifact. Updating transfers the new version to your Rule Designer workspace.
	An incoming (-) means that Decision Center no long contains an artifact. Updating deletes the artifact from your Rule Designer workspace.
	An outgoing (+) means that your Rule Designer workspace contains an artifact that is not in Decision Center. Publishing transfers the artifact to the remote project.
	An outgoing change means that Rule Designer contains a locally changed artifact that is not in the remote project. Publishing transfers the artifact to Decision Center and creates a new version.
	An outgoing (-) means that your Rule Designer workspace no longer contains an artifact. Publishing deletes the artifact from the remote Decision Center.
	A conflicting (+) means that Rule Designer and the remote Decision Center project contain a locally added artifact. You must resolve the conflict.
	A conflicting change means that Rule Designer and the remote Decision Center project contain a locally changed artifact. You must resolve the conflict.
	A conflicting (-) means that Rule Designer and the remote Decision Center project no longer contain an artifact. You must resolve the conflict.

Parent topic: [Synchronizing from Rule Designer](#)

Related concepts:
[Rule project items](#)

Related tasks:
[Creating projects from Decision Center](#)
[Publishing decision services to Decision Center](#)

Related information:
[Change management](#)
[Synchronization architecture](#)

Creating projects from Decision Center

You can create a project or in Rule Designer from an existing Decision Center project.

About this task

Business users and developers can collaborate on rule projects by copying the project or a branch of the project from Decision Center to Rule Designer.

You cannot copy a project or a branch of a project into an Eclipse workspace that already contains the project. You must copy the project or branch to a different workspace, or delete the project from the target workspace before you copy the project or branch to the workspace.

Note:

Decision services are created in Rule Designer, and then published to Decision Center.

Procedure

To create a project from an existing Decision Center project:

1. Click **File > New > Project**.
2. In the New Project wizard, select **Rule Project from Decision Center**, and then click **Next**.
3. In the **New Rule Project from Decision Center** window, complete the **URL** and **Data source** fields with information corresponding to your Decision Center session. By default, **Data source** is set to `jdbc/iLogDataSource`:

New Rule Project from Decision Center

Decision Center Configuration

Configure the project to be synchronized with a Decision Center project.

Connection

URL:

Not yet authenticated. Click the Connect button.

Data source:

Connect

Synchronization Filter

☐ Use the specified query to filter rule elements for synchronization

Project configuration

Remote project

Refresh list

Select a branch or release

Select a change activity

Local project contents

☒ Use defaults

Location **Browse ...**

< Back **Next >** **Finish** **Cancel**

Note: In the URL, you can use the `/teamserver` extension, or `/decisioncenter` extension indifferently.

4. Indicate where you want the copy to be stored on your computer if the location differs from your Eclipse workspace.

Remember that you cannot copy the project or a branch of the project to a workspace that already contains the same project.

5. Click **Connect** to establish a connection with Decision Center.
6. Proceed with the authentication by entering your Operational Decision Manager on Cloud credentials.
7. To limit the rule elements that are synchronized, select the **Use the specified query to filter rule elements for synchronization** check box, and then select a query from the list of queries. If you do not use a query, all the elements are copied.
8. In **Project configuration**, use the **Remote project** list to select the Decision Center project that you want to import.

Click **Refresh list** if your project is not in the list.

9. In **Select a branch or release**, select a branch or release if there is more than one in your project.
10. In **Select a change activity**, select a change activity if you want to import an activity from a release.
11. Click **Finish**.

The **Problems** view displays any errors, but the import continues. When the import finishes copying the project to your workspace, you can see the project in the **Rule** perspective.

Parent topic: [Synchronizing from Rule Designer](#)

Related concepts:

[Synchronization commands in Rule Designer](#)

Related tasks:

[Publishing decision services to Decision Center](#)

Related information:

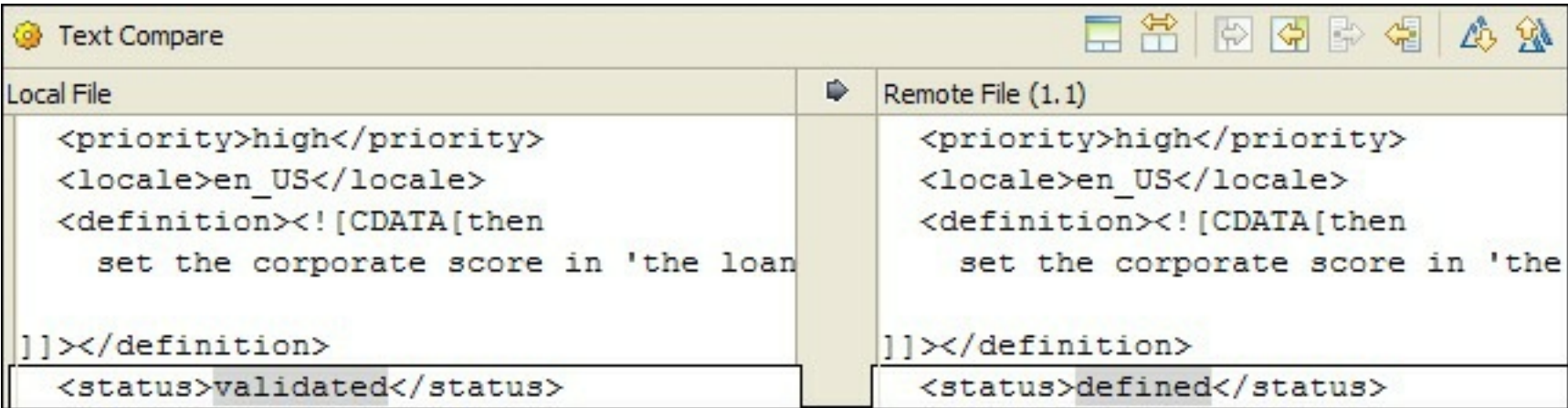
[Synchronization architecture](#)

[Change management](#)

Use of Text Compare in synchronization

Display conflicting rules and their differences in the Eclipse Text Compare Editor.

When the Synchronize view indicates a difference between the local Rule Designer and remote Decision Center versions of a rule, you can double-click the entry to display both rules and their differences in the Eclipse Text Compare Editor:



The Local File pane displays the local rule in its XML or modified text format. You can edit the rule in the Local File pane, but this practice is not the usual approach.

Note:

For details on the structure of the XML files, see [File types](#).

The Remote File pane displays the remote Decision Center version of the rule in read-only mode. The format of the remote version of the rule is an equivalent XML version, transformed so that you can compare them to find differences and resolve conflicts.

Note:

Saving a rule in Decision Center without changing the content increments the version number. As a result, the two different rules look the same in Text Compare.

After you check the contents of the local and remote versions, you can update, publish, or override to resolve any conflicts.

For details about the features available in the Text Compare Editor, see the Eclipse Help (**Workbench User Guide > Reference > User interface information > Views and editors > Compare editor**). In particular, use the **Show/Hide Ancestor Pane** button to verify the state of the common ancestor that is used in three-way comparisons.

Parent topic: [Synchronizing from Rule Designer](#)

Related concepts:

[Synchronization commands in Rule Designer](#)

Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

Synchronization and source code control

Use source code control (SCC) to share Rule Designer artifacts, and commit rule project resources.

You must know your SCC well before you implement synchronization with Decision Center. Section [Rule project items](#) provides a list of the resources to commit to your SCC, and items not to commit, for example the output folder and connection entry files.

Some SCC systems use the Team Synchronization feature in Eclipse. You access the systems through **Team > Share**.

The Synchronize view can handle multiple participants. Therefore, you can have one Decision Center participant and one SCC participant for each rule project in Rule Designer.

To have two different branches of the same project, import the projects into separate workspaces. If you update the UUIDs manually, you break synchronization.

Parent topic: [Storing and synchronizing rules](#)

Related concepts:

[Rule project items](#)

Related tasks:

[Publishing decision services to Decision Center](#)

Rule project items

A rule project is a container for organizing rule artifacts and setting up the business object model (BOM) and rule authoring vocabulary. Each rule project item is associated with a folder.

You implement a rule project in Rule Designer as an Eclipse project, which serves as a container for organizing rule-related items. A rule project can contain different types of folders:

Source folder (rules)

A root container for the rule packages and artifacts.

BOM folder (bom)

Contains the files that are related to the business object model (BOM). A BOM file that is stored in the BOM folder is part of the BOM path.

Deployment folder (deployment)

Contains the deployment configuration for a decision service.

Query folder (queries)

Contains query files.

Resource folder (resources)

Contains resource files, that is, files or folders that are not part of the rule model.

Template folder (templates)

Contains template files.

Reports folder (reports)

Contains the reports that are created by different operations, including deployment.

The folders are registered in the project properties, and must be located directly under the rule project. The source folder contains rule packages that contain rule artifacts. It is also the root package, so it can contain rule artifacts.

Note:

If you rename a folder or subfolder in Rule Designer and synchronize with Decision Center, you obtain a new version in Decision Center for every artifact in that folder.

Each rule project item is associated with a folder, a file, or both. The following table describes these associations. The **Decision Center** column indicates which project items synchronize with Decision Center. The **SCC** column indicates which project items must be committed when you use a source code control system.

Item	As so cia te d wi th	Comments	D e ci si o n C e n t e r	S C C
Rule proje ct	Th e roo t fol der		⊗	✓
	.p ro je ct file	XML file that stores the general Eclipse project information, such as the nature of the project or the launch configuration.	⊗	✓
	.r ul ep ro je ct	XML file that stores the project properties that are specific to a rule project: <ul style="list-style-type: none">• Categories• BOM path	✓	✓

	file	<ul style="list-style-type: none"> • XOM path • Output location • Relative path of the source, BOM, query, template, and resources folders. <p>Displayed with the “Project properties” label in the Synchronize view.</p>		
Rule package	A rule package folder	Called a folder in Decision Center.	✓	✓
	.rulpackage file	XML file that stores the information for the rule package. Displayed with the “Package properties” label in the Synchronize view.	✓	✓
Action rule	.brl file	XML file that stores the properties and the definition of an action rule.	✓	✓
Decision table	.dta file	XML file that stores the properties and the definition of a decision table.	✓	✓
Decision tree	.dtr file	XML file that stores the properties and the definition of a decision tree.	✓	✓
Function	.fct file	XML file that stores the properties and the definition of a function.	✓	✓
Ruleflow	.rfl file	XML file that stores the properties, the task definitions, and the description of a ruleflow diagram.	✓	✓
Technical rule	.trl file	XML file that stores the properties and the definition of a technical rule.	✓	✓
Deployment configuration	.dep file	The manner in which decision operations are packaged into RuleApps, managed, and then deployed.	✓	✓
Decision operation	.dop file	A function that defines the decision-making logic, and the input and output data for a decision. A decision operation is implemented as a ruleset.	✓	✓
Variable set	.var file	XML file that stores a list of variables.	✓	✓
Action rule template	.brt file	XML file that stores the template properties, and the properties and the definition of the action rule to instantiate.	✓	✓
Query	.qry file	XML file that stores the properties and the definition of a query.	✓	✓
BOM entry	.bom file	Plain text file that stores the structure of a BOM entry.	✓	✓
	<locale>.voc file	Key-value property file that stores the verbalization information that is attached to BOM elements. The first part of the keys corresponds to the fully qualified name of the BOM elements. The second part defines the verbalization of the BOM elements.	✓	✓

	.b2x file	XML file that stores the functions that map the BOM to the XOM.	✓	✓
Source folder	The source folder	The source folder is not a project item as such, but a container for the rule artifacts.	✗	✓
BOM folder	The bom folder	The BOM folder is not a project item as such, but a container for the BOM entries. All the BOM entries that are directly under the BOM folder or under folders in the BOM folder are part of the BOM and are referred to as the BOM path.	✗	✓
Deployment folder	The deployment folder	The deployment folder contains the deployment configuration for deploying a decision service.	✓	✓
Query folder	The queries folder	The queries folder is a container for the queries that can be used in the project.	✓	✓
Resource folder	The resources folder	<p>The resource folder is a container for files that are not part of the rule model. (See Defining a structure for rule project items and Creating a resource).</p> <p>The deployment.xml is created in the META-INF resource folder, and used with the managed XOM feature.</p> <p>If you selected the decision engine as rule engine, a B2X folder is created under the resources folder. The files in the B2X folder are taken into account during the synchronization with Decision Center and are required for the generation of ruleset archives from Decision Center.</p>	✓	✓
Template folder	The templates folder	The template folder is a container for the templates that can be used in the project and any dependent projects. The complete list of templates is computed by gathering the templates that are stored directly under the template folder or any of its subfolders.	✓	✓
Output folder	The output folder	The output folder stores compiled files that are generated when you build the project.	✗	✗
Reports folder	The reports folder	The reports folder holds the reports that are generated by different operations.	✗	✓
Decision Cent	The .s	File used to share the synchronization state between Rule Designer and Decision Center. Committing this file to SCC lets you work on the same project in a different workspace without generating	✗	✓

er conn ectio n entry	yn cE nt ri es file	conflicts.		
-----------------------------------	------------------------------------	------------	--	--

Parent topic: [Storing and synchronizing rules](#)

Related information:
[Synchronization architecture](#)

Setting up notifications from Decision Center with webhooks

You can use webhooks to receive notifications from Decision Center to the application of your choice.

A webhook is an HTTP custom callback, which you use to send notifications to external applications. A webhook is defined by the three following properties:

- Mandatory: A URL that specifies the endpoint where to send the event.
- Optional: A data source. If you do not specify one, the default data source is used.
- Optional: An authentication token to authenticate to your remote secure server.

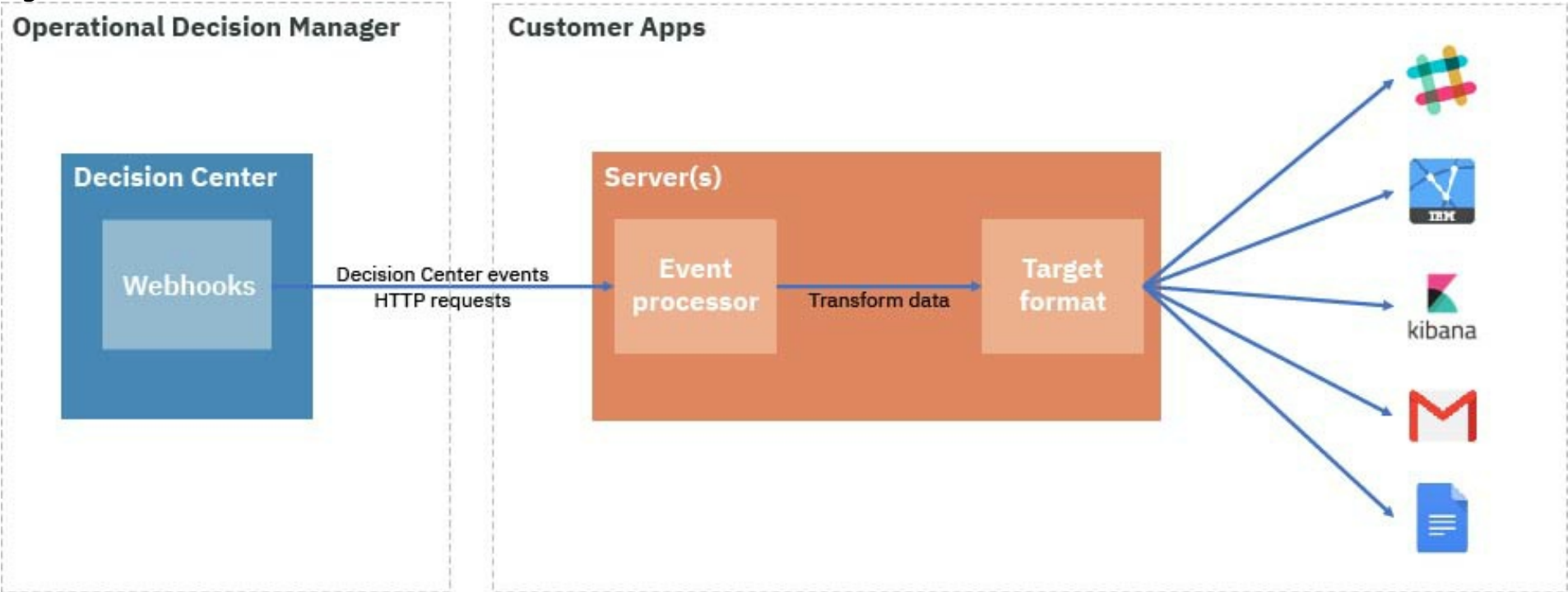
You can be notified in real time of the following types of events:

- When actions that are related to the decision governance framework happen in Decision Center, for example when a user creates a release, or approves an activity.
- When a deployment ends, you are notified of its status: completed, failed, or aborted.
- When a user creates or updates a rule or a decision table.

You register webhooks with the Decision Center REST API. Events are sent to the registered endpoints in JSON format. You must set up a server able to handle events and to convert them to a format compatible with the target application. For example, if you want to send notifications to Slack, your server must convert the events sent from Decision Center to a format that is compatible with Slack.

Notifications are asynchronous, which means they are done in the background.

Figure 1. Webhooks mechanism in Decision Center



Configuring your webhooks

The Decision Center REST API provides you with four endpoints to handle webhooks:

- PUT /webhook/notify to define one or more URLs where notifications are sent, and optionally provide an authentication token.
- GET /webhooks/notify to see the list of URLs getting notifications from this instance of Decision Center.
- DELETE /webhooks/{webhookId}/notify to delete a webhook.
- POST /webhooks/{webhookId}/notify to update an existing webhook.

For more information, see [IBM® ODM Decision Center API](#)

Receiving a webhook notification

The JSON payloads received by the server contain various information about your event. The following data is sent:

Table 1. Information sent by an event

Data	Description
id	Unique identifier that is given to an event. You use this ID to find the event in log files.
version	Version number of an event. This allows you to identify the version of the event.
author	The user that triggered the notification.
date	The date when an event was emitted, represented a Java™ date (long value).
sourcename	The name of the application that sends the event. This value is always Decision Center.
sourcelink	The URL of the Decision Center instance that sends the event.
type	The type of event that triggers a webhook, which allows you to filter events.

	<p>The following types are possible:</p> <p>Rule artifacts</p> <p>RuleCreated, when a rule is created. RuleUpdated, when a rule is updated. RuleDeleted, when a rule is deleted.</p> <p>Decision Governance Framework</p> <p>ReleaseCreated, when a release is created. ReleaseUpdated, when a release is updated. ReleaseDeleted, when a release is deleted. ActivityCreated, when an activity is created. ActivityUpdated, when an activity is updated. ActivityDeleted, when an activity is deleted.</p> <p>Deployment</p> <p>RuleAppDeployment, when a RuleApp is deployed.</p>
content	An array with information about the artifact associated with the event. For example, if you created a release, you have release details such as the approver’s name, due date, and so on.
Optional: details	An array with additional information about the event.
project	An array with information about the project associated with the event.
decisionService	An array with information about the decision service associated with the event.

Examples

The following example shows an event that is sent when a release is created:

```
{
  "version": "1.0",
  "id": "f3577511-c33b-4275-a5ef-de177598305b",
  "author": "rtsAdmin",
  "date": 1532519296400,
  "type": "ReleaseCreated",
  "content": [{
    "id": "faa9b2af-e2fe-42a4-9834-bb9cdd607334",
    "internalId": "brm.Release:866:866",
    "name": "New Release",
    "parentId": "7b360d2b-46ca-44c1-a945-fb91b9a0103e",
    "documentation": "",
    "buildMode": null,
    "status": "InProgress",
    "owner": "rtsAdmin",
    "targetDate": "2018-08-07T22:00:00.000Z",
    "approvers": [],
    "initial": false,
    "kind": "Release"
  }],
  "sourceName": "Decision Center",
  "sourceLink": "http://9.145.66.219:8081/decisioncenter?datasource=jdbc%2FilogDataSource",
  "project": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
  },
  "decisionService": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
  }
}
```

The following example shows an event that is sent when a change activity is updated. Here, the due date was changed. The details of the change are stored in the detail field.

```
{
  "version": "1.0",
  "id": "4eef8f4d-7c87-4022-b0aa-28bf88ffdb7f",
  "author": "rtsAdmin",
  "date": 1532519323387,
  "type": "ActivityUpdated",
  "content": [{
    "id": "0eade79c-14e5-4155-8a08-23c7d52a168b",
    "internalId": "brm.ChangeAct:881:881",
    "name": "Spring Activity",
    "parentId": "faa9b2af-e2fe-42a4-9834-bb9cdd607334",
    "documentation": "",
    "buildMode": "ClassicEngine",
    "status": "InProgress",
    "owner": "rtsAdmin",
    "targetDate": "2018-08-06T22:00:00.000Z",
    "approvers": [],
    "authors": [],
    "kind": "ChangeActivity"
  }],
  "details": [{
    "targetURL":
"http://9.145.66.219:8081/decisioncenter/t/library#overviewactivity?
datasource=jdbc%2FilogDataSource&baselineId=brm.ChangeAct%3A881%3A881",
    "updateType": "UPDATE_DUE_DATE",
    "oldValue": 1533679200000,
    "newValue": 1533592800000
  }],
  "sourceName": "Decision Center",
  "sourceLink": "http://9.145.66.219:8081/decisioncenter?
datasource=jdbc%2FilogDataSource",
  "project": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
  },
  "decisionService": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
  }
}
```

The following example shows an event that is sent when a RuleApp is deployed:

```
{
  "version": "1.0",
  "id": "e424a030-4aea-4452-bb5f-7450893a9803",
  "author": "rtsAdmin",
  "date": 1532519935332,
  "type": "RuleAppDeployment",
  "content": [{
    "id": null,
    "internalId": null,
    "name": "Report 2018-07-25_01-58-41-821",
    "status": "COMPLETED",
    "ruleAppName": "autoQuoteRuleApp",
    "messages": {
      "elements": [],
    }
  }]
```

```
        "totalCount": 0,
        "number": -1,
        "size": 0
    },
    "snapshot": {
        "id": "4758d793-e953-4f11-824f-9e9445cfa02a",
        "internalId": "dsm.DsDeploymentBsln:888:888",
        "name": "Deployment_to_QA_env_2018-07-25T11_58_39Z",
        "parentId": "0eade79c-14e5-4155-8a08-23c7d52a168b",
        "documentation": null,
        "buildMode": "ClassicEngine",
        "kind": "DeploymentSnapshot"
    },
    "servers": [],
    "archive": null
}],
"details": [{
    "targetURL":
"http://9.145.66.219:8081/decisioncenter/t/library#deploymentreport?
id=dsm.DSDeploymentReport%3A1%3A1&datasource=jdbc%2FilogDataSource&baselineId=
brm.ChangeAct%3A881%3A881"
}],
"sourceName": "Decision Center",
"sourceLink": "http://9.145.66.219:8081/decisioncenter?
datasource=jdbc%2FilogDataSource",
"project": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
},
"decisionService": {
    "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
    "internalId": "brm.RuleProject:229:229",
    "name": "AutoQuote",
    "buildMode": "ClassicEngine"
}
}
```


Deploying rule applications

You deploy a rule application, or RuleApp, from a decision service to an execution server. The RuleApp contains rulesets for validation or use with calling applications.

You use deployment configurations in decision services to deploy RuleApps. The deployment configurations apply decision operations that specify the rules to include in the rulesets of RuleApps. They also specify target execution servers, and the version policies for RuleApps and rulesets.

You also deploy the execution object model (XOM) that references the application objects and data on which you run your rules. You create and maintain the XOM in Rule Designer, and publish the XOM binary to Decision Center. You must keep the XOM synchronized in the components to ensure that the deployed RuleApps reference the correct XOM.

Note: For development purposes, you can deploy a RuleApp from Rule Designer to the Rule Execution Server in a development environment. However, you cannot deploy a RuleApp from Rule Designer directly to the Rule Execution Server in a test or production environment.

The following table summarizes the deployment tasks by user role:

User role	Deployment tasks
Rule developer	<div>Creates a decision service in Rule Designer and adds deployment configurations.</div> <div>Deploys a RuleApp for the decision service from Rule Designer or Decision Center to the development environment to verify and troubleshoot the deployment process.</div> <div>Publishes the decision service, its deployment configurations, and the XOM to Decision Center.</div>
Release manager	<div>Deploys RuleApps for decision services from Decision Center to the development, test, and production execution servers.</div> <div>Creates deployment configurations in the Decision Center Business console.</div>
Business user	<div>To validate a decision service, deploys RuleApps from Decision Center to the development and test execution servers.</div>
Integrator	<div>Deploys RuleApps to the development and test execution servers.</div>

Deployment configurations

You use deployment configurations to deploy RuleApps from decision services. They include the decision operations, target servers, and version policies for deploying rulesets in RuleApps.

Version policies

Version policies determine how client applications identify deployed rulesets.

Configuring deployment in Rule Designer

In Rule Designer, you configure the deployment of decision services from Decision Center.

Deploying from Rule Designer in the development environment

In an early stage of decision service development, you might want to deploy a decision service from your local Rule Designer to Rule Execution Server on the cloud to test the execution of the service before publishing it to Decision Center.

Deploying from Decision Center

You can deploy any branch (release, change activity, or regular branch) of a decision service from the Decision Center Business console.

XOM deployment

XOM deployment for decision services can be done with Rule Designer or with Decision Center. You synchronize the XOM binary file between Rule Designer and Decision Center to ensure that RuleApps deployed from either environment reference the correct XOM.

Embedded managed Java XOM in Decision Center

Embedded managed Java™ XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can download RuleApp archives with embedded managed Java XOMs by using the Decision Center REST API. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

Deploying to a hybrid cloud environment

To use Operational Decision Manager on Cloud in hybrid mode, you download and deploy a decision

service to a Decision Server that is outside the Operational Decision Manager on Cloud environment.

Related information:

[User roles, environments, and components](#)

Deployment configurations

You use deployment configurations to deploy RuleApps from decision services. They include the decision operations, target servers, and version policies for deploying rulesets in RuleApps.

You can create or edit a deployment configuration in Rule Designer or the Decision Center Business console. In the Business console, only release managers can create or edit a deployment configuration. When working within the governance framework, a deployment configuration can be created or edited only within a change activity.

A deployment configuration generates a RuleApp archive that contains rulesets. These rulesets are defined in the decision operations in the decision service. You can select one or more rules by applying either a custom validator or a query. Decision operations are defined in Rule Designer and cannot be changed in Decision Center.

You create three deployment configurations, for development, testing, and production.

A deployment configuration includes the following information:

- Configuration type (production or nonproduction): limits deployment to certain servers
- RuleApp name: used by the client application to request a decision
- Decision operations: lists the rules to be deployed
- Target server: runs the rule applications
- Version information: shows the ruleset base version number and versioning policy

In the Business console, the target server of a deployment configuration determines which user groups can use the configuration. For example, the business user group can deploy a decision service to the development environment if the target server of the deployment configuration is the development server.

For each operation, the deployment configuration contains a base ruleset version number whose role depends on the version policy. In the increment version policy, it defines the minimal version, and in the base version policy, the deployed ruleset version. In the user-defined version policy, it is used to calculate a proposed ruleset version at deployment time.

You can also set ruleset execution options in the deployment configuration:

- Enabled: You can enable execution of a ruleset in a decision service. When you disable a ruleset, this ruleset is not taken into account by the rule engine at execution time.
- Debug: You can enable debug mode on a ruleset in a decision service.
- Trace: You can monitor ruleset execution by generating execution traces.
- Bytecode generation: You can generate the ruleset with Java™ bytecode from the Business console.

Configuration type

You indicate whether a deployment configuration is for a production or nonproduction environment:

Production

When a release of a decision service completes its lifecycle in the governance framework, it can be deployed to a production server.

Nonproduction

When a release of a decision service is under development in the governance framework, it cannot be deployed to a production server.

Target servers

When deploying from a decision service, you choose the target Rule Execution Server that is defined in the deployment configuration.

The user role and the release state determine the servers to which a user can deploy. For example, if you are working in a change activity, you can deploy to a test environment but not to production. Typically, business users are limited to the development and test environments, while release managers can deploy to production.

Rule Execution Server Connections view

A deployment configuration specifies the target server for the deployment. Only a symbolic name for the target server is stored in the deployment configuration. The connection details for the server are defined in the Rule Execution Server Connections view of Rule Designer, which is separate from the deployment configuration.

Parent topic: [Deploying rule applications](#)

Version policies

Version policies determine how client applications identify deployed rulesets.

In a deployment configuration for a decision service, you establish the version policies to define the paths of the rulesets in the RuleApp deployment. The ruleset path determines how an application calls a ruleset through Rule Execution Server.

The ruleset path can have the following structure:

DeploymentConfiguration/RAVmajor.RAVminor/Operation/RSVmajor.RSVminor

DeploymentConfiguration identifies the decision service. RAVmajor.RAVminor identifies the version of the decision service interface that is being used. Client applications use the version number to ensure that they are compatible with the decision service interface. Operation is the name of the ruleset. The major version number of a ruleset (Vmajor) often represents a specific release of the ruleset, and the minor version number (Vminor) represents a deployment version of the ruleset. Typically, client applications call the latest enabled ruleset, but what they actually call depends on the ruleset path that is used in the client application.

The version numbers use base numbers that you define in the deployment configuration. You manually define a base number for the RuleApp and each ruleset. Deployment increments the minor ruleset version number, or replaces or creates a ruleset version number (Vmajor.Vminor).

Note: The version policies that are set in a Rule Designer deployment configuration are synchronized with Decision Center.

Select a policy by the type of deployment

The version policy that you use depends on the type of deployment:

Type of deployment	Version policy
Successive deployments of a release to a specific server	Increment minor version numbers
Development of a decision service	Use the base version numbers
Test fix to correct a deployed solution	Use the base version numbers
Test fixes or updates to an earlier release	Define the version numbers

Increment minor version numbers

This policy serves as the default setting. It deploys the decision operations of the release, changing the minor version number of each ruleset. It looks for the latest version number to increment on the server to which it has access, and the same deployed version number is used.

A client application uses a ruleset based on the ruleset path that is defined in the client application., but previous deployments remain available on Rule Execution Server. You can still use a previous deployment by specifying its full ruleset path. You use this policy with successive deployments of a release to a specific server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/2.1</div>

Use the base version numbers

This policy deploys the decision operations and replaces the base version on Rule Execution Server. The replacement ensures that the deployment configuration path corresponds exactly to what is deployed on the server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 (replaced)</div>

Define the version numbers

With this policy, you enter a specific ruleset version number at deployment time. You use this policy with test fixes or updates to an earlier release. Upon deployment, Rule Execution Server uses the specified ruleset version, and replaces the last version of the ruleset.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 (replaced) /ruleset/2.0</div>

Parent topic: [Deploying rule applications](#)

Configuring deployment in Rule Designer

In Rule Designer, you configure the deployment of decision services from Decision Center.

The rule developer uses Rule Designer to do the following deployment-related tasks:

- Set up deployment configurations, which define the decision operations and target servers for deployment from Decision Center.
- Run a decision service locally by using a run configuration, and deploy to the development environment to verify the deployment process.
- Publish the decision service so that the deployment configurations and the XOM are available in Decision Center.

Deployment artifacts

In Rule Designer, each rule project contained in a decision service has a **deployment** folder to store the deployment configurations and decision operations created by the rule developer. Typically, the deployment configurations are kept in the **main** project of the decision service.

See [Deployment configurations](#) for information about the different aspects of deployment configurations. Also, consider the following points for Operational Decision Manager on Cloud:

- Target servers are available for the development, testing, and production environments. The recommended approach is to create a deployment configuration for each environment.
- The configuration type is **Production** for the production server, and **Nonproduction** for the development and testing environments.
- From Rule Designer, you can only deploy to the development environment, and not to the test and production environments.

Parent topic: [Deploying rule applications](#)

Related concepts:

[Deploying from Decision Center](#)

[XOM deployment with decision services](#)

Related information:

[Synchronizing from Rule Designer](#)

Deploying from Rule Designer in the development environment

In an early stage of decision service development, you might want to deploy a decision service from your local Rule Designer to Rule Execution Server on the cloud to test the execution of the service before publishing it to Decision Center.

Procedure

1. In Rule Designer, right-click your decision service.
2. Click **Rule Execution Server > Deploy**.
3. In the RuleApp Deployment dialog, click the development deployment configuration, and then click **Next**.
4. Verify the deployment configuration settings, and then click **Next**.
5. Verify your authentication credentials.
6. Click **Connect** if you are not yet authenticated, and then click **Next**.
7. Verify the RuleApp and ruleset versions, and then click **Finish**.

When the RuleApp is deployed, a deployment report is displayed in Rule Designer.

Note: From Rule Designer, you cannot deploy to the test and production environments, but only to the development environment.
--

8. Use a test application to call the decision service that you have deployed.
9. Check that the behavior is as expected.

Parent topic: [Deploying rule applications](#)

Deploying from Decision Center

You can deploy any branch (release, change activity, or regular branch) of a decision service from the Decision Center Business console.

All the information required for deployment is contained in a deployment configuration. Deployment configurations can be synchronized with Rule Designer.

Only a release manager can create, edit, or delete deployment configurations in the Business console. This ensures a basic level of security on deployment. Deployment configurations contain information that includes the target servers, the decision operations that define the business rules, and settings for versioning of rulesets and RuleApps. The following aspects of a deployment configuration relate to making sure that deployment is secure:

- The configuration type as Production or Nonproduction.
- The target server where this deployment configuration can deploy to.
- User groups that can deploy with the configuration depend on the selected server.

Any user who can access a decision service in the Business console can deploy its branches (release, change activity, or regular branch) from any deployment configuration available to that user in that branch. The deployment configurations that are available for a given user depend on this user's role (see [Deploying rule applications](#)). The following conditions apply:

- In a change activity, you can deploy only with a deployment configuration whose type is Nonproduction.
- In a release, you can deploy with a deployment configuration whose type is Production only if the release is completed.

When you deploy a decision service, the XOM is also deployed if the version of the XOM is not already present in the target Rule Execution Server.

When working within the governance framework, a deployment configuration can only be created or edited within a change activity.

Deployment snapshots can be found in the list of snapshots. You can then redeploy from the deployment snapshot or from the report of the deployment.

Note: You can also automate rule deployment by using the Decision Center REST API .
--

Parent topic: [Deploying rule applications](#)

Related concepts:

[XOM deployment from Decision Center](#)

XOM deployment

XOM deployment for decision services can be done with Rule Designer or with Decision Center. You synchronize the XOM binary file between Rule Designer and Decision Center to ensure that RuleApps deployed from either environment reference the correct XOM.

[XOM deployment with decision services](#)

A RuleApp that you deploy must reference the correct XOM.

[XOM deployment from Decision Center](#)

When you deploy from Decision Center, the XOM is also deployed if the version is not already present in the target environment.

Parent topic: [Deploying rule applications](#)

XOM deployment with decision services

A RuleApp that you deploy must reference the correct XOM.

When you deploy a decision service, Rule Designer goes through the following process:

1. Builds the XOM binary from the source files that are referenced by the decision service.
2. Deploys the XOM if it is not already present in Rule Execution Server, and retrieves the XOM URI.
3. Copies this URI to each ruleset that is contained in the RuleApp. Specifically, Rule Designer copies the URI to a property of the ruleset called `ruleset.managedxom.uri`.

With this mechanism, each ruleset in Rule Execution Server references the correct XOM.

Note: If the XOM is based on an XML schema (.xsd file), the XML schema is packaged into the ruleset archive and deployed with the RuleApp.

XOM deployment in Decision Center

When Rule Designer builds the XOM from the source files, it places the XOM binary under `resources/xom-libraries`. The XOM binary is then published to Decision Center with the rest of the decision service.

With the XOM binary file synchronized between Rule Designer and Decision Center, RuleApps deployed from either environment reference the correct XOM.

XOM modifications

You modify the XOM source files in Rule Designer. Rule Designer always builds the XOM binary from the source files. You can then publish the XOM to Decision Center.

If you import a decision service from Decision Center into an empty workspace:

- You do not obtain the XOM source files. You must obtain these source files in the usual way, for example through source code control.
- The XOM binary obtained from Decision Center is copied to the `resources/xom-libraries` folder. The files in this folder should not be used in the XOM path.

Parent topic: [XOM deployment](#)

Related concepts:

[XOM deployment from Decision Center](#)

XOM deployment from Decision Center

When you deploy from Decision Center, the XOM is also deployed if the version is not already present in the target environment.

The XOM must first be published from Rule Designer to Decision Center. The XOM is stored in the resources of the BOM project of the decision service. All RuleApps created from this decision service reference this version of the XOM.

At deployment time, Decision Center creates a deployment snapshot, which contains the version of all rules and deployment artifacts at that moment in time. This ensures that any redeployed RuleApp references the correct XOM.

Important:

You cannot access the XOM source files from Decision Center. You modify XOM source files in Rule Designer.

In Decision Center, information on the XOM is visible in the deployment report. If you need to see the XOM files in Decision Center, create a smart folder that displays resources in the Enterprise console.

Parent topic: [XOM deployment](#)

Related concepts:

[XOM deployment with decision services](#)

Related information:



[Smart folders](#)

Embedded managed Java XOM in Decision Center

Embedded managed Java™ XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can download RuleApp archives with embedded managed Java XOMs by using the Decision Center REST API. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

The embedded managed Java XOM feature can be operated from the Decision Center REST API.

Table 1. Embedded managed XOMs

Module or tool	Deploy	Create or retrieve	Description
REST API		 You can create a RuleApp archive with or without the embedded managed Java XOM.	For more information, see downloadUsingGET . You set the parameter includeXOMInArchive to true to include the XOM in your RuleApp archive.

Parent topic: [Deploying rule applications](#)

Related concepts:
[Embedded managed Java XOM in Rule Execution Server](#)

Deploying to a hybrid cloud environment

To use Operational Decision Manager on Cloud in hybrid mode, you download and deploy a decision service to a Decision Server that is outside the Operational Decision Manager on Cloud environment.

Before you begin

To work in a hybrid cloud environment, the following prerequisites apply:

- Access to a source Operational Decision Manager on Cloud environment where a decision service (RuleApp and its XOM) is deployed to a Rule Execution Server. The RuleApp must be linked to the XOM, and both must be working properly in Operational Decision Manager on Cloud. You need credentials and the URL of the Rule Execution Server.
- Access to a target environment, for example:
 - On-premises installation of IBM® Operational Decision Manager Decision Server
 - Another Operational Decision Manager on Cloud environment
- Administration credentials and the URL of the target Rule Execution Server.

Important: The target Decision Server must be compatible with the source Operational Decision Manager components for them to work together correctly. Refer to [IBM Operational Decision Manager on Cloud Compatibility](#).

About this task

You can use the REST API or the Rule Execution Server console to download and deploy a decision service to another Decision Server.

[Using the Rule Execution Server REST API](#)

To deploy a decision service to a target Decision Server, you download the corresponding RuleApp and its XOM from the source Operational Decision Manager on Cloud environment, and use the REST API to place the RuleApp and XOM binary files on the target Decision Server. Then, you can run the decision service on this Decision Server.

[Using the Rule Execution Server console](#)

To deploy a decision service to a Decision Server that is outside Operational Decision Manager on Cloud, you download the corresponding executable assets (the RuleApp and its XOM) from the Rule Execution Server console of the cloud portal, and then deploy them to the target Decision Server through the Rule Execution Server console.

Parent topic: [Deploying rule applications](#)

Related information:

[IBM ODM on Cloud in a hybrid cloud environment](#)
[Testing a ruleset for REST execution](#)

Using the Rule Execution Server REST API

To deploy a decision service to a target Decision Server, you download the corresponding RuleApp and its XOM from the source Operational Decision Manager on Cloud environment, and use the REST API to place the RuleApp and XOM binary files on the target Decision Server. Then, you can run the decision service on this Decision Server.

Before you begin

You must authenticate with Operational Decision Manager on Cloud and deployed the decision service.

About this task

The example commands that are provided in the following procedure use cURL to authenticate with the cloud server and transfer archive files from the cloud server to another Decision Server. You can use a different program to send authentication credentials to the servers, access the REST API, and download files from the cloud server.

Procedure

1. Download the RuleApp from Operational Decision Manager on Cloud.

The command includes basic authentication credentials and the path to the RuleApp. Replace the user name and password variables with your basic authentication credentials. Provide the host URL of your cloud instance, and specify the correct name of the RuleApp in the path:

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/v1/ruleapps/ruleapp_name/1.0/archive --
output ruleapp_name.jar
```

2. Download the XOM:

- a. In the ruleset.managedxom.uris property in the output of the previous command, look for the managed XOM URI. This property gets the RuleApp details.
- b. If the property points to a XOM resource (resuri), skip this step. If it points to a library (reslib), check the content property in the output of the previous command for the XOM URI. The content property gets the library details. Use the name and version of the library that is returned in the previous command.

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/libraries/library_name/1.0
```

- c. Download the XOM. Use the name and version of the resource returned in the previous command.

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/xoms/xom_name/1.0/bytecode --output
xom_name.zip
```

3. Upload the XOM.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/xoms/xom_name.zip --data-binary @xom_name.zip -H
"Content-Type:application/octet-stream"
```

If the RuleApp references a library rather than a XOM, re-create the library and specify the XOM resource that it is supposed to contain.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/libraries/library_name1.0 -d
"resuri://xom_name.zip/1.0" -H "Content-Type:text/plain"
```

Note: Keep the names of the resources when downloading and uploading them.

4. Upload the RuleApp to the other Decision Server.

The command includes the user name, password, and URL to access the Rule Execution Server console on the other Decision Server.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/ruleapps --data-binary @ruleapp_name.jar -H
```

```
"Content-Type:application/octet-stream"
```

5. Test the execution of the ruleset (decision service) by using its REST API.

```
curl -X POST -k -u username@company_name.com:password  
server_URL/DecisionService/rest/v1/ruleappname/ruleappversion/rulesetname/  
rulesetversion -d jsonpayload -H "Content-Type:application/json"
```

Parent topic: [Deploying to a hybrid cloud environment](#)

Using the Rule Execution Server console

To deploy a decision service to a Decision Server that is outside Operational Decision Manager on Cloud, you download the corresponding executable assets (the RuleApp and its XOM) from the Rule Execution Server console of the cloud portal, and then deploy them to the target Decision Server through the Rule Execution Server console.

Before you begin

You must authenticate with Operational Decision Manager on Cloud and deployed the decision service.

Procedure

1. Download the RuleApp from Operational Decision Manager on Cloud:
 - a. Launch the Rule Execution Server console in your Operational Decision Manager on Cloud environment: development, test, or production.
 - b. In the **Explorer** tab, click the RuleApp that you want to download.
 - c. Click **Download Archive** in the RuleApp view.
 - d. Select the directory where you want to store the RuleApp archive.
2. Download the XOM:
 - a. Back in the **Explorer** tab of the Rule Execution Server console, under **Resources**, click the XOM that you want to download.
 - b. Click **Download Resource** in the Resource view.
 - c. Select the directory where you want to store the XOM archive.
3. Upload the XOM:
 - a. In the Resources view, click **Deploy Resource**.
 - b. Browse to the local path of the XOM archive, and then click **Deploy**. The managed XOM is deployed under **Resources** in the Rule Execution Server console. If the XOM is referenced from a library, you must re-create the library first (**Navigators > Libraries > Add Library**). Then, in the Library View of the newly created library, add the XOM as the internal resource reference (**Library View > Add Internal Resource Reference**).
 - c. Select the deployed XOM in the list of internal resources, or the library that contains the XOM in the list of libraries, and then click **Add**.
4. Upload the RuleApp to the target Decision Server:
 - a. Launch the Rule Execution Server console of the target Decision Server.
 - b. In the RuleApps view, click **Deploy RuleApp Archive**.
 - c. Browse to the local path of the RuleApp archive, and click **Deploy**. The RuleApp archive is deployed under **RuleApps** in the Rule Execution Server console.
5. To verify the deployment of the decision service to the target Decision Server, see [Testing a ruleset for REST execution](#)

Parent topic: [Deploying to a hybrid cloud environment](#)

Running rules

You run sets of rules, or rulesets, from decision services in Rule Execution Server to validate them and use them with calling applications.

[Managing rule execution in Rule Execution Server](#)

The following topics cover the features in Rule Execution Server. You can open certain topics from specific points in the Eclipse interface.

[Rule Execution Server console online help](#)

The Rule Execution Server console is a web-based graphical interface to access most of Rule Execution Server features.

Managing rule execution in Rule Execution Server

Managing rule execution involves tasks such as creating and deploying RuleApps, developing client applications, debugging, installing a production environment, and administering the production environment.

[Introducing Rule Execution Server](#)

Rule Execution Server architecture is based on a number of independent, but cooperating, software modules.

[Rule Execution Server components](#)

Rule Execution Server is an environment for executing rules. It provides management, performance, security, and logging capabilities.

[Improving the performance of Rule Execution Server](#)

You can improve the performance of Rule Execution Server.

[Rule Execution Server administration artifacts](#)

System administrators use the web-based Rule Execution Server console to manage and monitor RuleApps, ruleset execution, and decision services.

[Rule artifact management and deployment](#)

Rule Execution Server provides a comprehensive set of tools to manage the deployment and execution of rulesets.

Introducing Rule Execution Server

Rule Execution Server architecture is based on a number of independent, but cooperating, software modules.

Architects, administrators, developers, and testers work together to create, deploy, monitor, and administer decision services that run on Rule Execution Server instances in a development, test, or production environment.

The following steps provide the ordered flow of tasks to execute a ruleset using Rule Execution Server:

1. **Create:** Design and create a RuleApp.

A RuleApp is a deployment and management unit for Rule Execution Server. A RuleApp contains one or more rulesets. You deploy your RuleApps to Rule Execution Server in order to make the ruleset available to a client application. RuleApps can be deployed from a decision service deployment configuration.

Rule Designer does not explicitly show RuleApps. You use the decision service deployment configuration to hold information to make a RuleApp.

2. **Develop:** Design and create a client application.

To call a deployed ruleset that is contained in a RuleApp on Rule Execution Server, you must create a client application that calls the decision service through the chosen HTDS endpoint (SOAP or REST).

3. **Deploy to the test environment:** From a decision service that is ready for deployment, you can deploy one or more RuleApps. You deploy the rulesets that you defined in a decision operation to the Rule Execution Server instance that you defined in a deployment configuration.
4. **Test:** Run your client application and correct any errors that you find.

Before you deploy your decision service to a production server, you must ensure, by testing with sufficient data, that the client application and the decision service are stable and that the results are correct.

5. **Deploy to the production environment:** You deploy your validated decision services to the production environment.

Parent topic: [Managing rule execution in Rule Execution Server](#)

Rule Execution Server components

Rule Execution Server is an environment for executing rules. It provides management, performance, security, and logging capabilities.

With Rule Execution Server, you can change the business logic dynamically.

Note: For details of software requirements, see the [IBM® Operational Decision Manager on Cloud - Detailed System Requirements](#) page.

Rule Execution Server is a set of components that interact with the rule engine. The following table lists these components.

Table 1. Rule Execution Server components

Component	Features	For more information
Management console	<p>The management console provides the following services:</p> <ul style="list-style-type: none">• 24x7 execution with hot ruleset deployment. Hot deployment is the ability to publish changed applications to a running server without having to restart the server.• Web-based management of RuleApps• Views of runtime statistics• Views of runtime errors and warnings• Run server diagnostics• Lists of the transparent decision services that are linked to a ruleset, at the level of that ruleset. You can use the console to activate and deactivate a transparent decision service, or to view execution statistics.	<p>See the Rule Execution Server console online help.</p>
Transparent decision services	<p>Hosted transparent decision services are installed on the same application server as Rule Execution Server, then integrated with Rule Execution Server</p>	<p>See the following topics:</p> <ul style="list-style-type: none">• Hosted transparent decision services reference

Parent topic: [Managing rule execution in Rule Execution Server](#)

Improving the performance of Rule Execution Server

You can improve the performance of Rule Execution Server.

You improve the performance by designing and creating efficient rulesets. You can create efficient rulesets when you develop a business rule application. The goal of performance tuning is to decrease the amount of time and resources that your application server requires to process requests. Performance tuning allows your application server to complete more tasks in less time.

Improving performance during development

During the development phase of your application, you can enhance the performance of ruleset execution by creating efficient artifacts and choosing the execution mode for your artifacts.

The following table describes how to optimize the performance by creating more efficient artifacts.

Development artifact	Recommendation
Executable object model (XOM)	The size of the Java™ XOM deployed in your EAR affects ruleset parsing and execution. To improve performance, design the class loader to be as small as possible. Include only the JAR files that hold classes that are directly called by the XOM in the managed XOM persistence.
Ruleset size	<p>The size of a ruleset affects the total execution time.</p> <p>The main cause of resource usage when generating a RuleApp archive is the number and size of rule artifacts that are deployed as part of the RuleApp. Business rule applications work better if you limit the size of the resources that are deployed to the execution environment.</p>
Ruleset resources	<p>To reduce the resource usage cost of a business rule application:</p> <ul style="list-style-type: none">• Transform each large rule project into several smaller ones.• Generate multiple smaller rulesets from each rule project.• Use ruleset extractors, one per ruleset, to extract part of the rule project. In this case, you might have to change the application that executes the rulesets.
Rule flow	<p>Design the rule flow with mutually exclusive rule tasks to reduce the number of rules evaluated during the execution. In this way, the ruleset is segmented and only the rules from the relevant rule tasks are evaluated.</p> <p>This tip applies to rulesets that are built with the classic rule engine only.</p>
Threads	<p>The Java XOM must be thread-safe. The Java XOM must implement the Serializable API if you call the Rule Execution Server remotely.</p> <p>The toString method performance can have a huge influence.</p>
XML XOM	Configure rulesets that use an XML XOM to run in multiple simultaneous executions. Set the ruleset.xmlDocumentDriverPool.maxSize ruleset property to configure the execution pool of the XML document drivers. The default value is 1.
Rule flow	Keep rule flows simple. Complex rule flows reduce performance.
Rule tasks	Avoid dynamic filters in rule tasks. Use a fixed list of rules where possible.
Decision tables	Verify the size of the decision table in the Rule Designer technical view. Dividing a decision table into several smaller tables can reduce the size of the ruleset significantly.

For more information about the artifacts that can affect performance, see [Rule Execution Server administration](#)

[artifacts](#).

You can optimize the performance in specific circumstances by choosing an appropriate execution mode. For more information about execution modes, see [Choosing an execution mode](#).

Improving performance at run time

After you develop the application and deploy to an application server, a number of configuration steps can affect performance.

C on fi g ur at io n of	Recommendation
Gar b ag e col lec tor	On the IBM® JVM, the gencon garbage collector policy has good results on small and medium applications. Tune the garbage collector and memory size.
Rul es et up da te	Use asynchronous parsing to improve response times during ruleset updates. With asynchronous parsing, the parsing is done by a separate thread, which reduces the effect on concurrent ruleset execution threads. To enable this option, set the asynchronousRulesetParsing property to true in the ra.xml file. Ruleset parsing applies to rulesets that are built with the classic rule engine only. Rulesets that are built with the decision engine require the loading of Java classes only. The rules are already compiled to executable code before deployment.
Tra ce s	Execution without traces is the optimal mode in terms of memory usage and performance. For best performance, turn off the traces. When you choose to run the engine with a trace, the following side effects might impair performance: <ul style="list-style-type: none">• Memory usage increases as the execution trace is generated, the list of rules and tasks is cached, and the rule events are generated.• The response of the execution takes longer when the execution trace is integrated. The size of the execution trace depends on the number of rules and tasks in the ruleset.
Jav a 2 Se cu rit y	Enabling Java 2 Security decreases performance. You can tune your security configuration to minimize this effect. You can integrate Java 2 Security to the secure mode of the application server, but it is not mandatory. If Java 2 Security is enabled and an application is not prepared for it, the application is likely to run incorrectly and cause Java 2 Security access control exceptions at run time. For best performance, turn off Java 2 Security, especially on the Java Platform, Enterprise Edition.

Parent topic: [Managing rule execution in Rule Execution Server](#)

Rule Execution Server administration artifacts

System administrators use the web-based Rule Execution Server console to manage and monitor RuleApps, ruleset execution, and decision services.

System administrators use the Rule Execution Server console for the following tasks:

- To determine what rulesets are deployed
- To make real-time changes to business rules: typically, they can deploy, change, and manage business rules while the application is running.

The Rule Execution Server database stores all the changes that are made to a ruleset, such as information about the user who modified it, time of modification, and any comments that were added.

RuleApps

A RuleApp is a deployable management unit that contains rulesets. RuleApps keep records of the following details:

- RuleApp name
- RuleApp version
- Number of rulesets in the RuleApp
- RuleApp creation date

RuleApp archives

A RuleApp archive is an archive that stores RuleApps in a file.

Attention: RuleApp archives are saved in a strict directory structure: if you change that directory structure, you invalidate the RuleApp.

You search for a resource in a path that begins with the parent element in the descriptor:

- The path of a RuleApp is defined by the name and the version number of the RuleApp. No resources are associated with a RuleApp.
- The path of a ruleset is defined by the name and the version number of the RuleApp, followed by the name and the version number of the ruleset. This hierarchy is called the canonical ruleset path.

The following ruleset paths are all valid. The last example shows a canonical ruleset path.

- /rule8_8app/6_ruleset_6/1.0
- /rule8_8app/1.0/6_ruleset_6
- /rule8_8app/6_ruleset_6
- /rule8_8app/1.0/6_ruleset_6/1.0

Rulesets

You can add a ruleset to a RuleApp in one of the following ways:

- From the RuleApp View of the Rule Execution Server console in the cloud development environment.
- By deploying the ruleset within a RuleApp file.

A ruleset is deployed to Rule Execution Server along with a ruleset archive.

RuleApp archives contain largely compiled rules:

- Java™ bytecode when you compile from Rule Designer and select the **Optimize ruleset loading (Java bytecode generation)** option upon ruleset archive creation.
- Rules and ruleflows in an intermediate representation when you clear the **Optimize ruleset loading (Java bytecode generation)** option or when you generate rulesets from Decision Center.

Java XOM resources and libraries

When you set up Java XOM management, the deployment of Java XOM to Rule Execution Server generates Java XOM resources and libraries.

Parent topic: [Managing rule execution in Rule Execution Server](#)

Related information:

[Improving the performance of Rule Execution Server](#)

Rule artifact management and deployment

Rule Execution Server provides a comprehensive set of tools to manage the deployment and execution of rulesets.

The Rule Execution Server console is a management tool designed for system administrators, developers, or business analysts to implement changes in business requirements quickly and easily. Such changes can be uploaded to the rule engine while the application is running.

Management

Rule Execution Server provides an environment for you to manage the deployment and execution of your rules. Rule Execution Server includes a set of management facilities. System administrators can use them to change the behavior of the execution stack and disable rulesets in emergency situations.

Note:
All managed entities are versioned.

The following table shows what management tasks are available in what environment.

Table 1. Management tasks

Entity Operations	Edited by
RuleApps	
Create	<ul style="list-style-type: none">Rule Execution Server consoleRule Designer in the development environment only.Decision Center
Modify	<ul style="list-style-type: none">Rule Execution Server consoleDecision Center
Remove	<ul style="list-style-type: none">Rule Execution Server console
Get RuleApp Archive	<ul style="list-style-type: none">Rule Execution Server console
Rulesets	
Create	<ul style="list-style-type: none">Rule Execution Server consoleRule Designer in the development environment only.Decision Center
Modify (change ruleset archives and properties)	<ul style="list-style-type: none">Rule Execution Server consoleDecision Center
Enable/Disable	<ul style="list-style-type: none">Rule Execution Server consoleDecision Center
Remove	<ul style="list-style-type: none">Rule Execution Server console
Server	
Archive all RuleApps	<ul style="list-style-type: none">Rule Execution Server console
XOM resources (resuri)	
Create	<ul style="list-style-type: none">Rule Execution Server consoleDecision Center
Remove	<ul style="list-style-type: none">Rule Execution Server console
Get resource	<ul style="list-style-type: none">Rule Execution Server console
XOM resources (reslib)	
Create	<ul style="list-style-type: none">Rule Execution Server consoleDecision Center
Modify	<ul style="list-style-type: none">Rule Execution Server console
Remove	<ul style="list-style-type: none">Rule Execution Server console
Get library	<ul style="list-style-type: none">Rule Execution Server console

Parent topic: [Managing rule execution in Rule Execution Server](#)

Rule Execution Server console online help

The Rule Execution Server console is a web-based graphical interface to access Rule Execution Server features.

Use the Rule Execution Server console to manage and monitor deployed rulesets and decision services.

[Introducing the Rule Execution Server console](#)

The Rule Execution Server console provides a web-based graphical interface that you use for managing and monitoring RuleApps, ruleset archives, Java™ XOMs and libraries, and transparent decision services.

[Managing RuleApps](#)

After you have deployed a RuleApp to Rule Execution Server, you can manage it from the Rule Execution Server console.

[Managing rulesets](#)

You manage rulesets from the Rule Execution Server console.

[Managing Java XOM resources and libraries](#)

If you deploy a Java XOM to Rule Execution Server as JAR or ZIP resources, you can manage these resources through the Resources View of the Rule Execution Server console. You can deploy an ordered list of Java XOMs to Rule Execution Server as a library. When you do so, you can manage such libraries through the Libraries View of the Rule Execution Server console.

[Monitoring ruleset execution](#)

Using the monitoring capabilities of the Rule Execution Server console, you can enable the debug mode, enable ruleset monitoring, generate ruleset execution statistics, test ruleset execution, and view execution-related events in the XU log.

[Viewing or downloading an HTDS description file](#)

You can view or download the description file of a hosted transparent decision service to Web Service Description Language (WSDL), Web Application Description Language (WADL), or OpenAPI format.

Introducing the Rule Execution Server console

The Rule Execution Server console provides a web-based graphical interface that you use for managing and monitoring RuleApps, ruleset archives, Java™ XOMs and libraries, and transparent decision services.

You can also use the console to test ruleset execution and manage hosted transparent decision services.

User interface components

The Rule Execution Server console includes different user interface components to accomplish different tasks.

Bookmarking a view

To bookmark a RuleApp or ruleset view, you use the console permanent link feature and the bookmarking options of your browser.

Parent topic: [Rule Execution Server console online help](#)

User interface components

The Rule Execution Server console includes different user interface components to accomplish different tasks.

Top banner

From the top banner, you access the various features of the Rule Execution Server console.

Menu bar

Each view that you can display in the Rule Execution Server console has a menu bar with specific command buttons.

Tables and filters

In the Rule Execution Server console, you can sort the tables by column headers and you can filter their contents to view only certain artifacts.

Symbols

Icons are used throughout Rule Execution Server console to represent execution artifacts.

Parent topic: [Introducing the Rule Execution Server console](#)

Related tasks:

[Viewing debug traces for an execution unit \(XU\)](#)

[Testing ruleset execution](#)

Top banner

From the top banner, you access the various features of the Rule Execution Server console.

After you sign in to the Rule Execution Server console, its Home page displays a top banner with several tabs and links. Use these tabs and links to access the features of the console for the following purposes:

- Know what artifacts are deployed on Rule Execution Server and when they were deployed.
- Work with deployed artifacts: RuleApps, Java™ XOM resources, and Java XOM libraries.
- Monitor rulesets and execution units (XU).
- Enable or disable deployed resources.
- Read documentation about the Rule Execution Server console.

The top banner gives access to the following tabs and windows:

[The Home tab](#)

Displays a brief description of each of the other tabs.

[The Navigator panel of the Explorer tab](#)

Gives access to the RuleApps, XOM resources, XOM libraries, and Service Information.



Shows your login name.

Sign Out

Logs out the user from the current Operational Decision Manager on Cloud session.

Print View

Displays a plain view of the current page for printing. The link toggles to **Normal View**.

Help

Displays the documentation of the Rule Execution Server console in a separate window.

The Home tab

The Home tab is displayed each time you sign in to the Rule Execution Server console. The Home page is a welcome page that provides another way to access the other tabs and a brief description of each.

The Navigator panel of the Explorer tab

The **Explorer** tab gives access to views of the following execution artifacts:

- Deployed RuleApp and rulesets archives
- Java XOM JAR and ZIP resources
- Java XOM libraries

The Navigator panel is a side panel in the Explorer page. You can hide the Navigator panel by clicking the arrow handle. Click the arrow handle again to redisplay the Navigator panel.

Use the breadcrumbs that are provided in the Explorer pages to locate where you are and where you come from. The breadcrumbs are also links that you can click to move back along the same path.

Use the expandable tree to work with the artifacts.

RuleApps

Click **RuleApps** to display a RuleApps View and, from it, access a RuleApp View on a deployed RuleApp and a Ruleset View on a ruleset. For more information, see [Managing RuleApps](#) and [Managing rulesets](#).

Resources

Click **Resources** to display a Resources View and, from it, access a Resource View on a deployed Java XOM resource. For more information, see [Managing Java XOM resources and libraries](#).

Libraries

Click **Libraries** to display a Libraries View and, from it, access a Library View on a Java XOM library. For more information, see [Managing Java XOM resources and libraries](#).

Parent topic: [User interface components](#)

Menu bar

Each view that you can display in the Rule Execution Server console has a menu bar with specific command buttons.

At the top of each view in the Rule Execution Server console, a menu bar provides commands that are specific to that view. The menu commands available also depend on user rights.

Table 1. Explorer - RuleApps - Menu bar commands

Page and View	Command	Action	Comment
RuleApps View	Add RuleApp	Creates an empty RuleApp to which you can add rulesets later.	Available in the development environment only, not in test or production.
	Deploy RuleApp Archive	Imports the RuleApp archive that you select and deploys it using one of the version policies.	Available in the development environment only, not in test or production.
	Update RuleApps	Refreshes the RuleApps view to include the RuleApps that have just been deployed on the Rule Execution Server. The execution unit (XU) is updated.	
RuleApp View	Add Ruleset	Creates a ruleset within the selected RuleApp by importing a ruleset archive.	Available in the development environment only, not in test or production.
	Add Property	Adds a predefined or custom RuleApp property.	Available in the development environment only, not in test or production.
	Download Archive	Saves the RuleApp as a RuleApp archive.	
	Edit	Directly edits the display name and description of the RuleApp, and properties and included rulesets.	

Table 2. Explorer - Rulesets - Menu bar commands

Page and View	Command	Action	Comment
Ruleset View	View Statistics	Displays ruleset execution statistics, including all the visible execution units.	
	View Execution Units	Displays all execution units. Error and warning messages are attached to the selected ruleset.	
	Upload Ruleset Archive	Overrides the ruleset with an imported ruleset archive.	Available in the development environment only, not in test or production.
	Add Managed URI	Adds a reference to the URI of a managed Java™ XOM resource or library.	Available in the development environment only, not in test or production.
	Add Property	Adds a predefined or custom ruleset property.	Available in the development environment only, not in test or production.
	Edit	Edits the status and properties of the ruleset.	
	Retrieve WSDL	Displays or downloads the WSDL or WSDL code for a hosted transport	

	RTDS Description n File	WADL code for a hosted transparent decision service.	
Ruleset Statistics View	Refresh	Adds the latest execution in the statistics.	
	Reset	Resets execution statistics to null for the selected ruleset.	
Ruleset Execution Units	Refresh	Refreshes the list of execution units for the selected ruleset.	
	Reset Execution Unit Messages	Resets the messages that pertain to the selected ruleset for all visible execution units.	

Table 3. Explorer - Resources - Menu bar commands

Page and View	Command	Action	Comment
Resources View	Deploy resource	Deploys the selected resource to the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	Clean up resources	Displays the list of invalid and unused resources.	
Resource View	Remove	Removes the selected resource from the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	Download Resource	Downloads the selected resource as a JAR or ZIP file. The location depends on your browser settings.	
Explorer - Libraries			
Libraries View	Add Library	Adds a library to the libraries archive.	Available in the development environment only, not in test or production.
	Clean up Libraries	Displays the list of invalid and unused libraries.	
Library View	Remove	Removes the selected library from the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	Add Internal Resource Reference	Adds to this library a reference to a JAR or ZIP file that is already deployed to the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	Add Library Reference	Adds a reference to another library.	Available in the development environment only, not in test or production.

Parent topic: [User interface components](#)

Tables and filters

In the Rule Execution Server console, you can sort the tables by column headers and you can filter their contents to view only certain artifacts.

Tables

In the various views, execution artifacts such as RuleApps, rulesets, and Java™ XOM resources and libraries are listed in tables, with one row for each artifact and columns for meaningful information such as name, version, ruleset path, URIs. You can sort the contents of these tables by clicking any column header. For example, in a RuleApp view, you can order the listed rulesets on their ruleset path by clicking the **Ruleset Path** column header.

Filters

The Rule Execution Server console provides filters to limit what content is displayed inside a table. The filters are displayed at the top of each table.

The following filters are available:

View

The **View** filter is associated with the following check boxes, which you can select in combination:

- Select **Latest version** if you want to display only the latest version of each ruleset.
- Select **Enabled** if you want to display only the rulesets for which the **Enable** option is selected. This check box is available only in RuleApp Views.
- Select **Debug** if you want to display only the rulesets that have the debug mode enabled. This check box is available only in RuleApp Views.

Name

The **Name** field is not case-sensitive. It uses the [startsWith](#) method if you type directly into the field and also supports regular expressions, as documented on the [Class Pattern](#) page of the Java API documentation. For example, enter `.*xyz.*` in this field if you want to list only names that contain "xyz".
















Display by:

To select the number of artifacts to be displayed in the table, select **5**, **10**, **50**, or **100** from the drop-down menu.

Parent topic: [User interface components](#)

Symbols

Icons are used throughout Rule Execution Server console to represent execution artifacts.

Symbol	Represents
	In a RuleApps View, several RuleApps In a Resources View, several Java™ XOM resources In a Libraries View, several Java XOM libraries
	In a RuleApps View, a RuleApp In a Resources View, a Java XOM resource In a Libraries View, a Java XOM library
	In a RuleApp View, gives access to the RuleApp properties table In a Resource View or Library View, gives access to the tables of references
	In a RuleApp View, a ruleset In a Resource View or Library View, a ruleset reference
	In a Ruleset View, ruleset properties
	In a Ruleset View, ruleset input parameters
	In a Ruleset View, ruleset output parameters
	In a Ruleset View, introduces the table of ruleset parameters
	In a Ruleset Statistics View, ruleset statistics
	In a Ruleset View, ruleset archive elements
	In a Ruleset View, execution unit (XU)
	In a Ruleset View, execution unit (XU) messages on a ruleset
	The resource or library is valid and used
	The resource or library is valid but not used
	The resource or library is not valid

Parent topic: [User interface components](#)

Bookmarking a view

To bookmark a RuleApp or ruleset view, you use the console permanent link feature and the bookmarking options of your browser.

About this task

Most web browsers support bookmarking to help you keep track of important web pages. In the Rule Execution Server console, pages do not ordinarily display individual URLs. The permanent link feature provides a way to navigate directly to a specified page.

Procedure

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator pane.
3. Click the name of the relevant RuleApp.

The RuleApp View is displayed. You now have a full description of the RuleApp, including version, creation date, properties, and Rulesets.

4. Do one of the following steps:
 - For a RuleApp view, click **Permanent link** in the lower corner of the RuleApp description box.
 - For a ruleset view, click the relevant ruleset. In the ruleset view, click **Permanent link** in the lower corner of the ruleset description box.

The full URL is shown in the address bar of your browser.

5. Use the "bookmark" or "favorite" function of your browser to bookmark the page as you would any other web page.

Results

Tip:

One reason for not using bookmarks is that they might become out of date. Rule Execution Server console web pages are no different in this respect from other web pages because over time, users remove, archive, or version RuleApps and rulesets.

Parent topic: [Introducing the Rule Execution Server console](#)

Managing RuleApps

After you have deployed a RuleApp to Rule Execution Server, you can manage it from the Rule Execution Server console.

You can create, edit, and remove RuleApps and their properties. You can deploy and undeploy these RuleApps.

You can also archive a RuleApp to a JAR file. The RuleApp is not disabled or deleted from the server.

Archiving RuleApps

You can package a RuleApp into a JAR file as a RuleApp archive and download it.

Deploying RuleApp archives

You deploy a RuleApp archive from the RuleApps View. Managed Java™ XOMs can be included in the deployed RuleApp archive. You must choose versioning policies before you deploy the archive.

Adding RuleApps

You add new RuleApps from the RuleApps View. Use only valid characters for RuleApp names.

Setting RuleApp properties

You can add properties and rulesets to a deployed RuleApp, or remove existing properties or rulesets.

Versioning policy options

You use versioning options to control how RuleApp, ruleset, resource, and library versions are numbered and replaced.

Parent topic: [Rule Execution Server console online help](#)

Archiving RuleApps

You can package a RuleApp into a JAR file as a RuleApp archive and download it.

About this task

You can package and download a RuleApp archive that includes all rulesets or selected rulesets in the RuleApp:

- Download a RuleApp archive with **all rulesets** in the RuleApp. This feature is available in the following views in the **Explore** tab:
 - RuleApps View
 - RuleApp View
- Download a RuleApp archive with **selected rulesets** in the RuleApp. This feature is available only in the RuleApp View page.

Procedure

1. Click the **Explore** tab
2. In the Navigator panel, click **RuleApps** to display the RuleApps View page.
3. Download a RuleApp archive. Choose one of the options in the following table.

Table 1. How to download a RuleApp archive

Include all rulesets in the RuleApp archive		Include selected rulesets in the RuleApp archive
From RuleApps View	From RuleApp View	From RuleApp View
a. Select a RuleApp in the RuleApps table, and click Download Archive with All Rulesets next to a green arrow in the row.	a. Click a RuleApp in the RuleApps table. The RuleApp View page is displayed with a list of rulesets. b. Click Download Archive with All Rulesets next to a green arrow.	a. Click a RuleApp in the RuleApps table. The RuleApp View page is displayed with a list of rulesets. b. Select the rulesets that you want to include from the table, and click Download .

4. Save the file. The file is downloaded in the location that you specified.

Results

The RuleApp archive is now created and downloaded in the specified location. The archive contains the RuleApp and its rulesets. The RuleApp itself is not changed or deleted from the server.

Parent topic: [Managing RuleApps](#)

Deploying RuleApp archives

You deploy a RuleApp archive from the RuleApps View. Managed Java™ XOMs can be included in the deployed RuleApp archive. You must choose versioning policies before you deploy the archive.

Before you begin

Restriction: This action is available in the development environment only.

Procedure

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator panel to display the RuleApps View.
3. Click **Deploy RuleApp Archive** in the RuleApps View.
4. Click **Browse** and select the RuleApp file (.jar) to be deployed.

You can select the RuleApp archive file from your local computer or from the file system.

5. In the Deploy RuleApp Archive page, verify that appropriate versioning policies are selected in the **RuleApp Versioning Policy** section.
 - a. Optional: If there are managed Java XOMs that are included in the RuleApp archive, verify that appropriate versioning policies are selected in the **Resource Versioning Policy** and **Library Versioning Policy** sections.

Note: The selected resource and library versioning policy options are ignored when there is no managed Java XOM in the RuleApp archive.

For more information about the versioning policy options, see [Versioning policy options](#)

6. Click **Deploy**.

Deployment Report is displayed in the RuleApps View. See the **Details** section in the report to check what was deployed.

7. Click **OK** to return to the RuleApps View.

Results

When you deploy a RuleApp archive, you create a RuleApp on Rule Execution Server. The RuleApp is then listed in the RuleApp navigator.

Parent topic: [Managing RuleApps](#)

Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

Related tasks:

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM resources](#)

[Removing referenced Java XOM libraries](#)

Adding RuleApps

You add new RuleApps from the RuleApps View. Use only valid characters for RuleApp names.

About this task

This action is available in the development environment only.

Procedure

To add a RuleApp:

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator panel to display the RuleApps View.
3. Click **Add RuleApp** in the RuleApps View.

The Add New RuleApp page is displayed.

4. Click in the **Name** field and type the name of the new RuleApp.

Use only a–z, A–Z, 0–9, and underscore (_). For the first character, 0–9 is not allowed.

Note:

If you set Rule Execution Server persistence to file, remember that Windows environments do not support case sensitivity.

5. Click the default version number and type the new version number, where the major version number is ≥ 1 and the minor version number is ≥ 0 .
6. Click **Add**.

Results

The new RuleApp is added to the RuleApps list.

When you add a RuleApp, an empty container is created. It has a name and version number. You can edit the container to add one or more rulesets and a set of entities that specify how the rulesets are executed.

Parent topic: [Managing RuleApps](#)

Setting RuleApp properties

You can add properties and rulesets to a deployed RuleApp, or remove existing properties or rulesets.

You can add properties and rulesets to a deployed RuleApp in the development environment only.

You can add predefined or custom properties to a RuleApp from the RuleApps View. A RuleApp property is a property that applies to individual RuleApps. You can set RuleApp predefined or custom properties in one of the following ways:

- Before rule deployment in Rule Designer.
- In the Rule Execution Server console after the rules are deployed to Rule Execution Server.

The following table lists the predefined RuleApp properties.

Table 1. RuleApp properties

Name	Description
<code>ruleapp.interceptor.classname</code>	The full name of the ruleset execution interceptor to be used
<code>ruleapp.interceptor.description</code>	A description of the ruleset execution interceptor class

Parent topic: [Managing RuleApps](#)

Versioning policy options

You use versioning options to control how RuleApp, ruleset, resource, and library versions are numbered and replaced.

RuleApp version numbering is different depending on whether the RuleApp in the selected archive file exists in the server memory:

- If the RuleApp does not exist, it is deployed to Rule Execution Server with the version number 1.0, irrespective of the selected deployment policy.
- If the RuleApp already exists, it is deployed according to the versioning policy.

Here are examples that show the results of each versioning option.

Increment RuleApp major version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /RuleApp/2.0 /ruleset/1.0</div>

Increment RuleApp minor version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /RuleApp/1.1 /ruleset/1.0</div>

Replace RuleApp version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>

Increment rulesets major version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/3.0</div>

Increment rulesets minor version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>

<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div> <div>/ruleset/1.1</div>
---	---	---

Replace rulesets versions

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>

Increment resources major version

The following example shows the result after deployment if you select this option.

Deployed resource	Resource to be deployed	Result after deployment
<div>/resource/1.0</div>	<div>/resource/1.0</div> <div>Note: The checksum is different from the one in the Deployed resource column.</div>	<div>/resource/1.0</div> <div>/resource/2.0</div>

Increment resources minor version

The following example shows the result after deployment if you select this option.

Deployed resource	Resource to be deployed	Result after deployment
<div>/resource/1.0</div>	<div>/resource/1.0</div> <div>Note: The checksum is different from the one in the Deployed resource column.</div>	<div>/resource/1.0</div> <div>/resource/1.1</div>

Increment libraries major version

The following example shows the result after deployment if you select this option.

Deployed library	Library to be deployed	Result after deployment
<div>/library/1.0</div>	<div>/library/1.0</div>	<div>/library/1.0</div> <div>/library/2.0</div>

Increment libraries minor version

The following example shows the result after deployment if you select this option.

Deployed library	Library to be deployed	Result after deployment
<div>/library/1.0</div>	<div>/library/1.0</div>	<div>/library/1.0</div> <div>/library/1.1</div>

--	--	--

Parent topic: [Managing RuleApps](#)

Managing rulesets

You manage rulesets from the Rule Execution Server console.

Managing rulesets can require you to enable, disable, or debug rulesets, add ruleset properties, set WSDL options, manipulate ruleset archives, and set references to Java™ XOM resources.

You disable a ruleset to prevent its execution. When you disable a ruleset, this ruleset is not taken into account by the rule engine at execution time.

You can use the remote debugging feature to have Rule Designer add breakpoints in the ruleset. The debug URL points to the remote machine where Rule Designer runs. This URL consists of the machine name and the selected debugger port.

You can add predefined or custom properties to a ruleset to specify execution information. A ruleset property is a property that applies to individual rulesets. You set predefined or custom properties on a ruleset to specify execution information, in one of the following ways:

- Before rule deployment in Rule Designer.
- In the Rule Execution Server console after you have deployed your rules to Rule Execution Server.

You can add ruleset properties in the development environment only.

You can add managed URIs to a ruleset and modify the order of precedence between managed URIs at run time. By doing so, you modify the index of the selected reference. The index of a managed URI determines its position in the `ruleset.managedxom.uris` ruleset property, hence its precedence order at run time. The managed URI with index 1 executes first. Modify the order of the URIs to update, test, or diagnose various execution behaviors.

Changing target namespaces

When you generate the WSDL of a SOAP hosted transparent decision service (HTDS) or the WADL of a REST transparent decision service, you can change the namespace of the transparent decision service to avoid naming conflicts when more than one description file is generated at the same location.

Testing a ruleset for REST execution

From the Rule Execution Server console, you test REST API requests for execution of a ruleset in XML or in JSON format.

Setting ruleset properties

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

Parent topic: [Rule Execution Server console online help](#)

Changing target namespaces

When you generate the WSDL of a SOAP hosted transparent decision service (HTDS) or the WADL of a REST transparent decision service, you can change the namespace of the transparent decision service to avoid naming conflicts when more than one description file is generated at the same location.

About this task

When more than one description file is generated at the same location, naming conflicts can arise because the parameters might not be the same for different transparent decision services. Such is the case, for example, when more than one transparent decision service are added to a business integration project in IBM® Integration Designer. To avoid such conflicts, the URI for transparent decision service requests and responses, and their parameters, includes the ruleset name. The following example the URL for a ruleset named PreTradeChecks.

```
<wsdl:definitions name="PreTradeChecksDecisionService"
targetNamespace="http://www.ibm.com/rules/decisionservice/Execution/PreTradeChecks"
...

```

The schema importation mechanism references schema components from other schema documents.

```
...
<xsd:import
namespace="http://www.ibm.com/rules/decisionservice/Execution/PreTradeChecks/p
aram"/
...

```




The names of decision requests and responses are prefixed with the RuleApp name. For example:

```
<xsd:element name="PreTradeChecksRequest"
...
xsd:element name="PreTradeChecksResponse"

```

The value of the target namespace must be an absolute URI. To ensure the uniqueness of the target namespace, you can change the definition of this attribute in the Ruleset Parameters section of the Ruleset View.

Procedure

1. In the Rule Execution Server console, click the **Explorer** tab, expand the RuleApp that you want, and select a ruleset to open the Ruleset View.
 2. At the bottom of the Ruleset View, click **Show HTDS Options**.
 3. To change the target namespace of a SOAP transparent decision service, proceed as follows:
 - a. Next to the **Target namespace (SOAP only)** URI, click the  **Edit** icon. In the option name, **SOAP only** means that changing the namespace makes sense only for WSDL code generation. When you generate the WADL code for a ruleset, you can change the target namespace for ruleset parameters.
 - b. Type the custom URL in the field.
 - c. Click the  **Save** icon.
- You can cancel by clicking the **Undo** arrow next to the **Save** icon.
4. Optional: For either a SOAP or a REST transparent decision service, if applicable, edit the URL for the ruleset parameters as follows:
 - a. Click **Parameter target namespace**.
 - b. Type the custom URL in the field.
 - c. Click the  **Save** icon.

You can cancel by clicking the **Undo** arrow next to the **Save** icon.

Parent topic: [Managing rulesets](#)

Testing a ruleset for REST execution

From the Rule Execution Server console, you test REST API requests for execution of a ruleset in XML or in JSON format.

Before you begin

Restriction:

- Browser compatibility: The test feature is not supported by Internet Explorer 8 and earlier versions. If you work with Internet Explorer 9 or 10, make sure that the **Tools > Compatibility View** command is deactivated.
- Generation of OpenAPI description files is supported for only Java™ XOMs, while WSDL and WADL file generation are supported for XML XOMs and Java XOMs with JAXB annotations.

About this task

Before you execute a ruleset in XML or in JSON form from a client, you can test your execution request in the Rule Execution Server console.

Procedure

1. Select a ruleset and open the Ruleset View.
2. Click **Retrieve HTDS description file**.
3. Select **REST** in the **Service protocol type** section.
4. Select **WADL**, **OpenAPI - YAML**, or **OpenAPI - JSON** from the **Format** pull-down menu as explained in [Viewing or downloading an HTDS description file](#).

Remember: You can select **OpenAPI - YAML** and **OpenAPI - JSON** only for Java XOMs.

5. Click **Test**.


The execution request is displayed in a separate browser window.

If you selected WADL in the previous step, you can choose **XML** or **JSON** for Java XOMs as an execution format from the **Execution Request** drop-down menu. For XML XOMs, you have XML as an execution format.

If you selected OpenAPI, you have only one execution format: JSON.

6. Click **Execute Request**.

The response is displayed in the Server Response frame. If you selected **Decision trace information** before you click **Test**, the trace filter is added to the request and the decision trace is included in the response.

7. Optional: If you selected XML as an execution format, click the  **Validate XML** icon to validate the generated XML code.

Note: The JSON format does not support validation.

If the request is invalid, an error message indicates the reason and location of the failure. A red dot is displayed next to the incorrect line in the request.

8. Optional: Use the **Undo** and **Redo** icons to erase or restore what you type in the text areas.

Parent topic: [Managing rulesets](#)

Related tasks:

[Testing ruleset execution](#)

Setting ruleset properties

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

Ruleset properties apply to individual rulesets. You can set ruleset properties in one of the following ways:

- In Rule Designer before you deploy.
- In the Decision Center consoles before you deploy.
- In the Rule Execution Server console after deployment.
- Using the Rule Execution Server REST API.

Note: You can add or edit ruleset properties in the Rule Execution Server console only in the development environment.

To set values for the built-in ruleset properties in the Rule Execution Server console:

1. Click **Add Property** in the Ruleset View page.
2. In the New Ruleset Property page, select a ruleset property from the menu. Check **Predefined** in the **Property Type** section. Set a value for the selected ruleset property.

Tip:

- If the built-in ruleset property that you want to use is not listed in the menu when you select **Predefined**, select **Custom** instead and enter the name and the value of the property.
- You cannot rename ruleset properties. Instead, you must remove the property and create a new one.

[Built-in ruleset properties](#)

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

[Built-in ruleset properties for OpenAPI](#)

Some information in the OpenAPI description file can be overridden by specifying the ruleset properties for OpenAPI.

[Setting the ruleset property for increasing rule execution performance](#)

You can significantly increase rule execution performance by increasing the maximum size of the document driver pool.

Parent topic: [Managing rulesets](#)

Built-in ruleset properties

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

The following tables provide the list of built-in ruleset properties.

Table 1. Built-in ruleset properties

Name	Valid values	Default value	Description
ilog.rules.teamserver.permalink.report	A URL as a string		This property contains the URL to access the Decision Center report.
decisioncenter.url	A URL		The URL of the Decision Center server from which the deployment was done.
decisioncenter.buildnumber	A string		The number of the Decision Center build.
decisionservice.name	A string		The name of the decision service that was deployed, for example, loanvalidation-rules-service.
decisionservice.id	A string		The ID of the root project of the decision service that was deployed, for example, brm.RuleProject:44:44.
decisionservice.branch.name	A string		The name of the deployment snapshot if the user chose to create one, the branch, or the snapshot in the decision service that was deployed, for example, b or d-20150231-158621.
decisionservice.branch.id	A string		The ID of the deployment snapshot, branch, or snapshot in the decision service that

			was deployed, for example, brm.Branch :224:224.
decisionservice.branch.url	A URL		A link to a page in the Decision Center Business console that displays the deployed branch or snapshot.
decisionservice.deploymentConfiguration.name	A string		The name of the deployment configuration that was deployed, for example, d.
decisionservice.deploymentConfiguration.id	A string		The ID of the deployment configuration that was deployed, for example, dsm.Deployment:3:3.
decisionservice.deployer.name	A string		The display name of the Decision Center user who did the deployment.
decisionservice.deployer.id	A string		The login name of the Decision Center user who did the deployment.
ruleset.engine.version	A string		The MMU of the rule engine. for example, 1.20.0.
ilog.console.wsdl.endpoint	A URL as a string		Use this property to override the default HTDS option for the web service endpoint.
ilog.console.htds.context	A URL as a string		Use this property to override the default HTDS option for the location.
wsdl.targetnamespace	A URL as a string		Use this property to override the default HTDS option for the target namespace.
wsdl.paramtargetnamespace	A URL as a string		Use this property to override the default HTDS options for the parameter

			<p>target namespace.</p> <div><p>Note: This parameter can be used for both WSDL and WADL code generation , although wsdl is used as part of the parameter name. You cannot change wsdl to wadl in the parameter name even when this parameter is used for the WADL code generation .</p></div>
ruleset.managedxom.uris	<p>A comma-separated list of URIs. For example:</p> <p>resuri://common-classes.jar/1.0,resuri://LoanValidation.jar</p>	No default value	<p>This ruleset property controls Java™ XOM management. It locates the Java XOM resources for the ruleset.</p> <ul style="list-style-type: none">• If you do not set this property and the ruleset uses a Java XOM, the Java classes are loaded by the application class loader.• If you set this property (as a list of URIs) for a ruleset , the execution unit (XU)

			<div>creates and stores a dedicated class loader for the execution of the ruleset. Each URI must target a Java Archive .jar file or a .zip archive of classes and resources for Java execution.</div> <div>Only internal URIs are supported: resuri and reslib protocol</div>
<code>ruleset.maxIdleTime</code>	<div>Three possible values:</div> <ul style="list-style-type: none">• -1 or undefined : the ruleset is removed from the cache, depending on its usage by the JCA Connection Pool.• =>0 : the ruleset is removed from the cache		<div>This property enforces the ruleset pool policy on a ruleset. A ruleset stays in memory until the maximum idle time (in seconds) has reached the specified value set for this property. To avoid ruleset reparsing, you can use the special value of 0 to ensure that the ruleset is never released from memory.</div> <div>Important: Use the 0 value with caution as it might introduce a significant memory leak if the</div>

	<div>after the time out (in seconds) is reached and no SPI connections reference it any more.</div> <ul style="list-style-type: none">• 0: the ruleset is never removed from the cache except if the ruleset is redeployed.		<div>ruleset is not used any more.</div>
<code>ruleset.trace.enabled</code>	true, false		This property enables or disables the rule engine trace mode.
<code>ruleset.xmlDocumentDriverPool.maxSize</code>	≥ 0	1	<div>The 0 value means that an <code>XMLDocumentDriver</code> instance is created for each <code>XMLObjectTransformation</code>.</div> <div>Use a strictly positive value to specify the maximum size of the <code>IlrXMLDocumentDriverPool</code>, that is,</div> <div>the maximum number of used and unused <code>IlrXMLDocumentDriver</code></div>

			objects per ruleset.
<code>ruleset.xmlDocumentDriverPool.reserveTimeout</code>	<code>>=0</code>		This property specifies the number of milliseconds after which the call to reserve an <code>IlrXMLDocumentDriver</code> instance times out.

Parent topic: [Setting ruleset properties](#)

Built-in ruleset properties for OpenAPI

Some information in the OpenAPI description file can be overridden by specifying the ruleset properties for OpenAPI.

Table 1. Built-in ruleset properties for OpenAPI

Name	Valid values	Default value	Description
openapi.info.title	A string		See the description in the Info Object section in the OpenAPI Specification.
openapi.info.description	A string		See the description in the Info Object section in the OpenAPI Specification.
openapi.execute.operation.summary	A string		See the description in the Operation Object section in the OpenAPI Specification.
openapi.execute.operation.description	A string		See the description in the Operation Object section in the OpenAPI Specification.
openapi.execute.operation.operationId	A string		See the description in the Operation Object section in the OpenAPI Specification.

Parent topic: [Setting ruleset properties](#)

Setting the ruleset property for increasing rule execution performance

You can significantly increase rule execution performance by increasing the maximum size of the document driver pool.

About this task

By default, the **ruleset.xmlDocumentDriverPool.maxSize** ruleset property value is set to 1. This value might cause a bottleneck if multiple clients execute a hosted transparent decision service concurrently. By increasing the value of this property, you can significantly increase the performance of ruleset execution.

Set the value of the ruleset property in the Ruleset View of the Rule Execution Server console.

Procedure

1. Log in to the Rule Execution Server console.
2. Click the **Explorer** tab.
3. Click **RuleApps** in the Navigator panel to display the RuleApps View.
4. In the RuleApps View, click the name of the RuleApp that contains the ruleset.
5. In the RuleApp View, click the relevant ruleset.
6. In the Ruleset View, click **Add Property**.
7. In the New Ruleset Property form, select the predefined property **ruleset.xmlDocumentDriverPool.maxSize** in the drop-down list.
8. Set the required value.

The optimal value depends on the number of concurrent executions of the ruleset.

For example, for five concurrent clients, 5 can be a good value.

9. Click **Add**.

Parent topic: [Setting ruleset properties](#)

Managing Java XOM resources and libraries

If you deploy a Java™ XOM to Rule Execution Server as JAR or ZIP resources, you can manage these resources through the Resources View of the Rule Execution Server console. You can deploy an ordered list of Java XOMs to Rule Execution Server as a library. When you do so, you can manage such libraries through the Libraries View of the Rule Execution Server console.

Managed Java XOM resources (JAR) and libraries can be included in a deployed RuleApp archive. These resources and libraries are called embedded managed Java XOMs.

Java XOM resources

After you have deployed a Java XOM as JAR or ZIP resources, you can view these resources in the Rule Execution Server console.

Java XOM libraries

After you have deployed an ordered list of Java™ XOMs as a library, you view this library in the Libraries View of the Rule Execution Server console.

Embedded managed Java XOM in Rule Execution Server

Embedded managed Java XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can deploy RuleApp archives with embedded managed Java XOMs in the Rule Execution Server console. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

Removing referenced Java XOM resources and libraries

You want to remove a Java XOM resource or library at one point, but it might be referenced from rulesets or libraries. Before you remove a resource or library that is referenced, you must remove all links that point from these rulesets and libraries to the target resource or library. The library that you want to remove might be referencing resources or other libraries as well. Before you remove a library, all links that point from the library to these resources or libraries are removed as well.

Parent topic: [Rule Execution Server console online help](#)

Java XOM resources

After you have deployed a Java™ XOM as JAR or ZIP resources, you can view these resources in the Rule Execution Server console.

From the Resource View, you can:

- Deploy new Java XOM JAR or ZIP resources and increment the major version or minor version of the resource.
- Remove a previously deployed JAR or ZIP resource. Execution units (XU) are notified of the removal, which is logged as a message in the console.

Attention: When the value of the **Checksum** section of a resource is empty, it means that the resource file is empty. Using the empty resource file can corrupt the database.

You can deploy and remove resources in the development environment only.

Use the **Show references** buttons to show or hide the list of rulesets and the list of libraries that use the resource. The `ruleset.managedxom.uri` ruleset property allocates the resources to the ruleset at run time. Click **Show references from rulesets** to display the list of the rulesets that, directly or indirectly, use the resource in the `ruleset.managedxom.uri` ruleset property.

You can view invalid and unused resources so that you can clean up the resources archive. The Resources Clean-up View contains information that can help you correct the invalid references or remove the invalid or unused resources. Resources are diagnosed as invalid based on the algorithm supported by the **res-diagnose-xom-uri** Ant task.

Each deployed JAR or ZIP resource is flagged with an icon that reflects its status.

Green

The icon is green  if the following conditions are all satisfied:


- The data from the URI can be read.
- The checksum that is stored in the persistence layer is the same as the checksum that is computed from that data.
- At least one ruleset uses the resource. A ruleset can use the resource transitively through a library that might depend on another library.

Yellow

The icon is yellow  if the following conditions are all satisfied:

- The data from the URI can be read.
- The checksum that is stored in the persistence layer is the same as the checksum that is computed from that data.
- The resource is never used in any ruleset, even transitively through interdependent libraries.

Red

The icon is red  in one of the following cases:

- The data from the URI cannot be read.
- The checksum that is stored in the persistence layer is not the same as the checksum that is computed from that data.

Parent topic: [Managing Java XOM resources and libraries](#)

Java XOM libraries

After you have deployed an ordered list of Java™ XOMs as a library, you view this library in the Libraries View of the Rule Execution Server console.

From the Libraries View, you can:

- Add a library to an archive of libraries.
- Add references to custom resources, to internal resources, or to other libraries to a library.
- Remove a library from an archive of libraries. When you remove a library, you erase the selected library from the managed Java™ XOM but do not delete the internal resources and libraries that the library was referencing. To remove an internal reference to a resource or library, you remove its URI manually in the value of the **ruleset.managedxom.uris** property.

You can add a library, add references to a library, and remove libraries in the development environment only.

Use the **Show references** buttons to show or hide the list of rulesets that use this library in its **ruleset.managedxom.uris** property, directly or indirectly, through one or several libraries.

You can view invalid and unused libraries and then clean up the libraries archive. The Libraries Clean-up View contains information that can help you correct the invalid references or remove the invalid or unused libraries. Libraries are diagnosed as invalid based on the algorithm that the **res-diagnose-xom-uri** Ant task supports.

Each deployed library is flagged with an icon that reflects its status.

Green

The icon is green  if the following conditions are satisfied:


- All the referenced URIs are valid.
- At least one ruleset uses the library or is used by a ruleset-referenced library. Any action to remove this library leads to an inconsistent state.

Yellow

The icon is yellow  if the following conditions are satisfied:

- All referenced URIs are valid.
- No ruleset uses the library.

Red

The icon is red  if at least one referenced URI is not valid.



Parent topic: [Managing Java XOM resources and libraries](#)

Embedded managed Java XOM in Rule Execution Server

Embedded managed Java XOMs are managed Java™ XOM resources (.jar) and libraries that are included in a RuleApp archive. You can deploy RuleApp archives with embedded managed Java XOMs in the Rule Execution Server console. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

The embedded managed Java XOM feature can be operated from the Rule Execution Server console.

Table 1. Embedded Managed Java XOMs

Module or tool	Deploy RuleApp archive with embedded managed Java XOM	Create or retrieve RuleApp archive with embedded managed Java XOM	Description
Rule Execution Server console			For more information, see the following topics: <ul style="list-style-type: none">• Deploying RuleApp archives• Removing referenced Java XOM resources and libraries

Parent topic: [Managing Java XOM resources and libraries](#)

Removing referenced Java XOM resources and libraries

You want to remove a Java™ XOM resource or library at one point, but it might be referenced from rulesets or libraries. Before you remove a resource or library that is referenced, you must remove all links that point from these rulesets and libraries to the target resource or library. The library that you want to remove might be referencing resources or other libraries as well. Before you remove a library, all links that point from the library to these resources or libraries are removed as well.

About this task

Remove all links to/from the resource or library, and then remove the resource or library itself.

Note: If you do not remove the resource or library after removing these links, references might not be updated when you redeploy the RuleApp. If it occurs, you must update them manually.

Removing referenced Java XOM resources

Before you remove a Java XOM resource that is referenced from rulesets or libraries, you must first remove all links that point to the resource.

Removing referenced Java XOM libraries

Before you remove a Java XOM library that is referenced from rulesets or other libraries, you must first remove all links that point to the library. When the links pointing to the library are removed, links that are pointed from the library are removed at the same time.

Parent topic: [Managing Java XOM resources and libraries](#)

Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

Related tasks:

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources](#)

[Removing referenced Java XOM libraries](#)

Removing referenced Java XOM resources

Before you remove a Java™ XOM resource that is referenced from rulesets or libraries, you must first remove all links that point to the resource.

About this task

You want to remove a Java XOM resource at one point, but it might be referenced from rulesets or libraries. Before you remove a resource that is referenced from rulesets or libraries, you must remove the following links:

- Links pointing from rulesets to the resource that you want to remove
- Links pointing from libraries to the resource that you want to remove

Procedure

1. In the **Explorer** tab, open Resource View of the Java XOM resource that you want to remove.

See the following sections to check whether the resource is referenced from other artifacts:

- **Show References from Rulesets**
- **Show References from Libraries**

2. Click **Remove All References**.

A warning message is displayed.

Remember: Only the links that are pointing to the resource are deleted. The artifacts that reference the resource, such as rulesets and libraries, are not deleted.

3. Click **Confirm**.

Now the resource is not referenced from any ruleset or library. Check the sections that are listed in Step 1.

4. Click **Remove** to remove the resource.

Important: If you do not remove the resource, references might not be updated when you redeploy the RuleApp. You might need to update them manually.

Results

The resource is now removed cleanly.

You can remove multiple resources by using the **Remove All References** and **Remove** buttons respectively in Resources View.

Parent topic: [Removing referenced Java XOM resources and libraries](#)

Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

Related tasks:

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM libraries](#)

Removing referenced Java XOM libraries

Before you remove a Java™ XOM library that is referenced from rulesets or other libraries, you must first remove all links that point to the library. When the links pointing to the library are removed, links that are pointed from the library are removed at the same time.

About this task

You want to remove a Java XOM library at one point, but it might be referenced from rulesets or other libraries. Before you remove a library, the following links must be removed:

- Links pointing from rulesets to the library that you want to remove
- Links pointing from other libraries to the library that you want to remove

The library that you want to remove might be referencing other libraries or resources as well. In the Rule Execution Server console, the preceding links as well as the following links can be removed in one go:

- Links pointing from the library that you want to remove to Java XOM resources
- Links pointing from the library that you want to remove to other libraries

Therefore, the library can be removed cleanly without any link attached to it.

Procedure

1. In the **Explorer** tab, open Library View of the Java XOM library that you want to remove.

See the following sections to check whether the library is referenced from other artifacts, or if it references other artifacts:

- **Show Referenced Internal Resources**
- **Show Library References**
- **Show References from Rulesets**
- **Show References from Libraries**

2. Click **Remove All References**.

A warning message is displayed.

Remember: Only the links to the library are deleted. The artifacts that reference the library, such as rulesets and libraries, are not deleted. The artifacts that are referenced from the library are not deleted, either.

3. Click **Confirm**.

Now the library is not referenced from any ruleset or library, and it does not reference any resource or library. Check the sections that are listed in Step 1.

4. Click **Remove** to remove the library.

Important: If you do not remove the library, references might not be updated when you redeploy the RuleApp. You might need to update them manually.

Results

The library is now removed cleanly.

You can remove multiple libraries by using the **Remove All References** and **Remove** buttons respectively in Libraries View.

Parent topic: [Removing referenced Java XOM resources and libraries](#)

Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

Related tasks:

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM resources](#)

Monitoring ruleset execution

Using the monitoring capabilities of the Rule Execution Server console, you can enable the debug mode, enable ruleset monitoring, generate ruleset execution statistics, test ruleset execution, and view execution-related events in the XU log.

[Enabling and disabling the debug mode](#)

You can enable or disable the debug mode on a ruleset. When this mode is enabled, you can use Rule Designer to put breakpoints in rulesets by using the remote debugging feature.

[Generating ruleset statistics](#)

You can generate execution statistics for a particular ruleset from the Ruleset View.

[Ruleset statistics views](#)

Ruleset execution statistics are provided for each execution unit (XU) across the entire cluster.

[Viewing logged events on execution units \(XU\)](#)

You can view the execution events logged for each execution unit (XU) in the Rule Execution Server console. You can also modify how the events information is displayed.

Parent topic: [Rule Execution Server console online help](#)

Related tasks:

[Monitoring transparent decision services](#)

Enabling and disabling the debug mode

You can enable or disable the debug mode on a ruleset. When this mode is enabled, you can use Rule Designer to put breakpoints in rulesets by using the remote debugging feature.

About this task

The debug URL points to the remote server where Rule Designer runs. This URL consists of the server name and the chosen debugger port.

Procedure

To enable/disable the debug mode:

1. In the Navigator panel, click the relevant ruleset.
2. In the **Ruleset View**, click **Edit**.

The **Ruleset Properties** are now editable.

3. Select the check box next to the **Debug** property to enable the debug mode.
4. Click **Save** in the **Ruleset View**.

Results

When you enable the debug mode for a specific ruleset, that ruleset, when called, opens to a remote debugger.

Parent topic: [Monitoring ruleset execution](#)

Generating ruleset statistics

You can generate execution statistics for a particular ruleset from the Ruleset View.

About this task

Ruleset statistics provide information on ruleset execution such as the number of times a ruleset was executed and how long the execution took. When the ruleset is in debug mode, statistics are only on debug executions. If you disable the debug mode, ruleset statistics are reset. Debug execution statistics are not mixed with regular execution statistics.

Procedure

To generate statistics on the previous executions of a ruleset:

1. In the Navigator panel, click the relevant ruleset.
2. In the Ruleset View, click **View Statistics**.

Results

The Ruleset Statistics View is displayed.

Parent topic: [Monitoring ruleset execution](#)

Related information:

[Ruleset statistics views](#)

Ruleset statistics views

Ruleset execution statistics are provided for each execution unit (XU) across the entire cluster.

The Ruleset Statistics View displays a table of information about the execution of rulesets, both for each execution unit (XU) in the configuration and for the entire cluster.

The ruleset statistics table contains the following columns: Metric and Ruleset Execution.

The statistics table rows provide the following information. Durations are expressed in milliseconds.

Table 1. Ruleset statistics

Labels	Description
Count	The number of times the ruleset has been executed during this session.
Total time (ms)	The total time to execute the ruleset.
Average time (ms)	The average time to execute the ruleset. This figure is derived from the total execution time (Total time (ms)) and the number of executions (Count).
Max / Min time (ms)	The longest and shortest ruleset execution times
First / Last Execution Date	The dates and times of the first and last ruleset executions.
Last Execution Time (ms)	The time of the last ruleset execution.

Parent topic: [Monitoring ruleset execution](#)

Viewing logged events on execution units (XU)

You can view the execution events logged for each execution unit (XU) in the Rule Execution Server console. You can also modify how the events information is displayed.

Procedure

To view logged events on execution units (XU):

1. Click the **Explorer** tab.
2. In the Navigator panel, click the relevant ruleset.
3. In the Ruleset View page, click **View Execution Units**.

The Execution Units page is displayed. The execution unit (XU) table provides the number of warnings and errors logged on each server.

4. Click the name of the server in the list of deployed execution units.

Results

You can display events information in different ways:

- To limit the displayed log, use the **Display by: 5, 10, 50, 100** filter.
- To sort the log messages by column in ascending order, click the column name: **Level, Creation Date, Message**. To sort in descending order, click the column name again.
- To update the model and the log, click **Refresh** at the top of the page.

Parent topic: [Monitoring ruleset execution](#)

Viewing or downloading an HTDS description file

You can view or download the description file of a hosted transparent decision service to Web Service Description Language (WSDL), Web Application Description Language (WADL), or OpenAPI format.

Before you begin

Restriction:

- Generation of OpenAPI description files is supported for only Java™ XOMs, while WSDL and WADL file generation are supported for XML XOMs and Java XOMs with JAXB annotations.
- You cannot use the **Retrieve HTDS Description File** feature for rulesets that are built with an incompatible engine version. In case of incompatibility, the Ruleset View provides more information.

About this task

When you present a ruleset as a SOAP hosted transparent decision service, the description file is generated in WSDL. When you create a transparent decision service to execute a ruleset through the REST API, the description file is generated in WADL or OpenAPI.

WSDL

To invoke a hosted transparent decision service, you can generate a WSDL file from a ruleset from the Rule Execution Server console. You can download the generated WSDL for the following purposes:

- Import the transparent decision service into any product or application that supports WSDL code, such as Rule Designer or IBM® Integration Designer with XSD files.
- Host the WSDL on another computer.

WADL

When you want to execute a ruleset by using the REST API, you can generate a description file in .wadl format for the ruleset from the Rule Execution Server console.

OpenAPI

Another way to execute a ruleset by using the REST API is to generate an OpenAPI definition file in .yaml or .json format for the ruleset from the Rule Execution Server console, and execute it through an OpenAPI tool or framework.

When you download a WSDL or WADL file, you can select the ruleset path, add a decision trace request, and download the XSD files separately.

When you download an OpenAPI file, you can select the ruleset path and add a decision trace request.

Procedure

1. In the Rule Execution Server console, click the **Explorer** tab, expand the RuleApp you want, and select a ruleset to open the Ruleset View.
2. At the bottom of the Ruleset View, click **Show HTDS Options**.
3. Optional: If applicable, edit the transparent decision service target name space as explained in [Changing target namespaces](#).
4. Click **Retrieve HTDS Description File**. The ruleset path is displayed above the options. For example: /execution/1.0/PreTradeChecks/1.0
 - To generate a WSDL file, keep the **SOAP** option selected. If you select no options for WSDL generation, the WSDL code is generated as a single .wsdl file from the ruleset that you selected in the RuleApp View, with dynamic namespaces and decision identifiers, and with no decision trace filters.
 - To generate a WADL file, select the **REST** option, and then choose **WADL** from the **Format** pull-down menu.
 - To generate an OpenAPI file, select the **REST** option, , and then choose **OpenAPI - YAML** or **OpenAPI - JSON** from the **Format** pull-down menu.
5. Optional: For either format, select the appropriate options.

Latest ruleset version

Select this check box if you want to generate the WSDL, WADL, or OpenAPI file from the latest ruleset of the RuleApp displayed above the options.

Latest RuleApp version

Select this check box if you want to generate the WSDL, WADL, or OpenAPI file from the latest ruleset in the latest version of the RuleApp.

Decision trace information

Select this check box if you want the WSDL, WADL, or OpenAPI file to include decision trace filter definitions and trace.

In the WSDL and WADL code, descriptions of the decision trace filters are documented in `<xsd:documentation>` elements.

For information about the decision trace filters, see [HTDS decision trace filters](#)

Important:

The order between filters is meaningful: when a filter conflicts with another filter, the filter in the higher position is taken into account.

Inline types in separate XSD files

Select this check box if you want to download the WSDL or WADL code as a .zip archive in which the .xsd files are saved separately from the .wsdl or .wadl file. This option is useful if you import more than one decision from Decision Server into an IBM Integration Designer project, or if you turn a rule flow into a process. The separation of the object types from the WSDL avoids the duplication of these objects in IBM Integration Designer.

Proxy for API Connect

Select this check box if you want to use the OpenAPI file in API Connect.

6. Click **View** or **Download**.

Results

The WSDL, WADL, or OpenAPI file is displayed in a separate browser tab or window and saved to the disk to your default download directory. If you select **Inline types as separate XSDs**, the **View** button is not available and you can only download the generated files.

[HTDS decision trace filters](#)

You can include decision trace filter definitions when you view or download the WSDL, WADL, or OpenAPI description file of a hosted transparent decision service in the Rule Execution Server console.

Parent topic: [Rule Execution Server console online help](#)

Related tasks:

[Changing target namespaces](#)

[Configuring a web service endpoint](#)

[Configuring the hosted transparent decision service location](#)

HTDS decision trace filters

You can include decision trace filter definitions when you view or download the WSDL, WADL, or OpenAPI description file of a hosted transparent decision service in the Rule Execution Server console.

Important:

The order between filters is meaningful: when a filter conflicts with another filter, the filter in the higher position is taken into account.

The following decision trace filter definitions are included when you select the **Decision trace information** option for viewing or downloading the WSDL or WADL description file for a selected ruleset in the Rule Execution Server console:

Table 1. Decision trace filter definitions that are included in the WSDL or WADL description file

Filter	Type	Default value	Description
all	boolean	false	Records everything if it is set to true.
executionDuration	boolean	false	Records the duration of the execution.
executionDate	boolean	false	Records the date of the execution.
rulestProperties	boolean	false	Records static data about the ruleset.
systemProperties	boolean	false	Records static data about the execution server.
inetAddress	boolean	false	Records static data about the internet address of the server.
totalRulesFired	boolean	false	Records the number of rules executed.
totalRulesNotFired	boolean	false	Records the number of rules that have not been executed.
rules	boolean	false	Records the complete list of rules in the executed ruleset.
rulesFired	boolean	false	Records any rule that has been executed at least once.
rulesNotFired	boolean	false	Records the rules that have not been executed.
totalTasksExecuted	boolean	false	Records the number of tasks executed.
totalTasksNotExecuted	boolean	false	Records the number of tasks that have not been executed.
tasks	boolean	false	Records the complete list of tasks in the executed ruleset.
tasksExecuted	boolean	false	Records any task that has been executed at least once.
tasksNotExecuted	boolean	false	Records the tasks that have not been executed.
outputString	boolean	false	Records the

			stream output or the engine execution.
inputParameters	boolean	false	Records ruleset input parameters.
outputParameters	boolean	false	Records ruleset output parameters.
workingMemory	boolean	false	Records the state of the working memory.
workingMemoryFilter	string		Holds the working memory filter.
executionEvents	boolean	false	Records the events that have been executed. This filter is a shortcut to select the tasksExecuted and rulesFired filters.
boundObjects	boolean	false	Records the list of matched objects.
boundObjectsSerializationType	string enum: <ul style="list-style-type: none">• JAXB• ToString• ClassNa me	ClassName	Choose the type of serialization for bound objects.

Parent topic: [Viewing or downloading an HTDS description file](#)

Testing decision services

You can check the results and performance of a decision service by testing it before you put it into production.

As you develop a decision service, you can run it through tests and simulations to ensure that its rules produce the right results. In creating the decision service in Rule Designer, you can run its rules in Rule Execution Server. After publishing the decision service to Decision Center, you can run tests and simulations to validate its rules.

Testing decision services in the development environment

You can test a decision service by deploying its rules from Rule Designer to the development Rule Execution Server, and then running them with a test application.

Testing and simulation in Decision Center

Decision Center includes testing and simulation features to verify rules and evaluate their results.

Working with Excel scenario files

You store testing and simulation scenarios in Excel spreadsheets. Start by making sure the business object model (BOM) is configured correctly. Then, you can generate a scenario file and populate it with data.

Testing rule execution

You can test the execution of a deployed ruleset through a SOAP or REST web service.

Testing decision services in the development environment

You can test a decision service by deploying its rules from Rule Designer to the development Rule Execution Server, and then running run them with a test application.

Procedure

1. In Rule Designer, right-click your decision service.
2. Click **Rule Execution Server > Deploy**.
3. In the RuleApp Deployment dialog, click the development deployment configuration, and then click **Next**.
4. Verify the deployment configuration settings, and then click **Next**.
5. Verify your login credentials.
6. Click **Connect** if you are not yet authenticated, and then click **Next**.
7. Verify the RuleApp and ruleset versions, and then click **Finish**.

After deployment, Rule Designer displays a report.

Note: Rule Designer can only deploy to the development environment. It cannot deploy to the test or production environment.
--

8. Use a test application with Rule Execution Server to call the ruleset deployed from your decision service.
9. Check the results to determine whether the rules run as expected.

Parent topic: [Testing decision services](#)

Testing and simulation in Decision Center

Decision Center includes testing and simulation features to verify rules and evaluate their results.

You can improve the performance of a business application by validating its rules before deployment. Decision Center offers you two ways to check rule behavior:

- Testing: You run a set of rules on scenarios that are defined by you to compare the actual results with your expected results.
- Simulation: You run rules on real or fictitious operational data, and use the results to assess and refine the behavior of the rules. You apply key performance indicators (KPIs) in analyzing the rules.

You do rule testing and simulation in the Decision Center Business console. You create scenario files in Excel, and for simulations, you define KPIs, report formats, and simulation configurations.

Note: You can also run tests with the Decision Center REST API .

Related information:

[Testing sets of rules in the Business console](#)

[Simulating business application results](#)

Testing rulesets in Rule Designer

To validate rulesets against scenarios, you set up and run a testing configuration.

About this task

You can generate a template of the scenarios file where you enter the values that you want to test. Then, you set up the testing configuration by specifying a decision operation and the Excel file that contains the scenarios.

Procedure

1. If you don't already have one, create a scenario file. To generate a template scenario file, see [Generating an Excel scenario file](#).
2. Optional: You can add breakpoints in various artifacts of the project to pause the execution of the test in strategic places for debugging purposes. To learn more about breakpoints, see [Breakpoints](#).
3. Right-click the project that you want to test and select **Debug as > Debug configurations....**
4. In the list of configurations, right-click **Testing decision operation** and click **New**.
5. Give this configuration a name and specify the Excel file that contains the scenarios and the decision operation to use.
6. Click **Debug**.

Results

The results of the execution display in the Console view: you see whether each scenario was successful, the time it took to execute it, and the number of executed rules and tasks.

The results for the scenarios use the following statuses:

- **Successful:** A test is successful when the expected results match the actual results.
- **Unsuccessful:** A test is unsuccessful when the expected results are different from the actual results.
- **Error:** An error is reported when the test cannot run the scenarios, for example, when an entry in the scenario file is not correctly formatted.

If there are breakpoints in the rules of the project, the execution stops when it encounters one and the Debug view opens. You can then see which breakpoints stopped the execution and you can continue the resume the execution until the next breakpoint.

Example

Here is an example of the results of a test with three scenarios:

```
*****
Scenario Number 0 : 'Rejected loan' = Success
Time = 16, Nb Executed Rules = 1, Nb Executed Tasks = 3
*****
Scenario Number 1 : 'Big amount loan' = Failure
Expected result 'the loan is approved equals ': Failed
Time = 0, Nb Executed Rules = 1, Nb Executed Tasks = 2
*****
Scenario Number 2 : 'Approved loan' = Success
Time = 0, Nb Executed Rules = 0, Nb Executed Tasks = 3
*****
Nb Executed 3
Nb Failures 1
Nb Errors 0
End running the JVM for testing
```

The Rejected loan scenario and the Approved loan scenario were successful, meaning that the expected values defined in the scenarios match the actual results of the tests. The Big amount loan scenario failed because the rule that approves the loan did not produce the expected result.

Working with Excel scenario files

You store testing and simulation scenarios in Excel. Before you generate an Excel scenario file, make sure that the business object model (BOM) is configured correctly. You can then generate the file and populate it with scenario data.

[Excel scenario files](#)

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

[Relationship between the BOM and the Excel file](#)

You use Excel to format scenarios for tests and simulations. Some columns in the scenario file use the information contained in the BOM of the rule project.

[Adding columns to the Excel file](#)

You configure the BOM to add columns to the **Scenarios** and **Expected Results** sheets.

[Removing columns from the Excel file](#)

You configure the BOM to remove columns from the **Scenarios** and **Expected Results** sheets.

[Validating the project](#)

You verify that a rule project or decision operation is suitable for generating Excel scenario files.

Excel scenario files

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

An Excel scenario file can contain the following sheets:

- Scenarios: Contains the input data for scenarios. Both testing and simulation scenario files have this sheet.
- Data entry: Regroups information that is used in other sheets.
- Expected Results: Holds the results that you expect from tests.
- Expected Execution Details: Holds the execution details that you expect from tests.

Important:

Never modify the column structure of the sheets.

Scenarios sheet

Each row in the Scenarios sheet represents one scenario. Each scenario must contain all the information needed to process a transaction.

		the borrower					the loan		
Scenario ID	description	first name	last name	b	S	c	y	z	
Very low risk	Very low risk loan accepted	Sam	Adams	4			95	8/1/2009	# 100000
Low risk	Low risk loan accepted	Bob	Schwartz	4			94	8/7/2010	# 500000
Average risk	Average risk loan accepted	John	Johnson	4			85	9/1/2010	# 900000
Amount too high	Loan rejected when amount too high	Mary	Doe	4			32	8/15/2010	# 1100000

The columns indicate the information that you must provide for each scenario:

- **Scenario ID:** The name identifies the scenario and must be unique.
- **description:** A free text cell to describe the scenario.
- **business terms:** Values for the scenarios. Bold columns or subcolumns are mandatory.

Note:

The reduced rows between the column headers and the first scenario contain information that might be useful to developers. Never edit these rows.

The following visual aids help you complete your scenarios:

- Red triangle: When shown in a column header, it indicates the presence of a comment that explains the type of values to be entered, such as text, numbers, dates, or true or false. Hover over a triangle to display a comment as follows:

the borrower				
first name	last name	credit score	yearly income	
John	Smith	600	80000	
John	Smith	600	80000	

- Dark green triangle: When shown in a cell, it suggests a better format for the cell. For example, if you enter a numerical value in a text cell, Excel suggests that you format the cell for numerical values. To make sure that the scenario runs correctly, ignore this suggestion.
- Arrow: When shown before a column name, it indicates that the values must correspond to entries that are specified in a separate **data entry** sheet.

Data entry sheets

You create data entry sheets when you generate a scenario file. Data entry sheets regroup data to be used in other sheets.

The name of each data sheet serves as the type of data that is expected in the column of the other sheet that uses the data. For example, in the following figure, the address sheet **1** contains different addresses that are used in the addresses column **2** of the Scenarios sheet.

Entry ID	Street	City	State	Zip Code
billing address	2207 7th avenue	New York	NY	10027
shipping address	25 Elm Street	Dallas	TX	75201
personal address	110 Cactus Street	Phoenix	AZ	85003

1

the borrower						
description	first name	last name	credit score	yearly income	amount	→ addresses
Loan rejecte	John	Smith	600	80000	500000	personal address
Loan approve	John	Smith	600	80000	25000	billing address

Enter the name of an object defined in the sheet address

Note:

A column that uses values from a data entry sheet has an arrow in its column header.

You create an entry in a data entry sheet by completing a row that includes a unique name to identify the entry in the other sheets.

Expected Results sheet

The Expected Results sheet contains the results that you expect to obtain when you run tests that use the scenarios.

You can test many aspects of a set of rules, but the Expected Results sheet contains only the tests that you specified when you generated the scenario file. For example:

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Each blue column in the Expected Results sheet corresponds to a test. If you do not enter a value in a column, the corresponding test is skipped.

Tip:

You cannot create a new column in Excel. However, you can generate another empty scenario file template, and then copy and paste a column from it.

You link the sheets that contain the scenarios and their expected results by entering names in their respective Scenario ID columns. In addition to making sure that the names match between the two sheets, it is good practice to keep the scenarios and their expected results in the same order.

When you test for a list of values, you enter each item in the corresponding cell on a separate row. You do not have to duplicate the Scenario ID for each row.

Expected Execution Details sheet

The Expected Execution Details sheet contains the execution details that you expect to obtain when you run tests that use the scenarios.

The following are examples of execution details:

- List of rules fired
- List of executed ruleflow tasks
- Duration of execution

You can test many aspects of a run, but the Expected Execution Details sheet contains only the tests that you specified when you generated the scenario file. Each green column in the Expected Execution Details sheet

corresponds to a test.

You link the sheets that contain the scenarios and expected execution details by entering names in their respective Scenario ID columns.

When you test for a list of values, you enter each item in the corresponding cell on a separate row.

4			
5		Scenario ID	the list of fired rules contains
9		Big Loan	eligibility.checkIncome
10			eligibility.approval
11			eligibility.checkCreditScore
12		Small Loan	
13			
▶▶ Scenarios Expected Execution Details HELP			

You do not have to duplicate the Scenario ID for each row because the last ID entered is used.

Parent topic: [Working with Excel scenario files](#)

Related concepts:
[Relationship between the BOM and the Excel file](#)

Relationship between the BOM and the Excel file

You use Excel to format scenarios for tests and simulations. Some columns in the scenario file use the information contained in the BOM of the rule project.

When you use Excel as the format for your scenarios, columns in the **Scenarios** and **Expected Results** sheets of the Excel scenario file are generated using information contained in the BOM of the rule project.

This section explains how information retrieved from the BOM affects the sheets of the Excel scenarios file.

Scenarios sheet

The columns in the **Scenarios** sheet are defined by the BOM classes that define instances of input parameters required to execute the rulesets.

The **Scenarios** sheet is composed of:

Required columns

These provide the data that is required to execute your rulesets. The name of a required column displays in bold in the Excel sheet.

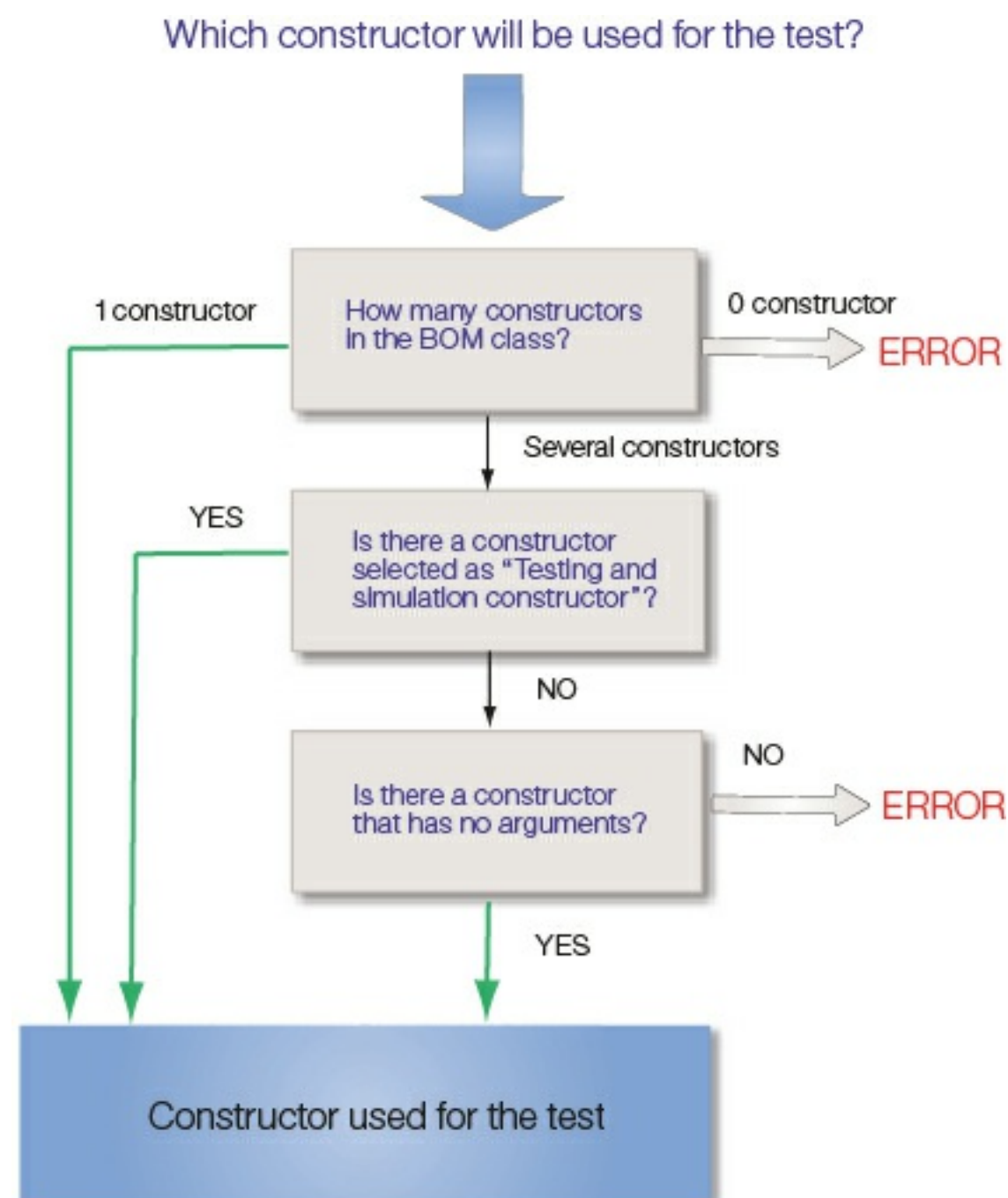
Optional columns

These provide additional information on the scenarios. The attributes of the BOM class define the optional column headings. Each optional column corresponds to an attribute. If the attribute is verbalized, the name in the column heading is the verbalization of the attribute.

Note: You can use only attributes that are non-static, and writable ("read/write" and "write only") to create optional columns.

A class or an interface in the BOM requires a constructor for it be useful in an Excel scenario file. A constructor of the BOM class defines the required columns. Choose the constructor that you want to use for the Excel scenario file.

A class sometimes has a default constructor that does not have any arguments. When there are several constructors in a class, the class uses the constructor with no arguments by default. If a class has several constructors, you must specify the one you want to use in your tests by selecting the **Testing and simulation constructor** option in the **General Information** section of the BOM editor.



Note: If the constructor has no arguments, a required column does not show in the Excel scenario file.

The name of the arguments in the constructor define the required column headings.

By default, when you generate a BOM from a XOM, the constructor arguments have generic names such as arg1, arg2, and arg3, as shown in the following figure:

Arguments

Edit the arguments of this member.

Name	Type
arg1	java.lang.String
arg2	java.lang.String
arg3	java.util.Date
arg4	java.lang.String
arg5	int

To obtain meaningful column headings, you must rename these arguments. Make sure that the name of the constructor arguments are meaningful to a business user.

You must rename each argument that references an attribute with the name of the attribute. For example, if arg1 sets the value of the attribute firstName, you must rename arg1 to firstName. You must rename other arguments with a name that is meaningful to a business user. The following figure shows an example of arguments that have been renamed:

Arguments

Edit the arguments of this member.

Name	Type
firstName	java.lang.String
lastName	java.lang.String
birthDate	java.util.Date
SSN	java.lang.String
creditScore	int

Note: You can also use annotations in your XOM to give a business name to the arguments, see [@BusinessName](#).

The name displayed in the column heading is the verbalized name of the attribute or constructor argument:

- If the attribute has an explicit verbalization, the verbalization is used as column heading.
- If the attribute does not have an explicit verbalization, or if the argument is an implementation parameter that does not correspond to an attribute, the column heading uses a default verbalization. For example, if the argument name is dateOfBirth, the default verbalization is date of birth.

Expected Results sheet

The ruleset output parameters and the BOM classes define the columns in the **Expected Results** sheet of the Excel scenario file.

The attributes of the BOM class define the column headings. Each column corresponds to an attribute.

You must verbalize each attribute because the name in the column heading is the verbalization of the attribute. If the attribute is a collection, you must define the domain.

Note: You can use only attributes that are non-static and readable ("read/write" and "read only") to create columns.

Parent topic: [Working with Excel scenario files](#)

Related concepts:
[Excel scenario files](#)

Adding columns to the Excel file

You configure the BOM to add columns to the **Scenarios** and **Expected Results** sheets.

About this task

The columns contained in the Excel scenario file are generated using information from the BOM. The input parameters, the attributes, and the constructor arguments are used to create the input columns in the Excel scenario file. You can also configure the BOM to add columns in the **Scenarios** and **Expected Results** sheets.

You can create attributes in the BOM to add new columns in the **Scenarios** sheet or in the **Expected Results** sheet.

Procedure

To add columns to the **Scenarios** and **Expected Results** sheets:

1. Create virtual attributes:
 - Writable attributes (“Read/Write” or “Write Only”) for optional columns in the Scenarios sheet
 - Readable attributes (“Read/Write” or “Read Only”) for columns in the Expected Results sheet
2. Define the BOM to XOM mapping for these virtual attributes:
 - Edit the setter part for a new column in the Scenarios sheet
 - Edit the getter part for a new column in the Expected Results sheet
3. Select the new available **Expected Results** options when you generate the Excel scenario file.

Results

To add **required columns**, you must modify the list of arguments of the testing and simulation constructor, or choose a different testing and simulation constructor.

Parent topic: [Working with Excel scenario files](#)

Related concepts:

[Relationship between the BOM and the Excel file](#)

Related tasks:

[Removing columns from the Excel file](#)

[Generating an Excel scenario file](#)

Removing columns from the Excel file

You configure the BOM to remove columns from the **Scenarios** and **Expected Results** sheets.

About this task

The Excel scenario file uses information from the BOM to generate columns. The input parameters, the attributes, and the constructor arguments are used to create the input columns in the Excel scenario file. You can also configure the BOM to remove columns from the **Scenarios** sheet.

By default, there is a column for each constructor argument and for each attribute (providing it is non-static and writable). If you do not want to display all the attributes as columns in the **Scenarios** sheet, you can exclude them.

Procedure

To remove optional columns from the **Scenarios** sheet:

1. In the BOM editor, double-click the attribute that you want to exclude from the Excel scenario file.
2. In the **General Information** area, select the **Ignore for testing and simulation** option.

Results

This option ensures that there is no column in the Excel scenario file for the attribute you have excluded.

To remove **required columns**, you can either modify the list of arguments of the testing and simulation constructor, or choose a different testing and simulation constructor (see [Relationship between the BOM and the Excel file](#)).

Parent topic: [Working with Excel scenario files](#)

Related concepts:

[Excel scenario files](#)

[Relationship between the BOM and the Excel file](#)

Related tasks:

[Adding columns to the Excel file](#)

Validating the project

You verify that a rule project or decision operation is suitable for generating Excel scenario files.

Before you begin

Important:
Before validating a project, make sure that you select the correct constructor in the BOM. For more information about the constructors and how to use them, see [Relationship between the BOM and the Excel file](#).

About this task

The default format for entering scenarios is Excel. When you use the default Excel format for your scenarios, you must validate that projects on which you want to run tests are suitable for generating the correct columns in the Excel scenario file.

You validate a project in the Testing and Simulation Project Validation view in Rule Designer. This view raises errors and warnings if the BOM or input parameters require modification. You must fix any errors in the view before you generate an Excel scenario file.

Procedure

To validate the project:

1. In the Rule Explorer, right-click the project on which you want to enable testing, and then click **Testing and simulation > Check project**.

The Testing and Simulation Project Validation view opens and lists the errors and warnings.

2. To view the solution description, select the warning or error.

The Solution Description section displays a description of the warning or error and indicates what you can do to solve the problem.

3. Double-click the warnings or errors to open the class or member to fix in the BOM editor.
4. In the Testing and Simulation Project Validation view, click the **Refresh View** button to view any remaining errors or warnings.

Results

If there are no remaining errors, you can generate the Excel scenario file in the Decision Center Business console.

Important:
Make sure that you publish to Decision Center the project containing any BOM changes that you made during project validation.

Parent topic: [Working with Excel scenario files](#)

Related concepts:
[Excel scenario files](#)

Testing rule execution

You can test the execution of a deployed ruleset from a client application through a SOAP web service or a REST web service.

Executing rules by using the REST service

Decision Server provides a Representational State Transfer (REST) service for ruleset execution. You can use it to execute rulesets through the HTTP protocol by using the XML or the JSON format.

Executing rules by using the SOAP service

Through a Simple Object Access Protocol (SOAP) web service, Rule Execution Server automatically presents as a web service any deployed ruleset that uses an XML schema or a Java™™ XOM.

Executing rules by using the REST service

Decision Server provides a Representational State Transfer (REST) service for ruleset execution. You can use it to execute rulesets through the HTTP protocol by using the XML or the JSON format.

Benefits

The REST service for ruleset execution provides XML and JSON generation, XSD validation, and execution services. The service is designed for the following benefits:

- You do not need a client library or a complex configuration to interact with a remote Rule Execution Server instance.
- You can work across environments or from various client applications, typically from JavaScript clients.
- You can easily move from local to remote Rule Execution Server execution.

Workflow

To use the REST service for a ruleset execution, use the following process. See [Executing a ruleset from a client](#).

1. Deploy a valid ruleset.
2. Use the REST service to generate an XML or a JSON payload. The REST service provides XML and JSON generation, XSD validation, and execution services. You can test the generation of the payload and its execution from the Rule Execution Server console. See [Testing ruleset REST execution](#).
3. After you are familiar with the input and output format, send the request as the payload of an HTTP call through a POST method to the corresponding URI. See [Executing a ruleset by creating a REST request](#).

[REST resources](#)

Resources for the Decision Server REST service for execution are rulesets.

[Endpoint URIs](#)

Endpoint URIs represent rulesets as Rule Execution Server resources for the REST execution service.

[HTTP methods](#)

The HTTP methods of the REST API provide the operations that are available on Rule Execution Server artifacts.

[HTTP headers and URI parameters](#)

The URIs for Rule Execution Server REST resources support some HTTP header fields and generic URI parameters.

[Content types](#)

The Content-Type field in HTTP headers indicates in which format the data is sent to, or returned by, the HTTP methods of the Rule Execution Server REST service.

[Request and response schema](#)

In the REST service for ruleset execution, requests and responses follow specific schemas, depending on whether the ruleset XOM is based on XML classes or on Java™ classes. The schema determines how the types are serialized.

[XML serialization of ruleset XOMs](#)

In the REST service for ruleset execution, primitive Java types, XSD types, and Java XOM classes are serialized differently.

[JSON serialization of ruleset XOMs](#)

Ruleset parameters of primitive Java types, arrays, and Java XOM classes can be serialized to JSON through a Jackson process.

[Executing a ruleset by creating a REST request](#)

You can execute a valid ruleset by creating an XML or a JSON request and using the POST method.

[Generating a WADL representation](#)

You can generate a WADL representation of request and response elements, and their documentation.

[Generating an OpenAPI representation](#)

OpenAPI is a standardized version of the Swagger specification. It provides a language-independent way to present REST APIs. You can generate OpenAPI representations for executing decision services. This capability is part of ODM Decision Connect which enables reusable decisions to be exposed as OpenAPI based public APIs to be invoked directly by applications. The OpenAPI based decisions can also be published to an IBM API Connect catalog for managed APIs. IBM API Connect Essentials is also an optional part of Decision Connect.

Parent topic: [Testing rule execution](#)

Related concepts:

[Endpoint URIs](#)

[Executing rulesets in the decision engine](#)

[Hosted transparent decision services](#)

Related tasks:

[Calling a decision service by using the REST API](#)
[Calling a decision service by using OpenAPI](#)
[Calling a decision service from API Connect](#)

REST resources

Resources for the Decision Server REST service for execution are rulesets.

Developers can use the REST service to execute rulesets. As in hosted transparent decision services, rulesets are identified by their signature. Short ruleset paths and canonical ruleset paths are both supported.

The primary artifact for a REST service is the resource. Each resource is represented as a unique resource identifier (URI), and every resource has an HTTP method that clients use to request the execution of a ruleset and receive the response. WADL and OpenAPI formats can be used to describe the signature of the REST service that can be called to execute the ruleset.

Examples of ruleset paths:

- Short ruleset path with no version numbers: `miniloanruleapp/miniloanrules/`
- Canonical ruleset path: `miniloanruleapp/1.0/miniloanrules/1.0/`

Parent topic: [Executing rules by using the REST service](#)

Endpoint URIs

Endpoint URIs represent rulesets as Rule Execution Server resources for the REST execution service.

URIs for REST endpoints have the following format:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetpath}/{filetype}?{options}
```

URIs are defined as follows:

Table 1. Items that make up an URI

Item	Description
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService	<p>The endpoint of the hosted transparent decision service (HTDS) application.</p> <p>The value of <odm_on_cloud_environment> must be either dev, test, or prod.</p> <p>Example: https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService .</p>
/rest	The REST service context root.
/v1	The version number of the REST service. This parameter is optional. The current version is numbered v1. If subsequent versions are released, the version number is incremented but the changes added by each version remains compatible with any code that implements the current /v1 version.
{rulesetpath}	<p>The short ruleset path or the canonical ruleset path.</p> <ul style="list-style-type: none">• A canonical ruleset path indicates the RuleApp name and version number and the ruleset name and version number. For example: myRuleApp/1.0/myRuleset/1.0• A short ruleset path leaves out one or both version numbers: For example: myRuleApp/myRuleset
{filetype}	The file type: wadl or openapi.
{options}	You can add options at the end of the URI. See Table 2 for details.

Table 2. Options

Option	Used for	Description
format	OpenAPI	<p>Select YAML or JSON. If no format is specified, YAML is used by default.</p> <p>Example: format=JSON</p>
schemes	OpenAPI	<p>Specify HTTP or HTTPS, separated by comma.</p> <p>Example: schemes=HTTP, schemes=HTTPS, or schemes=HTTP,HTTPS</p> <p>If this parameter is not specified, the same protocol as the request URL is used.</p>

extension	OpenAPI	Specify apiconnect to generate an OpenAPI file for IBM API Connect®. Example: extension=apiconnect
trace	OpenAPI WADL	If it is set to true, the trace is enabled. The trace filter is added to the request and the corresponding trace is added to the response. Example: trace=true
download	OpenAPI WADL	If it is set to true, the generated file is downloaded to the file system. Example: download=true
zip	WADL	If it is set to true, the WADL code and its XSD files are in a compressed file. When you specify the zip option, the inline option is not necessary. Example: zip=true
inline	WADL	If it is set to true, the XSD files are added in the WADL code. Example: inline=true

Example of adding an option:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?zip=true
```

Use & when you want to add more than one option:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetpath}/{filetype}?{option1}&{option2}&{option3}
```

Parent topic: [Executing rules by using the REST service](#)

HTTP methods

The HTTP methods of the REST API provide the operations that are available on Rule Execution Server artifacts.

The REST API for ruleset execution supports the GET and POST methods.

- You use the GET method to generate a sample XML or JSON payload or to retrieve WADL or OpenAPI for a specified ruleset. For example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/xml
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/json
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/ruleapp/1.0/miniloanrules/1.0/wadl
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/ruleapp/1.0/miniloanrules/1.0/openapi
```

- You use the POST method to create a request for execution of a ruleset or to validate an XML payload. The request body contains the XML or JSON payload. For example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0
```

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/validate
```

Parent topic: [Executing rules by using the REST service](#)

HTTP headers and URI parameters

The URIs for Rule Execution Server REST resources support some HTTP header fields and generic URI parameters.

The REST execution service supports the **Accept-Language** field name in HTTP headers. All other fields are ignored.

The Accept-Language header field

The Accept-Language field sends the list of languages that are valid for the response message. For example, Accept-Language: fr, pt-BR means that the preferred language is French but Brazilian Portuguese is also accepted. You can use the equivalent URI parameter instead of the HTTP header field.

The Accept-Language parameter

The **accept-language** URI parameter is equivalent to the **Accept-Language** HTTP header field. You can use it in request messages to send the list of languages that are valid for the response message.

Parent topic: [Executing rules by using the REST service](#)

Content types

The Content-Type field in HTTP headers indicates in which format the data is sent to, or returned by, the HTTP methods of the Rule Execution Server REST service.

The REST service for ruleset execution supports the `application/xml` and the `application/json` content types. XML is the default content type of the response. You can also find the object definitions by retrieving the Web Application Description Language (WADL).

For OpenAPI service definitions, only the `application/json` content type is supported. See the consumes and produces sections in the OpenAPI definition file.

Parent topic: [Executing rules by using the REST service](#)

Request and response schema

In the REST service for ruleset execution, requests and responses follow specific schemas, depending on whether the ruleset XOM is based on XML classes or on Java™ classes. The schema determines how the types are serialized.

The request and response schema results from the signature of the target ruleset.

- The request part is composed of the following elements:
 - The **IN** and **INOUT** parameters of the ruleset, in alphabetical order.
 - Optionally, a decision ID if you want to set it to a specific value.
 - Optionally, a trace filter.
- The response part is composed of the following elements:
 - The **INOUT** and **OUT** parameters of the ruleset, in alphabetical order.
 - The decision ID: either the default identifier or the value that you set in the request.
 - The returned trace, depending on the filter that you set in the request.

The XML payload is analyzed against the generated XSD files. The execution response is sent in the same format as the execution request (XML or JSON).

XML request validation

The validation response is returned in JSON format:

- If the request is valid, the response is an empty JSON list [].
- If the request is invalid, the tool returns the list of errors. Each error contains the following fields:
 - Type: the type of the error. Possible values are "Error", "Fatal" and "Warning".
 - Line: the number of the line that contains the error in the .xml file
 - Column: the number of the column that contains the error in the .xml file
 - Message: the error message itself

The JSON payload cannot be validated.

Here are examples of error messages:

```
{"type": "Error", "line": 8, "column": 32, "message": "cvc-datatype-valid.1.2.1: 'falseee' is not a valid value for 'boolean'."}
```

```
{"type": "Error", "line": 9, "column": 22, "message": "cvc-datatype-valid.1.2.1: '5d' is not a valid value for 'integer'."}
```

```
{"type": "Fatal", "line": 39, "column": 24, "message": "The element type\n\"par:longParam\" must be terminated by the matching end-tag\n\"</par:longParam>\"."}
```

Parent topic: [Executing rules by using the REST service](#)

Related concepts:

[XML serialization of Java types](#)

[XML serialization of ruleset XOMs](#)

Related tasks:

[Executing a ruleset by creating a REST request](#)

XML serialization of ruleset XOMs

In the REST service for ruleset execution, primitive Java™ types, XSD types, and Java XOM classes are serialized differently.

Ruleset parameters are handled differently depending on whether they are expressed as XML elements or as Java types.

Primitive types

Primitives types are Java types and classes that can be written as inline strings. The XML element that is used to serialize a primitive type as a ruleset parameter is the name of the ruleset parameter in the parameter namespace.

You can find more information here:

- [Changing target namespaces](#) explains how to customize the parameter namespace from the Ruleset View in the Rule Execution Server console.
- [XML serialization of Java types](#) shows the XML types and Java Architecture for XML Binding (JAXB) annotations for Java primitive types and simple types.

Dynamic XOM

For dynamic XOMs, the root element depends on how the ruleset parameters are defined. The XML description is serialized from the original XSD code of the dynamic XOM on which the ruleset is based.

Name of the root element

For rulesets that are based on a dynamic XOM, the root element name is determined as follows:

- If the parameter is an XML element in the ruleset signature, that element is used as the root element name of the parameter in requests and responses.
- If the parameter is a complex or primitive Java type, the parameter name is used as the root element name of the parameter in requests and responses, within the configured param namespace.

Note: A ruleset parameter is declared from an XML element if it meets the following condition: that parameter uses an XML complex type for which an XML element was declared from that complex type in the original imported XSD file. For example, starting from the following XSD definition:

```
<xsd:element name="MyConference">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="session" type="session"
maxOccurs="unbounded"/>
      <xsd:element name="participant" type="participant"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

the ruleset parameter is declared from the MyConference XML element:

```
<par:Request
xmlns:par="http://www.ibm.com/rules/decisionservice/myruleapp/myruleset
/param" xmlns:jav="http://www.acme.com/myconference">
  <jav:MyConference>
    <jav:session>
      [...]
    <jav:session>
      <jav:MyConference>
</par:Request>
```

Serialization

The root element depends on your XML types.

- For XML complex types, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.
- For XML elements, the parameter declaration uses the defined XML element as the root element in the original namespace. In this case, the name of the parameter that is attached to each node is provided in the WADL code as a comment message.

When multiple elements have the same name, element nodes are differentiated alphabetically. In this case, if you want one of the elements to be null, you must set the **xsi:nil** attribute to true. You can do that only if

the **nillable** attribute is set to `true` for the root element in the original XSD schema of the dynamic XOM. The default value of the **nillable** attribute is `false`.

Java XOM

For Java XOMs, the root element derives from the parameter declaration and XML serialization uses JAXB annotations. Arrays are supported for ruleset parameters.

Name of the root element

For rulesets that are based on a Java XOM, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.

Serialization

The XML structure of Java XOM classes is serialized through the standard JAXB process. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using JAXB annotations.

Note: A parameter is visible as an XML element only if you specify it with an `XmlElement` annotation or if valid `getXXX` and `setXXX` methods apply to the parameter.

Arrays as ruleset parameters

In Java XOM, you can specify ruleset parameters as simple or multidimensional arrays. XML serialization is processed differently for each type of arrays:

- If the ruleset parameter is a simple array, the parameter name repeats as a standard array in XML.
- If the ruleset parameter is a multidimensional array, the parameter name repeats as a standard array type in XML for the first dimension and then uses `item` elements for the subsequent dimensions. For example:

```
<myMultiDimArrayParam>
  <item>1</item>
  <item>2</item>
</myMultiDimArrayParam>
<myMultiDimArrayParam>
  <item>3</item>
  <item>4</item>
  <item>5</item>
</myMultiDimArrayParam>
```

Parent topic: [Executing rules by using the REST service](#)

Related concepts:

[Request and response schema](#)

[XML serialization of Java types](#)

JSON serialization of ruleset XOMs

Ruleset parameters of primitive Java™ types, arrays, and Java XOM classes can be serialized to JSON through a Jackson process.

Each ruleset parameter is written as a JSON name/value pair. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using Jackson or JAXB annotations. Jackson annotations are applied before JAXB annotations with matching precedence. Some JAXB annotations are not supported or do not have a JSON equivalent.

An empty constructor, either public or private, is needed to deserialize with Jackson.

Jackson version

Jackson currently exists as two major versions, 1.x and 2.x. Packaging and API differences make the two releases incompatible. The REST service for ruleset execution uses Jackson 2.x, which means that you must use Jackson 2.x to annotate the Java XOM classes.

Some known limitations

Static nested classes are supported. Nonstatic nested classes, also known as inner classes, are not supported.

Cyclic links between three or more classes are not supported.

The official Jackson documentation recommends not to use `java.sql.date`.

For more information, see the [Jackson JSON Processor Wiki](#).

If you use Java primitive types as input values in your execution requests, make sure that each value falls within its corresponding data type range. Otherwise, you might get unexpected output.

Parent topic: [Executing rules by using the REST service](#)

Executing a ruleset by creating a REST request

You can execute a valid ruleset by creating an XML or a JSON request and using the POST method.

About this task

To execute a deployed ruleset by using the REST service, you generate the XML or JSON payload and post the request. You can also generate a WADL representation of request and response elements. See [Generating a WADL representation](#).

In your client, you construct the request message as an XML or JSON packet, which depends on the XML signature in WADL format of the REST service. The client must specify the web service URL, pass in the request, and specify a variable to hold the response from the service.

Procedure

1. Generate an XML or JSON fragment by posting your request to the following URI.

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/{format}
```

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/xml
```

Then, you can use the result as a starting point to write an XML or JSON request.

2. Validate the XML structure of the request by posting your request to the following URI.

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/validate
```

Example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/validate
```

Tip: You can test the generation of the XML code also from the Rule Execution Server console as explained in [Testing a ruleset for REST execution](#).

If the request is not valid, error messages are returned in JSON format. For more information, see [Request and response schema](#). Validation (/validate) is not available for the JSON format.

3. Post the execution request to the following URI:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}
```

REST requests for execution use the POST method. The request body contains the XML or JSON payload. Example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0
```

The execution response is sent in the same format as the request.

Parent topic: [Executing rules by using the REST service](#)

Related concepts:
[Request and response schema](#)

Generating a WADL representation

You can generate a WADL representation of request and response elements, and their documentation.

About this task

To write XML input for the POST method to a specific ruleset endpoint URI, you can generate the WADL representation. The WADL representation contains two entries for a POST method and one for a GET method. A set of XSD files are attached to this method to describe the XML representation of the input and output objects.

To generate a WADL representation, you can use the endpoint URIs. The WADL format is described in the [Web Application Description Language](#) page of the W3C website.

Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/wadl
```

The URI variables are defined in [Endpoint URIs](#).

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl
```

2. Optional: To add the XSD files in the WADL code, use the **inline** query parameter. Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?inline
```

If you do not specify the **inline** parameter, the external .xsd files that are referenced by the .wadl file remain external. The .wadl file might not work correctly with some tools from independent software vendors. If you specify the **inline** parameter, the .xsd files are included in the .wadl file.

Note: In WebSphere® Application Server, you must explicitly set the **inline** value to true.

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?inline=true
```

3. Optional: To generate the WADL code and its XSD files to a compressed file, add the **zip** parameter.

If you specify the **zip** parameter, the **inline** parameter is not necessary.

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?zip=true
```

Results

If you try to generate WADL representation for an invalid URL or if an error occurs during the WADL generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

Note: The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

Parent topic: [Executing rules by using the REST service](#)

Related concepts:

[Endpoint URIs](#)
[Content types](#)

Generating an OpenAPI representation

OpenAPI is a standardized version of the Swagger specification. It provides a language-independent way to present REST APIs. You can generate OpenAPI representations for executing decision services. This capability is part of ODM Decision Connect which enables reusable decisions to be exposed as OpenAPI based public APIs to be invoked directly by applications. The OpenAPI based decisions can also be published to an IBM API Connect catalog for managed APIs. IBM API Connect Essentials is also an optional part of Decision Connect.

About this task

To write JSON input for the POST method to a specific ruleset endpoint URI, you can generate the OpenAPI representation. The OpenAPI representation contains one entry for the POST method.

To generate an OpenAPI representation, you can use the [Endpoint URIs](#). For more information about OpenAPI, see [The OpenAPI Specification](#) page of the Open API Initiative website.

Note: Version 2.0 of the OpenAPI specification is supported.

Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/openapi?format={format}
```

Set *{format}* to the format of the OpenAPI definition file, which is either YAML or JSON. The default format is YAML.

The URI variables are defined in [Endpoint URIs](#).

The following example shows a request to generate an OpenAPI representation in YAML format:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/openapi?format=YAML
```

2. Optional: To use the OpenAPI file in IBM API Connect®, use the **extension** query parameter and specify apiconnect.

The following example shows a request to generate an OpenAPI representation in YAML format and use it in IBM API Connect:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/openapi?format=YAML&extension=apiconnect
```

Results

You generated an OpenAPI representation in YAML or JSON format.

If you used an invalid URL or if an error occurs during the OpenAPI generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

Note: The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, a status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

Parent topic: [Executing rules by using the REST service](#)

Related concepts:

[Content types](#)

 [Decision Connect](#)

 [IBM API Connect](#)

Executing rules by using the SOAP service

Through a Simple Object Access Protocol (SOAP) web service, Rule Execution Server automatically presents as a web service any deployed ruleset that uses an XML schema or a Java™ XOM.

The SOAP web service automatically generates a Web Services Description Language (WSDL) file for each deployed ruleset archive.

SOAP web services support the first-level elements of the simple Java types, and the corresponding XML binding parameters in the WSDL code. A simple Java type is the finest-grain type from which a complex Java type can be expressed. A mapping table is provided in XML serialization of Java types.

The WSDL file references the XML schema types that are packaged in the RuleApp. You can use a tool such as Web Tools Platform (WTP) to generate a web service Java client. You can configure the WSDL generation through the Rule Execution Server console. You can configure the location and the web service endpoint.

Endpoint URIs

Endpoint URIs represent rulesets as Rule Execution Server resources for the SOAP execution service.

XML serialization of ruleset XOMs

In the SOAP service for ruleset execution, primitive Java types, XSD types, and Java XOM classes are serialized differently in the SOAP request and SOAP response.

Generating a WSDL representation

You can generate a WSDL representation of request and response elements, and their documentation.

Parent topic: [Testing rule execution](#)

Related concepts:

[XML serialization of Java types](#)

Related tasks:

[Calling a decision service as a SOAP web service](#)

Endpoint URIs

Endpoint URIs represent rulesets as Rule Execution Server resources for the SOAP execution service.

The endpoint URI for a decision service in the Operational Decision Manager on Cloud server uses the following format:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/ws/<ruleset_path>
```

The following list explains the parts of the URI:

- *<vhostname>*: Serves as the name of the host for the decision service.
- *<odm_on_cloud_environment>*: Uses dev, test, or prod depending on whether the decision service ruleset is in the development, test, or production environment.
- *<ruleset_path>*: Provides the path for the decision service rule execution and must take one of the following formats:
 - ruleAppName/rulesetName
 - ruleAppName/ruleAppVersion/rulesetName
 - ruleAppName/ruleAppVersion/rulesetName/rulesetVersion

The following example shows a URI to MiniloanServiceRuleset:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/MiniloanService/MiniloanServiceRuleset
```

Parent topic: [Executing rules by using the SOAP service](#)

XML serialization of ruleset XOMs

In the SOAP service for ruleset execution, primitive Java™ types, XSD types, and Java XOM classes are serialized differently in the SOAP request and SOAP response.

Ruleset parameters are handled differently depending on whether they are expressed as XML elements or as Java types.

Primitive types

Primitives types are Java types and classes that can be written as inline strings. The XML element that is used to serialize a primitive type as a ruleset parameter is the name of the ruleset parameter in the parameter namespace.

You can find more information here:

- [Changing target namespaces](#) explains how to customize the parameter namespace from the Ruleset View in the Rule Execution Server console.
- [XML serialization of Java types](#) shows the XML types and Java Architecture for XML Binding (JAXB) annotations for Java primitive types and simple types.

Dynamic XOM

For dynamic XOMs, the root element depends on how the ruleset parameters are defined. The XML description is serialized from the original XSD code of the dynamic XOM on which the ruleset is based.

Name of the root element

For rulesets that are based on a dynamic XOM, the root element name is determined as follows:

- If the parameter is an XML element in the ruleset signature, that element is used as the root element name of the parameter in requests and responses.
- If the parameter is a complex or primitive Java type, the parameter name is used as the root element name of the parameter in requests and responses, within the configured param namespace.

Note: A ruleset parameter is declared from an XML element if it meets the following condition: that parameter uses an XML complex type for which an XML element was declared from that complex type in the original imported XSD file. For example, starting from the following XSD definition:

```
<xsd:element name="MyConference">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="session" type="session"
maxOccurs="unbounded"/>
      <xsd:element name="participant" type="participant"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

the ruleset parameter is declared from the MyConference XML element:

```
<par:Request
xmlns:par="http://www.ibm.com/rules/decisionservice/myruleapp/myruleset
/param" xmlns:jav="http://www.acme.com/myconference">
  <jav:MyConference>
    <jav:session>
      [...]
    <jav:session>
      <jav:MyConference>
    </par:Request>
```

Serialization

The root element depends on your XML types.

- For XML complex types, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.
- For XML elements, the parameter declaration uses the defined XML element as the root element in the original namespace. In this case, the name of the parameter that is attached to each node is provided in the WSDL code as a comment message.

When multiple elements have the same name, element nodes are differentiated alphabetically. In this case, if you want one of the elements to be null, you must set the **xsi:nil** attribute to true. You can do that only if

the **nillable** attribute is set to `true` for the root element in the original XSD schema of the dynamic XOM. The default value of the **nillable** attribute is `false`.

Java XOM

For Java XOMs, the root element derives from the parameter declaration and XML serialization uses JAXB annotations. Arrays are supported for ruleset parameters.

Name of the root element

For rulesets that are based on a Java XOM, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.

Serialization

The XML structure of Java XOM classes is serialized through the standard JAXB process. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using JAXB annotations.

Note: A parameter is visible as an XML element only if you specify it with an `XmlElement` annotation or if valid `getXXX` and `setXXX` methods apply to the parameter.

Arrays as ruleset parameters

In Java XOM, you can specify ruleset parameters as simple or multidimensional arrays. XML serialization is processed differently for each type of arrays:

- If the ruleset parameter is a simple array, the parameter name repeats as a standard array in XML.
- If the ruleset parameter is a multidimensional array, the parameter name repeats as a standard array type in XML for the first dimension and then uses `item` elements for the subsequent dimensions. For example:

```
<myMultiDimArrayParam>
  <item>1</item>
  <item>2</item>
</myMultiDimArrayParam>
<myMultiDimArrayParam>
  <item>3</item>
  <item>4</item>
  <item>5</item>
</myMultiDimArrayParam>
```

Parent topic: [Executing rules by using the SOAP service](#)

Generating a WSDL representation

You can generate a WSDL representation of request and response elements, and their documentation.

About this task

To write the SOAP payload for the request to a specific ruleset endpoint URI, you must generate the WSDL representation. The WSDL representation contains the description of the SOAP request and SOAP response, as well as the endpoint of the web service. A set of XSD files are attached to the WSDL file to describe the XML representation of the input and output objects.

To generate a WSDL representation, you can use the format of SOAP resource URIs. The WSDL format is described in the [Web Services Description Language](#) page of the W3C website.

Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/ws/{rulesetPath}/wsdl
```

The URI variables are defined in [Endpoint URIs](#).

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/miniloanruleapp/1.0/miniloanrules/1.0/wsdl
```

2. Optional: To generate the WSDL code and its XSD files to a compressed file, add the **zip** parameter.

Use the **zip** parameter if you want to download the WSDL as a ZIP archive in which the XSD files are saved separately from the WSDL file. This option is useful if you import more than one decision from Decision Server into an IBM® Integration Designer project, or if you migrate a ruleflow to a process. The separation of the object types from the WSDL avoids the duplication of these objects in IBM Integration Designer.

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/miniloanruleapp/1.0/miniloanrules/1.0/wsdl?zip=true
```

Results

If you try to generate WSDL representation for an invalid URL or if an error occurs during the WSDL generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

Note: The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

Parent topic: [Executing rules by using the SOAP service](#)

Integrating business rules

After you deploy a ruleset from a decision service, you can call the ruleset as a SOAP web service or by using the REST API. You can also interact with it by using OpenAPI or API Connect. You must provide service credentials for authentication, and should keep them separate from the application code.

Calling decision service rulesets

Your application can call a decision service ruleset as a SOAP/XML, REST/XML, or REST/JSON web service. Incoming data is passed as HTTP content in the XML or JSON format. Security is provided over HTTPS through basic HTTP authentication that includes a user name and password. Use service credentials to authenticate client applications.

Authentication for REST and SOAP invocation

Operational Decision Manager on Cloud supports the use of basic authentication in invoking REST and SOAP APIs. Instead of hardcoding login credentials into your application, pull them in from a separate file.

Calling decision service rulesets

Your application can call a decision service ruleset as a SOAP/XML, REST/XML, or REST/JSON web service. Incoming data is passed as HTTP content in the XML or JSON format. Security is provided over HTTPS through basic HTTP authentication that includes a user name and password. Use service credentials to authenticate client applications.

Operational Decision Manager on Cloud uses the HTTPS secure communication protocol. It has the security level of TLS (TLS1.0, TLS1.1, or TLS1.2). If your client application does not support TLS, an exception occurs during the connection process.

You must use a cloud password in calling a decision service ruleset. Use service credentials because they are highly secure, and exempt from cloud portal policies for updating passwords (see [Service credentials for client applications](#)).

Using the REST API to call a ruleset

Your client application calls a decision service ruleset through the REST API of Operational Decision Manager on Cloud. It uses standard HTTP GET, POST, PUT, and DELETE commands.

Using a SOAP web service

A client application can invoke a decision service ruleset as a SOAP web service.

Using OpenAPI

You can generate a client from an OpenAPI file that is made for a decision service ruleset.

Using API Connect

You can expose a decision service ruleset to other organizations via IBM API Connect®, which manages access and uses built-in security mechanisms.

Parent topic: [Integrating business rules](#)

Using the REST API to call a ruleset

Your client application calls a decision service ruleset through the REST API of Operational Decision Manager on Cloud. It uses standard HTTP GET, POST, PUT, and DELETE commands.

Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud(see [Service credentials for client applications](#)).

About this task

A client application calls a decision service ruleset by connecting with an execution server and calling the ruleset through the REST API.

The endpoint URI for the ruleset in the server looks as follows:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/<ruleset_path>
```

The following list explains the parts of the URI:

- *<vhostname>*: Serves as the name of the host for the decision service.
- *<odm_on_cloud_environment>*: Uses dev, test, or prod depending on whether the decision service ruleset is in the development, test, or production environment.
- *<ruleset_path>*: Provides the path for the decision service rule execution and must take one of the following formats:
 - ruleAppName/rulesetName
 - ruleAppName/ruleAppVersion/rulesetName
 - ruleAppName/ruleAppVersion/rulesetName/rulesetVersion

The following example shows a URI to MiniloanServiceRuleset:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/MiniloanService/MiniloanServiceRuleset
```

The execution of the ruleset is performed through an HTTP POST at the specified endpoint URI. The user name and password for the authentication are passed with an authorization header (see [Authentication for REST and SOAP invocation](#)).

Example

The following example shows Java™ code that calls MiniloanServiceRuleset on the Operational Decision Manager on Cloud server. The example uses Apache HttpClient v4.3.

```
import java.io.IOException;

import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpStatus;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class DecisionServiceExecution {

    public static void main(String[] args) throws ClientProtocolException,
        IOException {

        // Replace <vhostname> with the name of the host of the cloud
        portal

        String endpointURI =
        "https://<vhostname>.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/MiniloanService/MiniloanServiceRuleset";

        // Replace "loginID" with the functional ID of the service
        account you are using to authenticate with your tenant in the Cloud
```

```

        String userName = "loginID";

        // Replace "password" with the password of the service account
        you are using to authenticate with your tenant in the Cloud
        String password = "password";

        String credentials = userName + ":" + password;
        String encodedValue =
Base64.encodeBase64String(credentials.getBytes());
        String authorization = "Basic " + new String(encodedValue);

        String contentType = "application/json";

        // Set the borrower and the loan
        String payload = "{\"borrower\": {" +
            "\"name\": \"John\", " +
            "\"creditScore\": 600, " +
            "\"yearlyIncome\": 80000 " +
            "}, " +
            "\"loan\": {" +
            "\"amount\": 500000, " +
            "\"duration\": 240, " +
            "\"yearlyInterestRate\": 0.05 " +
            "}" +
            "}";

        CloseableHttpClient client = HttpClients.createDefault();

        try {
            HttpPost httpPost = new HttpPost(endpointURI);
            // Add the basic authentication header
            httpPost.addHeader("Authorization", authorization);
            httpPost.setEntity(new StringEntity(payload));
            httpPost.addHeader("Content-Type", contentType);
            CloseableHttpResponse response =
client.execute(httpPost);
            try {
                if (response.getStatusLine().getStatusCode()
!= HttpStatus.SC_OK) {
                    System.err.println("Status Code: " +
response.getStatusLine().getStatusCode());
                    System.err.println("Status Line: " +
response.getStatusLine());

                    String responseEntity =
EntityUtils.toString(response.getEntity());
                    System.err.println("Response Entity: "
+ responseEntity);

                    throw new RuntimeException(
                        "An error occurred
when invoking Decision Service at: "
                        +
endpointURI
                        + "\n"
                        +
response.getStatusLine() + ": " + responseEntity);
                } else {
                    String result =
EntityUtils.toString(response.getEntity());
                    System.out.println("Result: " +
result);
                }
            } finally {
                if (response != null) {
                    response.close();
                }
            }
        }
    }
}

```

```
        }  
    }  
    } finally {  
        client.close();  
    }  
}  
  
}
```

Parent topic: [Calling decision service rulesets](#)

Using a SOAP web service

A client application can invoke a decision service ruleset as a SOAP web service.

Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#) and [Authentication for REST and SOAP invocation](#)).

About this task

For a client application to invoke a decision service ruleset as a SOAP web service, you must create proxy classes from a Web Services Description Language (WSDL) file generated for the ruleset path. The format of a WSDL file is independent of the language that generates it.

Procedure

1. Obtain a WSDL file for a decision service ruleset.
2. Generate proxy classes from the WSDL file.
3. Use the proxy classes to invoke the ruleset from your client application.

Example

To get a WSDL file from Rule Execution Server:

1. Deploy the ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click the ruleset that corresponds to your decision service.
4. In the Ruleset View, click **Retrieve HTDS Description File**.
5. Select **SOAP** as the service protocol type.
6. Check **Latest ruleset version** and **Latest RuleApp version** to generate the WSDL file for the latest versions.
7. Click **Download**.

To generate the proxy classes with Apache Axis:

1. Download and install [Eclipse IDE for Java™ EE Developers](#), which includes the Web Tools Platform (WTP) to generate the proxy classes for invoking the web service from a Java application.
2. Start Eclipse IDE for Java EE Developers and create a new Java Project (**File > New > Java Project**) to host the proxy classes.
3. Copy the WSDL file into this Java project.
4. Click **File > New > Other > Web Services > Web Service Client**.
5. In the Web Service Client wizard, click **Next**.
6. For the service definition, browse to the WSDL file.
7. Move the slider to select **Develop client**.
8. Under Configuration, ensure **Apache Axis** is selected as the Web service runtime, and then select the Java project as the Client project where you want to host the proxy classes that are generated.
9. Click **Finish**.

The following Java code sample imports the proxy classes for MiniloanServiceRuleset and calls the ruleset from a Java application:

```
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceMiniloanServiceRulesetBindingStub;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetDecisionServiceProxy;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetRequest;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetResponse;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.param
.Borrower;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.param
.Loan;

public class DecisionServiceExecution {
```

```

        public static void main(String[] args) {

            // Replace <vhostname> with the name of the host of the Cloud
            portal
            // NB: endpointURI is defined in the location attribute of the
            WDSL file
            String endpointURI =
            "https://<vhostname>.bpm.ibmcloud.com/odm/dev/DecisionService/ws/MiniloanService/MiniloanServiceRuleset/v75";

            MiniloanServiceRulesetDecisionServiceProxy proxy = new
            MiniloanServiceRulesetDecisionServiceProxy(endpointURI);

            MiniloanServiceMiniloanServiceRulesetBindingStub stub =
            (MiniloanServiceMiniloanServiceRulesetBindingStub)proxy.getMiniloanServiceRule
            setDecisionService_PortType();

            // Replace "loginID" with the functional ID of the service
            account you are using to authenticate with your tenant in the Cloud
            stub.setUsername("loginID");

            // Replace "password" with the password of the service account
            you are using to authenticate with your tenant in the Cloud
            stub.setPassword("password");

            // Set the borrower

            com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Borro
            wer borrower =

                                new
            com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Borro
            wer(

                                "John", // name
                                600, // credit score
                                80000); // yearlyIncome

            Borrower borrowerParam = new Borrower(borrower);

            // Set the loan

            com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Loan
            loan =

                                new
            com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Loan(
                                500000, // amount
                                240, // duration
                                0.05, // yearlyInterestRate
                                0, // yearlyRepayment (to be
            computed by the decision service)
                                true, // approved (set to true
            by default, to be computed by the decision engine),
                                null); // messages (to be
            computed by the decision service)

            Loan loanParam = new Loan(loan);

            // Set the decision ID
            String decisionID = "1";

            MiniloanServiceRulesetRequest request = new
            MiniloanServiceRulesetRequest(decisionID, borrowerParam, loanParam);

            try {
                MiniloanServiceRulesetResponse response =

```



```

proxy.miniloanServiceRuleset(request);
        System.out.println("Rules executed.");
        System.out.println("Approved: " +
response.getLoan().getLoan().isApproved());
        System.out.println("Yearly interest rate: " +
response.getLoan().getLoan().getYearlyInterestRate());
        System.out.println("Yearly repayment: " +
response.getLoan().getLoan().getYearlyRepayment());
        String[] messages =
response.getLoan().getLoan().getMessages();
        if (messages != null) {
            System.out.println("Messages: ");
            for (String message : messages) {
                System.out.println(message);
            }
        }
    }
    catch (Exception e) {
        throw new RuntimeException("An error occurred when
invoking Decision Service at: "
+ endpointURI, e);
    }
}
}

```

Parent topic: [Calling decision service rulesets](#)

Using OpenAPI

You can generate a client from an OpenAPI file that is made for a decision service ruleset.

Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#)).

Important:

You can use Swagger Editor or other [open source Swagger tools](#) to generate a client in your preferred language, for example: C#, Go, Apache Groovy, JavaScript, PHP, Python, Ruby, Scala, Swift, or TypeScript.

About this task

To call a decision service ruleset from a client application, you create proxy classes from an OpenAPI file generated for a ruleset path.

Procedure

1. Obtain a OpenAPI file for a decision service ruleset.
2. Generate proxy classes from the OpenAPI file.
3. Use the proxy classes to invoke the ruleset from your client application.

Example

To get the OpenAPI file from Rule Execution Server:

1. Deploy the decision service ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click a ruleset for your decision service.
4. In Ruleset View, click **Retrieve HTDS Description File**.
5. Select **REST** as a service protocol type.
6. Select **OpenAPI - YAML** or **OpenAPI - JSON** as a format to generate an OpenAPI file in YAML or JSON.
7. Check **Latest ruleset version** and **Latest RuleApp version** to generate the OpenAPI file for the latest versions.
8. Click **Download**.

To generate the proxy classes in Swagger Editor:

1. Open <http://editor.swagger.io/> in a web browser.
2. Click **File > Import File**.
3. Click **Browse** to select the OpenAPI file, and then click **Open**.
4. Click **Import**.
5. Click **Generate Client > Java**.

The following Java™ code sample imports the proxy classes generated in Swagger Editor for a ruleset and calls the ruleset from a Java application:

```
import java.util.List;

import io.swagger.client.ApiClient;
import io.swagger.client.ApiException;
import io.swagger.client.model.Borrower;
import io.swagger.client.model.Loan;
import io.swagger.client.model.Request;
import io.swagger.client.model.Response;

public class DecisionServiceExecution {

    public static void main(String[] args) {

        ApiClient apiClient = new ApiClient();

        // Replace "loginID" with the ID of a user who has access to
the Cloud portal
        apiClient.setUsername("loginID");

        // Replace "password" with the password of a user who has
access to the Cloud portal
        apiClient.setPassword("password");
```

```

DefaultApi api = new DefaultApi(apiClient);

// Create the request
Request request = new Request();

// Set the borrower
Borrower borrower = new Borrower();
borrower.setName("John");
borrower.setCreditScore(600);
borrower.setYearlyIncome(80000);
request.setBorrower(borrower);

// Set the loan
Loan loan = new Loan();
loan.setAmount(500000);
loan.setDuration(240);
loan.setYearlyInterestRate(0.05);
// approved (set to true by default, to be computed by the
decision engine)
loan.setApproved(true);
request.setLoan(loan);

// Retrieve the response
try {
    Response response =
api.callDecisionOperation(request);
    System.out.println("Rules executed.");
    System.out.println("Approved: " +
response.getLoan().getApproved());
    System.out.println("Yearly interest rate: " +
response.getLoan().getYearlyInterestRate());
    System.out.println("Yearly repayment: " +
response.getLoan().getYearlyRepayment());
    List<String> messages =
response.getLoan().getMessages();
    if (messages != null) {
        System.out.println("Messages: ");
        for (String message : messages) {
            System.out.println(message);
        }
    }
} catch (ApiException e) {
    throw new RuntimeException("An error occurred when
invoking Decision Service", e);
}
}
}

```

Parent topic: [Calling decision service rulesets](#)

Using API Connect

You can expose a decision service ruleset to other organizations via IBM API Connect®, which manages access and uses built-in security mechanisms.

Before you begin

You must use authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#)).

Important:

IBM API Connect can also generate client code for Swagger APIs. For more information, see [Viewing and testing APIs in the Developer Portal](#), and for API Connect, see [IBM API Connect](#).

About this task

To expose a decision service ruleset in API Connect, you must generate an OpenAPI file for API Connect, import it into API Connect, and configure the credentials.

Procedure

1. Obtain a OpenAPI file specific to API Connect for a decision service ruleset.
2. Import the OpenAPI file into API Connect, and set the credentials used when calling the ruleset.

Example

To get the OpenAPI file specific to API Connect from Rule Execution Server:

1. Deploy the decision service ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click a ruleset that corresponds to your decision service.
4. In the Ruleset View, click **Retrieve HTDS Description File**.
5. Select **REST** as a service protocol type.
6. Select **OpenAPI - YAML** or **OpenAPI - JSON** as a format to generate an OpenAPI file in YAML or JSON.
7. Check **Latest ruleset version** and **Latest RuleApp version** to generate the OpenAPI file for the latest versions.
8. Check **Proxy for API Connect** to generate an OpenAPI file for a proxy so that the decision service can be called from API Connect.
9. Click **Download**.

To import the OpenAPI file into API Connect and set the credentials:

1. Download and install API Designer for IBM API Connect, or use it on IBM® Bluemix®.
2. In the **APIs** tab in API Designer, click **Add > Import an existing OpenAPI**.
3. Click **Select File**.
4. Navigate to the OpenAPI file, and then click **Open**.
5. Click **Import**.
6. Click the **Assemble** tab to edit the policy assembly.
7. Click the **Calls Decision Service Operation** invoke policy.
8. In the property sheet, enter the user name and password of a user who has access to Operational Decision Manager on Cloud.
9. Click the **Save** icon.

Your decision service ruleset is now ready to be called from API Connect.

You can now continue configuring your API at the API Connect level by adding a custom security.

Parent topic: [Calling decision service rulesets](#)

Authentication for REST and SOAP invocation

Operational Decision Manager on Cloud supports the use of basic authentication in invoking REST and SOAP APIs. Instead of hardcoding login credentials into your application, pull them in from a separate file.

The recommended practice for using login credentials is to keep them in a separate file, and to call them from your application during authentication. With this approach, you do not have to alter the code of your application every time you change your credentials. The primary alternative is to add the credentials directly to your application. This approach risks the introduction of mistakes to your code, and requires you to redeploy the application every time you update the credentials.

The cloud portal offers two types of credentials:

- Service credentials: These credentials are exempt from the password renewal policies of the cloud portal. They are only valid for calling REST and SOAP APIs. Users cannot use them to log in to the portal or its components (see [Service credentials for client applications](#)).
- Local accounts: These user accounts have credentials that are subject to the password renewal policies of the cloud portal. Users and applications can use these credentials to log in to the portal and its components (see [Managing your account](#)).

Tip:

Use service credentials for authenticating your applications. They provide better security, and you can apply your own policies for updating login credentials.

SAML users

You cannot use SAML login credentials to authenticate a decision service. If you use SAML to log in to Operational Decision Manager on Cloud, you must request service credentials for REST or SOAP invocation (see [Getting a set of service credentials](#)). IBM® internal users automatically use SAML to log in to Operational Decision Manager on Cloud, and must request service credentials.

Calling service credentials

The following steps show you how to set up your application to call service credentials from a separate file. Depending on your IT strategy, you can use the credentials of a local account in place of the service credentials.

1. Obtain your service credentials from your cloud administrator. Only a cloud administrator can generate service credentials. The service credentials include a functional ID and a password.
2. Place the service credentials in a property file, for example, `sc.properties`:

```
odmoc.url=https://mytenant.bpm.ibmcloud.com/odm/dev
odmoc.functionalId=api1.fid@t1023
odmoc.password=xfmptNJsQ0NCvRxB1bYS1AxIcYyh1UR2y/LMpyx
```

The properties in this example are defined as follows:

Property	Description
<code>odmoc.url</code>	The URL of your tenant of the cloud portal
<code>odmoc.functionalId</code>	The functional ID in the service credentials
<code>odmoc.password</code>	The password in the service credentials

3. In your application, add the code that calls the service credentials from the file `sc.properties`. The following code is from a Java™ client:

```
package test;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Properties;

import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpResponse;
import org.apache.http.Header;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.client.HttpClients;
```

```

import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.util.EntityUtils;

import com.ibm.json.java.JSONObject;

public class OdmRestClient {

    public static void main(String[] args) throws Exception {
        // The builder should be configured to connect using HTTPS to ODM
on Cloud,
        // and must use a secure cipher with hostname verification. The
following
        // code does not reflect all the required settings.
        HttpClientBuilder builder = HttpClientBuilder.custom();
        builder.disableRedirectHandling();
        HttpClient client = builder.build();

        // This code calls the service credentials from the file
sc.properties.
        //It reads the url, functional Id and password from a property
file.
        String configFile = "sc.properties";
        FileInputStream fistream = null;
        try {
            fistream = new FileInputStream(configFile);
        }
        catch (FileNotFoundException e) {
            System.out.println("Could not open file: " + configFile);
            System.exit(0);
        }
        Properties props = new Properties();
        props.load(fistream);

        String baseUrl = props.getProperty("odmoc.url");
        String fid = props.getProperty("odmoc.functionalId");
        String pwd = props.getProperty("odmoc.password");
        String url = baseUrl + "/res/api/v1/ruleapps?count=true";

        // Basic Authentication header value
        String baHeader = fid + ":" + pwd;
        System.out.println("BA header is: " + baHeader);
        baHeader = Base64.encodeBase64String(baHeader.getBytes());

        // execute request
        HttpResponse response = null;
        System.out.println("** Calling: " + url);

        HttpGet request = new HttpGet(url);
        request.setHeader("Authorization", "Basic " + baHeader);
        try {
            response = client.execute(request);
        }
        catch (Exception e) {
            throw new Exception("Could not execute API call, reason: " +
e.getMessage());
        }

        int httpRespCode = response.getStatusLine().getStatusCode();
        System.out.println("Response code is: " + httpRespCode);

        if (httpRespCode==302) {
            // We are in one of these 3 cases: wrong ID, wrong password,
account locked
            System.out.println("Basic auth access denied.");
            System.out.println("Reason1: Wrong ID or password");
        }
    }
}

```

```
        System.out.println("Reason2: Account locked after some
unsuccessful attempts");
    }
    else if (httpRespCode==401) {
        // Aithorization is not granted to access the URL
        System.out.println("ODM API access was denied: you are not
authorized");
    }
    else if (httpRespCode==200){
        // Correct execution result, extract the HTTP response
        String json = EntityUtils.toString(response.getEntity(),
"UTF-8");
        System.out.println("Successful invocation, result : " +
json);
    }
    else {
        // Diagnose why such a code. Contact IBM support with all the
details.
        System.out.println("Unexpected return code: " +
httpRespCode);
    }
}
}
```

The HTTP client side can return the following codes during authentication:

Code	Description
Code 200	This code denotes a correct execution result. The return result is the actual response to the API call. You can get Code 200 in the case of wrong credentials when the redirect handling is not deactivated. To diagnose a connection problem accurately, deactivate the redirect handling.
Code 302	The HTTP request returns Code 302 when the functional ID or password is wrong, or the account is locked. The code does not provide the cause of the authentication failure. If the account is locked, you can try waiting one hour for the account to be unlocked automatically. If the account is not unlocked automatically, contact your cloud administrator. This code is returned when the redirect handling is deactivated
Code 401	You get this code when the credentials are correct, but they are not authorized to access the URL. You must ask your cloud administrator to grant the required authorization. (Note: This code is not currently returned but authorization could be enhanced in the future.)
Other codes	If you get any other code, contact IBM for assistance (see Troubleshooting and support).

Parent topic: [Integrating business rules](#)

Troubleshooting and support

Use the troubleshooting resources to resolve any technical problem that you might encounter while using Operational Decision Manager on Cloud.

Identifying problems

Start by using this troubleshooting process to try to identify the source of the problem. It includes recording the symptoms, re-creating the problem, and eliminating potential causes.

Support for Operational Decision Manager on Cloud

Customer support is available for the users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.

Limitations on OpenAPI specifications and tools

You might face some restrictions concerning the OpenAPI specifications and tools when you use the hosted transparent decision service (HTDS) through OpenAPI-based clients.

Identifying problems

Troubleshooting is the process of finding and eliminating the cause of a problem. The first steps for identifying a problem include recording the symptoms, re-creating the problem, and eliminating potential causes.

Recording the symptoms

When you get an error message in the product, make sure that you record this message. You might also receive multiple error messages that look similar, but have subtle differences. By recording the details of each one you can learn more about where your problem exists.

You can get error messages in the following sources:

- **Problems** view in the Eclipse workbench
- **Console** view in the Eclipse workbench
- **Error Log** view in the Eclipse workbench
- Error dialogs
- Log files in your workspace: `<workspace>/ .metadata/ .log`

Re-creating the problem

To reproduce the problem, try to identify the steps that you performed. If you have a consistently repeatable test case, you can have an easier time determining what solutions are necessary.

You can ask yourself the following questions:

- How did you first notice the problem?
- What was the first symptom of this problem?
- Were there other symptoms occurring around that time?
- Did you do anything different that made you notice the problem?
- Is this process a new procedure, or has it worked successfully before?
- If this process worked before, what has changed?

The change can refer to any type of change made to the system, ranging from adding new hardware or software, to configuration changes you might have made to existing software.

- Does the same problem occur elsewhere?

The problem could be on one machine only or on multiple machines.

Eliminating possible causes

You can narrow the scope of your problem by eliminating components that are not causing the problem. Consult the information that comes with the product and other available resources to help you with your elimination process.

Support for Operational Decision Manager on Cloud

Customer support is available for the users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.

Before contacting customer support

Start by collecting data on your problem (see [Collecting Data: Read first for all IBM Operational Decision Management Components](#)). Customer support needs this information to help you resolve your problem. In gathering the data, you can refer to online resources that might present a solution to your problem.

Business hours

Support is available during the following hours:

- 8:00 AM - 5:00 PM Eastern Standard Time (EST), Monday through Friday (excluding US holidays)
- 8:00 AM - 5:00 PM Central European Time (CET), Monday through Friday (excluding German holidays)

Submit your questions or issues through the [Client Success Portal](#). You must sign in to use the service.

After hours

For support outside normal business hours, open a ticket in an [IBM Service Request](#). Refer to the following table for severity levels.

Table 1. Severity levels and their definitions

Severity	Severity definition	Response time objectives	Response time coverage
1	Critical business impact or service down: Business critical functionality is inoperable or a critical interface has failed. This usually applies to a production environment and indicates an inability to access services, resulting in a critical impact on operations. This condition requires an immediate solution.	Within 1 hour	24 hours, 7 days a week
2	Significant business impact: A service business feature or function of the service is severely restricted in its use, or the client is in jeopardy of missing business deadlines.	Within 2 business hours	Monday-Friday, business hours
3	Minor business impact: The service or functionality is usable, and there is no critical impact on operations.	Within 4 business hours	Monday-Friday, business hours
4	Minimal business impact: An inquiry or nontechnical request.	Within 1 business day	Monday-Friday, business hours

Online support resources

- [dW Answers for Operational Decision Manager on Cloud](#): Answers common questions that are related to the product.
- [IBM Operational Decision Manager Developer Center](#): Serves as an online community for developers of reusable decisions for smarter processes and applications.
- [IBM Operational Decision Manager on Cloud: Terms of Use](#): Provides information about the service level agreements (SLAs) that apply to technical support.
- [Service requests and PMRs](#): Shows the status of service requests.
- [IBM Software as a Service \(SaaS\) Support Handbook](#): Provides information about support response times.
- [Directory of worldwide contacts](#): Lists the countries where IBM has offices. Click your country in the directory to obtain contact information about local IBM services.

Limitations on OpenAPI specifications and tools

You might face some restrictions concerning the OpenAPI specifications and tools when you use the hosted transparent decision service (HTDS) through OpenAPI-based clients.

Symptoms

You might encounter an issue in some specific use cases due to an incorrect behavior in some of the OpenAPI reference tools.

Note: You might encounter some of the issues that are described in the Table 1 when you use one or more of the following tools:

- swagger-core Java™ library (embedded in the product): V1.5.10
- Swagger Codegen code generation tool: V2.2.1
- Swagger Editor: V2.10.3

Diagnosing the problem

Table 1. Data types and limitations

Data type	Limitation
short type	<p>The OpenAPI specification has only <code>int32</code> and <code>int64</code> representations for integer, and standard integer numbers (<code>int</code> and <code>java.lang.Integer</code>) and long numbers (<code>long</code> and <code>java.lang.Long</code>) respectively for mapping.</p> <p>Short numbers (<code>short</code> and <code>java.lang.Short</code>) are mapped to <code>int32</code> and treated as regular integer numbers.</p> <p>As a consequence, when you use <code>short</code> or <code>java.lang.Short</code> in rule projects or decision services, clients or interfaces that are generated from the OpenAPI definition file provided by the HTDS allow you to input integer numbers instead of short numbers. Thus, entering a number bigger than the max value for short ends up in an execution error.</p>
password data type annotation	<p>Customizing Java XOM fields with the <code>@ApiModelProperty</code> annotation might be reflected on the OpenAPI definition file that is generated by the HTDS.</p> <p>In particular, you might use the datatype property to specify or override the type of the field of the Java object in OpenAPI. However, if you set password as a data type, the OpenAPI definition file might not be generated. You will see the following error:</p> <div>Unrecognized Type: [null]</div> <p>This problem is due to an incorrect behavior in the swagger-core Java library that is used for generating OpenAPI definition files. Avoid using the password data type.</p> <p>As a workaround, you can use a <code>String</code> field without an annotation (or at least without the password data type), and modify the generated OpenAPI definition file to specify <code>"format": "password"</code> in the corresponding field. It works with the Swagger Editor (the test form takes it into account, hiding characters as they are entered), and the Swagger Codegen tool can generate a client properly (the field is represented with a standard <code>String</code> type).</p>
binary data type annotation	<p>Similar to the password data type, a <code>String</code> field in a Java XOM can be annotated with the <code>@ApiModelProperty</code> annotation having</p>

	<p>binary as a data type property.</p> <p>Unlike the password data type, this binary data type does not prevent the OpenAPI definition file from being properly generated by the HTDS.</p> <p>However, if you use the Swagger Codegen tool to generate a Java client, the annotated field is represented as a <code>byte[]</code> field, which leads to a ruleset execution error at run time.</p> <p>In this particular use case, no such issue arises when you use the Swagger Editor.</p>
byte type	<p>When <code>byte</code> or <code>java.lang.Byte</code> is used as a XOM field or as a ruleset parameter, it becomes <code>byte[]</code> in a Java client that is generated by the Swagger Codegen tool, which leads to a ruleset execution error at run time.</p> <p>In this particular use case, no such issue arises when you use the Swagger Editor, since bytes are represented as strings, and the HTDS REST accepts and converts them to real bytes.</p>
arrays	<p>Arrays become lists when you use the Swagger Codegen tool to generate Java clients.</p> <p>For example, a field of type <code>String[]</code> becomes <code>List<String></code>.</p>

Trial version

The no-charge, 30-day trial version introduces you to Operational Decision Manager on Cloud.

The trial version contains the full version of Operational Decision Manager on Cloud, and is kept up to date with the latest features. You use it anonymously and securely, without risk of exposure to users from other organizations. You can take a decision service or a decision model service completely through its lifecycle.

Note: The platform is fully contained, so you cannot connect it to a production application outside the trial instance.

Introduction

The trial version gives you an opportunity to work on an instance of Operational Decision Manager on Cloud. It provides all the components you need to create, develop, and deploy a decision service or a decision model service. If you use the trial version with other users from your organization, you can discover the project management features along with the development features.

You can also use a sample client application to call business rules from an execution server. The application is included in the download files, and works with the product tutorials. You can see how to connect decision services with client applications.

While the trial version is fully functional, you cannot use it to develop rule applications for the production applications of your organization. You can use the tutorials and sample application to get a feel for the platform.

Access to the trial version is limited to 30 days. After 30 days, you must register again to continue using the trial version.

You use the trial version by following the information in this knowledge center. For additional information, you can contact the cloud team or visit [IBM Operational Decision Manager Developer Center](#). There is no customer support for the trial version, and you cannot obtain assistance through the Client Success Portal or IBM Service Request.

Important: The trial version includes data security and privacy (see [Data Processing and Protection Datasheet](#)).

Signing up

To use the trial version:

1. Sign up for the free trial version at [IBM® Operational Decision Manager on Cloud Trial](#). Your application form is checked, and an invitation is sent to your email address.
2. Follow the instructions in the invitation:

IBM Operational Decision Management on Cloud is available now

[Click here to activate your account](#)

Or copy and paste this link into your browser
<https://www.blueworksccloud.com/login/?token=0240b89a-0077-4652-8adb-434d65491e55&activation>

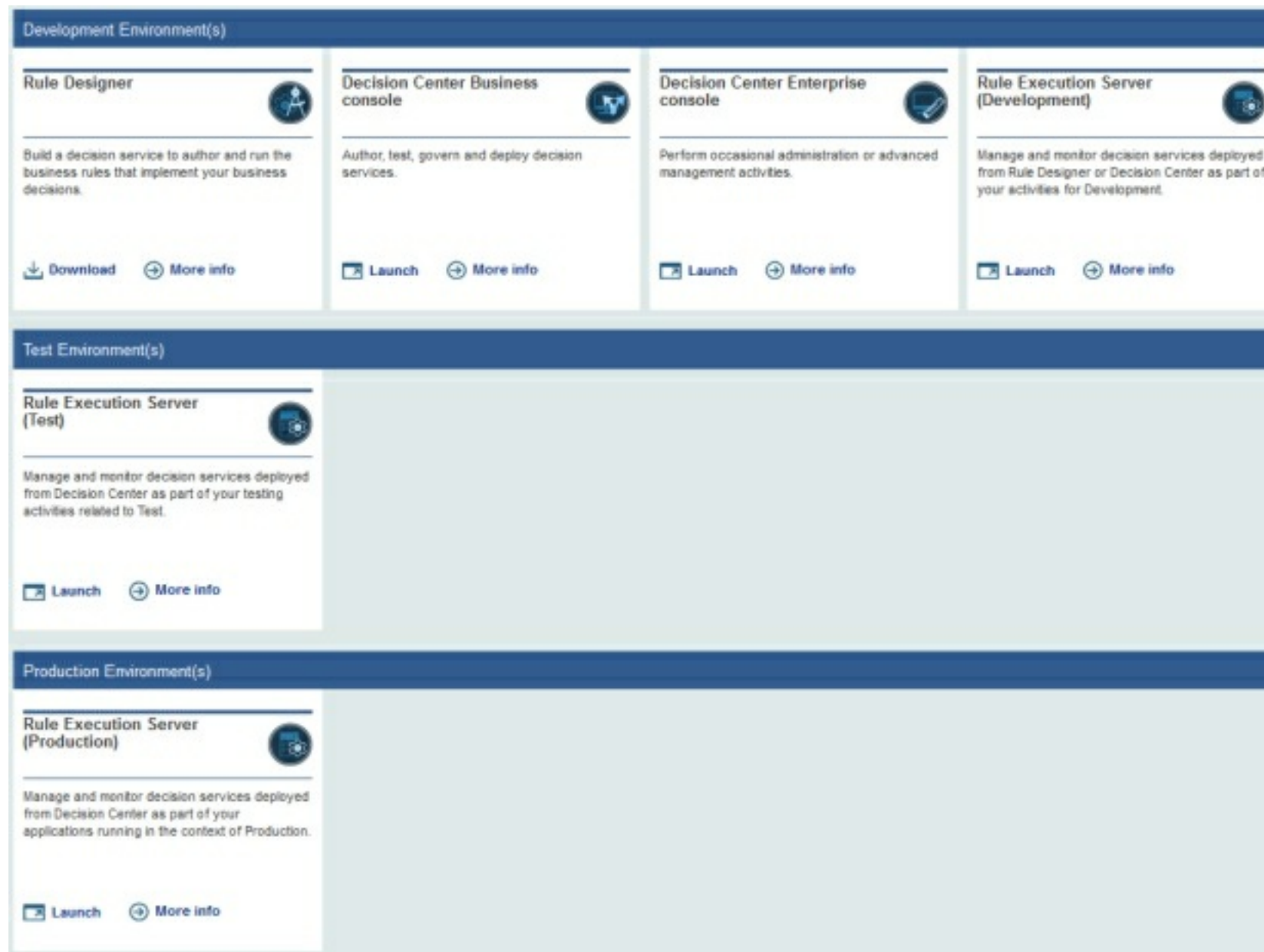
"Use the link to validate your email address and activate your account. Then, you can access the service at
<https://vhost108.blueworksccloud.com>

Use your email address as your login name.

3. When the Activate Your Account page opens, enter the requested information and click **Activate**. A message informs you that your account is being provisioned.

Tip: When prompted to enter a password, follow the guidelines at [Resetting your password](#).

4. Enter your email address as your user ID in the sign-in page, and click **Continue**.
5. Enter your password, and then click **Continue** again. The trial instance of Operational Decision Manager on Cloud opens. It contains the full version of the portal with the development, test, and production environments (see [Cloud environments](#)):



You can now use the trial version.

Tutorials

Use the following tutorials to become familiar with Operational Decision Manager on Cloud:

- [Getting started in Operational Decision Manager on Cloud](#): This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.
- [Creating a decision service in Rule Designer](#): This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.
- [Getting started with decision modeling](#): This tutorial shows the basics of creating a decision model service. You do all the modeling and authoring of a decision service in the Decision Center Business console.

You can find more tutorials at [Tutorials](#).

Reference

A set of reference topics for IBM® Decision Server.

[Rule Designer reference](#)

Refer to these topics for libraries, rule languages, and user interfaces.

[Decision Center reference](#)

Refer to these topics for messages.

[Rule Execution Server reference](#)

Refer to these topics for predefined ruleset and RuleApp properties.

Rule Designer reference

A set of reference topics that includes libraries, rule languages, and user interfaces.

[File types](#)

It can be useful to understand the different file types available in a rule project.

[Rule languages](#)

Decision Server provides three rule languages: Business Action Language (BAL), ILOG® Rule Language (IRL), and Advanced Rule Language (ARL).

[Messages](#)

Each listed message provides a detailed description about the error message and some steps to try and resolve the error.

File types

It can be useful to understand the different file types available in a rule project.

XML rule artifact files

The different types of XML rule artifact file all have the same structure.

BOM files

A BOM file (. bom) is a text file that defines part of the business object model (BOM)

Vocabulary files

A vocabulary file (_<locale>.voc) is a key-value property file that stores the verbalization data for the BOM.

BOM to XOM mapping files

A BOM to XOM mapping file (. b2x) is an XML file that stores the mapping between the BOM and the XOM.

Rule project files

A rule project file (. ruleproject) is an XML file that stores the properties that are specific to a rule project.

Parent topic: [Rule Designer reference](#)

XML rule artifact files

The different types of XML rule artifact file all have the same structure.

XML rule artifact files have the following extension: .rulepackage, .brl, .dta, .dtr, .fct, .rfl, .trl, .var, .brt, .qry. A description of each file type is provided in [Rule project items](#).

All XML rule artifact files are based on the following structure:

XML version header
Element type
Name
UUID
General properties (one per line)
Specific properties

Parent topic: [File types](#)

Related information:
[Rule project items](#)

BOM files

A BOM file (.bom) is a text file that defines part of the business object model (BOM)

The format of a BOM file is similar to the Java™ class definition format.

Here is an example of a .bom file:

```
public class MyClass
{
    public int attribute;
    public ilog.rules.brl.String method(float arg);
}
```

Parent topic: [File types](#)

Related information:

[business object model \(BOM\)](#)

[Rule project items](#)

Vocabulary files

A vocabulary file (`_<locale>.voc`) is a key-value property file that stores the verbalization data for the BOM.

You have one vocabulary file per locale, so a BOM entry can be associated with several vocabulary files. All the vocabulary files are published to Decision Center , which displays the rules in the user locale.

Here is an example of a `_<locale>.voc` file:

```
# MyClass
MyClass#label = my class
MyClass.attribute#sentence.action = set the attribute of {this} to {attribute}
MyClass.attribute#sentence.navigation = {attribute} of {this}
MyClass.method(float)#sentence.navigation = {this}.method({0})
```

Parent topic: [File types](#)

Related information:

[Vocabulary](#)

[Rule project items](#)

BOM to XOM mapping files

A BOM to XOM mapping file (.b2x) is an XML file that stores the mapping between the BOM and the XOM.

Here is an example of a BOM to XOM mapping file:

```
<translation>
  <class>
    <businessName>MyClass</businessName>
    <executionName>java.util.Vector</executionName>
    <import>java.util.Vector</import>
    <tester language="irl"><![CDATA[
      String type = (String)this.get(0);
      return type.equals("MyClass");
    ]]></tester>
    <attribute>
      <name>attribute</name>
      <getter language="irl"><![CDATA[
        Integer val = (Integer)this.get(1);
        return val.intValue();
      ]]></getter>
      <setter language="irl"><![CDATA[
        this.set(1, new Integer(value));
      ]]></setter>
    </attribute>
  </class>
</translation>
```

Parent topic: [File types](#)

Related information:

[Defining BOM to XOM mapping](#)

[Rule project items](#)

Rule project files

A rule project file (.ruleproject) is an XML file that stores the properties that are specific to a rule project.

Here is an example of a rule project file:

```
<?xml version="1.0" encoding="UTF-8"?>
<model.base:RuleProject xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:model.base="http://ilog.rules.studio/model/base.ecore"
xmlns:model.bom="http://ilog.rules.studio/model/bom.ecore"
xmlns:model.xom="http://ilog.rules.studio/model/xom.ecore">
  <name>rproject</name>
  <uuid>_srANAD1JEdqCPM8pR7aTDA</uuid>
  <outputLocation>output</outputLocation>
  <categories>any</categories>
  <paths xsi:type="model.xom:XOMPath" pathID="XOM"/>
  <paths xsi:type="model.bom:BOMPath" pathID="BOM"/>
  <modelFolders xsi:type="model.base:SourceFolder">
    <name>rules</name>
  </modelFolders>
  <modelFolders xsi:type="model.bom:BOMFolder">
    <name>bom</name>
  </modelFolders>
</model.base:RuleProject>
```

The structure of the files has been designed to be handled easily by diff tools, which you use to identify differences between different versions of the file. Being able to identify differences is an important feature when using a source code control system or when updating files from Decision Center.

Parent topic: [File types](#)

Related information:
[Rule project items](#)

Rule languages

Decision Server provides three rule languages: Business Action Language (BAL), ILOG® Rule Language (IRL), and Advanced Rule Language (ARL).

Business Action Language (BAL)

You use constructs to build rules. You use operators to do arithmetic operations, associate or negate conditions, and compare expressions. You use literals to declare values as being of a particular type. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

ILOG Rule Language (IRL)

The ILOG Rule Language (IRL) contains a set of keywords, and has its own syntax to structure each part of the rule. IRL does not support generics because IRL is not Java™. As a result you cannot use generics in: IRL, functions, BOM to XOM mapping, and initial/final actions.

Advanced Rule Language (ARL)

For each Business Action Language (BAL) expression that you create for BOM-to-XOM mapping in the decision engine, you can review its equivalent Advanced Rule Language (ARL) expression. ARL is a read-only rule language that is similar in syntax to Java 7.

Parent topic: [Rule Designer reference](#)

Business Action Language (BAL)

You use constructs to build rules. You use operators to do arithmetic operations, associate or negate conditions, and compare expressions. You use literals to declare values as being of a particular type. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

BAL constructs

You use BAL constructs to build business rules.

BAL operators

You use operators in rule statements to do arithmetic operations, associate or negate conditions, and compare expressions.

BAL literals

The Business Action Language (BAL) supports number, string, date, and time data types.

Punctuation in rules

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules (for example, in the Intellirule editor), mandatory punctuation is usually predicted in the completion menu.

Parent topic: [Rule languages](#)

BAL constructs

You use BAL constructs to build business rules.

all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

called (variable)

This construct provides a name for a variable declared in a `for each action` statement.

definitions

This construct introduces the part of the rule where you define variables.

else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

for each (object) in (list)

This construct specifies the actions that must be performed for every element of a collection.

from (object)

This construct accesses data from objects that are related to known objects.

if

This construct introduces the part of the rule in which you define conditions.

in (list)

This construct accesses data from object collections that are related to known objects.

it is not true that (a condition)

This construct negates a condition statement.

none of the following conditions are true

This construct negates a group of conditions.

print (string)

This construct prints a string, phrase or rule name to the standard output.

set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

set (variable) to (value)

This construct changes the value of a ruleset parameter or variable.

the name of this rule

This construct returns the name of the currently executed rule in the `then` part of a business rule.

the number of (objects)

This construct returns the number of objects in the current data set or in the specified collection.

then

This mandatory construct introduces the part of the rule that defines the actions that are executed if the `if` part of a rule is satisfied.

there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the current data set or the specified collection.

there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the current data set or the specified collection.

there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in the current data set.

there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

there is at least one (object)

This construct tests whether there is at least one object of a given type in the current data set.

there is at most one (object)

This construct tests whether there is at most one object of a given type in the current data set.

there is no (object)

This construct tests whether a data set contains no objects of the given type.

there is one (object)

This construct tests whether the current data set contains one and only one object of a given type.

where (test)

This construct matches objects against tests.

Parent topic: [Business Action Language \(BAL\)](#)

Related information:
[Working with action rules](#)

all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

Purpose

The construct `all of the following conditions are true` provides a more compact and convenient alternative to the logical operator `and` when you want to combine multiple conditions. The resulting statement is considered true only if each of the listed conditions is true.

Syntax

```
all of the following conditions are true:  
  - <condition>* [,]
```

Description

This is equivalent to writing `if <condition 1> and <condition 2> and ... <condition n>`. However, if you have a large number of conditions, using the `all of the following conditions are true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This separates those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is Gold junior member.

```
if  
  all the following conditions are true:  
    - the category of the customer is Gold  
    - the age of the customer is at most 15  
then...
```

Parent topic: [BAL constructs](#)

any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

Purpose

The construct `any of the following conditions are true` provides a more compact and convenient alternative to the logical operator `or` when you want to combine multiple conditions. The resulting statement is considered true if at least one of the listed conditions is true.

Syntax

```
any of the following conditions is true:
- <condition>* [,]
```

Description

This is equivalent to writing `if <condition 1> or <condition 2> or ... <condition n>`. However, if you have a large number of conditions, using the `any of the following conditions is true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Note:

Indentation in rules is for readability only; it does not influence the way the rule is processed.

Example

This example shows how to test if a customer is a Gold member or a senior customer.

```
if
  any of the following conditions is true:
    - the category of the customer is Gold
    - the age of the customer is more than 60
then...
```

The following more complex example shows the use of a comma to mark the end of a group of conditions. In this example, the order must be over \$50, the customer must be either a Gold customer or a senior customer, and the order must be shipping to NJ.

```
if
  all of the following conditions are true:
    - the order total is greater than $50
    - any of the following conditions is true:
      - the category of the customer is Gold
      - the age of the customer is more than 60,
    - the shipping address is in NJ
then...
```

Without the comma to mark the end of the `any of the following conditions are true` block, the final condition (the shipping address is in NJ) would be considered part of the previous group, and the rule would be interpreted to mean that the order must be over \$50 and (either the customer is a Gold customer or a senior customer or the shipping address is in NJ).

Parent topic: [BAL constructs](#)

called (variable)

This construct provides a name for a variable declared in a `for each` action statement.

Purpose

You use this construct in the outer loop of nested `for each` loops so that you can reference the named item from within the inner loop.

Syntax

```
called <variable>,
```

Description

When using a `for each` statement to perform an action on a set of items, a variable is created automatically to allow you to refer to each item in turn as 'this item' (see the first example below). However, if you nest one `for each` loop inside another, you must give the variable in the outer loop a name so that you can reference it from within the inner loop (see the second example below).

Example

The following rule shows a simple `for each` action statement. There is no need to use the `called` construct in this case.

```
if
  the category of the customer is Gold
then
  for each customer in 'senior customers':
    - send flowers to this customer;
```

The following rule shows two nested `for each` statements. The `called` construct is used so that each customer can be explicitly referred to from within the nested `for each` loop.

```
if
  the category of the customer is Gold
then
  for each customer called c, in 'senior customers':
    - for each account in 'new accounts':
      - send a statement for this account to the mailing address of c;
```

Parent topic: [BAL constructs](#)

Related reference:

[else](#)

[then](#)

[for each \(object\) in \(list\)](#)

definitions

This construct introduces the part of the rule where you define variables.

Purpose

You use this construct to define local variables that you can use elsewhere in the same rule. Use local variables to produce more concise and readable rules.

Syntax

```
definitions
    set <variable> to <definition> [in <list> / from <object>] [where <test>*]
;*
```

Description

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules. You define local variables using the [set \(variable\) to \(definition\)](#) construct.

You can also define global input/output variables (called ruleset parameters) at the application or rule project level. Ruleset parameters are accessible to any of the rules in your project. Ruleset parameters are created and edited in Rule Designer.

Example

The following example declares the local variable 'preferred customer' in the definitions part of the rule and uses it in the if and then parts of the rule.

```
definitions
    set 'preferred customer' to a customer;
if
    the age of 'preferred customer' is less than 18
then
    apply a 10% discount to the shopping cart of 'preferred customer';
```

Parent topic: [BAL constructs](#)

Related reference:

[from \(object\)](#)

[in \(list\)](#)

[set \(variable\) to \(definition\)](#)

[the number of \(objects\)](#)

[where \(test\)](#)

else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

Purpose

You use this construct to declare actions that are executed if the `if` part of a rule has not been satisfied. These actions are ignored if all local variables are not correctly initialized due to preconditions in the definitions part failing to be satisfied.

Syntax

```
else
    <action>;*
```

Description

The `else` part of a rule is optional and allows you to specify one or more actions to perform if the conditions in the `if` part of the rule are not met.

The actions in the `else` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

If one or more variables with preconditions are defined in the definitions part of a rule, the rule will only be processed if these preconditions are satisfied and all local variables are correctly initialized. This means that if these preconditions are not satisfied, the rule is not be processed at all, and the actions in the `else` part of the rule are never executed, whether or not the conditions in the `if` part of the rule are met.

Adding preconditions in the definitions part of a rule rather than in the `if` part of a rule can provide a small performance improvement, since it potentially reduces the number of conditions to be checked. However, since the logic of the rule is not the same in each case, the method to use depends on the result you want to obtain.

Example

In the following basic example, a customer will get a 10% discount if the customer is in the Gold category. Otherwise, the customer gets a 5% discount.

```
definitions
    set 'valued customer' to a customer
if
    the category of 'valued customer' is Gold
then
    apply a 10% discount;
else
    apply a 5% discount;
```

In the following example, a second condition has been added in the `if` part of the rule. Now, a customer receives a 10% discount only if the customer is in the Gold category and is aged over 20. All other customers receive a 5% discount.

```
definitions
    set 'valued customer' to a customer
if
    the category of 'valued customer' is Gold and 'valued customer' is older
    than 20
then
    apply a 10% discount;
else
    apply a 5% discount;
```

Now consider a third example. In the following rule, a minimum age requirement is added as a pre-condition in the definitions part of the rule. This means that a customer must first be over 20 years old to be considered a valued customer. Otherwise, the 'valued customer' variable is not initialized and the rest of the rule is not executed. With this rule, a customer under 20 receives no discount at all. A customer who is over 20 and in the Gold category receives a 10% discount. The `else` part of the rule is executed and a 5% discount applied only if the customer is over 20 but not in the Gold category.

```
definitions
    set 'valued customer' to a customer
    where this customer is older than 20;
```

```
if
  the category of 'valued customer' is Gold
then
  apply a 10% discount;
else
  apply a 5% discount;
```

Parent topic: [BAL constructs](#)

Related reference:
[then](#)
[for each \(object\) in \(list\)](#)
[set \(variable\) to \(value\)](#)

for each (object) in (list)

This construct specifies the actions that must be performed for every element of a collection.

Purpose

You use this construct to apply one or more actions to all objects in a collection.

Syntax

```
for each <object> [called <variable>,) in <list>:  
  - <action>*;
```

Description

The construct should be immediately followed by a colon (:) and a list of one or more actions, each on a separate line, each preceded by a dash. End each action statement with a semi-colon (;).

The implicit variable 'this <object>' can be used in each action statement within the for each loop to refer to the current object. If you nest a for each statement inside another, you can use the called construct to define an explicit name for the object in the 'outer' for each loop in order to reference it clearly from within the nested for each loop.

Example

The following rule shows how to apply a discount to all the expensive items in a customer's shopping cart, if the customer is a Gold customer.

```
definitions  
  set 'smith' to a customer ;  
  set 'expensive items' to all items  
    in the items of the shopping cart of smith  
    where the price of 'each item' is greater than 1000;  
if  
  the category of smith is Gold  
then  
  for each item in 'expensive items':  
    - apply 5% discount to this item;
```

Parent topic: [BAL constructs](#)

Related reference:

[else](#)

[then](#)

[called \(variable\)](#)

from (object)

This construct accesses data from objects that are related to known objects.

Purpose

You use this construct to expand the set of data that can be used by a rule.

Syntax

```
from <object>
```

Description

By default, rules are used to process a set of data established by the developer of an application. If you need to access data that is not in this set, but related to a data object in this set, you can use the `from` construct to define a variable that pulls a data object in from a related object.

The `from` construct is used to retrieve one distinct object from another one. You cannot use it to retrieve a collection of objects. To access a collection of objects, use the [in \(list\)](#) construct.

Example

The following rule declares a variable based on the relationship between a customer and the customer's preferred item.

```
definitions
  set 'preferred CD' to a CD
    from the preferred item of the customer;
if
  the price of 'preferred CD' is more than 25
then...
```

By default, the rule engine does not know what a "CD" object is. To write a rule that uses a CD object, you must first declare a variable of type CD that pulls in data from the preferred item object of the customer. This is possible because the customer object (and therefore the customer's preferred item) is already known to the rule engine.

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[definitions](#)

if

This construct introduces the part of the rule in which you define conditions.

Purpose

You use this construct in a rule to define one or more condition statements.

Syntax

```
if
    <condition>*
```

Description

If the conditions in the `if` part are met, the actions in the `then` part of the rule are executed. The `if` part of a rule is optional. If there is no `if` part, all actions in the `then` part of the rule are performed each time the rule is executed.

If the conditions in the `if` part of the rule are not met, the actions in the `else` part of the rule are executed. If the conditions are not met and the rule does not have an `else` part, the rule does nothing.

Example

The following rule shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if
    the age of the customer is less than 18
then
    apply 10% discount to the shopping cart of the customer
```

Parent topic: [BAL constructs](#)

Related reference:

[from \(object\)](#)

[in \(list\)](#)

[the number of \(objects\)](#)

[where \(test\)](#)

[there are \(number\) \(objects\)](#)

[there is at most one \(object\)](#)

[there is no \(object\)](#)

[there is one \(object\)](#)

in (list)

This construct accesses data from object collections that are related to known objects.

Purpose

You use this construct to expand the set of data that can be used by a rule.

Syntax

```
in <list>
```

Description

By default, rules are used to process a set of data established by the developer of an application. If you need to access data that is not in this set, but related to data in this set, you can use the `in` construct to define a variable that pulls data in from a related data collection.

The `in` construct is used to retrieve data from a collection of objects related to an existing object. The `<list>` parameter of the `in` construct must be an expression that denotes a collection. To retrieve one distinct object from another distinct object, use the [from \(object\)](#) construct.

The `in` construct can be used in the definitions part of a rule and in count conditions specified in the `if` part of a rule.

Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
  set 'item' to a CD
    in the items of the shopping cart of the customer ;
if
  there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
  there is an item
    in the items of the shopping cart of the customer ;
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[definitions](#)
[there are \(number\) \(objects\)](#)
[there is at most one \(object\)](#)
[there is no \(object\)](#)
[there is one \(object\)](#)

it is not true that (a condition)

This construct negates a condition statement.

Purpose

You use this construct when there is no sentence in the vocabulary or no operator to express the opposite of a condition.

Syntax

```
it is not true that <condition>
```

Description

When there is no sentence in the vocabulary or no operator to express the opposite of a condition, you can insert the `it is not true that` construct in front of an existing condition to negate it.

This can be useful in cases where a boolean (true/false) attribute defined in your vocabulary is verbalized as an expression such as 'customer is a loyalty member', but where you want to test for the opposite of this expression ('customer is not a loyalty member'). If the opposite expression has not been defined in your vocabulary, you could write 'it is not true that customer is a loyalty member' to achieve the required result.

In general, the logic of a negative expression is more difficult to interpret, so it is advisable to avoid them where possible.

Example

The following example uses the `it is not true that` construct to check the age of the customer, assuming that the boolean property 'is a minor' has been verbalized in the vocabulary, but that the opposite of the expression 'is a major' has not been defined.

```
if
  it is not true that the Customer is a minor
```

The following example shows how to negate a set of conditions that have been grouped using the `all of the following conditions are true` construct. The resulting expression will return True except if the customer is both a gold member and over 60.

```
if
  it is not true that all of the following conditions are true:
    - the category of the Customer is gold
    - the age of the customer is more than 60
```

Parent topic: [BAL constructs](#)

none of the following conditions are true

This construct negates a group of conditions.

Purpose

You use this construct to groups together a series of condition statements that you want to negate.

Syntax

```
none of the following conditions are true:  
  - <condition>* [,]
```

Description

The group evaluates to true only if none of the conditions listed are true.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is not a Gold senior member and not under 60.

```
if  
  none of the following conditions are true:  
    - the category of the customer is gold  
    - the age of the customer is least 60  
then...
```

Parent topic: [BAL constructs](#)

print (string)

This construct prints a string, phrase or rule name to the standard output.

Purpose

You use this construct to define an action phrase that by default outputs a string to the standard output.

Syntax

```
print "<string>;
```

Description

You can modify this behavior by setting a note handler on the note method of the rule engine.

Example

The following action prints the string "Hello World!" to standard output.

```
print "Hello World!" ;
```

Parent topic: [BAL constructs](#)

Related reference:
[the name of this rule](#)
[the number of \(objects\)](#)

set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

Purpose

You use this construct in the definitions part of a rule to declare a local variable.

Syntax

```
set '<variable>' to <definition> [in <list> / from <object>] [where <test>*] ;
```

Description

Variables produce more concise rules by replacing a value or the result of an expression with a short, convenient identifier. A local variable can be used anywhere within the rule in which it is defined, but is not available in other rules.

Enclose the name of each variable in single quotes. This is compulsory for variable names that contain spaces. While it is not essential to enclose one-word variable names in single quotes, it makes them easier to identify and to reduce the risk of confusion.

Example

The following examples show how to define a variable as an object.

```
definitions
  set 'i' to an item;
  set 'house' to a house
    where the price of this house is more than 1000;
```

The following examples show how to define a variable as a literal, string, or collection.

```
definitions
  set 'category' to Gold ;
  set 's' to "a string";
  set 'expensive items' to all items
    in the items of the shopping cart of the customer
    where the price of each item is more than 200;
```

The following example shows how to define a variable as the result of an expression.

```
definitions
  set 'expr' to the price of the car of the customer + 100;
  set 'h2' to the house of the customer
    where the price of this house is more than 1000;
```

The following example shows how to define a variable as another variable.

```
definitions
  set 'expr2' to 'expr';
```

Parent topic: [BAL constructs](#)

Related reference:

[from \(object\)](#)

[in \(list\)](#)

[where \(test\)](#)

[definitions](#)

set (variable) to (value)

This construct changes the value of a ruleset parameter or variable.

Purpose

You use this construct to modify the value of a ruleset parameter or variable in the action part of a rule.

Syntax

```
set <variable> to <value>;
```

Description

The variable can be set to a literal value, an expression, or a list.

Example

The following action changes the value of a ruleset parameter to the age of a customer.

```
then  
    set param1 to the age of the customer;
```

Parent topic: [BAL constructs](#)

Related reference:

[else](#)
[then](#)

the name of this rule

This construct returns the name of the currently executed rule in the then part of a business rule.

Purpose

You use this construct to provide an audit trail, for example, by logging a message that quotes the name of the rule whenever it is executed.

Syntax

```
the name of this rule
```

Description

This construct displays the name of the current rule as a string. It can be useful as a means of providing an audit trail. Each time a rule is executed, you can log a message that this rule was executed.

The variable is of type String and contains the name of this rule. It is only available in the action part of the rule.

Example

The following action would return the name of the executed rule, Retired_Customer_Policy for example.

```
if
    the age of the customer is greater than 65
then
    print the message "A 5% discount has been applied to policy: " + the name of this rule;
```

Parent topic: [BAL constructs](#)

Related reference:

[else](#)

[then](#)

[for each \(object\) in \(list\)](#)

the number of (objects)

This construct returns the number of objects in the current data set or in the specified collection.

Purpose

You use this construct to count the number of objects of the specified type and return the total as a number.

Syntax

```
the number of <objects> [in <list>] [where <test>,*]
```

Description

You can use the optional `in <list>` clause to count the objects in a specific list. You can use one or more optional `where` clauses to apply additional conditions to the objects that are included in the returned count.

This construct cannot be used in the definitions part of the rule.

Example

The following condition is met if there are at least 6 vehicles:

```
if
    the number of vehicles is more than 5
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[in \(list\)](#)
[definitions](#)

then

This mandatory construct introduces the part of the rule that defines the actions that are executed if the `if` part of a rule is satisfied.

Purpose

You use this construct to introduce one or more actions to be executed if the conditions of the rule are met.

Syntax

```
then  
    <action>;*
```

Description

Every rule must have a `then` part. If a rule does not have an `if` part, the actions in the `then` part of the rule are always executed whenever the rule is executed.

The actions in the `then` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

Example

The following example shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if  
    the age of the customer is less than 18  
then  
    apply 10% discount to the shopping cart of the customer;
```

Parent topic: [BAL constructs](#)

Related reference:

[else](#)

[for each \(object\) in \(list\)](#)

[set \(variable\) to \(value\)](#)

there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the current data set or the specified collection.

Purpose

You use this construct to determine whether the current data set contains exactly the specified number of occurrences of a particular object.

Syntax

```
there are <number> <objects> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests if the number of Gold customers is equal to 8.

```
if
    there are 8 customers
        where the category of each customer is Gold
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[in \(list\)](#)

there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the current data set or the specified collection.

Purpose

You use this construct to determine whether the current data set contains at least the specified number of occurrences of a particular object.

Syntax

```
there are at least <number> <objects> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are at least 10 gold customers.

```
if
  there are at least 10 customers
    where the category of each customer is gold
then...
```

Parent topic: [BAL constructs](#)

there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in the current data set.

Purpose

You use this construct to determine whether the current data set or the specified collection contains no more than the specified number of occurrences of a particular object.

Syntax

```
there are at most <number> <objects> [in <list>] [where <test>,*]
```

Description

Use the optional in clause to apply the test to a specific collection of objects. Use one or more optional where clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the if part of a rule.

Note:

The <number> argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than or equal to 3.

```
if
  there are at most 3 customers
    where the category of each customer is Gold
then...
```

Parent topic: [BAL constructs](#)

there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the current data set or the specified collection contains fewer than the specified number of occurrences of a particular object.

Syntax

```
there are less than <number> <objects> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than 3.

```
if
    there are less than 3 customers
        where the category of each customer is Gold
then...
```

Parent topic: [BAL constructs](#)

there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the current data set or the specified collection contains more than the specified number of occurrences of a particular object.

Syntax

```
there are more than <number> <objects> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are more than 10 customers with the gold category.

```
if
    there are more than 10 customers
        where the category of each customer is Gold,
then...
```

Parent topic: [BAL constructs](#)

there is at least one (object)

This construct tests whether there is at least one object of a given type in the current data set.

Purpose

You use this construct to determine whether the current data set or the specified collection contains at least one occurrence of a particular object.

Syntax

```
there is at least one <object> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests whether or not a Customer object exists.

```
if
    there is at least one customer
        where...
then...
```

The following condition tests if a senior Gold customer exists.

```
definitions
    set 'gold customers' to all customers
        where the category of each customer is Gold;
if
    there is at least one customer in 'gold customers'
        where the age of this customer is at least 60,
then...
```

Parent topic: [BAL constructs](#)

there is at most one (object)

This construct tests whether there is at most one object of a given type in the current data set.

Purpose

You use this construct to determine whether the current data set or the specified collection contains zero or one occurrence of a particular object.

Syntax

```
there is at most one <object> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests if there is at most one Gold customer in the set of objects provided to the rule engine for execution.

```
if
  there is at most one customer
    where the category of this customer is Gold,
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[in \(list\)](#)

there is no (object)

This construct tests whether a data set contains no objects of the given type.

Purpose

You use this construct to determine whether the current data set or the specified collection contains no occurrences of a particular object.

Syntax

```
there is no <object> [in <list>] [where <tests>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests that there is no object of type `Customer` in the object set provided to the rule engine for execution.

```
if
    there is no customer where...
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[in \(list\)](#)

there is one (object)

This construct tests whether the current data set contains one and only one object of a given type.

Purpose

You use this construct to determine whether the current data set or the specified collection contains one and only one occurrence of a particular object.

Syntax

```
there is one <object> [in <list>] [where <test>,*]
```

Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests if there is one Gold customer in the set of objects provided to the rule engine for execution.

```
if
    there is one customer
        where the category of this customer is Gold
then...
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[where \(test\)](#)
[in \(list\)](#)

where (test)

This construct matches objects against tests.

Purpose

Matches objects against tests.

Syntax

```
where <test>[,]*
```

Description

In the definitions part of a rule, you can use one or more where clauses with the set construct to test that an object meets one or more conditions in order to be a valid value for the variable being declared. In the definitions part of the rule, the where statement does not end with a comma. Instead, each set construct should end with a semi-colon (;).

In the if part of a rule, you can use one or more where clauses in count conditions (there is more than, there is less than, and so on) to test that an object meets one or more conditions before being counted.

In a where construct, an implicit variable that refers to the current instance of the object being tested is automatically declared. In the test, you can thus refer to potential instances of the variable as this <object type> (or each <object type> if they are part of a collection.)

Example

The following example shows the definitions part of a rule that defines the local variable 'expensive items' as a collection of items whose price is greater than 100. The implicit variable each item is used in the where clause to refer to each item object in the collection.

```
definitions
  set 'expensive items' to all items
    where the price of each item is more than 100;
```

The following example shows the where clause used together with the there is one construct in the if part of a rule to test if there is a customer older than 100. The implicit variable this customer is used in the where clause to refer to the customer object being tested.

```
if
  there is one customer
    where the age of this customer is more than 100,
```

Parent topic: [BAL constructs](#)

Related reference:

[if](#)
[from \(object\)](#)
[in \(list\)](#)
[set \(variable\) to \(definition\)](#)
[definitions](#)
[there are \(number\) \(objects\)](#)
[there is no \(object\)](#)
[there is one \(object\)](#)

BAL operators

You use operators in rule statements to do arithmetic operations, associate or negate conditions, and compare expressions.

Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Date operators

Date operators define conditions based on dates.

Number operators

Number operators define conditions based on numbers.

Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Text operators

Text operators define conditions based on strings.

Parent topic: [Business Action Language \(BAL\)](#)

Related information:

[Inserting an arithmetic expression](#)

[Controlling how condition statements are combined](#)

[Decision trees](#)

[Working with action rules](#)

Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

Table 1. Arithmetic operators

Operator	Description	Example
- <number>	Makes a number negative.	<pre>if there is more than one error then set 'status code' to - the number of errors;</pre>
<number> + <number>	Adds a number to another number.	<pre>if there is at least one additional driver then set the rental cost of the car to the base rental cost of the car + 100;</pre>
<number> - <number>	Subtracts a number from another number.	<pre>if the number of 'items purchased' - the number of 'items returned' is more than 0 then...</pre>
<number> / <number>	Divides a number by another number. If both numbers are integers, the result is also an integer (the result of the division without the remainder.)	<pre>if there is at least one order then set the average order quantity to the number of items ordered / the number of orders;</pre>
<number> * <number>	Multiplies a number by another number.	<pre>if the price of the item * 'sales tax' is more than 100 then...</pre>
<text> + <text>	Concatenates two strings.	<pre>definitions set 'full name' to 'first name' + ' ' + 'last name'</pre>
<number> + <text>	Concatenates a number and a string. The result is treated as a string.	<pre>if the order total is less than 'discount threshold' then display the message: "Spend just \$" + ('discount threshold' - order total) + " more to receive a 10% discount!";</pre>
<text> +	Concatenates a string and a number. The result is treated as	<pre>if</pre>

<code><number></code>	a string.	<pre>if the order total is more than 'discount threshold' then apply a 10% discount; display the message: "You received a 10% discount because your order was over \$" + 'discount threshold' ;</pre>
-----------------------------	-----------	---

Parent topic: [BAL operators](#)

Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Table 1. Logical operators

Operator	Description	Example
<condition > and <condition >	Associates two conditions such that the resulting expression is only true if both conditions are true.	<pre>if the category of the customer is Gold and the age of the customer is at least 60 then...</pre>
<condition > or <condition >	Associates two conditions such that the resulting expression is true if either (or both) conditions are true.	<pre>if the category of the customer is Gold or the age of the customer is at least 60 then...</pre>

Parent topic: [BAL operators](#)

Date operators

Date operators define conditions based on dates.

Table 1. Date operators

Operator	Description	Example
<a day of week> is after <a day of week>	Tests that a day of the week comes after another day of the week. <div>Note: Sunday is considered to be the first day of the week.</div>	<pre>if the day of the return date of 'rented car' is after Friday then...</pre>
<a day of week> is before <a day of week>	Tests that a day of the week comes before another day of the week. <div>Note: Sunday is considered to be the first day of the week.</div>	<pre>if the day of the return date of 'rented car' is before Friday then...</pre>
<date> is after <date>	Tests that a date comes after another date.	<pre>if the return date of 'rented car' is after 05/07/2007 05:00:00 PM then...</pre>
<date> is after <date> and on or before <date>	Tests that a date is in a range in which the first date is excluded, and the last date is included.	<pre>if the return date of 'rented car' is after 'pickup date' and on or before 'scheduled return date' then...</pre>
<date> is after <date> and before <date>	Tests that a date is in a range in which both dates are excluded.	<pre>if the return date of 'rented car' is after 'pickup date' and before 'scheduled return date' then...</pre>
<date> is after or the same as <date>	Tests that a date comes after another date, or is the same as that other date.	<pre>if the return date of 'rented car' is after or the same as 05/07/2007 05:00:00 PM then...</pre>
<date> is at <time>	Tests that the time part of a date is at the specified time.	<pre>if the return date of 'rented car' is at 05:00:00 PM then...</pre>
<date> is before <date>	Tests that a date comes before another date.	<pre>if the return date of 'rented car' is before 05/07/2007</pre>

		car' is before 05/07/2007 05:00:00 PM then...
<date> is before or the same as <date>	Tests that a date comes before another date, or is the same as that other date.	if the return date of 'rented car' is before or the same as 05/07/2007 05:00:00 PM then...
<date> is in <month>	Tests that a date is in a specific month.	if the return date of 'rented car' is in October then...
<date> is in <month> the year <year>	Tests that a date is in the specified month of the specified year.	if the return date of 'rented car' is in October the year 2007 then...
<date> is in the year <year>	Tests that a date is in the specified year.	if the return date of 'rented car' is in the year 2007 then...
<date> is on <a day of week>	Tests that a date is on a specific day of the week.	if the return date of 'rented car' is on Thursday then...
<date> is on <simple date>	Tests that a date is on a specific date, specified without a time.	if the return date of 'rented car' is on 05/07/2007 then...
<date> is on <day of week> at <time>	Tests that a date is on a specific day of the week, at a specific time.	if the return date of 'rented car' is on Thursday at 05:00:00 PM then...
<date> is on or after <date> and before <date>	Tests that a date is in a range in which the first date is included, and the last date is excluded.	if the return date of 'rented car' is on or after 'pickup date' and before 'scheduled return date' then...
<date> is on or after <date>	Tests that a date is in a range in which both dates	

after <date> and on or before <date>	range in which both dates are included.	if the return date of 'rented car' is on or after 'pickup date' and on or before 'scheduled return date' then...
--	--	--

Parent topic: [BAL operators](#)

Number operators

Number operators define conditions based on numbers.

Table 1. Number operators

Operator	Description	Example
<number> does not equal <number>	Tests that a number is not equal to another number.	<pre>if the number of rentals does not equal 3 then...</pre>
<number> equals <number>	Tests that a number is equal to another number. This is equivalent to the is <number> operator	<pre>if the number of rentals equals 3 then...</pre>
<number> is <number>	Tests that a number is equal to another number. This is equivalent to the equals <number> operator	<pre>if the number of rentals is 3 then...</pre>
<number> is at least <number>	Tests that a number is greater than or equal to another number.	<pre>if the number of rentals is at least 3 then...</pre>
<number> is at least <number> and less than <number>	Tests that a number is included in a range in which the first value is included and the last value is excluded.	<pre>if the age of the customer is at least 12 and less than 25 then...</pre>
<number> is at most <number>	Tests that a number is less than or equal to another number.	<pre>if the number of rentals is at most 3 then...</pre>
<number> is between <number> and <number>	Tests that a number is included in a range in which both values are included.	<pre>if the age of the customer is between 12 and 25 then...</pre>
<number> is less than <number>	Tests that one number is less than another.	<pre>if the number of rentals is less than 3 then...</pre>

<number> is more than <number>	Tests that one number is greater than another.	if the number of rentals is more than 3 then...
<number> is more than <number> and at most <number>	Tests that a number is included in a range in which the first value is excluded and the last value is included.	if the age of the customer is more than 12 and at most 25 then...
<number> is strictly between <number> and <number>	Tests that a number is included in a range in which the first value is excluded and the last value is excluded.	if the age of the customer is strictly between 12 and 25 then...

Parent topic: [BAL operators](#)

Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Table 1. Object operators

Operator	Description	Example
<object> is <object>	Tests that two objects are equivalent.	<pre>if the current customer is the customer on the rental agreement then...</pre>
<object> is not <object>	Tests that two objects are not equivalent.	<pre>if the current customer is not the customer on the rental agreement then...</pre>
<object> is one of <list>	Tests that an object is part of a set.	<pre>if the item of the order is one of 'discounted items' then...</pre>
<object> is not one of <list>	Tests that an object is not part of a collection.	<pre>if the customer category is not one of 'all categories' then...</pre>
<list> contain <object>	Tests that a collection contains an object. This operator is functionally equivalent to <object> is one of <list>.	<pre>if the customer categories contain Platinum then... apply a 10% discount;</pre>
<list> do not contain <object>	Tests that a collection does not contain an object. This operator is functionally equivalent to <object> is not one of <list>.	<pre>if the customer categories do not contain Normal then... apply a 10% discount;</pre>

Parent topic: [BAL operators](#)

Text operators

Text operators define conditions based on strings.

Table 1. Text Operators in BAL

Operator	Description	Example
<text> contains <text>	Tests that a string contains another string.	<pre>if the name of the customer contains "Smith" then...</pre>
<text> does not contain <text>	Tests that a string does not contain another string.	<pre>if the name of the customer does not contain "Smith" then...</pre>
<text> does not end with <text>	Tests that a string does not end with another string.	<pre>if the rental agreement code of the customer does not end with "XG5" then...</pre>
<text> does not start with <text>	Tests that a string does not start with another string.	<pre>if the rental agreement code of the customer does not start with "XG5" then...</pre>
<text> ends with <text>	Tests that a string ends with another string.	<pre>if the rental agreement code of the customer ends with "XG5" then...</pre>
<text> is empty	Tests that a string is empty. An empty string contains no characters and is equivalent to "". A string is not empty if it contains a space (" ")	<pre>if the rental agreement code of the customer is empty then...</pre>
<text> is not empty	Tests that a string is not empty. A string is not empty if it contains a space (" ")	<pre>if the rental agreement code of the customer is not empty then...</pre>
print <text>	Prints a string to standard out.	<pre>if</pre>

		<pre>11.... then print the message "Hello World"</pre>
<p><text> starts with <text></p>	Tests that a string starts with another string.	<pre>if the rental agreement code of the customer starts with "XG5" then...</pre>
<p>the length of <text></p>	Returns the number of characters in a string.	<pre>if the length of 'customer name' is more than 3 then...</pre>

Parent topic: [BAL operators](#)

BAL literals

The Business Action Language (BAL) supports number, string, date, and time data types.

Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

Dates and times

You can use the Date, Simple Date, and Universal Date value types to express date values. You can use the Time and Universal Time value types to express time values.

Parent topic: [Business Action Language \(BAL\)](#)

Related information:

[Overview: Ways to express business rules](#)

[Decision trees](#)

[Working with action rules](#)

[Working with decision tables](#)

[Working with decision trees](#)

Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

- 10 (integer)
- 10.5 (decimal, with decimal point separator)
- 150,500.51 (decimal, with '000 grouping separator and decimal point separator)
- -10 (negative integer)
- 10E-5 (exponent)

The lexical representation of numbers (the decimal point separator and the grouping separator) is locale-specific.

Note:

Big number literals are not supported in the business rule language. All literals manipulated in a rule must be within the range of the Java™ Double class, otherwise they are rounded to the closest Double value. The internal representation of numbers corresponds to Java doubles, and precision can be lost in conversion everywhere a number is passed as a parameter to a BOM method expecting a narrow type. This also happens when assigning an expression to a variable or a field. The IRL view of the rule editor allows you to see exactly where these conversions – represented by C-style casts – take place.

Parent topic: [BAL literals](#)

Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

To include certain special characters within a String, you must prefix them with a backslash (\). For example, to include a double quotation mark within a String, you would write \". The backslash is required here to indicate that the String has not yet ended.

The following special character sequences are accepted within text strings:

- \n (new line character)
- \t (tab character)
- \b
- \r (carriage return character)
- \f (line feed character)
- \' (single quotation mark)
- \" (double quotation mark)
- \\ (backslash)

Parent topic: [BAL literals](#)

Dates and times

You can use the Date, Simple Date, and Universal Date value types to express date values. You can use the Time and Universal Time value types to express time values.

Several types of date and time are available.

In the following table, a is the symbol for AM or PM, and z is the time zone information, for example +0200.

Date	Standard date format that includes time, and is written as: m/d/y h:mm:ss a
Simple Date	Simple date format that does not include time, and is written as: m/d/y
Time	Time format that is written as either: h:mm:ss a h:mm a
Universal Time	Time format that includes the time zone, and is written as either: h:mm:ss a z h:mm a z
Universal Date	Date and time format that includes the time zone, and is written as: m/d/y h:mm:ss a z

Note:

The business rule language supports only the Gregorian calendar format.

Parent topic: [BAL literals](#)

Punctuation in rules

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules (for example, in the Intellirule editor), mandatory punctuation is usually predicted in the completion menu.

Backslash \

The backslash is used to prefix special characters in text strings. For further details, see [Strings](#).

Comma ,

The comma marks the end of Where statements.

Example

This example shows a Where statement that is closed by a comma.

```
if
  there is at least one car
    where
      the color of this car is blue,
then ...
```

Some language constructs may be terminated by an optional comma.

Example

This example shows a construct that is terminated by an optional comma. This may prevent the construct becoming ambiguous in a particular context.

```
all following conditions are true:
- the color of the car is blue
- the price of the car is more than 1000,
```

Double quotation mark "

Double quotation marks are used to enclose string literals; that is, text strings. Avoid using them in the verbalization of BOM classes and members.

Example

This example shows an action phrase in which the message to be displayed is a text string enclosed in double quotation marks.

```
then
  apply a 10% discount;
  display the message: "You received a discount!";
```

Parentheses ()

Parentheses can be used to group any expression. They can help clarify the default precedence of logical operators linking conditions to one another. You can also use parentheses to regroup condition statements and hence change the order in which they are interpreted.

Example

This example shows nested parentheses in an action phrase.

```
then
  print "the customer: " + (the name of the customer of (the shopping cart))
```

Semicolon ;

The semicolon marks the end of variable definitions and action phrases.

Single quotation mark '

Single quotation marks are used to enclose variables. Single quotation marks are optional for single-word variables and mandatory for variables that consist of several words. Automatic variables are not enclosed in single quotation marks. Because single quotation marks are frequently used in the rule language, avoid using them in the verbalization of BOM classes and members.

Example

In this example, the variables `customer` and `the cart` are defined. The `customer` variable does not need to be enclosed in single quotation marks because it is a single word, whereas `the cart` needs to be delimited because it consists of two words.

definitions

```
set customer to a customer;  
set 'the cart' to the shopping cart of customer;
```

Parent topic: [Business Action Language \(BAL\)](#)

ILOG Rule Language (IRL)

The ILOG® Rule Language (IRL) contains a set of keywords, and has its own syntax to structure each part of the rule. IRL does not support generics because IRL is not Java™. As a result you cannot use generics in: IRL, functions, BOM to XOM mapping, and initial/final actions.

[IRL keywords](#)

The ILOG Rule Language defines IRL keywords to support rule authoring, management, and execution.

[IRL grammar](#)

The ILOG Rule Language (IRL) grammar conforms to a specific notation and has naming restrictions. It is composed of several operators that are a subset of the operators provided in the Java programming language.

Parent topic: [Rule languages](#)

IRL keywords

The ILOG® Rule Language defines IRL keywords to support rule authoring, management, and execution.

after

The `after` keyword defines a binary temporal constraint in the condition part of a rule in an event condition.

agendafilter

The `agendafilter` keyword defines an agenda filter in a rule task.

algorithm

The `algorithm` keyword defines an execution mode.

allrules

The `allrules` keyword sets all the rules in a rule task as eligible to be executed.

as

The `as` keyword defines a type conversion.

before

The `before` keyword defines a binary temporal constraint.

body (in flowtask)

The `body` keyword defines the body of the ruleflow.

body (in action task)

The `body` keyword defines the body of an action task.

body (in ruletask)

The `body` keyword defines the body of the rule task.

break

The `break` keyword exits a loop.

case

The `case` keyword identifies a statement block in a switch statement.

catch

The `catch` keyword designates exceptions that are caught if thrown.

collect

The `collect` keyword constructs a collection object.

continue

The `continue` keyword skips an iteration.

default (in ruletask)

This `default` keyword selects the RetePlus execution mode for a rule task.

default (in switch)

This `default` keyword executes the `else` condition statement.

dynamic

The `dynamic` keyword specifies the dynamic ordering of rules.

dynamicselect

The `dynamicselect` keyword specifies the dynamic selection of a rule.

else (in if)

In functions or rule actions, the `else` keyword executes an alternate statement block.

else (in rule)

In rule definitions, the `else` keyword executes an alternate action block depending on the value of an `evaluate` statement.

evaluate

The `evaluate` keyword specifies tests on objects.

event (in rule conditions)

The `event` keyword declares an event condition or an event object.

event (in rule actions)

The `event` keyword declares an event condition or an event object.

exists

The `exists` keyword tests whether a class condition is true.

filter

The `filter` keyword defines an agenda filter.

finalaction

The finalaction keyword declares a final action.

finally

The finally keyword introduces a control block that is executed after a try block.

firing

The firing keyword determines whether all the rules are executed.

firinglimit

The firinglimit keyword specifies the number of rules to be executed.

flowtask

The flowtask keyword declares a subflow task.

for

The for keyword introduces an execution loop.

foreach

The foreach keyword executes a statement block several times.

fork

The fork keyword executes several statement blocks sequentially.

from

The from keyword supports relations between objects.

function

The function keyword defines a function to be called.

functiontask

The functiontask keyword declares an action task.

goto

The goto keyword jumps to another ruleflow statement.

hasher

The hasher keyword defines a hashing expression.

hierarchy

The hierarchy keyword defines a hierarchical property.

if

The if keyword executes one of two statement blocks.

if (in ruleflow)

The if keyword executes one statement block or another, depending on a Boolean expression value.

import

The import keyword imports a Java class or package into a rule file.

in

The in keyword supports relations between object values.

in — !in (predicates)

In any expression, the in keyword tests whether a value belongs to a collection and or the !in keyword tests whether a value does not belong to a collection.

in (ruleset parameter)

The in keyword defines a ruleset parameter.

initialaction

The initialaction keyword declares an initial action.

inout

The inout keyword defines a ruleset parameter that is both an input parameter and an output parameter.

insert

The insert keyword inserts an object into the working memory.

instanceof

The instanceof keyword tests whether an expression can be cast into a type.

instances

The instances keyword declares class instances.

isknown

The isknown keyword tests whether a class attribute has an initialized value.

isunknown

The `isunknown` keyword tests whether a class attribute has an initialized value.

logical (in wait)

In a `wait` statement, the `logical` keyword designates that conditions must remain true for a `wait` statement to become true.

match

The `match` keyword deals with hierarchical property values.

modify

The `modify` keyword modifies objects and updates the agenda.

new

The `new` keyword creates a new object or array.

not

The `not` keyword tests the negation of a rule condition.

occursin

The `occursin` keyword defines a unary temporal constraint in an event condition.

ordering

The `ordering` keyword sets the order of the rules executed in a rule task.

out

The `out` keyword defines a ruleset output parameter.

overriding, overrides

The `overriding` and `overrides` keywords declare overriding relations.

package

The `package` keyword declares a package.

property

The `property` keyword declares a rule, ruleset, or ruleflow property.

propertydefinition

The `propertydefinition` keyword predeclares the types of rule properties.

refresh

The `refresh` keyword requests a refresh of the agenda.

retract

The `retract` keyword removes an object from the working memory.

return

The `return` keyword returns to the caller.

rule

The `rule` keyword declares a rule.

rule (in ruletask)

The `rule` keyword sets a single rule as eligible for execution.

ruleset

The `ruleset` keyword declares a ruleset.

ruletask

The `ruletask` keyword declares a rule task.

scope

The `scope` keyword defines a scope in a rule task.

select

The `select` keyword selects a rule in a rule task.

sequential

The `sequential` keyword specifies the execution mode of a rule task.

switch

The `switch` keyword executes one statement block or another according to an integer expression value.

then

The `then` keyword declares the action part of a rule.

throw

The `throw` keyword throws an exception.

timeof

The `timeof` keyword accesses an event timestamp.

timeout

The timeout keyword is associated with a wait statement.

try

The try keyword executes control statements.

until

The until keyword specifies an absolute time.

update

The update keyword updates an object.

use

The use keyword imports an IRL package or artifact.

variables

The variables keyword declares variables.

wait

The wait keyword incorporates time as a rule parameter.

when

The when keyword declares the condition part of a rule.

where

The where keyword defines a constraint in a collect statement.

while

The while keyword executes a statement block.

while/break/continue (in ruleflow)

The while keyword executes a loop.

Parent topic: [ILOG Rule Language \(IRL\)](#)

after

The after keyword defines a binary temporal constraint in the condition part of a rule in an event condition.

Purpose

A binary temporal constraint in an event condition.

Context

Rule conditions

Syntax

```
[?var:] event className (?eventVar1 after [interval] ?eventVar2);
```

Description

Deprecated as of V7.5.

The after keyword is used in the condition part of a rule in an event condition. You express a temporal constraint between two events with the after and before keywords. An interval is of the form [lowerBound, upperBound], where a bound is either an expression evaluating to an integer or the \$ sign which denotes infinity.

Considering that the timestamp of ?event ₁ is t₁ and that the timestamp of ?event ₂ is t₂, you can express the after operator as follows:

- ?event ₂ after[min,max] ?event ₁
is satisfied if the value of t₂ - t₁ is between the values of min and max, inclusive.
- ?event ₂ after[min,\$] ?event ₁
is satisfied if the value of t₂ - t₁ is greater than, or equal to, the value of min.
- ?event ₂ after[\$,max] ?event ₁
is satisfied if the value of t₂ - t₁ is less than, or equal to, the value of max.
- ?event ₂ after ?event ₁
is satisfied if the value of t₂ - t₁ is positive or null.

Example

This example assumes that a User instance is in the working memory and that an Alarm instance is inserted. A partial instance of the ReportAlarmPairsToUsers rule is created and maintained for four ticks. If the User instance is then retracted, the partial instance of the rule is immediately deleted. An instance of the ReportAlarmsToUsers rule is created on each instance of the User class in the working memory when a second alarm event ?a2 is inserted no later than four ticks after ?a1.

```
rule ReportAlarmPairsToUsers {  
  when {  
    ?u: User();  
    ?a1: event Alarm();  
    ?a2: event Alarm(?this after[1, 4] ?a1);  
  }  
  then {  
    ?u.report(?a1, ?a2);  
  }  
};
```

Parent topic: [IRL keywords](#)

Related reference:

[before](#)
[event \(in rule conditions\)](#)
[event \(in rule actions\)](#)
[occursin](#)
[timeof](#)

agendafilter

The agendafilter keyword defines an agenda filter in a rule task.

Purpose

This agenda filter is used if the execution algorithm selected for the rule task is RetePlus.

Context

Rule task definitions

Syntax

```
(1) ruletask ruleTaskName
{
    agendafilter = filter (variableName)
    {
        action1
        ...
        actionn
    }
};

(2) ruletask ruleTaskName
{
    agendafilter = value;
};
```

Description

You can write agenda filters in two different syntaxes:

Syntax (1)

You write the code of the agenda filter within the task definition. This code is similar to an IRL function with an [IlrRuleInstance](#) parameter and a Boolean return type. The *variableName* placeholder represents this [IlrRuleInstance](#) object. It can be referenced in the inline code of the agenda filter. Ruleset variables are accessible from the agenda filter code. This agenda filter is applied on each instance of the rules that compose the task body. Any instance for which the agenda filter returns true is executed. The other instances are not.

Syntax (2)

You write the agenda filter as an instance of a class that implements the [IlrAgendaFilter](#) interface. This filter is applied on each instance of the rules that compose the task body. The instance for which the agenda filter returns true is executed. The other instances are not.

Example

These two examples illustrate the use of the agendafilter keyword.

Example 1

This example defines a rule task ruletask1 in which the body is composed of all the rules of the ruleset. The agenda filter defined on the rule task filters the rule instances contained in that task. Only the rule instances whose names are contained in the input parameter of the ruleset are executed. The others are not.

```
ruletask ruletask1
{
    body = select(?rule) {return true;}
    agendafilter = filter(?instance)
    {
        String name = ?instance.ruleName;
        return inputParam.contains(name);
    }
};
```

Example 2

This example defines another rule task in which the body is composed of all the rules of the ruleset. The

agenda filter defined on the task is an instance of the Java™ class `MyAgendaFilter`, which implements the [IlrAgendaFilter](#) interface.

```
ruletask ruletask1
{
    body = select(?rule) {return true;}
    agendafilter = new MyAgendaFilter();
};
```

Parent topic: [IRL keywords](#)

Related reference:
[ruletask](#)

algorithm

The `algorithm` keyword defines an execution mode.

Purpose

This key is used to define an execution mode for a rule task, either `RetePlus`, `sequential`, or `Fastpath`.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    algorithm = default|sequential;
};
```

Description

The value that follows the `algorithm` keyword indicates which execution mode is used to execute the rules in that rule task. You can apply one of the following execution modes to rule tasks:

- The `RetePlus` mode: the value of the `algorithm` property is then `default` because `RetePlus` was the default execution mode in versions earlier than Operational Decision Manager V8.6.
- The `sequential` mode: the value of the `algorithm` property is then `sequential`.
- The `Fastpath` mode: `Fastpath` is the default algorithm. To use it, you must set the `algorithm` property to `sequential` and the custom property `ilog.rules.engine.sequential.fastpath` to `true`. For example:

```
ruletask R1
{
    property ilog.rules.engine.sequential.fastpath = true;
    algorithm = sequential;
};
```

Parent topic: [IRL keywords](#)

Related reference:

[ruletask](#)

allrules

The allrules keyword sets all the rules in a rule task as eligible to be executed.

Purpose

This keyword is used to set all the rules as eligible for execution in a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    firing = allrules|rule;
};
```

Description

The value that follows the firing keyword indicates whether all the rules are executed (allrules, the default value) or whether only one rule is executed (rule).

Note:

All instances of one rule are executed. If only one rule is executed, use the keyword [firinglimit](#). See also [ruletask](#) for more information.

Parent topic: [IRL keywords](#)

Related reference:
[ruletask](#)

as

The as keyword defines a type conversion.

Purpose

This keyword is used to convert the type of an expression to another type.

Context

Any expression

Syntax

```
expression as type ;
```

Description

The as statement attempts to convert the type of the expression into the requested type, without raising a `ClassCastException` instance. If the conversion is possible, the as statement returns an expression with the correct type. It returns null if the conversion failed.

Example

```
rule AsStatement {
  when {
    ?a: Alarm(?e : equipment);
  }
  then {
    Multiplex m = ?e as Multiplex;
    if (m != null);
    System.out.println("Alarm occurs on Multiplex");
  }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[instanceof](#)

before

The before keyword defines a binary temporal constraint.

Purpose

This keyword is used to define a binary temporal constraint in an event condition.

Context

Rule conditions

Syntax

```
[?var:] event className(?eventVar1  
before [interval] ?eventVar2);
```

Description

Deprecated as of V7.5.

Use the before keyword in the condition part of a rule in an event condition. To express a temporal constraint between two events, you use the after and before keywords. An interval is of the form [lowerBound, upperBound], where a bound is either an expression evaluating to an integer, or the \$ sign which denotes infinity.

Considering that the timestamp of ?event 1 is t1 and that the timestamp of ?event 2 is t2, you can express the before operator as follows:

1. ?event 1 before[min,max] ?event 2
is satisfied if the value of t2 - t1 is between the values of min and max, inclusive.
2. ?event 1 before[min,\$] ?event 2
is satisfied if the value of t2 - t1 is greater than, or equal to, the value of min.
3. ?event 1 before[\$,max] ?event 2
is satisfied if the value of t2 - t1 is less than, or equal to, the value of max.
4. ?event 1 before ?event 2
is satisfied if the value of t2 - t1 is positive or null.

Example

This example assumes that a User instance is in the working memory and that an Alarm instance is inserted. A partial instance of the ReportAlarmPairsToUsers rule is created and maintained for four ticks. If the User instance is then retracted, the partial instance of the rule is immediately deleted. An instance of the ReportAlarmsToUsers rule is created on each instance of the User class in the working memory when ?a1 is no greater than 4 ticks before a second alarm event ?a2 is inserted.

```
rule ReportAlarmPairsToUsers {  
  when {  
    ?u: User();  
    ?a1: event Alarm();  
    ?a2: event Alarm(?a1 before[1, 4] ?this);  
  } then {  
    ?u.report(?a1, ?a2);  
  }  
};
```

Parent topic: [IRL keywords](#)

Related reference:

[after](#)
[event \(in rule conditions\)](#)
[event \(in rule actions\)](#)
[occursin](#)
[timeof](#)

body (in flowtask)

The body keyword defines the body of the ruleflow.

Purpose

This keyword is used to define the body of a flow task in flowtask statements.

Context

Flow task definitions

Syntax

```
flowtask flowTaskName
{
    body {ruleflow}
};
```

Description

The flow task body defines a ruleflow. It is composed of task calls that are chained together through control statements. The control statements are:

sequence, if, switch, while (break and continue), fork, goto.

Example

```
flowtask main
{
    body =
    {
        while(!ending)
        {
            if (turn == Constants.Player1) ChooseMovePlayer1;
            else ChooseMovePlayer2;

            CheckMove;
            if (ending) break;

            UpdateDistance;
            ExpandObjects;
            DetectConnect4;
            if (ending) break;

            DetectGridFull;
            if (ending) break;
            ChangeTurn;
        }
        EndOfGame;
    }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[goto](#)

[flowtask](#)

[fork](#)

[if \(in ruleflow\)](#)

[switch](#)

[while/break/continue \(in ruleflow\)](#)

body (in action task)

The body keyword defines the body of an action task.

Purpose

This keyword is used to define the body of an action task in `functiontask` statements.

Context

Function task definitions

Syntax

```
functiontask functionTaskName
{
    body
    {
        action1;
        ...
        actionn;
    }
};
```

Description

The action task body is equivalent to an IRL function with no arguments and with the `void` return type. The task execution consists in executing the statements that compose its body after its initial actions and before its final actions if they are defined.

Example

```
functiontask UpdateDistance
{
    body =
    {
        int x = move.x;
        int y = move.y;
        for (int i = y + 1; i < 10; i++)
        {
            int p = grid.position(x,i);
            if (p == null) break;
            p.distance--;
            update(p);
        }
    }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[evaluate](#)

[for](#)

[functiontask](#)

[if \(in ruleflow\)](#)

[modify](#)

[retract](#)

[return](#)

[try](#)

[update](#)

[while](#)

body (in ruletask)

The body keyword defines the body of the rule task.

Purpose

This keyword is used to define a rule task body in ruletask statements.

Context

Rule task definitions

Syntax

```
(1) ruletask  ruleTaskName
    {
        body {ruleName1, ..., ruleNamen}
    };

(2) ruletask  ruleTaskName
    {
        body = select([variableName])
        {
            action1;
            ...
            actionn;
        }
    };

(3) ruletask  ruleTaskName
    {
        body = dynamicselect([variableName])
        {
            action1;
            ...
            actionn;
        }
    };

(4) ruletask  ruleTaskName
    {
        body = dynamicselect([variableName])
        {
            action1;
            ...
            actionn;
        } in Expression;
    };
```

Description

You can write the body of a rule task in different syntaxes.

Syntax (1)

The rule task body is composed of the rules that are explicitly listed by their names. These rules must belong to the same ruleset as the rule task.

Syntax (2)

The rule task body has been specified by comprehension: the rules that compose the task body are not explicitly listed. The contents of the task body is computed when the code given as body for each rule in the ruleset is executed.

This body definition can be seen as either of the following IRL functions:

- As an IRL function with a Boolean return type and an argument of type [IIRule](#). In this case, a variable name is specified. The code is executed for each rule of the ruleset. When the execution returns true, the rule becomes part of the rule task. Otherwise it does not.

- As an IRL function with no argument and with `ilog.rules.engine.IlrRule[]` rule array as its return type. In this case, the rule task body contains the rules returned by the code.

In all cases for syntax (2), the code is executed only once for the task, even if the task is executed several times. It is evaluated the first time it is needed.

Syntax (3)

The difference with Syntax (2) is the time at which the body is computed before each task is executed. This is useful when the body content depends on variables whose values change during the ruleflow execution.

Important:

Use a [scope](#) keyword in conjunction with syntax (2) or (3) because it limits the scope of the execution of the [select](#) or [dynamicselect](#).

Example

```
ruletask DetectGridFull
{
    ordering = literal;
    firinglimit = 1;
    body = { DetectGridFull }
};

ruletask ExpandObjects
{
    ordering = dynamic;
    firing = allrules;

    body = select(?rule)
    {
        String ?task = ?rule.getProperties().getString("task","");
        return ?task.equals("ExpandObjects");
    }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[algorithm](#)

[ruletask](#)

[rule \(in ruletask\)](#)

[scope](#)

break

The break keyword exits a loop.

Purpose

A side statement for exiting a loop.

Context

Functions or technical rule actions

Syntax

```
break;
```

Description

The break statement is used in the action part of a rule or in functions. Use it to exit a while loop or a for loop. This statement forces the rule engine to go to the end of the containing statement block and to continue to the next instruction.

Example

The AlarmSurveillance rule tests for a new Alarm object and loops while the state is equal to ON unless the level is equal to 3. The single rule condition matches an object Alarm if the field state equals the static value NEW. If such a Alarm object is matched, the engine can execute the action part. The modify statement is used to change the field state from NEW to ON. The while statement is used to loop as long as the field state has the value ON. An if statement tests the field level. If the level is equal to 3, a break statement is executed and the program exits the while loop.

```
rule AlarmSurveillance {
  when {
    ?a: Alarm(state == NEW);
  }
  then {
    modify ?a {
      state = ON;
    }
    while (?a.state == ON ) {
      ...
      if (?a.level == 3)
        break;
    }
  }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[continue](#)
[for](#)
[foreach](#)
[while](#)

case

The case keyword identifies a statement block in a switch statement.

Purpose

This keyword is used to identify a statement block in a switch ruleflow statement.

Context

switch statements

Syntax

```
switch (expression)
{
    case value1:
        {ruleflowStatement}
        ...
    case valuen:
        {ruleflowStatement}
    default:
        {ruleflowStatement}
}
```

Description

The switch statement is a conditional ruleflow statement. The integer expression is evaluated. If the value of a case block is equal to the evaluated expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

Parent topic: [JRL keywords](#)

Related reference:
[switch](#)

catch

The catch keyword designates exceptions that are caught if thrown.

Purpose

This keyword is used within a try statement to designate exceptions that are caught if thrown.

Context

Functions or rule actions

Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

Description

The try-catch-finally statements establish a block of code for exception handling. Such blocks all begin and end with curly braces.

The catch statements are designed to handle a specific type of exception. The code within a catch statement is executed if an exception is caught. A catch statement is declared with an `exceptionType` that specifies the type of exception that the statement can handle. It also provides an identifier that the statement can use to refer to the exception object that it is currently handling. The identifier must be of type `Throwable` or one of its subclasses.

Parent topic: [JRL keywords](#)

Related reference:

[finally](#)

[try](#)

collect

The collect keyword constructs a collection object.

Purpose

This statement is used to construct a collection object.

Context

Rule conditions

Syntax

```
[?variable:] collect [(expression)] collectionTarget  
[where (collectionTest1 ... collectionTestn)];
```

Description

Use the collect statement in the condition part of a rule to create a collection object. The collection object stores instances of the class `collectionTarget` that match the condition. This condition can contain tests on the class fields. The collection object can be bound to a variable for the scope of the rule. The expression is optional, but if provided, it must return a collector object that implements the [IlrCollection](#) interface, as shown in the example below. The collect statement can contain a list of tests on the collection object in the where part of the statement.

If a rule contains some conditions that are matched and a collect condition, the eligibility of the rule depends on the existence of a where statement, as follows:

- Without a where statement, the rule is eligible to be executed, even if no object has been matched by `collectionTarget`. In other words, the rule is eligible and the collection object is empty.
- With a where statement, the rule is eligible only if the where statement is evaluated to `true`.

The collector object, declared by an expression, must implement the `IlrCollection` interface and its public methods: [addElement](#), [updateElement](#), and [removeElement](#). The `IlrCollection` interface is defined as follows:

```
public interface IlrCollection {  
    public void addElement(java.lang.Object element);  
    public void updateElement(java.lang.Object element);  
    public void removeElement(java.lang.Object element);  
}
```

If the *expression* argument is left empty, a [IlrDefaultCollector](#) object is used.

The where part of the statement can contain tests that the collection object must fulfill. Or it can be left empty. The [IlrCollection](#) interface and [IlrDefaultCollector](#) class provide methods for all collections, such as `size`, `isEmpty`, `contains`, and `elements`.

Example

The `MusicCollector` rule collects `Music` objects and stores them in an `ItemCollector` object. The first condition returns in variable `?s` the `Store` objects that have the field `domain` equal to `MUSIC`. The variable `?c` refers to the `ItemCollector` object. The constructor for the `ItemCollector` object takes a `Store` object as an argument.

```
rule MusicCollector {  
    when {  
        ?s: Store(domain==MUSIC);  
        ?c: collect(new ItemCollector(?s))  
            Music(store==?s; category==JAZZ;  
                artist equals "Miles Davis";  
                production > 1956 & < 1965)  
            where ( size() > 5 && ItemCollector.averagePrice()  
                < 15.0);  
    }  
    then {  
        Enumeration ?enum = ?c.elements();  
        while (?enum.hasMoreElements()) {
```

```

        Music ?cd = (Music)enum.nextElement();
        System.out.println("At " + ?s.name
                           "you can find the title: " + ?cd.title);
    }
}
}

```

The collectionTarget is the class Music with the following field values:

- store equal to the result in ?s
- category equal to JAZZ
- artist equal to the string Miles Davis
- production between the dates 1956 and 1965

Any object that match this condition is inserted into the ItemCollector object. For the collect statement to be true, two tests are defined in the where part of the statement. The default method size tests that the number of items in the collector is greater than 5 and the user-defined method averagePrice tests that the average price of the items is lower than 15.

If the when part of the rule is true, the then part is executed. The variable ?c refers to the ItemCollector object. The first statement creates a variable ?enum which refers to an enumeration of the elements of the ItemCollector object. The while statement prints the name field of the Store object and the corresponding title field of each Music object.

Parent topic: [IRL keywords](#)

Related reference:

[exists](#)
[from](#)
[in](#)
[not](#)

continue

The continue keyword skips an iteration.

Purpose

A side statement for skipping an iteration.

Context

Rule actions or functions

Syntax

```
continue;
```

Description

Use the continue statement in the action part of a rule and in a function to skip an iteration of a while or for loop. This statement forces the rule engine to skip to the next test of the loop statement.

Example

The rule StockDropWarning checks the percent change of a customer's stocks and issues a warning if a stock price dropped by more than a certain percentage. The condition Client returns an account number in variable ?a and a decimal value percentWarningMark in variable ?p. The second condition, ClientStockList, returns a stock list in variable ?s corresponding to the account number equal to ?a. In the action part of the rule, the while statement iterates over every stock in the stock list ?s. If a stock type is equal to LONG, the continue statement cause the iteration to skip the rest of the statements in the while block and proceed to the while test. If a stock type is not equal to LONG, the difference between the purchase price and the current price is calculated. If the stock price decreased in ?p percent below the purchasePrice, a message is printed.

```
rule StockDropWarning {
  when {
    ?c:Client(?a:account_number;?p:percentWarningMark);
    ClientStockList(account_number == ?a; ?s:stockList);
  }
  then {
    Enumeration ?enum = ?s.elements();
    while (?enum.hasMoreElements()) {
      Stock ?x = (Stock)enum.nextElement();
      if (?x.type == Stock.LONG)
        continue;
      if ( ((1.-?p)*?x.purchasePrice) > ?x.currentPrice ) {
        System.out.println("Dear "+ ?c.name + " Your stock "
          + ?x.ticker + " has dropped " +
          ((?x.purchasePrice - ?x.currentPrice) /
            ?x.purchasePrice ) + " percent.");
      }
    }
  }
};
```

Parent topic: [JRL keywords](#)

Related reference:

[break](#)

[for](#)

[foreach](#)

[while](#)

default (in ruletask)

This default keyword selects the RetePlus execution mode for a rule task.

Purpose

In the context of execution modes, the default keyword is used to select the RetePlus execution mode for a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    algorithm = default|sequential;
};
```

Description

The value that follows the algorithm keyword indicates which execution mode is used to execute the rules:

- RetePlus if the algorithm value is default, because RetePlus was the default execution mode in versions earlier than Operational Decision Manager V8.6.
- The sequential mode if the algorithm value is sequential.

Fastpath is an option within the sequential mode.

Parent topic: [IRL keywords](#)

Related reference:

[algorithm](#)
[ruletask](#)

default (in switch)

This default keyword executes the else condition statement.

Purpose

This keyword is used in the switch statement to execute the else condition.

Context

switch statements

Syntax

```
switch (expression)
  case value1:
    {ruleflowStatement}
    ...
  default:
    {ruleflowStatement}
```

Description

The switch statement is a conditional ruleflow statement. The integer expression is evaluated. If a value of the case block is equal to the evaluated expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

Parent topic: [JRL keywords](#)

Related reference:
[switch](#)

dynamic

The `dynamic` keyword specifies the dynamic ordering of rules.

Purpose

This keyword is used to specify the dynamic ordering of the rules that are executed in a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    [ordering = dynamic|sorted|literal;]
};
```

Description

Rule tasks execute rules. You can use parameters to specify how the rules are ordered and how many rules are executed.

The keyword `ordering` specifies how the rules are sorted.

- `dynamic`: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- `sorted`: The rules are sorted in decreasing order of static priority.
- `literal`: The rules are listed in the same order as they have been listed in the rule task body.

Parent topic: [IRL keywords](#)

Related reference:

[ordering](#)
[ruletask](#)

dynamicselect

The `dynamicselect` keyword specifies the dynamic selection of a rule.

Purpose

This keyword is used to specify the dynamic selection of a rule in a rule task body.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    body = dynamicselect(?rule) {action1 ... actionr}
};
```

Description

The body of a rule task contains a list of rules. You can either specify each rule name explicitly (extension) or have the list of rules computed from the code, which uses the `select` or `dynamicselect` keywords (comprehension). The code must return a value of type `boolean`. The selection method changes the way the filter is applied to the rules. With dynamic rule selection, the filter is evaluated each time the rule task is invoked.

There is no difference between `dynamicselect` and `select`. They behave the same way in that the statement is called each time the task is executed.

To optimize performance, the decision engine does not execute the code statements the same number of times depending on whether they are before the use of the `(?rule)` variable or after. Statements that come before the statement in which the `(?rule)` variable is used are calculated only once per `ruletask` execution, when the task body is evaluated. Statements including and after the first statement in which the `(?rule)` variable is used are calculated for every rule under review in the task scope. For example, `?contract` is a ruleset variable with a `country` attribute and an `effectiveDate` attribute, and the task applies only for a selection of countries and only to the rules whose `effectiveDate` covers the one of the contract:

```
body = dynamicselect(?rule) {
return ?contract.country in { "France", "Spain", "Italy"} &&
    ?contract.effectiveDate.after((java.util.Date)?
rule.getPropertyValue("effectiveDate"));
}
```

In the previous example, `?contract.country in { "France", "Spain", "Italy"}` is evaluated once per task execution and `?contract.effectiveDate.after((java.util.Date)?rule.getPropertyValue("effectiveDate"))` is calculated once per rule for each task execution.

However, if you write the following code, then both conditions are calculated once per rule for each task execution:

```
body = dynamicselect(?rule) {
return  ?contract.effectiveDate.after((java.util.Date)?
rule.getPropertyValue("effectiveDate")) &&
    ?contract.country in { "France", "Spain", "Italy" };
}
```

Parent topic: [IRL keywords](#)

Related reference:

[body \(in ruletask\)](#)
[ruletask](#)
[select](#)

else (in if)

In functions or rule actions, the `else` keyword executes an alternate statement block.

Purpose

A side statement within an `if` statement to execute an alternate statement block depending on a Boolean expression value.

Context

Functions or rule actions

Syntax

```
if (test) {statement}  
else {statement}
```

Description

Use the `if` statement in the action part of a rule, a function, or a ruleflow. The test can be any legal test as in the Java™ programming language. If the test returns `true`, the first statement block is executed and the `else` block is skipped over. If the test returns `false`, the first block is skipped over and the statement block that follows the `else` clause is executed.

In a rule or function, any IRL statement, arithmetic expressions, and method calls can be executed within the statement block. In a ruleflow, any ruleflow statement can be executed within the statement block. A statement block can contain one or more statements. If it contains only one statement, the braces (`{}`) are not required. The `else` keyword and its statement block is optional.

Parent topic: [IRL keywords](#)

Related reference:

[if](#)

else (in rule)

In rule definitions, the `else` keyword executes an alternate action block depending on the value of an `evaluate` statement.

Purpose

In the condition part of a rule, this keyword is used to execute an alternate action block depending on the value of an `evaluate` statement.

Context

Rule definitions

Syntax

```
rule ruleName {  
  when {condition evaluate (expression)}  
  then {[action1 ... actionm]}  
  else {[action1 ... actionp]}  
};
```

Description

When the condition part of the rule ends with an `evaluate` statement, the action part can have an `else` part, which is executed if the last `evaluate` statement returns `false`. If it returns `true`, the `then` part is executed.

Parent topic: [IRL keywords](#)

Related reference:

[then](#)
[evaluate](#)
[rule](#)

evaluate

The evaluate keyword specifies tests on objects.

Purpose

This evaluate statement is used in the condition part of a rule to test objects in the working memory.

Context

Rule conditions

Syntax

```
evaluate (expression);
```

Description

Before the evaluate statement, you must write a simple condition that binds a variable to an object or a value. The engine can test any such variable bound to an object or a value. Note that the statements not, exists, and collect are not simple conditions. An evaluate statement is true if all the tests carried out in the expression are true. The expression can be made of multiple tests enclosed in braces ({}).

When the condition part of a rule ends with an evaluate statement, the action part can have an else part, which is executed if the last evaluate statement returns false. If it returns true, the then part is executed.

The evaluate statement differs from the not and exists statements. The not and exists statements apply tests only within the scope of their statement. The evaluate statement is more general in that it specifies tests that must be satisfied by the objects that exist in the working memory.

Example

The following two CloseOrder rules show equivalent ways of testing objects of the working memory. The first rule has two conditions followed by an evaluate statement. In the second rule, the test of the evaluate statement is incorporated in the second condition.

Rule 1

```
rule CloseOrder {
  when {
    ?s: CustomerShipment(?id:CustomerId; ?sl:shipmentList);
    ?o: CustomerOrder(?id==CustomerId; ?ol:orderList);
    evaluate (?sl.equals(?ol));
  }
  then {
    retract(?s);
    insert Shipped(?o,currentDate());
  }
};
```

Rule 2

```
rule CloseOrder {
  when {
    ?s: CustomerShipment(?id:CustomerId; ?sl:shipmentList);
    ?o: CustomerOrder(?id==CustomerId; ?ol:orderList;
                      ?sl.equals(?ol))
  }
  then {
    retract(?s);
    insert Shipped(?o,currentDate());
  }
};
```

For both rules, variable ?s refers to a CustomerShipment object, variable ?id returns the CustomerId field, and variable ?sl returns the shipmentList field. In the second condition, variable ?o refers to a CustomerOrder object where the CustomerId field matches ?id and variable ?ol contains the orderList field. The evaluate statement calls a test, the equals method, which returns true if the two lists contain the same elements. If the conditions are true, the action part is executed. The CustomerShipment object is deleted from the working memory by the retract statement, and a Shipped object is inserted into the working

memory by the insert statement.

Parent topic: [IRL keywords](#)

Related reference:

[else \(in if\)](#)

[exists](#)

[instanceof](#)

[isknown](#)

[isunknown](#)

[not](#)

[rule](#)

event (in rule conditions)

The event keyword declares an event condition or an event object.

Purpose

This keyword is used for declaring an event condition or an event object in rule conditions.

Context

Rule conditions

Syntax

```
[?var:] event className (test1;...;testn);  
where a temporal test test i has the form:  
{after|before|occursin} [interval]
```

Description

Deprecated as of V7.5.

On the side of a rule, an event condition can be bound to a variable. The event class name is defined in the statement together with tests.

Tests can be temporal or nontemporal.

- In a temporal test, you can use the following keywords: The interval, when provided, restricts the allowed delay between the events.
 - To express a temporal constraint, use the test operators after, before, or occursin.
 - To express temporal constraints between two events, use the after and before operators.
 - To express a temporal constraint on a single event, use the occursin operator. An occursin test is satisfied if the timestamp of the event is included in the interval.

Do not write references to bound variables or in other event conditions in the expression. The expression is evaluated once, when the events occurring as operands of the temporal constraint operator are inserted.

- You can combine nontemporal tests and temporal constraints as shown in the following code lines:

```
?a: event Alarm();  
?b: event Alarm(sev >= HIGH || ?this after[1,5] ?a);
```

Example

The alarmManager rule prints a message when two alarms occur on the same managed object with a time difference between 1 and 5 clock ticks.

```
rule alarmManager {  
  when {  
    ?mo: ManagedObject();  
    ?operator: Operator() in ?mo.subscribers();  
    ?alarm1: event Alarm(managedObject == ?mo);  
    ?alarm2: event Alarm(managedObject == ?mo; ?this after[1, 5]  
      ?alarm1);  
  }  
  then {  
    System.out.println("Seen two alarms on " + ?mo.name);  
    ?operator.report(?alarm1, ?alarm2);  
  }  
}
```

The when keyword introduces four conditions:

1. In the first condition, a variable ?mo is matched by a ManagedObject object.

2. The second condition returns the subscribed Operator objects in the variable ?operator.
3. The third condition is an event statement. The variable ?alarm1 points to an event object named Alarm with a single constraint: the value of the field managedObject is equal to the variable ?mo.
4. The fourth condition is also an event statement, where the variable ?alarm2 points to an event object named Alarm with the following constraints: the value of the field managedObject is equal to the variable ?mo and the difference between the first event object timestamp and the second event object timestamp is greater than, or equal to, 1 and less than, or equal to, 5 clock ticks.

The then keyword introduces two actions:

1. The first action prints a message that names a managed object.
2. The second action executes an ?operator.report method using the event object variables ?alarm1 and ?alarm2.

Any alarm is automatically retracted from the working memory 6 clock ticks after its insertion.

Parent topic: [IRL keywords](#)

Related reference:

[after](#)
[before](#)
[occursin](#)
[timeof](#)

event (in rule actions)

The event keyword declares an event condition or an event object.

Purpose

This keyword is used for declaring an event condition or an event object in rule actions.

Context

Functions or rule actions

Syntax

```
insert event [(timeExpression)] object [{statement1;...;statementn}];
```

Description

Deprecated as of V7.5.

Next to a rule, when an object is inserted into the working memory as an event, a timestamp is automatically associated with it. The `timeExpression` argument defines an integer value which is used as the timestamp when an event is inserted. The term `object` must be a constructor for the class. For the constructor to apply, you must declare it as `public` in your Java™ code.

Following the optional `timeExpression`, the `insert event` statement can either terminate with a semicolon (;) or specify a block of executable statements. The statements are executed on the object before this object is inserted into the working memory.

You can remove an event object explicitly from the working memory by using the `retract` statement or the `IlrContext.retract` method.

Example

The `alarmManager` rule prints a message when two alarms occur on the same managed object with a time difference between 1 and 5 clock ticks.

```
rule alarmManager {
  when {
    ?mo: ManagedObject();
    ?operator: Operator() in ?mo.subscribers();
    ?alarm1: event Alarm(managedObject == ?mo);
    ?alarm2: event Alarm(managedObject == ?mo; ?this after[1, 5]
                        ?alarm1);
  }
  then {
    System.out.println("Seen two alarms on " + ?mo.name);
    ?operator.report(?alarm1, ?alarm2);
  }
}
```

The `when` keyword introduces four conditions:

1. In the first condition, a variable `?mo` is matched by a `ManagedObject` object.
2. The second condition returns the subscribed `Operator` objects in the variable `?operator`.
3. The third condition is an event statement. The variable `?alarm1` points to an event object named `Alarm` with a single constraint: the value of the field `managedObject` is equal to the variable `?mo`.
4. The fourth condition is also an event statement, where the variable `?alarm2` points to an event object named `Alarm` with the following constraints: the value of the field `managedObject` is equal to the variable `?mo`, and the difference between the first event object timestamp and the second event object timestamp is greater than or equal to 1 and less than or equal to 5 clock ticks.

The `then` keyword introduces two actions:

1. The first action prints a message naming a managed object.
2. The second action executes an `?operator.report` method that uses the event object variables `?alarm1` and `?alarm2`.

Any alarm is automatically retracted from the working memory 6 clock ticks after its insertion.

Parent topic: [IRL keywords](#)

Related reference:

[after](#)
[before](#)
[occursin](#)
[timeof](#)

exists

The exists keyword tests whether a class condition is true.

Purpose

This keyword is used in the condition part of a rule to test whether the condition is true.

Context

Rule conditions

Syntax

```
exists condition;
```

Description

Use the exists statement in the condition part of a rule, on a simple condition, an in statement, or a from statement. The exists statement tests whether the condition is true. The contrary of exists is not (avoid not exists).

The exists statement returns true when any object in the working memory matches the condition.

- It returns true for a simple condition if the working memory contains at least one object that can match the condition.
- It also returns true with an in or from statement when the in or from statement returns true.

Because the condition does not discriminate which object was matched, you cannot bind an exists statement to an external variable. Any variable bound within the exists statement is local to the statement. No statement of the rule can contain any reference to the bound variable.

You can use the special variable ?this within the exists statement to designate the current working memory object being tested.

Example

The GeneralCustomerRequest rule uses the exists statement to verify that an item requested by a customer exists. The exists statement returns true when a single instance of the object is found, which is faster than providing all the items that match the customer's request.

```
rule GeneralCustomerRequest {
    when {
        CustomerRequest(?item:request);
        exists Item(?this == ?item);
    }
    then {
        System.out.println(?item + " are in stock.");
    }
};
```

In the CustomerRequest condition, the variable ?item is matched with the field request. In the exists statement, the Item object is represented by the variable ?this. The statement tests whether the Item object is equal to the variable ?item. If an Item object is matched, the println command in the action part prints that the items are in stock.

Parent topic: [JRL keywords](#)

Related reference:

[collect](#)
[fork](#)
[from](#)
[in](#)
[not](#)

filter

The filter keyword defines an agenda filter.

Purpose

This keyword is used to define the filter of an `agendafilter` in a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
  agendafilter = filter (variableName)
  {
    action1
    ...
    actionn
  }
};
```

Description

This format presents a way to define an agenda filter in the rule task. In this case, you write the code of the agenda filter within the task definition. Such code is similar to an IRL function with an [IlrRuleInstance](#) parameter and a Boolean return type. The *(variableName)* placeholder represents this `IlrRuleInstance` object. The inline code of the agenda filter can hold a reference to it. Ruleset variables are accessible from the agenda filter code. This agenda filter is applied to each instance of the rules that compose the task body. Only the instance for which the agenda filter returns `true` is executed. Others are not.

Parent topic: [IRL keywords](#)

Related reference:

[agendafilter](#)
[ruletask](#)

finalaction

The finalaction keyword declares a final action.

Purpose

This keyword is used to declare a final action in a ruleflow task.

Context

Rule task definitions

Syntax

```
finalaction {action1 ... actionn}
```

Description

A subflow task can have an initial action or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as functions. They are similar to an IRL function with the void return type and no arguments.

Parent topic: [IRL keywords](#)

Related reference:

[flowtask](#)

[functiontask](#)

[ruletask](#)

finally

The `finally` keyword introduces a control block that is executed after a `try` block.

Purpose

This keyword is used within a `try` statement that specifies a control block to be executed after the `try` block has been executed, including its exceptions.

Context

Functions or rule actions

Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

Description

The `try-catch-finally` statements establish a block of code for exception handling. The blocks all begin and end with curly braces.

The `finally` statement is executed if any portion of the `try` statement is executed, regardless of how the code in the `try` and `catch` statements completes.

Parent topic: [IRL keywords](#)

Related reference:

[catch](#)

[try](#)

firing

The `firing` keyword determines whether all the rules are executed.

Purpose

This keyword is used to determine whether all the rules of a rule task are executed or only one of them.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    [firing = allrules|rule;]
};
```

Description

The value following the `firing` keyword indicates whether all rules are executed (`allrules`, the default) or whether only one rule is executed (`rule`).

Note:

All instances of one rule are executed. If only one rule is executed, use the keyword [firinglimit](#). See also [ruletask](#) for more information.

Parent topic: [IRL keywords](#)

Related reference:

[allrules](#)
[rule](#)
[ruletask](#)

firinglimit

The `firinglimit` keyword specifies the number of rules to be executed.

Purpose

This keyword is used to specify the maximum number of rules to be executed in a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    [firinglimit = integer value;]
};
```

Description

The purpose of a rule task is to execute rules. You can modify the execution order and the number of rules that are executed.

The keyword `firinglimit` specifies how many rules are executed.

Parent topic: [IRL keywords](#)

Related reference:

[body \(in ruletask\)](#)
[ruletask](#)

flowtask

The flowtask keyword declares a subflow task.

Purpose

This keyword is used to declare a flow task.

Context

At the top level of rulesets

Syntax

```
flowtask flowTaskName
{
    [property propertyName = value;]
    [initialaction {action1 ... actionm}]
    [finalaction {action1 ... actionn}]
    [completionflag = value;]
    body {ruleflow}
};
```

Description

A flow task is one of the three kinds of tasks available in a ruleflow. A flow task describes how task executions are chained together and under which conditions.

Note:

1. The ruleflow syntax is described in [Grammar specification](#).
2. If a formal comment (/**...*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

A subflow task can have the following attributes:

property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value later using the API.

initialaction, finalaction

A subflow task can have an initial or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as IRL functions. They are similar to an IRL function with the void return type and no arguments.

body

The flow task body defines a ruleflow. It consists of task calls that are chained together through control statements. The control statements are: fork, goto, if, sequence, switch, while (break and continue)

Example

This example illustrates a flow task definition.

```
flowtask main
{
    initialaction =
    {
        turn = Constants.Player2;
        saved = new SavedGame(grid);
        for (var i = 0; i < 7; i++) for (var j = 0; j < 6; j++)
            insert(grid.array[i][j]);
    };
    finalaction =
    {
        grid = null;
        move = null;
        winner = Constants.None;
    };
};
```

```

    connect4 = null;
    ending = false;
    message = null;
};
body =
{
    while(!ending)
    {
        if (turn == Constants.Player1) ChooseMovePlayer1;
        else ChooseMovePlayer2;

        CheckMove;
        if (ending) break;

        UpdateDistance;
        ExpandObjects;
        DetectConnect4;
        if (ending) break;

        DetectGridFull;
        if (ending) break;
        ChangeTurn;
    }
    EndOfGame;
}
};

```

This task is executed in this order:

1. Its initial actions are executed first.
2. The body is executed.

The body consists of a while loop. Each loop corresponds to a move from a player in the Connect4 game. If the move causes a Connect4 object to be accomplished or the grid to be full, the game is finished and the while loop can be interrupted by a break statement. Here the ruleflow resumes after the while loop. In this particular example, the EndOfGame task is executed.

3. When the flow finishes execution, the final actions of the task are executed.

Parent topic: [JRL keywords](#)

Related reference:

[body \(in flowtask\)](#)
[functiontask](#)
[ruletask](#)

for

The for keyword introduces an execution loop.

Purpose

A statement for executing a statement block as many times as matching objects are found to start the action.

Context

Functions or rule actions

Syntax

```
for (initialize; test; increment) statement;
```

Description

Use the for statement in the action part of rules and in functions to execute multiple times the statement or a block of statements enclosed in braces ({}). The `initialize`, `test`, and `increment` arguments can be any valid expression, as in the Java™ programming language. The statement block can execute any IRL statement and any arithmetic expressions and method calls.

Example

The `ConnectivityUpdate` rule tests whether a new node exists in the network and adds a link to it from each of its neighbors.

```
rule ConnectivityUpdate{
  when {
    ?n: Node(state == NEW; ?neighbors: neighbors() );
  }
  then {
    for (int ?i = 0; ?i < ?neighbors().size(); ?i++) {
      ( (Node)?neighbors().elementAt(i) ).addLink(?n);
    }
    update (?n);
  }
};
```

This rule is processed as follows:

1. The rule condition matches an object `Node` when the field `state` equals the static value `NEW` and it binds the variable `?neighbors` with the vector field `neighbors()`.
2. If such a `Node` object is found, the action part can be executed. The variable `?i` is initialized to 0. The for statement loops for each neighbor and the `addLink(?n)` method updates the links from the neighbors to the new node.
3. The last action uses the update command to update the agenda with the new node.

Parent topic: [IRL keywords](#)

Related reference:

[break](#)
[continue](#)
[foreach](#)
[while](#)

foreach

The foreach keyword executes a statement block several times.

Purpose

The foreach statement is used in the action part of a rule for executing a statement block numerous times.

Context

Functions or rule actions

Syntax

```
foreach (type variable in expression ) statement;
```

Description

Use the foreach statement in the action part of rule and in functions to execute the statement, or a block of statements enclosed in braces ({}), on each element of a collection or array. The foreach statement introduces a new variable which you can use in the remainder of the execution block.

The variable type can be any valid type, as in the Java™ programming language. The expression can be any legal expression that returns a Collection or Array value type, as in the Java programming language. The statement block can execute any IRL statement and any arithmetic expressions and method calls.

This foreach statement slightly differs from the corresponding statement in Java in that it automatically filters out the objects that are not of the expected type. See the second example below.

Example

Here are two examples of foreach statements.

Example 1

The first example shows the basic behavior of the foreach statement:

```
rule ConnectivityUpdate{
  when {
    ?n: Node(state == NEW; ?neighbors: neighbors() );
  }
  then {
    foreach ( Node node in ?neighbors ) {
      node.addLink(?n);
    }
    update (?n);
  }
};
```

The ConnectivityUpdate rule tests whether a new node exists in the network and adds a link to it from each of its neighbors. This rule is processed as follows:

1. The rule condition matches an object Node when the field state equals the static value NEW, and it binds the variable ?neighbors with the vector field neighbors().
2. If such a Node object is found, the action part can be executed. The foreach statement loops for each neighbor. A new variable node is defined and the method addLink(?n) updates the links of the neighbors to the new nodes.
3. The last action uses the update command to update the agenda concerning the new node.

Example 2

The second example shows the filtering capability of the foreach statement<

```
List l = new ArrayList();
l.add(1);
l.add(null);
l.add("a string");

foreach ( String s in l ) {
```

```
System.out.println(s);  
}
```

This statement block produces a `string` because the instances that are not of the expected type (here `String`) are filtered out automatically.

Parent topic: [IRL keywords](#)

Related reference:

[break](#)
[continue](#)
[for](#)
[in](#)
[while](#)

fork

The fork keyword executes several statement blocks sequentially.

Purpose

The fork statement is a ruleflow statement used to execute several statement blocks in sequence.

Context

Body of flowtask blocks

Syntax

```
fork
{ruleflowStatement} &&
{ruleflowStatement}
[&& {ruleflowStatement}] *
```

Description

Use this statement to define several ruleflow statement blocks that execute sequentially. The flow continues after the fork statement when all the statement blocks of the fork statement have been executed.

Example

In the following fork statement, the first block is executes T1, then T2. When T2 finishes execution, the second block executes T3, then T4. When T4 finished, all the fork blocks are finished, T5 can be executed.

```
flowtask main
{
    body =
    {
        fork
        { T1; T2; }
        &&
        { T3; T4; }
    }
    T5;
};
```

Parent topic: [IRL keywords](#)

Related reference:

[body \(in flowtask\)](#)
[flowtask](#)

from

The `from` keyword supports relations between objects.

Purpose

The `from` statement is used in the condition part of a rule to express relations between objects.

Context

Rule conditions

Syntax

```
condition from expression;
```

Description

This statement returns `true` when the `from` condition matches an object that is returned by the `from` expression.

Using the `from` statement, you can directly access objects in the working memory and objects that are linked to other objects by fields or by method calls. Methods that are attached directly to the context object can be called in the expression part of the statement.

The `from` statement fosters performance because pattern matching is done on a reduced number of objects and not on all the objects in the working memory (see the first example below).

You can also use the `from` statement to access objects in a database. Using SQL queries in the expression part of the statement, you can retrieve information from a relational database (see the second example below).

Example

The first example uses the `from` keyword to extract a subset of objects while the second example uses the `from` keyword to query a relational database.

Example 1

The rule `FindBookStore` returns bookstores or stores with a book department in the city of London.

```
rule FindBookStore {
    when {
        ?s: Store(city == London);
        ?d: Department(type == Books) from ?s.department;
    }
    then {
        System.out.println(
            "The following book store/department was found: "
            + ?s.address);
    }
}
```

This rule is processed as follows:

1. The first condition provides the stores in London in variable `?s`.
2. The second condition uses this reduced set of stores to find which store has a book department.
3. If the `when` part is successful, the action part prints the stores found.

Example 2

The rule `FindCarOwner` finds the owner of a car by means of an SQL query to a relational database where employee information resides. The first condition matches a `CarToMove` object and returns a license number in variable `?l`. The second condition returns, in variable `?c`, a `Car` object found using `?l`, the license number. The `from` statement uses the `Person` object to return a name in variable `?n`, which matches the name of the car owner. An SQL query launched by the `getOwner()` method retrieves the car owner. The action part of the rule prints the person's name and telephone number, the car model, and the driver license number.

```
rule FindCarOwner {
    when {
        CarToMove(?l:license);
```

```
    ?c: Car(license == ?l);
    ?p: Person(?n:name) from ?c.getOwner();
}
then {
    System.out.println("Contact " + ?n + " at telephone "
        ?p.telephone + " to move a " + ?c.model +
        " with license: " + ?l);
}
}
```

Parent topic: [IRL keywords](#)

Related reference:

[collect](#)

[exists](#)

[in](#)

[not](#)

function

The function keyword defines a function to be called.

Purpose

The function keyword is used to declare a function that can be invoked in the action part of a rule.

Context

At the top level of rulesets

Syntax

```
function returnType functionName (argList) {statement}
```

Description

You can declare functions in a rule file before rule declarations. The function declaration consists of a header part and the statement block. The header part contains the return type of the function, the function name, and a list of arguments. The arguments are listed in pairs: the argument type and the argument name, separated by a comma. The statement block can contain any ILOG® Rule Language side statement, arithmetic expressions, and method calls. A statement block contains one or more statements. Use the keyword return to declare the value returned by the function, of the type returnType.

Example

```
function String buildMessage(Client ?c, ShoppingCart ?s)
{
    int ?a = ?s.getAmount();
    int ?t = ?c.getTotalAmount();
    if (?a > 100.0 && ?t > 1000.0)
        return("Dear "+ ?c.getName() + ", you are a GOLD
        customer and will receive a 10% discount today!!!");
    else if ( ?a > 100.0 )
        return("Dear "+ ?c.getName() + ", you are a SILVER
        customer and will receive a 5% discount today!!!");
    else if (?t > 1000.0)
        return("Dear "+ ?c.getName() + " , we can offer you a 10%
        discount on a shopping cart valued at over $100 today.");
    else
        return("Dear "+ ?c.getName() + " , we can offer you a 5%
        discount on a shopping cart valued at over $100 today.");
}
rule Promotion
{
    when {
        ?s:ShoppingCart(?a:amount;?id:clientNumber);
        ?c:Client(clientNumber == ?id; ?t:totalPurchaseToDate);
    }
    then {
        System.out.println(buildMessage(?c,?s));
    }
}
};
```

The function buildMessage takes a Client and a ShoppingCart as arguments and returns a message with the possible discount. The variable ?a is the current shopping cart amount and the variable ?t is the total purchased to date for the client. Three if statements identify which discount corresponds to the client.

Four cases are treated:

- The current shopping cart is valued at over \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at over \$100 and past purchases have not exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have not exceeded \$1000.

The Promotion rule provides a client with a promotion depending on the value of the shopping cart and past purchases. The condition ShoppingCart returns an amount in variable ?a and a client number in variable ?c . The second condition, Client, returns a total amount spent in variable ?t corresponding to the client number equal to ?c. In the action part of the rule, the function buildMessage is called and returns the appropriate message, which is displayed.

Parent topic: [IRL keywords](#)

Related reference:

[evaluate](#)

[for](#)

[if](#)

[modify](#)

[retract](#)

[return](#)

[throw](#)

[try](#)

[update](#)

[while](#)

functiontask

The functiontask keyword declares an action task.

Purpose

This keyword is used to declare an action task that defines a ruleflow function.

Note:

In product versions earlier than 7.1.x, action tasks were called function tasks.

Context

At the top level of rulesets

Syntax

```
functiontask functionTaskName
{
    [property propertyName = value;]
    [initialaction {action1 ... actionm}]
    [finalaction {action1 ... actionn}]
    [completionflag = value;]
    body {ruleflow}
};
```

Description

An action task is one of the three kinds of tasks available in a ruleflow.

Note:

1. The ruleflow syntax is described in [Grammar specification](#).
2. If a formal comment (/**...*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

An action task can have the following attributes:

property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value later using the API.

initialaction, finalaction

An action task can have an initial or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. Initial and final actions for action tasks are composed of inline IRL code, such as IRL functions. They are similar to IRL functions with a void return type and no arguments.

body

An action task is composed of inline IRL code, such as IRL functions. The body is similar to an IRL function with a void return type and no arguments.

Example

This example defines an action task whose body is the code given between the braced brackets.

```
functiontask UpdateDistance
{
    body =
    {
        int x = move.x;
        int y = move.y;
        for (var i = y + 1; i < 10; i++)
        {
            int p = grid.position(x,i);
            if (p == null) break;
            p.distance--;
        }
    }
};
```

```
        update(p);  
    }  
}  
};
```

This code is executed when the task is called to be executed in a ruleflow.

If initial actions are provided, they are executed before the task body. If final actions are defined, they are executed after the task body.

Parent topic: [IRL keywords](#)

Related reference:

[body \(in action task\)](#)

[flowtask](#)

[ruletask](#)

goto

The goto keyword jumps to another ruleflow statement.

Purpose

The goto keyword is a ruleflow statement used to jump to another statement in the ruleflow.

Context

Body of flowtask blocks

Syntax

```
goto label;
```

Description

The target statement must be labeled and this label is the one referenced in the goto statement.

You can use a goto statement in a fork or while statement if the following rules are respected:

- In a fork statement, the goto statement must not jump to a statement that does not belong to one of the fork blocks.
- In a while statement, the goto statement must not jump to a statement outside the while loop.

Example

```
flowtask main
{
  body =
  {
    start: if (turn == Constants.Player1) ChooseMovePlayer1;
           else ChooseMovePlayer2;
           CheckMove;
           if (ending) goto end;
           UpdateDistance;
           ExpandObjects;
           DetectConnect4;
           if (ending) goto end;
           DetectGridFull;
           if (ending) goto end;
           ChangeTurn;
           goto start;
    end: EndOfGame;
  }
};
```

The ruleflow consists of a loop implemented with the goto statement goto start, which jumps to the beginning of the flow. The loop can be interrupted under some conditions and this interruption is implemented with the goto statements goto end. The start and end parts are the labels of the first and last ruleflow statements, respectively.

Parent topic: [IRL keywords](#)

Related reference:

[body \(in flowtask\)](#)
[flowtask](#)

hasher

The hasher keyword defines a hashing expression.

Purpose

This keyword is used in a ruleset declaration to define a hashing expression.

Context

At the top level of rulesets

Syntax

```
ruleset rulesetName
{
    hasher (typeName variableName) = value;
};
```

Description

With the hasher keyword, you can define a hashing expression to optimize rule execution.

Parent topic: [IRL keywords](#)

Related reference:
[ruleset](#)

hierarchy

The hierarchy keyword defines a hierarchical property.

Purpose

This keyword is used to define a hierarchy that defines values and gives a partial order between those values.

Context

At the top level of rulesets

Syntax

```
hierarchy hierarchyName
{ "rootName"
  { "nodeName1" { "nodeName11" "nodeName12" ... }
    "nodeName2"
    "nodeName3" { "nodeName31" { "nodeName311" } }
  }
}
```

Description

IRL rules can have hierarchical properties. Specific predicates are defined to deal with hierarchical properties and get the rules that have a specific property value.

When a rule hierarchical property is accessed, its value is considered to be a string.

Example

This example defines three rules r1, r2, and r3. All define the location property which is a hierarchical property.

```
hierarchy Geography
{ "North"
  { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } }
    "Europe" { "France" { } "Germany" }
  }
}
propertydefinition { Geography location; }
rule r1 { property location = "North/USA/Texas/Paris" when { ... } then { ... }
}
rule r2 { property location = "France" when { ... } then { ... } }
rule r3 { property location = "North/Europe/France/Paris" when { ... } then { ... } }
```

In r1 and r3, the location value is given as a path from the hierarchy root to the final leaf, so as to resolve the ambiguity between the two nodes named Paris. The r2 location property value is France which refers to a unique node in the hierarchy.

Parent topic: [IRL keywords](#)

Related reference:

[match](#)
[propertydefinition](#)
[rule](#)

if

The `if` keyword executes one of two statement blocks.

Purpose

This statement is used for executing one of two statement blocks in the action part of rules or in functions.

Context

Functions or rule actions

Syntax

```
if (test) {statement}  
[else {statement}]
```

Description

The test can be any valid test, as in the Java™ programming language. If the test returns `true`, the first statement block is executed and the `else` block is skipped over. If the test returns `false`, the first block is skipped over and the statement block following the `else` keyword is executed. Any IRL statement can be executed within the statement block, as well as arithmetic expressions and method calls. A statement block consists of one or more statements. A single statement does not require the braces (`{}`). The `else` keyword and its statement block are optional.

Example

In this example, the rule `PromotionLevel` provides a client with a promotion depending on the value of the shopping cart and past purchases. The rule has a priority value of `1,000,000`. The condition `ShoppingCart` returns an amount in variable `?a` and a client number in variable `?c`. The second condition, `Client`, returns a total amount spent in variable `?t` corresponding to the client number equal to `?c`. In the action part of the rule, `if` statements are used to identify which promotion, if any, corresponds to the client.

```
rule PromotionLevel {  
    priority = 1,000,000;  
    when {  
        ?s:ShoppingCart(?a:amount;?id:clientNumber);  
        ?c:Client(clientNumber == ?id; ?t:totalPurchaseToDate);  
    }  
    then {  
        if (?a > 100.0 && ?t > 1000.0)  
            insert Promotion() {level = 1;}  
        else { if ( ?a > 100.0 )  
            insert Promotion() {level = 2;}  
            else if (?t > 1000.0)  
                System.out.println("Dear "+ ?c.name +  
                    ", we can offer you a 10% discount on a shopping  
                    cart valued at over $100 today.");  
        }  
    }  
};
```

Three cases are treated:

- The current shopping cart is valued at over \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at over \$100 and past purchases have not exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have exceeded \$1000.

Parent topic: [IRL keywords](#)

Related reference:

[else \(in if\)
rule](#)

if (in ruleflow)

The `if` keyword executes one statement block or another, depending on a Boolean expression value.

Purpose

This statement is a conditional ruleflow statement that executes one statement block or another depending on the return value of a Boolean expression.

Context

In a ruleflow body

Syntax

```
if (test)
  {ruleflowStatement}
[else {ruleflowStatement}]
```

Description

The test can be any valid test, as in the Java™ programming language. If the Boolean expression returns `true`, the ruleflow statement block that follows the `if` is executed. If the expression returns `false`, the ruleflow statement block corresponding to the `else` part is executed. The `else` part is not mandatory.

Example

```
flowtask main
{
  body =
  {
    while(!ending)
    {
      if (turn == Constants.Player1) ChooseMovePlayer1;
      else ChooseMovePlayer2;
      CheckMove;
      if (ending) break;

      UpdateDistance;
      ExpandObjects;
      DetectConnect4;
      if (ending) break;
      DetectGridFull;
      if (ending) break;
      ChangeTurn;
    }
    EndOfGame;
  }
};
```

Parent topic: [JRL keywords](#)

Related reference:

[else \(in rule\)](#)

[body \(in flowtask\)](#)

[flowtask](#)

import

The `import` keyword imports a Java™ class or package into a rule file.

Purpose

This statement is used to specify which Java classes a ruleset uses.

Context

At the top level of rulesets

Syntax

```
import [packageName
.] className |packageName
.*;
```

Description

The `import` declarations must be the first statements in a rule file. Import statements are not mandatory, but when present, they are always placed at the beginning of the rule file, before the ruleset header. After it is imported, the class or package is accessible from the rules. The scope of the `import` declarations is the ruleset file.

It is possible to import a specific class or an entire package.

For some Java packages, the `import` declaration is implicit.

```
import java.lang.*;
import ilog.rules.engine.*;
```

The rule interpreter adds these packages automatically, therefore you do not need to import any classes from these packages explicitly.

Example

If you declare the following imports:

```
import java.util.*;
import userPackage.UserClass;
```

the rule interpreter reads:

```
import java.lang.*;
import ilog.rules.engine.*;
import java.util.*;
import userPackage.UserClass;
```

Here are some examples of class names that might be used as a result of the two imports above:

Class Name	Qualified Class Name
Vector	java.util.Vector
java.util.Vector	java.util.Vector
UserClass	userPackage.UseClass
Number	java.lang.Number
IlrRule	ilog.rules.engine.IlrRule

Parent topic: [IRL keywords](#)

in

The `in` keyword supports relations between object values.

Purpose

This statement is used in the condition part of a rule to express the relations between values.

Context

Rule conditions

Syntax

```
condition in expression;
```

Description

Use this statement to test whether the value of the condition is a member of the set of values returned by the `in` expression. The expression can be an array, a `java.util.Vector`, `java.util.Enumeration`, or a `java.util.Collection` instance.

Using the `in` statement, you can retrieve not only objects in the working memory but also objects that are linked to other objects by fields or by method calls.

In combination with the `collect` statement, the `in` statement expresses logical statements such as `all`, `at least`, or `none` (see the examples below).

The `in` statement provides performance advantages since the pattern matching is done on a reduced number of objects and not on all the objects in the working memory.

Example

The first example retrieves objects even if they are not in the working memory. The other examples combine `in` and `collect` statements.

Example 1

This example uses the `in` statement to retrieve the items bought by the customer even if they are not inserted in the working memory. The variable `?d` returns the items bought. The method `boughtItems()` can return a user-defined array, a `java.util.Vector` value, or a `java.util.Enumeration` value.

```
?c: Customer();  
?d: Item() in ?c.boughtItems();
```

Example 2

This example uses the `in` statement within a `collect` statement to retrieve all the items bought by the customer with a price higher than 10. The `where` part uses the method `size()` to test that the collection has at least one element.

```
?c: Customer();  
collect Item(price > 10) in ?c.boughtItems() where (size() > 0);
```

Example 3

This example is similar to the previous example except that the `where` part now tests that all the items bought had a price higher than 10, by testing the equality of the results returned by the `size()` and `length()` methods.

```
?c: Customer();  
collect Item(price > 10) in ?c.boughtItems() where (size() == ?  
c.boughtItems().length() );
```

Parent topic: [JRL keywords](#)

Related reference:

[collect](#)
[exists](#)
[from](#)
[not](#)

in — !in (predicates)

In any expression, the `in` keyword tests whether a value belongs to a collection and or the `!in` keyword tests whether a value does not belong to a collection.

Purpose

A predicate to test whether a value belongs or does not belong to a collection.

Context

Any expression

Syntax

```
(1) [?var:] className (?var1 in|!in ?var2);
(2) [?var:] className (?var1 in|!in {? var1, var2, ..., varn});
```

Description

Use the `in` keyword as a binary predicate in a condition to test whether an operand is part of another operand. The `!in` keyword does the negation of this test.

The operand can be expressed in two ways:

- In syntax (1), the operand type must be an array or implement the `java.util.Collection` package.
- In syntax (2), the values that compose the enumeration must be homogeneous. The enumeration contains either primitive types only or objects only.

The tests are done with equals, as in Java™ collections.

Example

```
rule InTestEnum
{
    when {
        ?i: Integer(intValue() in {1, 2, 3});
    }
    then {
        out.println("InTestEnum rule");
    }
}

rule InTestCollection
{
    when {
        ?i: Integer(intValue() in getValues());
    }
    then {
        out.println("InTest rule");
    }
}
```

with:

```
function int[] getValues()
{
    int values = new int[2];
    values[0] = 1;
    values[1] = 2;
    return values;
}
```

Parent topic: [JRL keywords](#)

in (ruleset parameter)

The `in` keyword defines a ruleset parameter.

Purpose

This keyword is used in a ruleset declaration to define an `in` ruleset parameter.

Context

Ruleset declarations

Syntax

```
ruleset rulesetName
{
    [in typeName variableName;]
    [inout typeName variableName;]
    [out typeName variableName [= value];]
};
```

Description

Ruleset parameters are a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: `in`, `inout`, and `out`.

You cannot specify an initial value in the ruleset for the `in` and `inout` parameters because they are initialized by the [setParameters](#) method. However, you can initialize the `out` parameter.

Parent topic: [IRL keywords](#)

Related reference:
[ruleset](#)

initialaction

The `initialaction` keyword declares an initial action.

Purpose

This keyword is used to declare an initial action in a ruleflow task.

Context

Rule task definitions

Syntax

```
initialaction {action1 ... actionnn}
```

Description

A subflow task can have an initial action or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as IRL functions. They are similar to an IRL function with the void return type and no arguments.

Parent topic: [IRL keywords](#)

Related reference:

[flowtask](#)

[functiontask](#)

[ruletask](#)

inout

The `inout` keyword defines a ruleset parameter that is both an input parameter and an output parameter.

Purpose

The keyword in a ruleset declaration to define a parameter value that is provided as input to a ruleset when it is executed and can be modified by the execution process to produce an output value when the execution is completed.

Context

At the top level of rulesets

Syntax

```
ruleset rulesetName
{
    [in typeName variableName;]
    [inout typeName variableName;]
    [out typeName variableName [= value];]
};
```

Description

Ruleset parameters are a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: `in`, `inout`, and `out`.

You cannot specify an initial value in the ruleset for the `in` and `inout` parameters because they are initialized by the [setParameters](#) method. However, you can initialize the `out` parameter.

Parent topic: [JRL keywords](#)

Related reference:
[ruleset](#)

insert

The insert keyword inserts an object into the working memory.

Purpose

This statement is used in the action part of rules or in functions to create an object and insert it into the working memory.

Context

Functions or rule actions

Syntax

```
insert [ event[(timeExpression)] | logical ] object | typeName[arguments]
[ {statement1 ... statementn} ] ;
```

Description

The insert statement creates a new object with the *typeName* [arguments], executes statements on the scope of the object, and inserts the object into the working memory. When you specify *typeName*, you must specify the type of the created object and *arguments* are the arguments that you pass to the constructor. If you do not pass any arguments, the object is created with the default constructor. To allow the rule engine to apply the constructor, you must declare it as public in your Java™ code.

Optionally, statements can follow the object to be inserted into the working memory. Such statements operate implicitly on the inserted object. They are executed before the object is inserted into the working memory. If the statement block contains only one statement, the braces ({}) are not required.

When an object is inserted into the working memory as an event, a timestamp is automatically associated with it. The *timeExpression* expression defines an integer value that is used as the timestamp when an event is inserted.

You can remove an event object explicitly from the working memory by using the retract statement or the `IlrContext.retract` method.

Note: The usage of the event (in rule actions) keyword is deprecated as of V7.5.

To specify the uniqueness of the object, you must redefine the `java.lang.Object.equals` method in your Java class (see the example below). The process goes on as follows:

1. An object is created normally from the specified constructor.
2. This object is then tested by the `equals` method to determine whether an object that equals this object already exists in the working memory.
 - If an existing object in the working memory equals the object, the new object is not inserted into the working memory. The existing object is set to have a new justification that corresponds to the condition part of the rule that has just been executed.
 - If no existing object in the working memory equals the object, the new object is inserted into the working memory. This object is then maintained by the condition part of the rule that inserted it.

Maintenance of a logical object means that, as long as the condition part of a rule that justified the object remains true, the object is kept in the working memory. If the condition part becomes false, the object loses a justification. A logical object that loses its last justification is automatically retracted from the working memory.

To benefit of the Truth Maintenance System, you must redefine the `java.lang.Object.equals` method and define a `hashCode` method in your Java class, as follows:

```
public boolean equals (Object obj)
{
    if (obj == null || !(obj instanceof className))
        return(false);
    className myObj = (className)obj;
    return (fieldName.equals(myObj
j.fieldName));
}
...
```

```
public int hashCode()
```



```
{
    return (fieldName.hashCode());
}
```

Note:

If more than one field discriminates the object, the equals and hashCode methods must apply to each field.

Optionally, after an object has been inserted into the working memory, you can have a daemon run on them. To execute a daemon, the class must implement the interface [IlrAssertDemon](#) and define the method [asserted](#). For example, these code lines print a message every time a `className` object is inserted into the working memory:

```
public void asserted(IlrContext context)
{
    System.out.println("Asserted className
object in memory
                        with fieldName: " + fieldName
);
}
```

Example

Here are three different ways of using the insert keyword.

Example 1

This example uses an insert statement to insert an object into the working memory and another to pass an object value.

```
integer v = new Integer(2);
insert v;

insert Integer(13)
{
    System.out.println("Inserting the value: " + intValue());
}
```

This example runs as follows:

1. The first line creates an Integer object.
2. The first insert statement inserts this object into the working memory.
3. The second insert statement passes the value 13 as a parameter.
4. The engine interprets the block of executable statements as applying to the current object. When the method `intValue` is encountered, it is considered as the call to the method `intValue` on the current object, which returns 13 as a result.
5. The print statement prints out:

```
Inserting the value: 13
```

and the object is then inserted into the working memory.

Example 2

This example calls a constructor with no parameters, then executes the statement block. The code assigns values to the fields `color` and `shape` of the class `Form` *before* the object is inserted into the working memory.

```
insert Form()
{
    color=Red;
    shape=Circle;
}
```

Example 3

This example uses logical objects to manage alarms in a chemical plant.

```
rule CheckTemperature {
  when {
    Sensor(type==Temperature; value>150);
  }
  then {
    insert logical (new Alarm());
  }
};
rule CheckPressure {
  when {
    Sensor(type==Pressure; value>2);
  }
  then {
    insert logical Alarm();
  }
};
```

The CheckTemperature rule issues an alarm if the temperature is greater than 150. The CheckPressure rule issues the same alarm if the pressure is greater than 2. Here, the purpose is to have a single alarm if both temperature and pressure are exceeded.

If the rule CheckTemperature is executed, an alarm is created. Because the object is logical, it is maintained by the condition of the rule. If the temperature changes and is no longer greater than 150, the alarm is automatically removed from the working memory. However, if the temperature does not change and the rule CheckPressure is executed, instead of inserting a new alarm, the code justifies the existing alarm a second time, based on the condition part of the rule.

In an insert statement, you can write the object to be inserted in either of the following ways;

- Write the keyword `new` before the object and use parentheses around the object, as in the first example.
- Use neither parentheses nor the `new` keyword, as in the second example.

Parent topic: [IRL keywords](#)

Related reference:
[event \(in rule conditions\)](#)
[retract](#)

instanceof

The instanceof keyword tests whether an expression can be cast into a type.

Purpose

This keyword is used in the condition part or in the action part of rules and in functions to test whether an expression can be cast into a referenced type.

Context

Any expression

Syntax

```
[?var:] className (expression instanceof type)
```

Description

Use the instanceof operator in the condition part or in the action part of rules and in functions to test whether the passed expression (an operand of instanceof) can be cast into the passed type (another operand of instanceof) without raising a `ClassCastException` exception. If the expression is passed successfully, this test returns `true`. It returns `false` if the test is unsuccessful.

Example

```
rule InstanceofTest {
    when {
        ?a: Alarm(equipment instanceof Multiplex);
    }
    then {
        System.out.println("Alarm occurs on Multiplex");
    }
};
```

Parent topic: [JRL keywords](#)

Related reference:

[as](#)

instances

The instances keyword declares class instances.

Purpose

This keyword is used in a ruleset declaration to declare class instances.

Context

At the top level of rulesets

Syntax

```
ruleset rulesetName
{
    instances (className) = value|{value1,...,valuen};
};
```

Description

Use the instances keyword to declare on which instances of a specified class the rule engine works, instead of looking for objects of this class in the working memory.

There are two ways to define class instances:

- Give a value whose type implements the `java.util.Collection` package. This collection contains the objects of the specified class on which the rule engine works.
- Explicitly specify the instances used by the engine. Values are given to compute these instances. The value type is derived from the specified class.

Parent topic: [JRL keywords](#)

Related reference:

[ruleset](#)

isknown

The `isknown` keyword tests whether a class attribute has an initialized value.

Purpose

This keyword is used in the condition part or in the action part of a rule or in a function to test whether the attribute has a value.

Context

Any expression

Syntax

```
[?var:] className (?attribute isknown);
```

Description

In the case of XML binding, an attribute has no value in the following cases:

- When the value is not given in the XML file, or
- When the value is given in the XML file and set to `xsi:nil`.

In the case of Java™ classes, the test always returns `true` because a Java attribute of a Java class is always considered initialized.

Example

```
rule TestknownValue {  
    when {  
        ?u: User(address isknown);  
    }  
    then {  
        sendToAddress(?u.address);  
    }  
};
```

Parent topic: [IRL keywords](#)

Related reference:
[isunknown](#)

isunknown

The `isunknown` keyword tests whether a class attribute has an initialized value.

Purpose

This keyword is used in the condition part or in the action part of a rule or in a function to test whether the attribute has a value.

Context

Any expression

Syntax

```
[?var:] className (?attribute isunknown);
```

Description

In the case of XML binding, an attribute has no value in the following cases:

- When the value is not given in the XML file, or
- When the value is given in the XML file and set to `xsi:nil`.

In the case of Java™ classes, the test always returns `false` because a Java attribute of a Java class is never considered as not initialized.

Example

```
rule TestUnknownValue {
  when {
    ?u: User(address isunknown);
  }
  then {
    askForAddress(?u);
  }
};
```

Parent topic: [IRL keywords](#)

Related reference:
[isknown](#)

logical (in wait)

In a `wait` statement, the `logical` keyword designates that conditions must remain true for a `wait` statement to become true.

Purpose

This keyword is used in the `wait` statement to test whether conditions become valid or remain true during a designated waiting period.

Context

Rule conditions

Syntax

```
(1) wait logical [[until] expression] [{condition}];  
(2) ?variable  
: wait [logical] [until] expression {condition}  
    ...  
    then ...  
    timeout ?variable {action}
```

Description

In `wait` statements, use the keyword `logical` to indicate that the previously satisfied conditions must remain true for the `wait` statement to become true. If you do not use the `logical` keyword, all the previously satisfied conditions are ignored: they might become false during the waiting period and the `wait` statement might succeed.

Parent topic: [IRL keywords](#)

Related reference:

[wait](#)

match

The match keyword deals with hierarchical property values.

Purpose

The match predicates are used to handle hierarchical property values.

Context

Any expression involving objects

Syntax

```
rule property access value
match
string value;
rule property access value
  match up
string value;
rule property access value match updown
string value;
rule property access value match down
string value;
```

Description

The match predicates have two arguments: the first argument is an access to a rule hierarchical property. To make this access possible, you must define the rule property in a propertydefinition section. The second argument is a string.

Example

This example uses the following element:

- The Geography property, a hierarchical property that has been defined as follows:

```
hierarchy Geography { "North"
  { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } }
  "Europe" { "France" { } "Germany" } } }
```

- The location property, a rule property of type Geography,
- The ?rule variable

You can use the following keywords:

match

?rule.?location match "California" returns true if the location value is equal to California.

match up

?rule.?location match up "California" returns true if the location value is one of the values from the hierarchy root to California node.

match down

?rule.?location match down "California" returns true if the location value is one of the values from the "California" node to the hierarchy leaves (children of California node).

match updown

?rule.?location match updown "California" returns true if the location value either matches up or matches down California.

In this example, the rule task selects all the rules except the rule France.

```
hierarchy Geography { "North"
  { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } } }
```



```
    "Europe" { "France" { } "Germany" } }  
}  
propertydefinition { Geography location; }  
rule usa { property location = "USA"; when { ... } then { ... } }  
rule california { property location = "California"; when { ... } then { ... }  
}  
rule france { property location = "France"; when { ... } then { ... } }  
ruletask main  
{  
    body = dynamicselect()  
    {  
        Collection rules = collect IlrRule(?this.?location match up "California")  
            in getRuleset().getAllRules();  
        return rules;  
    }  
}
```

Parent topic: [IRL keywords](#)

Related reference:

[hierarchy](#)
[rule](#)

modify

The modify keyword modifies objects and updates the agenda.

Purpose

This statement is used in the action part of rules or in functions to modify objects and updates the agenda accordingly.

Context

Functions or rule actions

Syntax

```
modify [refresh] object {statement1 ... statementn};
```

Description

The statement block after the object argument can modify the state of an object of the working memory. Those statements can be arithmetic expressions or method calls. When an object of the working memory is modified, the agenda is also updated. If the block contains only one statement, the braces ({}) are not required.

Note:

In a modify statement, the statements that modify the object are specified directly in the statement block that follows the object, as shown in the syntax above. This is different from the update statement, where the modifications are specified before the update keyword, independently of the update statement.

If you apply the refresh keyword, the rules that remain true or become true after the modification of the object are reinserted into the agenda. Without the refresh keyword, only the rules that become true as a result of the modification are inserted into the agenda.

After the object has been updated in the working memory, the update can optionally launch a daemon. For this purpose, the class must implement the [IlrUpdateDemon](#) interface and must define the updated method (see the [update](#) keyword for details).

Example

```
rule InventoryUpdate {
    priority = 1,000,000;
    when {
        Sold(item == book; ?ISBN:ISBN);
        ?b:Book(?ISBN == ISBN; currentStock > 0);
    }
    then {
        modify ?b {
            currentStock-=1;
        }
    }
};
```

The InventoryUpdate rule has a 1,000,000 priority. The first condition matches an object Sold with field item equal to book and instantiates the variable ?ISBN with the value of the field ISBN. The second condition uses the variable ?ISBN to match an object Book and tests that the field current_stock is greater than 0. The condition is true only if the current stock is one or more. When both conditions are fulfilled, the action part can be executed. The object referenced by ?b is updated in the working memory by the modify statement; the field currentStock is decremented by one.

```
import tmp.*;

rule IncrementClock {
    when {
        ?t:Time();
    }
    then {
        modify refresh ?t {
            ?t.setSeconds(?t.getSeconds()+1);
        }
    }
};
```

```
};  
  }  
}
```

If the refresh keyword were not used, the rule would be executed only once. The refresh keyword is used to reinsert the rules matching a Time object into the agenda. In the example, the rule is executed forever.

Parent topic: [IRL keywords](#)

Related reference:

[refresh](#)

[update](#)

new

The new keyword creates a new object or array.

Purpose

This allocation expression is used in the action part of a rule or function to create a new object.

Context

Functions or rule actions

Syntax

```
new className
(
arguments
) ;
new className
[
dimension
] [
dimension
]* { initialValues
} ;
```

Description

In the case of arrays, you can specify how to initialize it with the `initialValues` method.

Example

```
function Integer getAnInt(int value)
{
    return new Integer(value);
}
function Integer[][] getAnArray(int dim1, int dim2)
{
    return new Integer[dim1][dim2];
}
function Integer[][] getAnInitializedArray()
{
    int[][] array2 = {{1,2}, {3,4}, {5,6,7}};
    return array2;
}
```

This example consists of three functions:

- `getAnInt` returns an `Integer` object.
- `getAnArray` returns a two-dimensional array with lengths equal to `dim1` and `dim2`.
- `getanInitializedArray` returns a two-dimensional array. The first dimension is itself an array of length 3, that is, the number of braced elements in the `initialValues` list.

Each array consists of an `Integer` array:

1. The first one is an array of length 2, initialized with the `Integer` values 1 and 2.
2. The second one is an array of length 2, initialized with the values 3 and 4.
3. The last one is an array of length 3, initialized with the values 5, 6, and 7.

Parent topic: [JRL keywords](#)

Related reference:

[function](#)
[return](#)

not

The not keyword tests the negation of a rule condition.

Purpose

This statement is used in the condition part of rules to test the negation of a rule condition.

Context

Rule conditions

Syntax

```
not condition;
```

Description

You use the not statement in the condition part of rules, to a simple condition, an in statement, or a from statement. The not statement returns true if the specified condition is false. A not condition returns true for a simple condition if the working memory does not contain any object that can match the condition. A not statement returns true with an in or from statement when the in or from statement returns false. Use not for the contrary of both exists *class* or just *class*.

A not condition cannot be bound to an external variable. The not condition returns true when no object in the working memory matches the condition. Hence it is invalid to bind a false condition to a variable.

Any variable bound within the not condition is local to the condition. The remainder of the rule cannot contain any reference to it.

You can use the special variable ?this within the not statement to designate the current working memory object being tested.

Example

```
rule GuestPromotion {
    when {
        ?g:Guest(?i:id);
        not GroupId(this.contains(?i));
    }
    then {
        ?g.promotions();
    }
}
```

The GuestPromotion rule verifies that a guest is not part of a group, in order to provide a promotion. The first condition uses the variable ?g to reference a Guest object and the variable ?i to return the field value id. The not statement is used to test whether this id is contained in the object GroupId. The method contains is applied to the variable ?this. The then part launches the method promotions().

```
rule LargestPercentStockChange {
    when {
        ?s: Stock(?c:currentValue; ?l:lastClosing; ?p:(?c-?l)/?l);
        not Stock(?c1:currentValue; ?l1:lastClosing; ?p < (?c1-?l1)/?l1);
    }
    then {
        System.out.println(?s.ticker + "has the largest change of: "+ ?p +"%");
    }
};
```

The purpose of the LargestPercentStockChange rule is to print out the stock with the largest percent change.

- The first condition, in the line starting with ?s, specifies a stock, and the rule variables named ?c and ?l are bound to the current stock value and the last closing value, respectively. The stock percent change is calculated and saved as ?p.
- In the second condition, the line contains the not keyword. As in the previous condition, a percent change is calculated with rule variables ?c1 and ?l1. The condition then compares the result with the previous calculated change ?p. The not keyword causes the condition to fail for any percentage change except the largest.

Parent topic: [IRL keywords](#)

Related reference:

[collect](#)
[exists](#)
[from](#)
[fork](#)
[in](#)

occursin

The occursin keyword defines a unary temporal constraint in an event condition.

Purpose

This keyword is used in the condition part of rules in an event condition to express a temporal constraint on a single event.

Context

Rule conditions

Syntax

```
[?var:] event className (?eventVar occursin interval);
```

Description

Deprecated as of V7.5.

An occursin test is satisfied if the timestamp of the event is included in the interval.

An interval is of the form [*lowerBound*, *upperBound*L], where a bound is either an expression evaluating to an integer, or the \$ sign which denotes infinity.

Example

```
rule CheckErrorReport {
  when {
    ?u: User();
    ?a1: not event Check(?this occursin [1, 60]);
  } then {
    ?u.report(?a1);
  }
};
```

An instance of the CheckErrorReport rule is created on each instance of the User class in the working memory when the timestamp of the Check event ?a1 does not occur between 1 and 60.

Parent topic: [IRL keywords](#)

Related reference:

[after](#)
[before](#)
[event \(in rule conditions\)](#)
[timeof](#)

ordering

The ordering keyword sets the order of the rules executed in a rule task.

Purpose

This keyword is used to specify how the rules are ordered and how many rules are executed in a rule task.

Context

Rule task definitions

Syntax

```
ruletask ruleTaskName
{
    [ordering = dynamic|sorted|literal;]
};
```

Description

The keyword ordering specifies how the rules are sorted. It takes these values:

- **dynamic**: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- **sorted**: The rules are sorted in decreasing order of static priority.
- **literal**: The rules are listed in the same order as they have been listed in the rule task body.

Parent topic: [IRL keywords](#)

Related reference:

[dynamic](#)
[ruletask](#)

out

The out keyword defines a ruleset output parameter.

Purpose

This keyword is used in a ruleset declaration to define a ruleset output parameter to exchange data between the application and the ruleset.

Context

At the top level of rulesets

Syntax

```
ruleset rulesetName
{
    [in typeName variableName;]
    [inout typeName variableName;]
    [out typeName variableName [= value];]
};
```

Description

Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: in, inout, and out.

You cannot specify an initial value in the ruleset for the in and inout parameters because they are initialized by the [setParameters](#) method. However, you can initialize the out parameter.

The out ruleset variables can be initialized in the ruleset.

You access the in, inout, and out parameters through the [EngineInput](#) and [EngineOutput](#) objects that are created when you execute your ruleset. Use the [setParameters](#) method to initialize the in and inout parameters.

Parent topic: [IRL keywords](#)

Related reference:
[ruleset](#)

overriding, overrides

The overriding and overrides keywords declare overriding relations.

Purpose

The overriding and overrides keywords are used to declare overriding relations.

Context

At the top level of rulesets

Syntax

```
overriding { "groupName1" overrides "groupeName2";  
  "groupName3" overrides "groupName4", "groupName5"; }
```

Description

You can organize rules in groups by using an `ilog.rules.group` rule property. The overriding keyword introduces overriding declarations which apply to the rules that compose a rule task and take groups as parameters. In the parameter block, the `overrides` keyword indicates which rule group is executed instead of which other rule groups: a declaration in which `group1` overrides `group2` means that when rules of `group1` are together with rules of `group2` in a rule task, the rules of `group2` are removed from the task and therefore not executed. Rule overriding is the last rule selection done before the task is executed.

If a rule does not define an `ilog.rules.group` property, a default one is created. Its value is the fully qualified name of the rule.

Example

```
import java.util.Collection;  
  
hierarchy Geography  
{  
  "North" {  
    "USA" {  
      "California"  
      ...  
    }  
    ...  
  }  
}  
  
propertydefinition  
{  
  Geography location;  
}  
  
overriding  
{  
  "california" overrides "usa";  
  "blacklisted" overrides "california";  
}  
  
rule usa  
{  
  property location = "USA";  
  when {  
    ...  
  }  
  then {  
    ...  
  }  
}  
rule california  
{
```

```

    property location = "California";
    when {
        ...
    }
    then {
        ...
    }
}
rule blacklisted
{
    property ilog.rules.group = "blacklisted";
    when {
        ...
    }
    then {
        ...
    }
}

ruletask main
{
    body = dynamicselect(){
        Collection rules = collect IlrRule(?this.?location match up "California")
in getRuleset().getAllRules();
        return rules;
    }
}

```

This example defines a rule task that selects all the rules in which the location property matches up the value California. The resulting selection contains the rules in which the value of the location property is one of the values in the Geography hierarchy from the root (North) to the California node.

The collected rules are usa and california. These rules have not defined an `ilog.rules.group` property. Therefore, the property value for these rules is their fully qualified names, here `usa` and `california`. The overriding declaration removes the `usa` rule from the selection and keeps only the `california` rule.

Parent topic: [IRL keywords](#)

Related reference:
[ruletask](#)

package

The package keyword declares a package.

Purpose

This keyword is used to declare rule packages that you can use to better manage IRL artifacts such as variables, functions, rules, and tasks.

Context

At the top level of rulesets

Syntax

```
package packageName {variables, functions, rules, tasks}
```

Description

Rule packages are held by the ruleset. A ruleset contains at least one package, the default package. The default package name is the empty string. IRL artifacts defined outside a package definition are considered to be part of the default package. An IRL artifact has a short name, which is the name used for its definition, and a fully qualified name, which is its short name prefixed with its package name. You can use IRL artifacts defined in a package A in another package B by either referencing the artifacts by their fully qualified name or if their package is imported into the package B, by using the use keyword. You cannot import the default package, but you can import the artifacts of the default package.

Example

```
function void displayPrice(double price)
{
    ...
}
package pricing
{
    variables {
        Customer customer;
    }
    rule isEligible
    {
        when {
            ...
        }
        then {
            ...
        }
    }
}

package europe.pricing
{
    use pricing.*;
    use displayPrice(double);
    ruletask main
    {
        body {
            isEligible
        }
        finalaction {
            displayPrice(3.2d);
        }
    }
}
rule FlightDisplayExpired {
};
```

This example defines two packages, pricing and europe.pricing. The displayPrice function belongs to

the default package. The `europa.pricing` package uses the `pricing` package and the `displayPrice` function.

Parent topic: [IRL keywords](#)

Related reference:

[flowtask](#)

[function](#)

[functiontask](#)

[rule](#)

[ruletask](#)

[use](#)

property

The `property` keyword declares a rule, ruleset, or ruleflow property.

Purpose

This keyword is used to declare specific properties for a rule, a ruleset, or a ruleflow in a rule file.

Context

Rulesets, ruleflows, and rules

Syntax

```
property propertyName = value;
```

Description

The *propertyName* argument is any identifier and value is of type `String`.

You can use the `property` keyword at two levels: the ruleset level and the individual rule level.

- At ruleset level, the property declaration must be shown after the `import` statement.
- At rule level, the property declaration is part of the rule header.

Users can declare properties at both the ruleset level and the rule level.

As a ruleflow property, this property is used to define a user property on the task. The engine does not interpret this property. You can retrieve its value later by using the API.

Example

The first example shows a ruleset property. The second example shows a rule property.

Example 1

When used at the ruleset level, as in this example, the `property` keyword is enclosed by a declaration that begins with the word `ruleset`, followed by any identifier. Two user-defined properties store the first and last names of the ruleset author.

```
ruleset Vacation {  
    property authorlastname = "Brans";  
    property authorfirstname = "Sue";  
}
```

Example 2

When used at the rule level, the `property` keyword is part of the header declarations. A user-defined property declares the date at which the rule was last changed. Note that the value is of type `String`.

```
Rule CdUpdate {  
    priority = 100;  
    property lastChangeDate = "22 02 06";  
    when { ...
```

Parent topic: [JRL keywords](#)

Related reference:

[flowtask](#)

[functiontask](#)

[rule](#)

[ruleset](#)

[ruletask](#)

propertydefinition

The `propertydefinition` keyword predeclares the types of rule properties.

Purpose

This keyword is used to predeclare the type of rule properties to make rule properties accessible as regular members of the rule and to make type checking possible.

Context

At the top level of rulesets

Syntax

```
propertydefinition { Type propertyName; }
```

Description

You must define rule properties in the `propertydefinition` section. For example, if a `location` property is defined in the `propertydefinition` section, you can write in a rule selection body:

```
body = select(?rule)
{
    String location = ?rule.location;
    return location.equals("London");
}
```

For hierarchical properties, use the hierarchy name as the property type. The property value is type-checked and considered as a string.

Example

```
hierarchy Geography
{
    "North America" {
        "USA" "Canada"
    }
}

propertydefinition
{
    Geography location;
    java.util.Date effectiveDate;
}

rule usa
{
    property location = "USA"
    property effectiveDate = java.util.Date(120000);
    when {
        ...
    }
    then {
        ...
    }
}

function void displayRuleProperties()
{
    IlrRule[] rules = getRulesetI().getAllRules();
    for(int i=0; i<rules.length; i++)
    {
        out.println("Location of " + rules[i].getName() + " is " +
rules[i].?location);
        out.println("Date of " + rules[i].getName() + " is " +
```

```
rules[i].?effectiveDate);  
    }  
}
```

A Geography hierarchy is defined. The propertydefinition section predeclares two properties, location and effectiveDate. The usa rule has two properties: location and effectiveDate. Their values are compatible with the type declared in the propertydefinition statement. The displayRuleProperties function displays the properties of the rules (here only usa), made easily accessible by the propertydefinition statement.

Parent topic: [IRL keywords](#)

Related reference:
[hierarchy](#)
[rule](#)

refresh

The refresh keyword requests a refresh of the agenda.

Purpose

This keyword is used in the modify or update statement to request a refresh of the agenda after an update.

Context

Functions or rule actions

Syntax

```
modify [refresh] object {statement1 ... statementn};  
update [refresh] object;
```

Description

If you specify the refresh keyword, the rules that remain true or become true after the modification of the object are reinserted into the agenda. If you do not specify the refresh keyword, only the rules that become true as a result of the modification are inserted into the agenda.

Note: In the decision engine you must add the **com.ibm.rules.engine.repeatabl**e property, see [Repeatabl](#)e rules.

Parent topic: [IRL keywords](#)

Related reference:
[modify](#)

retract

The retract keyword removes an object from the working memory.

Purpose

This keyword is used in the action part of rules or in functions to remove an object from the working memory.

Context

Functions or rule actions

Syntax

```
retract object;
```

Description

The object can be defined by a variable or by an expression that denotes an object associated with the current rule engine or the engine that you specify (context in the code sample below).

After the object has been retracted from the working memory, the removal can optionally execute a daemon. To execute a daemon, the object class must implement the interface [IlrRetractDemon](#) and define the method `retracted`.

Example

The first example shows a basic retract statement while the second example shows the implementation of a daemon after the object is removed.

Example 1

```
rule FlightDisplayExpired {
    when {
        ?f: Flight( status==ARRIVED;
                    landingTime + 30 < currentTime() );
    }
    then {
        retract ?f;
    }
};
```

The `FlightDisplayExpired` rule removes a `Flight` object after the arrival time has passed 30 minutes. The first condition returns a `Flight` object in the variable `?f`. The object field `status` must be equivalent to the static constant `ARRIVED`. The field `landingTime + 30` must be less than the time returned by the method `currentTime()`. If the condition is true, the object `?f` is removed from the working memory by the `retract` statement.

Example 2

```
public void retracted(IlrContext context)
{
    System.out.println("Retracted a className object from
                       memory with fieldName: " + fieldName);
}
```

This example prints a message each time a `className` object is retracted from the working memory.

Parent topic: [IRL keywords](#)

return

The return keyword returns to the caller.

Purpose

The return statement returns to the caller of the code that contains this statement.

Context

Only in IRL functions or inline IRL code in a task definition, such as initial actions, final actions, the action task body, an agenda filter, or a rule task body defined with the select or dynamicselect keyword.

Syntax

```
return [expression];
```

Description

A return statement with no expression is allowed in a function with the return type void. In the task definition, such a statement is authorized in initial actions, final actions, and the action task body.

A return statement with an expression is allowed in a function with any return type other than void. The type of the returned expression must be assignable to the function return type.

A return statement is not allowed in a rule on either side. It causes a parsing error.

Example

Example 1

This example illustrates a return statement with an expression. A function returning an int is declared. The return statements in this function are followed by int expressions.

```
function int getOtherPlayer(int player)
{
    if (player == Constants.Player1) return Constants.Player2;
    else return Constants.Player1;
}
```

Example 2

This second example illustrates a return statement with an expression. The return statement is shown in a rule task body declared with the select keyword. The returned value is of boolean type.

```
ruletask ExpandObjects
{
    ordering = dynamic;
    firing = allrules;
    body = select(?rule)
    {
        String ?task = ?rule.getProperties().getString("task","");
        return ?task.equals("ExpandObjects");
    }
};
```

Parent topic: [IRL keywords](#)

Related reference:
[function](#)

rule

The rule keyword declares a rule.

Purpose

This keyword is used to declare a rule.

Context

At the top level of rulesets

Syntax

```
rule ruleName {[priority = value;]
                [property propertyName = value;]
when {condition1 ... conditionn [evaluate (expression)]}
then {[action1 ... actionm]}
[else {[action1 ... actionp]}]
};
```

Description

A rule in IRL is composed of three parts:

- A header part, which defines the name of the rule, its priority, and properties.
- The condition part, which begins with the keyword when, also referred to by its side of the rule.
- The action part, which begins with the keyword then, also referred to by its side of the rule.

When the condition part ends with an evaluate statement, the action part can have an else part, executed if the evaluate statement returns false. If it returns true, the then part is executed.

If a formal comment (/**...*/) precedes the rule definition, the comment is saved so that it can be retrieved later using the API.

Example

```
rule VideoCallBilling {
  when {
    ?c:Customer(?p:phoneNo);
    ?v:collect (new videoCallCollection())
      Usage(phoneNo == ?p; type == videoCall)
      where (size() > 0);
  }
  then {
    float ?t = 0.;
    Enumeration ?enum = ?v.elements();
    while (?enum.hasMoreElements()) {
      Usage ?x = (Usage)enum.nextElement();
      ?t += ?x.charge();
    }
    System.out.println("Dear "+ ?c.name + "Your bill for
                      Video conference calls is : "+ ?t);
  }
};
```

The VideoCallBilling rule collects appropriate Usage objects, sums the charge, and prints the result with the client's name. The first condition returns the Customer object in variable ?c, with the field phoneNo stored in variable ?p. The second condition is a collect statement. This statement returns a videoCallCollection object in variable ?v. This collection object contains Usage objects with the field phoneNo equal to the variable ?p, phoneNo from the previous condition, and field type equal to videoCall. The where part of the collect statement verifies that one or more Usage objects have been collected. The then part of the rule declares two variables ?t, initialized to 0, and variable ?enum, set to the elements of the collection using the default collection method elements(). Using the while statement, the elements of the collection are enumerated and the Usage objects are accessed and referenced using the variable ?x. Variable ?t and the method charge() are used to sum the charge for each Usage object. The println statement prints the client's name and the total charge.

```
rule CheckTemperature {  
    priority = 1,000,000 + 1;  
    when {  
        Sensor(type==Temperature; value>150);  
    }  
    then {  
        insert Alarm();  
    }  
};
```

The CheckTemperature rule has a priority of 1,000,001. If an object Sensor with a field type equal to Temperature has a value greater than 150, the condition part is true and the action part is executed. An object Alarm is inserted into the working memory using the command insert.

Parent topic: [IRL keywords](#)

Related reference:

[evaluate](#)

[property](#)

[ruleset](#)

[ruletask](#)

[then](#)

[when](#)

rule (in ruletask)

The rule keyword sets a single rule as eligible for execution.

Purpose

This keyword is used in rule tasks to set a single rule as eligible to be executed.

Context

Rule tasks

Syntax

```
ruletask ruleTaskName
{
    firing = allrules|rule;
};
```

Description

The value that follows the firing keyword indicates whether all rules are executed (allrules, the default) or whether only one rule is executed (rule). When the firing value is set to rule, it means that all instances of the first eligible rules are executed.

Parent topic: [IRL keywords](#)

Related reference:
[ruletask](#)

ruleset

The ruleset keyword declares a ruleset.

Purpose

This keyword is used to declare a ruleset.

Context

At the top level of rulesets

Syntax

```
ruleset rulesetName
{
    [property propertyName = value;]
    [instances (className) = value|{value1,...,valuen};]
    [hasher (typeName variableName) = value;]
    [in typeName variableName [= value];]
    [inout typeName variableName [= value];]
    [out typeName variableName [= value];]
};
```

Description

It is not mandatory to declare rulesets. If you choose to do so, you must specify at least the ruleset name.

In addition to the ruleset name, a ruleset declaration can also contain the following elements:

- **property**

You can define properties on the ruleset.

- **instances**

You can specify class instances to declare on which instances for a specified class the rule engine works. If you do not specify class instances, the engine looks for objects of this class in the working memory.

There are two ways to define class instances:

- Give a value whose type implements `java.util.Collection`. Such a collection contains the objects of the specified class on which the rule engine works.
- Explicitly specify the instances used by the engine. Values are given to compute these instances. The value type is derived from the specified class.

- **hasher**

You can define a hashing expression to optimize rule execution.

- **in, out, inout**

You can define ruleset parameters as a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, function, or task definition in the ruleset.

Example

```
ruleset Connect4
{
    // Provides the grid, the player and the adversary
    in Grid grid;
    out SavedGame saved;

    // The latest move
    int turn;
    Position move;

    // Who's the winner, what is the connect4, and any other reason
    // to end the game.
    int winner = Constants.None;
```

```
Connect4 connect4;  
boolean ending = false;  
String message;  
  
// Where to find the position objects.  
instances(Position) = ?grid.getAllPositions();  
};
```

This example illustrates a ruleset declaration. The ruleset name is Connect4. Ruleset parameters are declared. One of them, named grid, is an input parameter. Its initial value is set by a call to `IlrContext.setParameters`. The parameter saved is an output parameter. Its value can be obtained from the application with the different API calls listed above. The ruleset variables turn, move, winner, connect4, ending, and message are local.

Instances are defined for the class Position. Each object of type Position is bound in a rule condition part. The engine does not look for those objects in the working memory but in the specified list obtained with `?grid.getAllPositions()`.

Parent topic: [IRL keywords](#)

Related reference:

[flowtask](#)

[function](#)

[functiontask](#)

[import](#)

[property](#)

[rule](#)

[ruletask](#)

ruletask

The ruletask keyword declares a rule task.

Purpose

The keyword used to declare a rule task.

Context

At the top level of rulesets

Syntax

```
ruletask ruleTaskName
{
    [property propertyName = value;]
    [algorithm = default|sequential;]
    [matchedclasses = matchedClasses]
    [iterator = value;]
    [ordering = dynamic|sorted|literal;]
    [firing = allrules|rule;]
    [firinglimit = integer value;]
    [agendafilter = agendaFilter]
    [initialaction {action1 ... actionm}]
    [finalaction {action1 ... actionn}]
    [completionflag = value;] [scope = scope]
    [body body]
};
matchedClasses is given a value in one of two ways:
matchedclasses = {class1, class2, ..., classn}
matchedclasses = value;
agendaFilter is defined in one of two ways:
agendafilter = filter(?instance) {action1 ... actions}
agendafilter = value;
body is defined in one of three ways:
body {ruleName1, ruleName2,..., ruleNamep}
body = select(?rule) {action1 ... actionq}
body = dynamicselect(?rule) {action1 ... actionr}
```

Description

A rule task is one of the three kinds of tasks available in a ruleflow. A rule task lists the rule or rules that define the rule task.

Note:

1. The ruleflow syntax is described in [Grammar specification](#).

2. If a formal comment (/*...*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

A rule task can have the following attributes:

- property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value by defining API class and methods.
- algorithm

The value that follows this keyword indicates which execution mode is used to execute the rules: the RetePlus mode (the algorithm value is then default, which is its default value) or the sequential mode (the algorithm value is sequential).
- ordering, firing, firinglimit

A rule task is designed to execute rules. You can set up some rule execution parameters, such as how the rules are ordered and how many rules are executed.

The keyword `ordering` specifies how the rules are sorted.

You can have rules sorted according to the following values:

- **dynamic**: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- `agendafilter`

Use the `agendafilter` keyword in the rule task definition to filter the rules that are actually executed, according to specified criteria.
- `initialaction`, `finalaction`

You can define an initial action or a final action, or both, for a rule task. Initial actions are executed before the task body. Final actions are executed after the task body. They are composed of inline IRL code, such as IRL functions. They are similar to an IRL function with the `void` return type and with no arguments.
- `scope`

A scope defines the rules that can be used in the ruletask, before the optional selection defined in the body is applied. The use of a scope is optional.
- `body`

A rule task consists of a list of rules that compose its body. You can specify the rule list either by passing the rule names explicitly (extension), or by using `select` or `dynamicselect` keywords so that the list is computed from the code (comprehension). The given code must return a `boolean` value.

The use of a task body is optional, but there must be at least a scope or a body.

Example

Example 1

In this rule task, the body consists of the rules `DetectConnect4` and `DetectGridFull`. The rule ordering is set to `literal`: the order in which the rules are listed in the body is kept for the execution.

```
ruletask DetectConnect4
{
    ordering = literal;
    body = { DetectConnect4, DetectGridFull }
};
```

Example 2

In this rule task, the body is defined by comprehension. To compute the body, the code is executed for each rule in the ruleset. The rules for which the return value is `true` are part of the rule task body. The others are not part of the body.

The ordering here is set to `sorted`: the rules are sorted in decreasing order of static priority before being executed.

The firing value is set to `rule`: only one rule is executed among the rules that compose the body, even if more than one are eligible to be executed.

The initial actions are executed before the task body. If final actions are provided, they are executed after the task body.

```
ruletask ChooseMovePlayer1
{
    initialaction =
    {
        move = null;
    };

    body = select(?rule)
    {
        String ?task = ?rule.getProperties().getString("task","");
        return ?task.equals("ChooseMovePlayer1");
    }
    ordering = sorted;
    firing = rule;
};
```

Parent topic: [IRL keywords](#)

Related reference:

[agendafilter](#)

[body \(in ruletask\)](#)

[flowtask](#)

[functiontask](#)

scope

The `scope` keyword defines a scope in a rule task.

Purpose

This keyword is used to define a scope in a rule task, that is, a superset of the rules that compose the rule task.

Context

Rule tasks

Syntax

```
ruletask main { body = select(?rule) { ... }  
  scope = {packageName1.*, ..., packageNamem.*,  
    group(groupName1), ..., group(groupNamep),  
    ruleName1, ..., ruleNamen} }  
ruletask main { body = select(?rule) { ... } scope = all; }
```

Description

You can define a scope with packages or rules. If you specify a package, all the rules of the package are part of the scope. When you choose to define a scope, the final task body cannot contain rules that are not part of the scope. Domains are converted into a scope. You cannot use domains and scope together; otherwise a parsing error is raised.

The `all` value for a scope means that the scope includes all the rules of the ruleset. This is equivalent to not defining a scope.

If the rule task runs in sequential mode and you have defined a scope, the bytecode generation is done at parsing time.

Example

In this example, the rule task keeps all the rules defined in the scope (always returns true): `pricing.r1` and `europe.pricing.r2`.

```
package pricing  
{  
  rule r1  
  {  
    when {  
      ...  
    }  
    then {  
      ...  
    }  
  }  
}  
package europe.pricing  
{  
  rule r1  
  {  
    when {  
      ...  
    }  
    then {  
      ...  
    }  
  }  
  rule r2  
  {  
    when {  
      ...  
    }  
    then {  
      ...  
    }  
  }  
}
```

```
    }  
  }  
}  
  
ruletask main  
{  
  scope = { pricing.*, europe.pricing.r2}  
  body = dynamicselect(?rule) {  
    return true;  
  }  
}
```

Parent topic: [IRL keywords](#)

Related reference:

[body \(in flowtask\)](#)

[body \(in ruletask\)](#)

[ruletask](#)

select

The select keyword selects a rule in a rule task.

Purpose

This keyword is used to select a rule in the list of rules that compose the rule task body.

Context

Rule tasks

Syntax

```
ruletask ruleTaskName
{
    body = select(?rule) {action1 ... actionq}
};
```

Description

You can either specify each rule name explicitly (extension) or have the list of rules computed from the code, which uses the select or dynamicselect keywords (comprehension). The given code must return a value of the type boolean. The selection method changes the way the filter is applied to the rules.

Note: There is no difference between dynamicselect and select. They behave the same way in that the statement is called each time the task is executed.

Parent topic: [IRL keywords](#)

Related reference:

[body \(in ruletask\)](#)
[dynamicselect](#)
[ruletask](#)

sequential

The `sequential` keyword specifies the execution mode of a rule task.

Purpose

This keyword is used to select the sequential execution mode for a rule task.

Context

Rule tasks

Syntax

```
ruletask ruleTaskName
{
    algorithm = default|sequential;
};
```

Description

The value that follows the `algorithm` keyword indicates which execution mode is used to execute the rules: the RetePlus mode (the `algorithm` value is then `default`, which is its default value) or the sequential mode (the `algorithm` value is `sequential`).

Parent topic: [IRL keywords](#)

Related reference:
[ruletask](#)

switch

The switch keyword executes one statement block or another according to an integer expression value.

Purpose

This ruleflow conditional statement is used to execute one statement block or another according to an integer expression value.

Context

Ruleflow body

Syntax

```
switch (expression)
{
    case value1:
        {ruleflowStatement}
    ...
    case valuen:
        {ruleflowStatement}
    default:
        {ruleflowStatement}
}
```

Description

The integer expression *expression* is evaluated. If a case block evaluates the specified expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

Example

```
ruleset r
{
    int count;
}

flowtask main
{
    body =
    {
        switch(count)
        {
            case 1:
            {
                T1;
            }
            case 3:
            {
                T2;
            }
            default:
            {
                T3;
            }
        }
    }
};
```

This example illustrates a switch statement. The count variable on which the switch is executed is a ruleset variable in this example. When the switch is executed, the value of the count variable is evaluated. Depending on its value, one of the switch statement blocks is executed. If the count equals 1, the first case block (case 1) is executed. If it equals 3, the second case block (case 3) is executed. Otherwise, the default block is executed.

Parent topic: [JRL keywords](#)

Related reference:

[flowtask](#)

[body \(in flowtask\)](#)

then

The then keyword declares the action part of a rule.

Purpose

This keyword is used to specify the action part of a rule, also known as the right-hand side.

Context

Rule actions

Syntax

```
then {  
  [insert|if|modify|update|retract|timeout|while]  
  action1 ...  
  [insert|if|modify|update|retract|timeout|while]  
  actionn  
}
```

Description

It can contain one or more action statements to be done when the rule is executed, or it can be empty. The variables defined in the condition part of a rule can be used in the action part of the rule, except the variables defined within not and exists statements. The timeout action must be associated with a wait condition.

Example

In this example, the ConnectivityUpdate rule tests whether a new node exists in the network and adds a link to such new node from each of its neighbors. The single rule condition matches an object Node when the field state equals the static value NEW and it binds the variable ?neighbors with the vector field neighbors(). If such a Node object is found, the action part can be executed. The variable ?i is initialized to 0. The for statement loops for each neighbor, updating the links of the neighbors with the new node by calling the method addLink(?n). The last action uses the update command to update the agenda concerning the new node.

```
rule ConnectivityUpdate{  
  when {  
    ?n: Node(state == NEW; ?neighbors: neighbors() );  
  }  
  then {  
    for (int ?i = 0; ?i < ?neighbors().size(); ?i++) {  
      ( (Node)?neighbors().elementAt(i) ).addLink(?n);  
    }  
    update (?n);  
  }  
};
```

Parent topic: [IRL keywords](#)

Related reference:

[else \(in rule\)](#)

[for](#)

[if](#)

[modify](#)

[retract](#)

[throw](#)

[try](#)

[update](#)

[wait](#)

[while](#)

throw

The throw keyword throws an exception.

Purpose

This statement is used in the action part of rules or in functions to throw an exception that makes control flow immediately to an exception handler.

Context

Functions or rule actions

Syntax

```
throw  
expression
```

Description

The throw statement requires a single expression that is a throwable object. Throwable objects are instances of any subclass of the Java™ Throwable class.

The exception in the throw statement is not the exception actually thrown. The thrown exception is an instance of the [IlrUserRuntimeException](#) class, a subtype of [IlrRuntimeException](#), which encapsulates the exception thrown in the throw statement. The inherited [getTargetException](#) accessor retrieves the exception thrown in the throw statement. This method has the following signature: public Throwable getTargetException()

Example

```
function void replaceValue(String name, Object newValue)  
{  
    String ?attr = find(name);  
    if (?attr == null)  
        throw new NoSuchAttributeException(name,this);  
    ?attr.valueOf(newValue);  
}
```

This example shows a function that replaces the attribute valueOf by an object called newValue. The function assigns the name string to an attr object and then tests whether the attr object can be found. If the attr object does not exist, the NoSuchAttributeException is thrown. The thrown exception can be caught either in Java code by catching an IlrUserRuntimeException or superclass object, or in an IRL rule or function by catching a NoSuchAttributeException or superclass object.

Note that, unlike the Java code, the declaration of the replaceValue function does not contain a throws statement specifying that the function can throw a NoSuchAttributeException instance.

Parent topic: [IRL keywords](#)

Related reference:

[try](#)

timeof

The timeof keyword accesses an event timestamp.

Purpose

This keyword is used to access an event timestamp.

Context

Functions or rule actions

Syntax

```
timeof (?eventVar)
?time
```

Description

Deprecated as of V7.5.

The value of the timeof operator applied to an event is the event timestamp returned as a long number. The variable ?time is synonymous with timeof(?this).

Example

The following rule prints a timestamp each time an Alarm object is inserted in the working memory.

```
rule TraceAlarms {
  when {
    ?a: event Alarm();
  } then {
    System.out.println(timeof(?a));
  }
};
```

Parent topic: [IRL keywords](#)

Related reference:

[after](#)

[before](#)

[event \(in rule conditions\)](#)

[occursin](#)

timeout

The `timeout` keyword is associated with a `wait` statement.

Purpose

This keyword is used in the action part of a rule and is executed if the associated `wait` statement fails.

Context

Functions or rule actions

Syntax

```
?variable
: wait [logical] [until] expression {condition}
    ...
  then ...
    timeout ?variable {action}
```

Description

Deprecated as of V7.5.

Use the `wait` keyword in the condition part of a rule to test whether conditions become valid or remain true during a specified waiting period. The `wait` statement creates a timer that delays the execution of a rule. You can include multiple timers in a rule and you can also specify an optional `timeout` statement for each timer. The `timeout` statement is part of the action part of a rule. If the `wait` statement fails, the `timeout` statement is executed.

Parent topic: [IRL keywords](#)

Related reference:

[then](#)
[wait](#)

try

The `try` keyword executes control statements.

Purpose

The `try` keyword is used to execute control statements with the ability to predeclare named exception handlers. A `catch` statement specifies exceptions that are caught if thrown within a `try` block. A `finally` statement specifies a control block that is executed after a `try` block is finished processing, including its exceptions, if any.

Context

Functions or rule actions

Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

Description

Use `try-catch-finally` statements in the action part of rules or in functions. Such statements define a block of code for exception handling. The `try`, `catch`, and `finally` blocks all begin and end with curly braces.

The `try` block must be followed by at least one `catch` or a `finally` statement, or both. The `try` statement governs the statements enclosed within it and defines the scope of any exception handlers associated with it.

The `catch` statements are designed to handle a specific type of exception. The code within a `catch` statement is executed if an exception is caught. The *exceptionType* parameters declared in the `catch` statement specify the type of exception that the statement can handle and also provide identifiers that the `catch` statement can use to refer to the exception object that it is handling. The identifier must be of type `Throwable` or one of its subclasses.

The `finally` statement is guaranteed to be executed if any portion of the `try` statement is executed, regardless of how the code in the `try` and `catch` statements completes.

Example

Example 1

In this example, the `division` rule includes a `try` statement in its action part. When the `?x` variable is equal to zero, the division throws an `ArithmeticException`. The exception is caught by the `catch` clause which prints a message.

```
rule division {
    when {
        ?x = Integer() ;
        ?y = Integer() ;
    }
    then {
        try {
            int ?z = ?y/?x;
        }
        catch (ArithmeticException e) {
            System.out.println("Exception message : " + e) ;
        }
    }
}
```

Example 2

This example describes two `catch` statements, with one handler for each of the two types of exceptions that can be thrown within the `try` block: `ArrayIndexOutOfBoundsException` and `IOException`. When an exception occurs, the first `catch` statement that has an exception type compatible with the thrown exception is executed.

```
try {
```

```
    ...
}
catch (ArrayIndexOutOfBoundsException e) {
    System.err.println("Caught ArrayIndexOutOfBoundsException: "
        + e);
}
catch (IOException e) {
    System.err.println("Caught IOException: " + e);
}
```

Example 3

This example shows a finally statement.

```
try{
    ...
}
finally {
    note("in finally");
}
```

Parent topic: [JRL keywords](#)

Related reference:

[catch](#)
[finally](#)
[throw](#)

until

The `until` keyword specifies an absolute time.

Purpose

This keyword is used in the `wait` statement to specify an absolute time period.

Context

Rule conditions

Syntax

```
(1) wait [[until] expression] {condition};  
(2) wait logical [[until] expression] [{condition}];  
(3) ?variable  
: wait [logical] [until] expression {condition}  
    ...  
    then ...  
        timeout ?variable {action}
```

Description

Deprecated as of V7.5.

Use the `wait` keyword to test whether conditions become valid or remain true during a specified waiting period. The `until` keyword indicates that the expression specifies an absolute time by which the `wait` statement must succeed. If you do not use the `until` keyword, the expression specifies a relative time, that is, a specific duration as the waiting period. The expression must return an integer value.

Parent topic: [IRL keywords](#)

Related reference:

[wait](#)

update

The update keyword updates an object.

Purpose

This keyword is used in the action part of rules or in functions to update a modified object in the working memory. You can use it in RetePlus mode to notify the rule engine of an object state change. The engine then matches the rules against the new object state, which can result in new rule instances being added to the agenda.

Context

Functions or rule actions

Syntax

```
update [refresh] object;
```

Description

You can define the *object* parameter as a variable or an expression that denotes an object associated with the current context or a specified context.

When an object is modified (for example, in a function or in Java™ code), the rules of the agenda might not be in a consistent state with respect to the new contents of the object. In such cases, you must use the update statement to notify the rule engine of the modification.

If the refresh keyword applies, rules that remain true or become true after the modification of the object are reinserted into the agenda. Without this keyword, only the rules that become true as a result of the modification are inserted into the agenda.

Note: In the decision engine you must add the **com.ibm.rules.engine.repeatabable** property, see [Repeatabable rules](#).

After an object has been updated in the working memory, the update statement can optionally execute a daemon. For this purpose, write a class that implements the interface [IlrUpdateDemon](#) and defines the method updated. For example, this method prints a message each time a *className* object is updated in the working memory:

```
public void updated(IlRuleContext context)
{
    System.out.println("Updated a className object in
                        memory with fieldName: " + fieldName);
}
```

Example

```
rule JobProcessing
{
    priority = -?a;
    when {
        ?n:NewJob(?s:size;?p:priority);
        ?proc:Server(?a:activity; ?id:identifier);
    }
    then {
        retract(?n);
        insert( new Job(?s,?p,?id));
        ?a = ?a + ?proc.updateActivity(?n);
        update refresh ?proc;
    }
};
```

The JobProcessing rule assigns new jobs to servers based on the server activity level. The rule uses a dynamic priority that is equal to the negation of the activity level. The lower the activity level, expressed by the variable ?a, the higher the priority. Hence, the server with the lowest activity level executes first.

The rule has two conditions. The first condition matches an object NewJob which is returned by the variable ?n. This condition contains variable ?s, which returns the value of field size, and variable ?p, which returns the value of field priority. The second condition matches an object Server, referenced by the variable ?proc.

This condition contains variable ?a, which returns the value of field activity, and variable ?id, which returns the value of field identifier.

The action part of the rule follows the keyword then and works as follows.

1. The first action applies the retract statement to the NewJob object, pointed to by variable ?n.
2. The second action applies an insert statement to a Job object with three fields, size, priority, and identifier, represented by the three variables ?s, ?p, and ?id. A new activity value is calculated by incrementing the activity ?a with the result of the method updateActivity. This method is called on the Server object pointed to by the variable ?proc.
3. Finally, the Server object is updated in the working memory using the update statement with the refresh keyword. Any rule matching a Server object in the condition part can be reinserted into the working memory.

Parent topic: [IRL keywords](#)

Related reference:
[modify](#)

use

The use keyword imports an IRL package or artifact.

Purpose

This keyword is used to import an IRL package or artifact.

Context

Packages

Syntax

```
use [packageName.] artifactName | packageName.*;
```

Description

The effect of referencing artifacts is different depending on whether they belong to the default package or not.

Using an artifact from another package

To use an artifact from another package, write this statement if the artifact is a variable, rule, or task:

```
use artifactName;
```

For functions, the syntax is more complex because you must uniquely identify the artifact that you want to import. To do so, include the function signature in the use statement:

```
use displayPrice(double);
```

Referencing artifacts in the default package

To reference artifacts from the default package in package B, you can import the artifacts one by one (not necessarily all of them) from the default package, using the use keyword. Only these artifacts are visible in package B.

When artifact names conflict, the following rule applies: If the default package defines an artifact named `artifactName` and package B defines an artifact also named `artifactName` and uses package A, all references to `artifactName` in B are considered to be references to the B artifact and priority is given to the local artifact. There is no way to reference the artifact in the default package.

Referencing another package that is not the default

To reference artifacts from package A (A not being the default package) in package B, you can do one of the following:

- You can write a use statement to import package A into package B. In this case, all its artifacts are visible in package B.
- If you can import artifacts from package A one by one (not necessarily all of them) using the use keyword. In this case, only the imported artifacts are visible in package B.

When artifact names conflict, the following rules apply:

- If package A defines an artifact named `artifactName` and package B defines an artifact also named `artifactName` and uses package A, all references to `artifactName` in package B are considered to be references to the B artifact and priority is given to the local artifact. To reference the A artifact, you must use its fully qualified name.
- If two packages A and B define an artifact named `artifactName`, and if package C uses packages A and B and references the `artifactName` artifact, an error is raised because it is impossible to give priority to a package against another. To avoid any ambiguity, use the fully qualified name.

Example

This example defines two packages, `pricing` and `europa.pricing`. The `displayPrice` function belongs to the default package. The `europa.pricing` package uses the `pricing` package and the `displayPrice` function.

```
function void displayPrice(double price)
{
```

```

    ...
}
package pricing
{
    variables {
        Customer customer;
    }
    rule isEligible
    {
        when {
            ...
        }
        then {
            ...
        }
    }
}

package europe.pricing
{
    use pricing.*;    All the pricing package is imported. All its artifacts are
visible in this package
    use displayPrice(double); The function displayPrice from the default package
is imported
    ruletask main
    {
        body {
            isEligible
        }
        finalaction {
            displayPrice(3.2d);
        }
    }
}

```

Parent topic: [IRL keywords](#)

Related reference:
[package](#)

variables

The variables keyword declares variables.

Purpose

This keyword is used to declare the variables that are associated with a ruleset.

Context

Packages

Syntax

```
variables {  
    typeName variableName [= value];  
    ...  
}
```

Description

You can declare the variables that are attached to a package. You can then reference these variables in rules, tasks, and functions, and in other variable initialization. You can use variables defined in a package from another package as soon as these variables are referenced with their fully qualified name or imported with the use keyword. The default package can also use other packages.

Note:

A fully qualified name is the short name of the variable prefixed with its package name.

use statements are at the same place as import statement in IRL code.

Example

```
package pricing  
{  
    variables {  
        Customer customer;  
    }  
    rule isEligible  
    {  
        when {  
            ...  
        }  
        then {  
            out.println("Customer is " + customer.name);  
        }  
    }  
}  
  
package europe.pricing  
{  
    use pricing.*;  
  
    ruletask main  
    {  
        initialaction {  
            out.println("Customer is " + customer.name); customer is the variable of  
the pricing package.  
        }  
        body {  
            isEligible  
        }  
    }  
}
```

The pricing package has defined a customer variable. This variable is used in a rule that belongs to the

pricing package. The europe.pricing package uses the pricing package. Therefore, all pricing artifacts are visible in the europe.pricing package, (for example, the customer variable).

Parent topic: [IRL keywords](#)

Related reference:
[package](#)

wait

The wait keyword incorporates time as a rule parameter.

Purpose

This keyword is used in the condition part of rules to add time as a rule parameter.

Context

Rule conditions

Syntax

```
(1) wait [[until] expression] {condition};
(2) wait logical [[until] expression] [{condition}];
(3) ?variable
: wait [logical] [until] expression {condition}
    ...
    then ...
        timeout ?variable {action}
```

Description

Deprecated as of V7.5.

Use the wait statement in the condition part of rules to test whether conditions become valid during a specified waiting period or whether conditions remain true for a waiting period. The waiting period or time frame can be an absolute time or a specific duration relative to the current time. If you provide no *expression* parameter, the wait is indefinite.

The wait statement creates a timer that delays the execution of a rule. You can include multiple timers in a rule and, optionally, a timeout statement for each timer.

In syntaxes (2) and (3), the logical keyword is used to specify that the previously realized conditions must remain true for the wait statement to become true. If the logical keyword is not used, all the previously realized conditions are ignored, that is, they might become false during the waiting period and the wait statement might succeed.

The keyword until is used to designate that the expression specifies an absolute time by which the wait statement must succeed. If until is not used, the expression specifies a relative time, that is, a specific duration as the waiting period. The expression must return an integer value.

The wait statement (3) might include a timeout statement that is part of the action part of a rule. If the wait statement fails, the timeout statement is executed.

The rule engine has a time counter that is updated by the application according to the application requirements. The initial value of the time counter is 0.

You can synchronize the time counter of the rule engine with an external clock by using the following methods of [llrContext](#):

- The [time](#) method returns the current time.
- The [nextTime](#) method increments time by 1.
- The [nextTime](#) method increments time by the specified increment.
- The [setTime](#) method sets time to a specified time.

Note:

It is the developer's responsibility to synchronize the time counter with any external clock according to the application requirements.

Example

Example 1

In this example, the DisplayScreen rule displays either a user-defined screen or a default screen.

The rule contains two condition statements and two action statements.

1. The first condition matches a Display object.

2. The second condition is a wait statement which specifies a wait duration of 5 time units. This condition tests whether a User object is not found in the working memory within the 5 time units.
3. The first action launches a displayScreen method applied to the user's response, represented by the variable ?r. This statement is executed if the wait statement succeeds.
4. If the wait statement fails, that is, a User object is not found in the working memory within the 5-unit time duration, the second action executes the timeout statement which calls the defaultScreen method.

```
rule DisplayScreen {
  when {
    Display();
    ?to: wait 5 {User(?r:response);}
  }
  then {
    displayScreen(?r);
    timeout ?to { defaultScreen(); }
  }
};
```

Example 2

In this example, the NetworkMonitoring rule checks the traffic load of network elements and inserts an alarm if an element load passes 95% of its maximum load value.

This rule contains six condition statements and two action statements.

1. The first condition matches a Network object with variable ?e returning the value of field element, the field status being equal to alive and the variable ?m returning the value of the field maxLoad.
2. The second condition tests that an Alarm object for the element referenced by variable ?e does not exist in the working memory.
3. The third condition returns the Traffic object in variable ?t, where the field element is equal to variable ?e and the variable ?l returns the value of the field load.
4. The first wait logical statement simply causes a wait of one time unit.
5. The second wait logical statement matches the object Traffic within one time unit, as in the third condition, and the current value of the field load, returned in variable ?l2, is verified to be less than 95% of the maxLoad, using variable ?m.
6. If the load is below this threshold, the first action statement applies a modify statement on object Traffic with the calculated derivative of the load change.
7. If the wait statement fails, that is, the load is above 95% for over one time unit, the timeout statement inserts an Alarm object with the element pointed to by the variable ?e, using the insert statement.

```
rule NetworkMonitoring {
  when {
    Network(?e:element; status == alive; ?m:maxLoad);
    not Alarm(?e);
    ?t:Traffic(element == ?e; ?l:load );
    wait logical 1;
    ?a:wait logical 1 {Traffic(element == ?e; ?l2:
    load && ?l2 < 0.95*?m);}
  }
  then {
    modify ?t {?t.derivative = ?l2-?l}
    timeout ?a {insert Alarm(?e);}
  }
}
```

Example 3

The WatchDogSample rule checks whether an Alarm exists for a network element and tests whether the problem is regulated within 10 time units.

This rule contains two condition statements and two action statements.

1. In the first condition, the variable ?a points to an Alarm object with the following constraints: variable ?id contains the value of the field networkElement, the value of the field severity is either LOW or MEDIUM, and variable ?t contains the value of the field time.
2. The second condition is a wait statement. This statement does not use the keyword logical. As a consequence, the previous conditions might turn false and the rule might still succeed. The until keyword means that the expression value is an absolute time and not a relative time. Therefore, the wait condition must become valid by time ?t+10, 10 time units past the value of ?t. The wait condition tests for an Alarm object with the value of the field networkElement equal to ?id, the field severity equal to CLEAR, and field time greater than ?t. If within 10 time units an Alarm object is found that fulfills these conditions, the retract action statement is executed.
3. The retract statement removes the Alarm object pointed to by ?a from the working memory. If the wait statement fails, the timeout statement is executed. The variable ?to designates the corresponding timeout statement for the wait statement.
4. The timeout statement prints a message We have a problem with ?id at time: ?t.

```
rule WatchDogSample {
  when {
    ?a: Alarm(?id:networkElement; severity == LOW ||
              severity == MEDIUM; ?t:time;)
    ?to: wait until ?t+10 {Alarm(networkElement == ?id;
                                severity == CLEAR; time > ?t);}
  }
  then {
    retract ?a;
    timeout ?to {
      System.out.println("We have a problem with "
                        + ?id " at time: " +?t);}
  }
};
```

Parent topic: [JRL keywords](#)

Related reference:

[timeout](#)

[logical \(in wait\)](#)

[until](#)

when

The when keyword declares the condition part of a rule.

Purpose

This keyword is used to introduce the condition part of a rule which consists of one or more conditions that must be satisfied for the rule to be executed.

Context

Rule conditions

Syntax

```
when {  
    [collect|evaluate|exists|not|wait] condition1  
    ...  
    [collect|evaluate|exists|not|wait] condition2  
}
```

Description

A condition can be instantiated with objects in the working memory. Using a class name, the rule engine matches instances of that class, or instances of any derived class. The keyword not, exists, or evaluate can precede a condition.

Use conditions for the following purposes:

- Match patterns with objects of the working memory to test their validity. For example, `Car(color == blue)` tests whether a Car object has a color field equal to the value blue.
- Instantiate a rule variable with a field of an object in the working memory. For example, given a Car object in the working memory with a field color, the condition `Car(?c:color)` instantiates the variable ?c with the color of the car.
- Bind a rule variable with an object of the working memory that fulfills the constraints of the condition. For example, `?c:Car(color == blue)` return a Car object with the field color equal to blue to variable ?c. If no such object exists in the working memory, the condition fails.

For each rule, the cardinality of rule instances in the agenda equals the product of the number of matched instances for each condition of the rule. For example, suppose we have two classes, Depart and Destination, and for each, three city instances: Paris, New York, and Tokyo.

```
rule example  
{  
    when {  
        Depart(?c1:city);  
        Destination(?c2:city);  
    }  
    then {  
        System.out.println("Possible city pairs are: "  
                             + ?c1 + ":" + ?c2);  
    }  
}
```

This rule generates nine instances of the rule in the agenda, including the three naive results such as Tokyo:Tokyo.

Example

Example 1

In this example, the first condition of the CarColor rule is true if a Car object exists with the field color equal to red. The second condition tests that no Car object exists with the field color equal to green. If both conditions are true, the action is done, printing the message There is a red car but no green car.

```
rule CarColor  
{  
    when {
```

```

    Car(color == red);
    not Car(color == green);
}
then {
    System.out.println("There is a red car but no green car.")
}
}

```

Example 2

In this example, the first condition matches a Request object, binding the three variables ?m, ?h, and ?l with the fields mark, highPrice, and lowPrice, respectively.

The rule CarsAvailableAtPriceRange searches for a car from the information supplied by the user. The rule has two condition statements and one action statement. To execute the action statement, the two condition statements must be fulfilled.

The second condition matches a UsedCar object, with the following constraints: the field mark equals ?m and the field price has a value that is between ?h and ?l. If a UsedCar object is found that fulfills these constraints, it is bound to the variable ?car. The action part of the rule prints out any car found, for example, A 1996 BMW 573i is available for you. The action part might print multiple cars because each UsedCar object fulfilling the constraints generates a separate rule instance. For more information on rule instances of the agenda, see [Agenda](#)).

```

rule CarsAvailableAtPriceRange {
    when {
        Request(?m: mark; ?h: highPrice; ?l: lowPrice );
        ?car: UsedCar(mark equals  ?m ; price < ?h & > ?l)
    }
    then {
        System.out.println("A " + ?car.year + " " + ?car.mark
                           + " " + ?car.model + " is available for you.");
    }
}

```

Parent topic: [IRL keywords](#)

Related reference:

[collect](#)
[evaluate](#)
[exists](#)
[not](#)
[then](#)
[wait](#)

where

The where keyword defines a constraint in a collect statement.

Purpose

This keyword is used in a collect statement in the condition part of a rule to contain tests that the collection object must fulfill.

Context

Rule conditions

Syntax

```
[?variable:] collect [(expression)] collectionTarget  
[where (collectionTest1 ... collectionTestn)];
```

Description

You can leave this statement empty.

Parent topic: [IRL keywords](#)

Related reference:
[collect](#)

while

The `while` keyword executes a statement block.

Purpose

This statement is used in the action part of rules or in functions to execute a statement block numerous times.

Context

Functions or rule actions

Syntax

```
while (test) [{statement1; ... statementn;}]
```

Description

The *test* argument can be any legal test, as in the Java™ programming language. The *statement* block can execute any IRL statement, arithmetic expressions, and method calls. Empty statement blocks are valid. If you specify only one *statement* block, the braces `{}` are not mandatory.

Example

```
rule StockDropWarning {
  when {
    ?c:Client(?a:account_number;?p:percentWarningMark);
    ClientStockList(account_number == ?a; ?s:stockList);
  }
  then {
    Enumeration ?enum = ?s.elements();
    while (?enum.hasMoreElements()) {
      Stock ?x = (Stock)enum.nextElement();
      if (((1.-?p)*?x.purchasePrice) > ?x.currentPrice) {
        System.out.println("Dear "+ ?c.name + "Your stock "
          + ?x.ticker + " has dropped " + ( (?x.purchasePrice
            - ?x.currentPrice) / ?x.purchasePrice ) +
          " percent.");
      }
    }
  }
};
```

The rule `StockDropWarning` checks the percent change of a client's stocks and warns if a stock price is below its purchase price by a certain percentage. The condition `Client` returns an account number in variable `?a` and a decimal value `percentWarningMark` in variable `?p`. The second condition, `ClientStockList`, returns a stock list in variable `?s` corresponding to the account number equal to `?a`. In the action part of the rule, the `while` statement iterates on every stock in the stock list `?s` and check the difference between the purchase price and the current price. If the stock price decreased in `?p` percent below the purchase price, the `println` function in the action statement prints a message.

Parent topic: [IRL keywords](#)

Related reference:

[break](#)
[continue](#)
[for](#)
[foreach](#)

while/break/continue (in ruleflow)

The while keyword executes a loop.

Purpose

This ruleflow conditional statement is used to execute a loop while a Boolean expression value remains true.

Context

Ruleflow body

Syntax

```
while (test)
    {ruleflowStatement}
```

Description

The *test* argument can be any legal test, as in the Java™ programming language. As long as this value remains true, the statements inside the while block are executed. Any ruleflow statement can be in a while block. When the value becomes false, the engine stops executing the while block and resumes ruleflow execution after the while statement.

A break statement can interrupt a while loop. When the engine encounters a break statement, the while loop is interrupted and ruleflow execution continues after the while statement.

You can include a continue statement in a while loop. When the engine encounters a continue statement, ruleflow execution returns to the while statement again, reevaluates the *test*, and continues ruleflow execution according to this value.

Example

```
ruleset Connect4
{
    // The latest move
    int turn;
    // Who's the winner, what is the connect4, and any other reason
    // to end the game.
    int winner = Constants.None;
    Connect4 connect4;
    boolean ending = false;
};
flowtask main
{
    body =
    {
        while(!ending)
        {
            if (turn == Constants.Player1) ChooseMovePlayer1;
            else ChooseMovePlayer2;
            CheckMove;
            if (ending) break;
            UpdateDistance;
            ExpandObjects;
            DetectConnect4;
            if (ending) break;
            DetectGridFull;
            if (ending) break;
            ChangeTurn;
        }
        EndOfGame;
    }
}
```

This example illustrates a while statement. The while block is executed while the ending variable value remains false (because then !ending remains true).

If the ending Boolean value becomes true, a break statement can interrupt the loop.

Parent topic: [IRL keywords](#)

Related reference:

[body \(in flowtask\)](#)

[break](#)

[continue](#)

[flowtask](#)

IRL grammar

The ILOG® Rule Language (IRL) grammar conforms to a specific notation and has naming restrictions. It is composed of several operators that are a subset of the operators provided in the Java™ programming language.

Operator summary

The operators of the ILOG Rule Language are a subset of the operators provided in the Java programming language.

Naming restrictions

A number of keywords are reserved and naming restrictions apply to package names.

Grammar notation

The IRL grammar notation follows the notation presented in *The Java Language Specification* by James Gosling, Bill Joy and Guy Steele (Addison Wesley, 2nd Edition, 2000).

Grammar specification

The ILOG Rule Language (IRL) defines a lexical grammar.

Parent topic: [ILOG Rule Language \(IRL\)](#)

Operator summary

The operators of the ILOG® Rule Language are a subset of the operators provided in the Java™ programming language.

The ILOG Rule Language operators are summarized in the table below.

The column marked P presents the precedence of the operator: the larger the value the higher the precedence. The column marked A represents the Associativity of the operator. Given an expression with several operators of the same precedence, the associativity determines the priority.

Table 1. ILOG Rule Language Operators

Operator	Operand Type(s)	Operation Performed	P	A
.	object, member	object member access	10	L
[]	array, int	array element access	10	L
(args)	method, arglist	method invocation	10	I
+	String, String	String concatenation	10	L
++, --	variable	post-increment, decrement	10	L
++, --	variable	pre-increment, decrement	9	R
+, -	number	unary plus, unary minus	9	R
!	boolean	boolean NOT	9	R
new	class, arglist	object creation	8	R
(type)	type, any	cast (type conversion)	8	R
*, /, %	number, number	multiplication, division, remainder	7	L
+, -	number, number	addition, subtraction	6	L
+	string, any	string concatenation	6	L
<, <=	number, number	less than, less than or equal	5	L
>, >=	number, number	greater than, greater than or equal	5	L
instanceof	reference, type	type comparison	5	L
as	reference, type	type conversion	5	L
==	primitive, primitive	equal (have identical values)	4	L
!=	primitive, primitive	not equal (have different values)	4	L
==	reference, reference	equal (refer to same object)	4	L
!=	reference, reference	not equal (refer to different objects)	4	L
&&	boolean, boolean	conditional AND	3	L
	boolean, boolean	conditional OR	2	L
=	variable, any	assignment	1	R
*, /=, %=, +=, -=,	variable, any	assignment with operation	1	R

Parent topic: [IRL grammar](#)

Naming restrictions

A number of keywords are reserved and naming restrictions apply to package names.

Deprecated as of V7.5.

You must adhere to the naming restrictions.

The reserved keywords are:

- break
- catch
- collect
- continue
- else
- evaluate
- event
- exists
- finally
- hasher
- if
- instanceof
- instances
- isknown
- isunknown
- logical
- new
- not
- refresh
- return
- throw
- timeof
- timeout
- try
- until

Important:

Reserved keywords cannot be used as identifiers, unless otherwise indicated in the grammar.

Package Names

The reserved keyword event cannot be used as an identifier but can be used as a package name.

@ Operator

The @ character can be used to prefix an identifier that might otherwise be interpreted as an IRL keyword. An identifier can be a variable name, a field, a method, or a class name.

Parent topic: [IRL grammar](#)

Grammar notation

The IRL grammar notation follows the notation presented in *The Java™ Language Specification* by James Gosling, Bill Joy and Guy Steele (Addison Wesley, 2nd Edition, 2000).

Terminal symbols, that is, the language keywords, are shown in **Courier bold font** in the syntactic and lexical rules presented in this chapter.

Nonterminal symbols are shown in *italic font* in the syntactic rules. A definition of a nonterminal symbol starts with the symbol followed by a colon. For example:

```
PropertyEntry:  
    property  
    PropertyName  = PropertyValue;
```

states that the nonterminal symbol *PropertyEntry* has a single definition, which comprises the terminal token `property` followed by a *PropertyName*, the token equals sign, followed by an *Identifier*, followed by the token semicolon.

A definition might have several rules. Each rule is described on a separate line. The following example shows a nonterminal symbol with two rule definitions:

```
TestAssignmentList:  
    TestAssignment  
    TestAssignmentList;  TestAssignment
```

The syntactic definition of the nonterminal symbol *TestAssignmentList* can be either *TestAssignment* or *TestAssignmentList*, followed by the token semicolon, followed by *TestAssignment*.

The subscripted suffix “*opt*” indicates an optional symbol. It might be shown after a terminal or nonterminal symbol. For example:

```
WaitStatement :  
    wait untilopt Expression
```

is equivalent to:

```
WaitStatement :  
    wait Expression  
    wait until Expression
```

When the words “one of” follow the colon in a grammar definition, they signify that each of the symbols on the following line or lines is an alternative definition. For example, the following lexical grammar rule:

```
AssignmentOperator: one of  
    = *= /= %= += -=
```

is a convenient abbreviation for:

```
AssignmentOperator:  
    =  
    *=  
    /=  
    %=  
    +=  
    -=
```

The words “but not” are used to indicate symbols that are excluded from the rule definition. For example:

```
ReducedAssignmentOperator:  
    AssignmentOperator  but not -=
```

is equivalent to:

ReducedAssignmentOperator: one of
= *= /= %= +=

An explanatory phrase in parentheses might be shown after some expressions.

Parent topic: [IRL grammar](#)

Grammar specification

The ILOG® Rule Language (IRL) defines a lexical grammar.

Note:
A bullet (•) before a grammar rule indicates that the terminal symbol in the rule is detailed in [IRL keywords](#).

Table 1. General form of an IRL program. Table shows the general form of an IRL program.

General form of an IRL program	
	<i>RuleSetDefinition</i> : <i>ImportOrUseDeclaration</i> opt <i>RulesetPropertyDefinition</i> opt <i>HierarchicalPropertyDeclaration</i> opt <i>PropertyTypingDeclaration</i> opt <i>RuleOverridingDeclaration</i> opt <i>VariableDeclaration</i> opt <i>PackageList</i> opt
Package definition	
	<i>PackageList</i> : <i>Package</i> <i>PackageList</i> opt
	<i>Package</i> : <i>PackageDefinition</i> <i>PackageContent</i>
•	<i>PackageDefinition</i> : package <i>Identifier</i> { <i>ImportOrUseDeclaration</i> opt <i>VariableDeclaration</i> opt <i>PackageContentList</i> }
	<i>PackageContentList</i> : <i>PackageContent</i> <i>PackageContentList</i> opt
•	<i>PackageContent</i> : <i>FunctionDefinitionList</i> opt <i>RuleDefinitionList</i> opt <i>TaskDefinitionList</i> opt
Import or use declaration	
	<i>ImportOrUseDeclaration</i> : <i>SingleImportOrUseDeclaration</i> <i>ImportOrUseOnDemandDeclaration</i>
•	<i>SingleImportOrUseDeclaration</i> : ImportOrUseKeyword <i>PackageName</i> . *;
•	<i>ImportOrUseOnDemandDeclaration</i> : ImportOrUseKeyword <i>TypeName</i> ;
	<i>ImportOrUseKeyword</i> : one of

	<div>import</div> <div>use</div>
Ruleset property definition	
	<div>RulesetPropertyDefinition :</div> <div>ruleset identifier</div> <div>{ PropertyListopt RulesetParameterListopt HasherDefinitionListopt InstanceDefinitionListopt }</div>
	<div>PropertyList:</div> <div>PropertyEntry PropertyListopt</div>
<div>•</div>	<div>PropertyEntry:</div> <div>property PropertyName = PropertyValue;</div>
	<div>RulesetParameterList:</div> <div>RulesetParameter RulesetParameterListopt</div>
	<div>RulesetParameter:</div> <div>InRulesetParameter InOutRulesetParameter OutRulesetParameter</div> <div>LocalRulesetParameter</div>
<div>•</div>	<div>InRulesetParameter:</div> <div>in Type Variable;</div> <div>in Type Variable = Expression;</div>
<div>•</div>	<div>InoutRulesetParameter:</div> <div>inout Type Variable;</div>
<div>•</div>	<div>OutRulesetParameter:</div> <div>out Type Variable;</div> <div>out Type Variable = Expression;</div>
<div>•</div>	<div>LocalRulesetParameter:</div> <div>Type Variable;</div> <div>Type Variable = Expression;</div>
<div>•</div>	<div>HasherDefinitionList:</div> <div>hasher (ReturnType Variable) = Expression;</div>
<div>•</div>	<div>InstanceDefinitionList:</div> <div>instances (ReturnType) = Expression;</div> <div>instances (ReturnType) = {InstancesList};</div>
	<div>InstancesList:</div>

	<i>Expression</i> <i>Expression, InstanceList</i>
	<i>Type</i> : one of boolean char byte short int long float double String <i>ClassName</i>
	<i>ReturnType</i> : void <i>Type</i>
Variable declaration	
•	<i>VariableDeclaration</i> : variables { <i>LocalRulesetParameterList</i> }
	<i>LocalRulesetParameterList</i> : <i>LocalRulesetParameter</i> <i>LocalRulesetParameterList</i>
Hierarchical property declaration	
	<i>HierarchicalPropertyDeclaration</i> : hierarchy <i>identifier</i> { <i>HierarchicalNodeDefinition</i> }
•	<i>HierarchicalChildren</i> : { <i>HierarchyDefinitionList</i> }
	<i>HierarchyDefinitionList</i> : <i>HierarchicalNodeDefinition</i> <i>HierarchyDefinitionList</i> opt
	<i>HierarchicalNodeDefinition</i> <i>STRING_LITERAL</i> <i>HierarchicalChildren</i> opt
Property typing declaration	
•	<i>PropertyTypingDeclaration</i> : propertydefinition { <i>PropertyTypingList</i> }
	<i>PropertyTypingList</i> : PropertyTyping <i>PropertyTypingList</i> opt
	<i>PropertyTyping</i> : <i>Type</i> <i>Variable</i> ;
Rule overriding declaration	
•	<i>RuleOverridingDeclaration</i> : overriding { <i>RuleOverridingList</i> }

	<i>RuleOverridingList :</i> <i>RuleOverriding RuleOverridingListopt</i>
	<i>RuleOverriding:</i> <i>STRING_LITERAL overrides OverriddenList</i>
	<i>OverriddenList:</i> <i>STRING_LITERAL</i> <i>STRING_LITERAL , OverriddenList</i>
Function definition	
	<i>FunctionDefinitionList :</i> <i>FunctionDefinition FunctionDefinitionList</i>
•	<i>FunctionDefinition :</i> <i>function ReturnType FunctionName (</i> <i>FunctionArgumentList)</i> <i>{ FunctionActionList }</i>
	<i>FunctionArgumentList:</i> <i>FunctionArgument</i> <i>FunctionArgument, FunctionArgumentList</i>
	<i>FunctionArgument:</i> <i>ReturnType Variable</i>
	<i>FunctionActionList:</i> <i>FunctionAction FunctionActionListopt</i>
	<i>FunctionAction: one of</i> <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i> <i>ForFunctionAction IfElseFunctionAction</i> <i>ModifyAction RetractAction</i> <i>ReturnStatement ThrowAction</i> <i>TryFunctionAction UpdateAction</i> <i>WhileFunctionAction</i>

Table 2. Syntax of a rule

Rule header	
	<i>RuleDefinitionList :</i> <i>RuleDefinition RuleDefinitionList</i>
•	<i>RuleDefinition :</i> <i>rule Identifier { RuleParametersopt</i> <i>RuleConditions RuleActions } ;</i>
	<i>RuleParameters:</i>

	<i>RulePriority</i> opt <i>RuleProperty</i> opt
•	<i>RulePriority:</i> priority = <i>PriorityExpression</i> ;
•	<i>RuleProperty:</i> <i>ActivationProperty</i> opt <i>PropertyList</i>
	<i>ActivationProperty:</i> property activation = true;
Rule conditions	
•	<i>RuleConditions :</i> when { <i>FirstRuleCondition</i> <i>RuleConditionList</i> opt }
	<i>RuleConditionList:</i> <i>RuleCondition</i> <i>RuleConditionList</i> <i>RuleCondition</i>
	<i>FirstRuleCondition:</i> <i>RuleCondition</i> except <i>WaitStatementList</i> , <i>EvaluateCondition</i>
	<i>RuleCondition:</i> one of <i>SimpleCondition</i> <i>NotCondition</i> <i>ExistsCondition</i> <i>EvaluateCondition</i> <i>FromExpression</i> <i>InExpression</i> <i>CollectCondition</i> <i>WaitStatementList</i>
•	<i>SimpleCondition :</i> <i>Variable</i> opt:opt eventopt <i>ClassCondition</i>
	<i>Variable:</i> ? opt <i>Identifier</i>
	<i>ClassCondition:</i> <i>ClassName</i> (<i>TestAssignmentList</i>);
•	<i>NotCondition:</i> not <i>ClassCondition</i> ;
•	<i>ExistsCondition:</i> exists <i>ClassCondition</i> ;
•	<i>EvaluateCondition:</i> evaluate (<i>TestAssignmentList</i>);
•	<i>FromExpression:</i>

	<i>Variable</i> <i>opt SimpleCondition</i> from <i>PrimaryExpression</i> ;
•	<i>InExpression</i> : <i>Variable</i> <i>opt SimpleCondition</i> in <i>PrimaryExpression</i> ;
•	<i>CollectCondition</i> : <i>Variable</i> <i>opt collect (Expression</i> <i>opt)</i> <i>SimpleCondition</i> where <i>TestAssignmentSet</i> ;
	<i>WaitStatementList</i> : <i>WaitLogicalStatement</i> <i>WaitStatement</i>
•	<i>WaitLogicalStatement</i> : wait logical until <i>opt Expression WaitCondition</i> <i>opt</i> <i>Variable</i> wait logical until <i>opt Expression WaitCondition</i> <i>opt</i> <i>TimeoutAction</i>
•	<i>WaitStatement</i> : wait <i>Expression</i> <i>opt WaitConditions</i> wait until <i>Expression WaitConditions</i> <i>Variable</i> wait <i>Expression</i> <i>opt WaitConditions</i> <i>TimeoutAction</i> <i>Variable</i> wait until <i>Expression WaitConditions</i> <i>TimeoutAction</i>
	<i>WaitConditions</i> : { <i>WaitConditionList</i> }
	<i>WaitConditionList</i> : <i>WaitCondition</i> <i>WaitConditionList</i>
	<i>WaitCondition</i> : one of <i>SimpleCondition NotCondition ExistsCondition</i> <i>CollectCondition</i>
Tests and variable declarations	
	<i>TestAssignmentList</i> : <i>TestAssignment</i> <i>TestAssignmentList</i> ; <i>TestAssignment</i>
	<i>TestAssignment</i> : one of <i>AndExpresssionList OrExpressionList</i>

	<i>VariableAssignment TestAssignmentSet</i> <i>BooleanExpression NotAssignment</i> <i>TemporalConstraint</i>
	<i>AndExpressionList:</i> <i>TestAssignment && TestAssignment</i>
	<i>OrExpressionList:</i> <i>TestAssignment TestAssignment</i>
	<i>VariableAssignment :</i> <i>Variable Expression BooleanAssignmentListopt</i>
	<i>BooleanAssignmentList:</i> <i>BooleanAssignment</i> <i>BooleanAssignmentList BooleanAssignment</i>
	<i>BooleanAssignment:</i> <i>& BooleanArgument</i>
	<i>TestAssignmentSet :</i> <i>(TestAssignment)</i>
	<i>NotAssignment :</i> <i>! TestAssignment</i>
	<i>TemporalConstraint:</i> <i>Expressionopt.opt TemporalOperator</i> <i>Arguments</i>
•	<i>TemporalOperator:</i> one of after before occursin
	<i>BooleanExpression :</i> <i>Expression</i> <i>BooleanArgumentList</i>
	<i>BooleanArgumentList :</i> <i>BooleanArgument</i> <i>BooleanArgumentList & BooleanArgument</i>
•	<i>BooleanArgument:</i> <i>BooleanArgument</i> instanceof <i>Identifier</i> isknown isunknown InTest

	<i>BooleanTerm</i> : one of <i>PredefinedPredicate Expression</i> Identifier Expression (use of custom predicate)
	<i>PredefinedPredicate</i> : one of == != < > >= <=
	InTest: one of in <i>Expression</i> in { InList } !in Expression !in { InList }
	<i>InList</i> : one of <i>Expression</i> <i>InList, Expression</i>
Expressions	
	<i>Expression</i> : <i>AdditiveExpression</i>
	<i>AdditiveExpression</i> : <i>MultiplicativeExpression</i> <i>MultiplicativeExpressionList</i>
	<i>MultiplicativeExpressionList</i> : + <i>MultiplicativeExpression</i> - <i>MultiplicativeExpression</i> + <i>MultiplicativeExpressionList</i> - <i>MultiplicativeExpressionList</i>
	<i>MultiplicativeExpression</i> : <i>UnaryExpression</i> <i>UnaryExpressionList</i>
	<i>UnaryExpressionList</i> : * <i>UnaryExpression</i> / <i>UnaryExpression</i> % <i>UnaryExpression</i> * <i>UnaryExpressionList</i> / <i>UnaryExpressionList</i> % <i>UnaryExpressionList</i>
	<i>UnaryExpression</i> : + <i>UnaryExpression</i>

	<div>- <i>UnaryExpression</i></div> <div><i>++ UnaryExpression</i></div> <div><i>-- UnaryExpression</i></div> <div><i>UnaryExpression ++</i></div> <div><i>UnaryExpression - -</i></div> <div><i>CastExpression</i></div> <div><i>TimeOfExpression</i></div> <div><i>PrimaryExpression</i></div> <div><i>AsExpression</i></div>
	<div><i>CastExpression:</i></div> <div><i>(ClassName) UnaryExpression</i></div>
<div>•</div>	<div><i>AsExpression:</i></div> <div><i>UnaryExpression as (ClassName)</i></div>
	<div><i>TimeOfExpression:</i></div> <div><i>timeof (Expression);</i></div>
Primary Expressions	
<div>•</div>	<div><i>PrimaryExpression:</i></div> <div><i>PrimaryPrefix</i></div> <div><i>PrimaryPrefix PrimarySuffixList</i></div>
	<div><i>PrimaryPrefix:</i> one of</div> <div><i>Literal Name AllocationExpression</i></div> <div><i>(Expression)</i></div>
	<div><i>Literal:</i> one of</div> <div><i>INTEGER_LITERAL FLOATING_POINT_LITERAL</i></div> <div><i>CHARACTER_LITERAL STRING_LITERAL true</i></div> <div><i>false null</i></div>
	<div><i>Name:</i></div> <div><i>Variable IdentifierListopt</i></div>
	<div><i>IdentifierList:</i></div> <div><i>DotIdentifier</i></div> <div><i>DotIdentifier IdentifierList</i></div>
	<div><i>DotIdentifier:</i></div> <div><i>. Identifier</i></div>
	<div><i>AllocationExpression:</i></div> <div><i>new ClassName Arguments</i></div> <div><i>new Type DimensionList</i></div> <div><i>new Type OptDimensionList</i></div>

	<i>BracedInitializationList</i> <i>BracedInitializationList</i>
	<i>Arguments:</i> (<i>ExpressionList</i>)
	<i>ExpressionList:</i> <i>Expression</i> <i>ExpressionList, Expression</i>
	<i>DimensionList:</i> <i>Dimension</i> <i>DimensionList Dimension</i>
	<i>Dimension:</i> [<i>Expression</i>]
	<i>OptDimensionList:</i> <i>OptDimension</i> <i>OptDimensionList OptDimension</i>
	<i>OptDimension:</i> [<i>Expression</i> opt]
	<i>BracedInitializationList:</i> { <i>InitializationList</i> }
	<i>InitializationList:</i> <i>Initialization</i> <i>InitializationList, Initialization</i>
	<i>Initialization :</i> <i>Expression</i> <i>BracedInitializationList</i> { <i>ExpressionList</i> }
	<i>PrimarySuffixList:</i> <i>PrimarySuffix</i> <i>PrimarySuffixList PrimarySuffix</i>
	<i>PrimarySuffix: one of</i> <i>. Variable</i> (access to variable or class field) <i>Arguments</i> (access to IRL function or class method) <i>DimensionList</i> (access to array element)
Rule actions	

•	<i>RuleActions :</i> <i>then {RuleActionListopt}</i>
	<i>RuleActionList:</i> <i>RuleAction</i> <i>RuleActionList RuleAction</i>
	<i>RuleAction:</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i> <i>ForAction IfElseAction ModifyAction</i> <i>RetractAction ThrowAction</i> <i>TimeoutAction TryAction UpdateAction</i> <i>WhileAction</i>
•	<i>InsertAction :</i> <i>insert logicalopt eventopt ClassName</i> <i>Argumentsopt</i> <i>ExecutableActionStatementsopt</i>
•	<i>VarDeclAction:</i> <i>Type Variable = Expression;</i>
•	<i>ForAction :</i> <i>for(ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i> <i>LoopRuleAction</i> <i>for(ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i> <i>{LoopRuleActionList}</i>
•	<i>ForFunctionAction :</i> <i>for (ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i> <i>LoopFunctionAction</i> <i>for (ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i> <i>{LoopFunctionActionList}</i>
	<i>ForInit:</i> <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i>
	<i>ForUpdate:</i> <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i>
•	<i>IfElseAction :</i> <i>if TestAssignmentSet RuleAction</i> <i>ElseStatement opt</i>

	<i>if TestAssignmentSet {RuleActionList} ElseStatementopt</i>
•	<i>ElseStatement:</i> <i>else RuleAction</i> <i>else {RuleActionList}</i>
•	<i>IfElseFunctionAction :</i> <i>if TestAssignmentSet FunctionAction</i> <i>ElseFunctionActionopt</i> <i>if TestAssignmentSet {FunctionActionList}</i> <i>ElseFunctionActionopt</i>
•	<i>ElseFunctionAction:</i> <i>else FunctionAction</i> <i>else {FunctionActionList}</i>
•	<i>ModifyAction :</i> <i>modify refreshopt PrimaryExpression</i> <i>ExecutableActionStatements;opt</i>
•	<i>RetractAction :</i> <i>retract PrimaryExpression;</i>
•	<i>ThrowAction:</i> <i>throw Expression;</i>
•	<i>TimeoutAction :</i> <i>timeout Variable {NonTimeoutRuleActionList}</i>
	<i>NonTimeoutRuleActionList:</i> <i>NonTimeoutRuleAction</i> <i>NonTimeoutRuleActionList</i> <i>NonTimeoutRuleAction</i>
	<i>NonTimeoutRuleAction:</i> <i>RuleAction</i> but not <i>TimeoutAction</i>
	<i>RuleAction :</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i> <i>ForAction IfElseAction ModifyAction</i> <i>RetractAction</i> <i>UpdateAction WhileAction</i> break; continue;
	<i>FunctionAction:</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i>

	<i>ForFunctionAction IfElseFunctionAction ModifyAction</i> <i>RetractAction UpdateAction WhileFunctionAction</i> <i>break; continue; ReturnStatement</i>
•	<i>ReturnStatement:</i> <i>return (expression)</i>
	<i>RuleActionList:</i> <i>RuleAction RuleActionListopt</i>
	<i>FunctionActionList:</i> <i>FunctionAction FunctionActionListopt</i>
•	<i>TryAction:</i> <i>try {RuleActionListopt}</i> <i>Catches</i> <i>or</i> <i>try {RuleActionListopt}</i> <i>Catchesopt</i> <i>Finally</i>
	<i>Catches:</i> <i>CatchClause</i> <i>Catches CatchClause</i>
•	<i>CatchClause:</i> <i>catch (ExceptionType Identifier)</i> <i>{RuleActionListopt}</i>
•	<i>Finally:</i> <i>finally {RuleActionListopt}</i>
•	<i>TryFunctionAction:</i> <i>try {FunctionActionListopt}</i> <i>FunctionCatches</i> <i>or</i> <i>try {FunctionActionListopt}</i> <i>FunctionCatchesopt</i> <i>Finally</i>
	<i>FunctionCatches:</i> <i>FunctionCatchClause</i> <i>FunctionCatches FunctionCatchClause</i>
•	<i>FunctionCatchClause:</i>

	catch (<i>ExceptionType Identifier</i>) { <i>FunctionActionList</i> opt}
•	<i>FunctionFinally:</i> finally { <i>FunctionActionList</i> opt}
•	<i>UpdateAction :</i> update refreshopt <i>PrimaryExpression</i> ;
•	<i>WhileAction :</i> while <i>TestAssignmentSet LoopRuleAction</i> while <i>TestAssignmentSet</i> { <i>LoopRuleActionList</i> }
•	<i>WhileFunctionAction :</i> while <i>TestAssignmentSet LoopFunctionAction</i> while <i>TestAssignmentSet</i> { <i>LoopFunctionActionList</i> }
Executable statements	
	<i>ExecutableStatementList:</i> <i>ExecutableStatement</i> <i>ExecutableStatementList</i> <i>ExecutableStatement</i>
	<i>ExecutableActionStatements:</i> { <i>ExecutableStatementList</i> }
	<i>ExecutableStatement:</i> <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i> opt;
	<i>AssignmentOperatorExpression:</i> <i>AssignmentOperator Expression</i>
	<i>AssignmentOperator : one of</i> = *= /= %= += -=

Table 3. Syntax of a ruleflow

Ruleflow tasks	
	<i>TaskDefinitionList :</i> <i>TaskDefinition TaskDefinitionList</i>
	<i>TaskDefinition :</i> <i>RuleTaskDefinition</i> <i>FunctionTaskDefinition</i> <i>FlowTaskDefinition</i>
•	<i>RuleTaskDefinition :</i> ruletask identifier {

	<i>CommonTaskParameters</i> <i>RuleTaskBody</i> <i>RuleTaskScope</i> <i>RuleTaskParametersopt</i> };opt
•	<i>FunctionTaskDefinition :</i> functiontask <i>identifier</i> { <i>CommonTaskParameters</i> <i>FunctionTaskBody</i> };opt
•	<i>FlowTaskDefinition :</i> flowtask <i>identifier</i> { <i>CommonTaskParameters</i> <i>FlowTaskBody</i> };opt
•	<i>CompletionFlagParameter:</i> completionflag = <i>Expression</i>
•	<i>InitialActions:</i> initialaction { <i>FunctionActionList</i> }
•	<i>FinalActions:</i> finalaction { <i>FunctionActionList</i> }
	<i>RuleTaskBody:</i> <i>ExtendedBodyDefinition</i> <i>ComprehensiveBodyDefinition</i> <i>DynamicComprehensiveBodyDefinition</i>
•	<i>RuleTaskScope:</i> scope { <i>ScopeDefinition</i> }
	<i>ScopeDefinition:</i> <i>RuleName</i> <i>PackageName.*</i> <i>ScopeDefinitionopt</i>
•	<i>ExtendedBodyDefinition:</i> body { <i>RulesList</i> };opt
	<i>RulesList:</i>

	<i>identifier</i> <i>identifier, RulesList</i>
•	<i>ComprehensiveBodyDefinition:</i> body = select (<i>Variable</i>) { <i>FunctionActionList</i> };opt body = select () { <i>FunctionActionList</i> };opt
•	<i>DynamicComprehensiveBodyDefinition:</i> body = dynamicselect(<i>Variable</i>) { <i>FunctionActionList</i> } EndDynamicComprehensiveBodyDefinition or body = dynamicselect () { <i>FunctionActionList</i> } EndDynamicComprehensiveBodyDefinition
	EndDynamicComprehensiveBodyDefinition: ;opt or <i>DomainValueDefinition</i>
•	<i>DomainValueDefinition:</i> in Expression;
	<i>RuleTaskParameters:</i> <i>AgendaFilterParameter</i> <i>AlgorithmParameters</i> <i>FiringParameter</i> <i>FiringLimitParameter</i> <i>OrderingParameter</i>
•	AgendaFilterParameter: agendafilter = <i>Expression</i> ; agendafilter = filter (<i>Variable</i>) { <i>FunctionActionList</i> };opt
•	<i>AlgorithmParameters:</i> algorithm = <i>AlgorithmValues</i> ; iterator = <i>Expression</i> ; <i>MatchedClasses</i>
	<i>AlgorithmValues:</i> default sequential
•	<i>MatchedClasses:</i> matchedclasses = <i>Expression</i> ; matchedclasses = { <i>MatchedClassesList</i> }; opt
	<i>MatchedClassesList:</i> <i>ReturnType</i>

	<i>ReturnType, MatchedClassesList</i>
•	<i>FiringParameter:</i> firing = <i>FiringValues</i> ;
	<i>FiringValues:</i> allrules rule
•	<i>FiringLimitParameter:</i> firinglimit = <i>Expression</i> ;
•	<i>OrderingParameter:</i> ordering = <i>OrderingValues</i> ;
	<i>OrderingValues:</i> dynamic sorted literal
•	<i>FunctionTaskBody:</i> body { <i>FunctionActionList</i> }; opt
•	<i>FlowTaskBody:</i> body { <i>FlowActionList</i> }; opt
	<i>FlowActionList:</i> <i>FlowAction FlowActionList</i> opt
	<i>FlowAction:</i> one of <i>FlowTaskInvocationAction FlowForkAction</i> <i>FlowGotoAction</i> <i>FlowIfElseAction FlowSwitchAction</i> <i>FlowWhileAction</i>
	<i>FlowTaskInvocationAction :</i> <i>Label</i> opt <i>identifier</i> ;
	<i>Label:</i> <i>identifier:</i>
•	<i>FlowForkAction :</i> <i>Label</i> opt fork <i>FlowForkActionList</i> ;
•	<i>FlowForkActionList:</i> { <i>FlowActionList</i> }; opt { <i>FlowActionList</i> } && <i>FlowForkActionList</i>
•	<i>FlowGotoAction :</i> goto <i>identifier</i> ;
•	

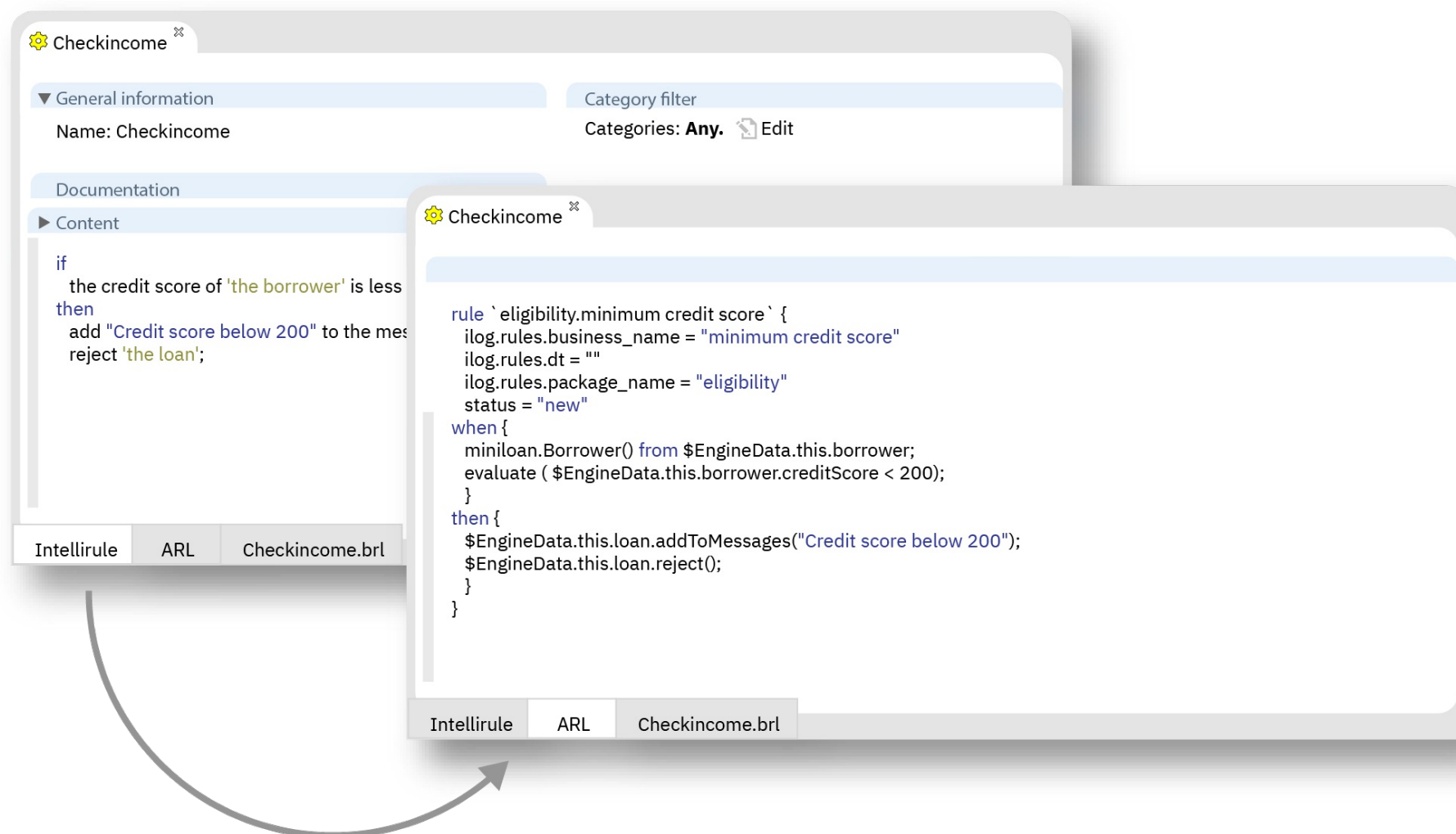
	<i>FlowIfElseAction</i> : <i>Label</i> opt if <i>TestAssignmentSet</i> <i>FlowAction</i> <i>FlowElseAction</i> opt ; <i>Label</i> opt if <i>TestAssignmentSet</i> { <i>FlowActionList</i> } <i>FlowElseAction</i> opt;
•	<i>FlowElseAction</i> : else <i>FlowAction</i> else { <i>FlowActionList</i> }
•	<i>FlowSwitchAction</i> : <i>Label</i> opt switch (<i>Expression</i>) { <i>Cases</i> opt <i>Default</i> opt }
•	<i>Cases</i> : <i>CaseClause</i> <i>Cases</i> <i>CaseClause</i>
•	<i>CaseClause</i> : case <i>INTEGER_LITERAL</i> : <i>FlowAction</i> case <i>INTEGER_LITERAL</i> : { <i>FlowActionList</i> }
•	<i>Default</i> : default: <i>FlowAction</i> default: { <i>FlowActionList</i> }
•	<i>FlowWhileAction</i> : <i>Label</i> opt while <i>TestAssignmentSet</i> <i>LoopFlowAction</i> while <i>TestAssignmentSet</i> { <i>LoopFlowActionList</i> }
•	<i>LoopFlowAction</i> : <i>FlowAction</i> break; continue;
	<i>LoopFlowActionList</i> : <i>LoopFlowAction</i> <i>LoopFlowActionList</i> opt

Parent topic: [IRL grammar](#)

Advanced Rule Language (ARL)

For each Business Action Language (BAL) expression that you create for BOM-to-XOM mapping in the decision engine, you can review its equivalent Advanced Rule Language (ARL) expression. ARL is a read-only rule language that is similar in syntax to Java 7.

If you are familiar with Java 7, you can review how your BAL expressions are interpreted by the decision engine by reviewing the content in the ARL tab in Rule Designer.



BAL and ARL are equivalent in meaning, as you can see in the following two examples. In the first rule condition that is written in BAL, the credit score of a borrower is examined to check whether the borrower has a credit score less than 200:

```
if
  the credit score of 'the borrower' is less than 200
then
  add "Credit score below 200" to the messages of 'the loan' ;
  reject 'the loan';
```

The decision engine interprets the previous expression and produces its equivalent in ARL, readable from the ARL tab in Rule Designer. The following ARL expression describes the exact same rule condition as its counterpart in BAL:

```
rule `eligibility.minimum credit score` {
  ilog.rules.business_name = "minimum credit score"
  ilog.rules.dt = ""
  ilog.rules.package_name = "eligibility"
  status = "new"
when {
  miniloan.Borrower() from $EngineData.this.borrower;
  evaluate ( $EngineData.this.borrower.creditScore < 200);
}
then {
  $EngineData.this.loan.addToMessages("Credit score below 200");
  $EngineData.this.loan.reject();
}
}
```

[Differences between ARL and Java 7](#)

ARL is a read-only rule language that is similar in syntax to Java 7. To understand the difference between ARL and Java 7, review this list of ARL keywords and operators.

[ARL rules examples](#)

The ARL we are going to describe in this section is only used to describe rules to help users to

disambiguate BAL.

Parent topic: [Rule languages](#)

Differences between ARL and Java 7

ARL is a read-only rule language that is similar in syntax to Java 7. To understand the difference between ARL and Java 7, review this list of ARL keywords and operators.

Aggregates

Unlike Java, ARL has aggregate operators that are used in rule descriptions.

Anonymous classes

In Java, you can create anonymous class directly in the body of a method or constructor. In ARL, you cannot create anonymous classes.

Back-quotes

Unlike Java, ARL uses back-quotes (``) to name variables that are not parsed as Java identifies. See the following example:

```
String `the lazy dog` = "pluto";
```

Cast Operators

Much like in C#, you can use the cast operator "as" to do certain types of conversions between compatible reference types or nullable types. See the following example:

```
AClass anObject = o as AClass;
```

Goto

ARL does not support goto.

Inner classes

You can refer to the fields of an enclosing class by using the syntax 'A.this', where A represents the enclosing class. This mechanism is extended to refer to certain classes of the engine. For example, if 'ruleName' is ambiguous, you can print a rule name by writing the following code:

```
if (RuleInstance.this != null) note("ARL mapping of method M being called by rule "+ RuleInstance.this.ruleName);
```

ARL Keywords

The following list is the complete set of keywords in the ARL rule language:

abstract	continue	final	in	not	queryTemplate	select	throw	refresh
aggregate	default	finally	insert	null	query	signature	throws	retract
as	do	for	instanceof	operator	once	static	transient	void
assert	else	from	interface	out	over	stipulation	true	volatile
break	enum	goto	match	package	repeatable	strictfp	try	when
case	evaluate	groupby	method	priority	restricts	super	typedef	where
catch	exists	if	modal	private	return	switch	update	while
class	explicit	implements	modify	property	rule	synchronized	updateEngineData	xpath
conditionTemplate	extends	implicit	native	protected	ruleset	then	updateGenerator	xs
const	false	import	new	public	ruleTemplate	this	updateGenerator	

Loops

foreach loops have a different syntax.

Switch

It is not necessary to use the keyword "break" at the end of each "case" section.

Parent topic: [Advanced Rule Language \(ARL\)](#)

ARL rules examples

The ARL we are going to describe in this section is only used to describe rules to help users to disambiguate BAL.

We describe the most common statements corresponding to the BAL constructs in [BAL constructs](#).

From parameters

BAL and ARL are equivalent in meaning, as you can see in the following comparison. The same rule condition is presented side by side:

BAL	ARL
<pre>definitions set b to 'the borrower'; if the credit score of b is less than 200 then add "Credit score below 200" to the messages of 'the loan' ; reject 'the loan';</pre>	<pre>rule `eligibility.minimum credit score` { ilog.rules.business_name = "minimum credit score" ilog.rules.dt = "" ilog.rules.package_name = "eligibility" status = "new" when { miniloan.Borrower() from \$EngineData.this.borrower; b : miniloan.Borrower() from \$EngineData.this.borrower; evaluate (b.creditScore < 200); } then { \$EngineData.this.loan.addToMess ages("Credit score below 200"); \$EngineData.this.loan.reject(); } }</pre>

In the previous BAL example, a new variable (b) is defined by writing the following expression: ‘set b to ‘the borrower’;’. In the Miniloan ServiceParameters, one can see that ‘the borrower’ is the verbalization of the variable ‘borrower’, which is an instance of the BOM class Borrower. Notice also that ‘the loan’ is a verbalization of an instance of Loan.

If you are familiar with Java, you can see from the previous example that ARL has some particularities:

- The identifiers are written in free text between back-quotes: as `eligibility.minimum credit score`.
- The keywords rule and when are unique to ARL. Other examples are: ruleset, signature, evaluate, from, ruleset, match, many, in, flowtask, ruletask, and aggregate. For a complete list of keywords, see [ARL keywords](#).
- The syntactic construction \$EngineData.this allows BOM objects and ruleset parameters to be referenced. See [Execution class mapping](#).
- The declaration b : miniloan.Borrower() from \$EngineData.this.borrower; is called a binding. It means b is a new variable whose value is the ruleset parameter ‘borrower’ referenced by \$EngineData.this.borrower.

The ‘if’ expression in BAL corresponds to the ‘when’ expression in the ARL translation. Both languages have a then expression:

BAL	ARL
<pre>the credit score of b is less than 200</pre>	<pre>miniloan.Borrower() from \$EngineData.this.borrower; b : miniloan.Borrower() from \$EngineData.this.borrower;</pre>

	<pre>evaluate (b.creditScore < 200);</pre>
--	--

ARL is designed to express pattern matching, which is the operating principle of the underlying rule engine. The expression `miniloan.Borrower()` from `$EngineData.this.borrower`; means that the ruleset parameter 'borrower' must be an instance of the class `Borrower`, the binding: `'b : miniloan.Borrower()'` means that if an instance of `Borrower` is matched, it is referred to as 'b', finally the expression: `'evaluate (b.creditScore < 200);'` returns the value of the expression: `'b.creditScore < 200'`. Considering that the semicolon behaves like a logical and, end-to-end these expressions mean the same as 'the credit score of b is less than 200' in a java-like dialect.

Let's go back to the notation `'$EngineData.this'`. It's necessary to use such a notation because there are many structures used below the keyword 'rule', in particular:

- `EngineData` to give access to BOM and ruleset parameters object
- `RunningEngine` to give access to methods of the underlying engine

For this reason, the keyword 'this' becomes ambiguous. A simple way to lift this ambiguity is to use the prefix 'this'. See [Execution class mapping](#).

The action of the conditions in both languages (introduced by `then`) are self-explanatory: If the condition, introduced by `when` or `if`, is true, then the corresponding actions are taken, in the ARL version, a series of calls to methods of the 'Loan' instance 'loan' which is, as told above, a ruleset parameter.

Match object in working memory

What happens when you modify the definitions section? Notice that the `from` keyword, previously used to reference a parameter, vanishes. Additionally, the binding `b : miniloan.Borrower()`; acts on each instance of `Borrower` found in the working memory. (The working memory is the set of objects to which the rule engine is applied.)

BAL	ARL
<pre>definitions set b to 'the borrower'; if the credit score of b is less than 200 then add "Credit score below 200" to the messages of 'the loan' ; reject 'the loan';</pre>	<pre>rule `eligibility.minimum credit score` { ilog.rules.business_name = "minimum credit score" ilog.rules.dt = "" ilog.rules.package_name = "eligibility" status = "new" when { b : miniloan.Borrower(); evaluate (b.creditScore < 200); } then { \$EngineData.this.loan.addToMess ages("Credit score below 200"); \$EngineData.this.loan.reject(); } }</pre>

Collections

Imagine a scenario in which a single loan has several co-borrowers. The loan could be refused if one of the co-borrowers has a low credit score.

BAL	ARL
<pre>definitions set b to a borrower in 'several co-borrowers' ; if the credit score of b is less than 200</pre>	<pre>rule `eligibility.minimum credit score` { ilog.rules.business_name = "minimum credit score" ilog.rules.dt = "" ilog.rules.package_name =</pre>

<pre>then add "Credit score below 200" to the messages of 'the loan' ; reject 'the loan';</pre>	<pre>"eligibility" status = "new" when { b : miniloan.Borrower() in \$EngineData.this.borrowers; evaluate (b.creditScore < 200); } then { \$EngineData.this.loan.addToMess ages("Credit score below 200"); \$EngineData.this.loan.reject(); } }</pre>
---	---

The variable ‘several co-borrowers’ is the verbalization of the variable ‘borrowers’ defined as a collection of borrowers. Noticed that as borrowers is a collection the binding, ‘b : miniloan.Borrower() in \$EngineData.this.borrowers;’ is written with the keyword in instead of from because in this case it is an inclusion.

Binding and loops

The following rule condition defines the variable risky borrowers, which corresponds to borrowers whose yearly income is below 1000. If this list is not empty it prints all its elements. In the following ARL example, the definition of this variable is made by using a binding. The variable name followed by a colon (:), and an expression:

```
`risky borrowers`:aggregate {
  collect_class_1 : miniloan.Borrower($EngineData.this.borrower.yearlyIncome
< 1000);
}
```

The expression used to define `risky borrowers` is an *aggregate* (see below). In the action part of the expression, there is a loop iterating over the elements of the just created variable: it has the same syntax as in Java.

Similar to the example above, in the following example, the variable b is defined in ARL by using the binding: b : miniloan.Borrower() in \$EngineData.this.borrowers;;

BAL	ARL
<pre>definitions set 'risky borrowers' to all borrowers where the yearly income of 'the borrower' is less than 1000; if the number of objects in 'risky borrowers' is more than 0 then for each borrower called b, in 'risky borrowers': - print the name of b;</pre>	<pre>rule `doc.called` { ilog.rules.business_name = "called" ilog.rules.dt = "" ilog.rules.package_name = "doc" status = "new" when { miniloan.Borrower() from \$EngineData.this.borrower; `risky borrowers`:aggregate { collect_class_1 : miniloan.Borrower(\$EngineData.t his.borrower.yearlyIncome < 1000); } do { java.util.ArrayList<miniloan.Bo rrower>{collect_class_1}; } evaluate (</pre>

	<pre>\$EngineData.this.borrower.name.equals("Jack")); } then { { for (miniloan.Borrower b : `risky borrowers`){ { ilog.rules.brl.System.printMess age((b.name); } } }</pre>
--	--

Evaluate

The constraints expressed in BAL on a variable are often translated in ARL by using the operator evaluate::

```
definitions
  set b to a borrower in 'several co-borrowers' ;
if
  the credit score of b is less than 200
then
  add "Credit score below 200" to the messages of 'the loan' ;
  reject 'the loan';
```

The constraint on the credit score b is expressed in ARL as evaluate (b.creditScore < 200);.

The same constraint could be done directly in the binding: b : miniloan.Borrower(b.creditScore < 200) in \$EngineData.this.borrowers; This way of expressing constraints is often used in aggregates.

Aggregates

As in database aggregations, aggregates allow to group multiple values together to form a single value. There are various types of aggregates depending on the way values are grouped. The following aggregates are often used in BAL to ARL translations: collections, count, exists, match any.

Collections

Values can be accumulated in collections. For example, in the following rule we want to create a collection regrouping all borrowers whose yearly income is below a given threshold of 1000. To create this collection in ARL, we create a collection aggregate: collect_class_1 : miniloan.Borrower(\$EngineData.this.borrower.yearlyIncome < 1000); and add it to the ArrayList:

```
do {
  java.util.ArrayList<miniloan.Borrower>{collect_class_1};
}
```

The result returned by the previous do statement is bound to the variable of the binding.

BAL	ARL
<pre>definitions set 'risky borrowers' to all borrowers where the yearly income of 'the borrower' is less than 1000; if there is at most one borrower in 'risky borrowers' then print "there are risky borrowers";</pre>	<pre>rule `doc.Collect` { ilog.rules.business_name = "Collect" ilog.rules.dt = "" ilog.rules.package_name = "doc" status = "new" when { miniloan.Borrower() from \$EngineData.this.borrower; risky borrowers:aggregate { collect_class_1 : miniloan.Borrower(\$EngineData.t</pre>

	<pre>his.borrower.yearlyIncome < 1000); } do { java.util.ArrayList<miniloan.Bor rower>{collect_class_1}; } evaluate (`risky borrowers`.size() <= 1); } then { ilog.rules.brl.System.printMess age("there are risky borrowers"); } }</pre>
--	--

Count

Aggregation functions such as the count function can be used, for example, in the following rule condition expression to determine whether there are 8 borrowers with a yearly income superior to 1200. The aggregate count function counts the number of borrowers whose yearly income is more than 12000. The where clause attached to the aggregate is similar to the evaluate operator already defined. The boolean results of the where clause is bound to the variable of the binding:

BAL	ARL
<pre>if there are 8 borrowers where the yearly income of each borrower is more than 12000, then print "there are at least 8 of them";</pre>	<pre>rule `doc.There are more` { ilog.rules.business_name = "There are more" ilog.rules.dt = "" ilog.rules.package_name = "doc" status = "new" when { aggregate_1:aggregate { collect_1 : miniloan.Borrower(this.yearlyIn come > 12000); } do { count{collect_1}; } where (aggregate_1 == 8); } then { ilog.rules.brl.System.printMess age("there are at least 8 of them"); } }</pre>

Exists

The operator exists tests that there is *at least one* item in a collection of items. So, in the following rule testing there is at least one borrower is translated by an 'exists' clause:

BAL	ARL
<pre>if there are 8 borrowers</pre>	<pre>rule `doc.There is at least one` {</pre>


```

    where the yearly income of
each borrower is more than
12000,
then
    print "there are at least 8
of them";

```

```

        ilog.rules.business_name =
        "There is at least one"
        ilog.rules.dt = ""
        ilog.rules.package_name =
        "doc"
        status = "new"
        when {
            exists {

miniloan.Borrower(this.borrower
.yearlyIncome > 12000 &&
this.borrower.yearlyIncome <
10000000);
            }
            }
        then {

ilog.rules.brl.System.printMess
age("it exists");
        }
    }
}

```

Match many

The match many instruction is used to translate decision tables. In the decision table below, the match many instruction follows its semantic. The first case in the first match many corresponds to the first condition debt to income of the first two lines. If the condition is satisfied, an embedded match many evaluates the corresponding conditions on the credit score lines. If the first condition on the credit score line is satisfied (a credit score between 0 and 200), then a then clause describes the action to be taken: the loan is refused.

The other match many evaluates the other conditions of the table and takes the corresponding actions. The conditions corresponding to disabled actions in the decision table do not have corresponding cases in the match many instruction:

ARL translation of the decision table:

```
rule `eligibility.repayment and score` {
  ilog.rules.business_name = "repayment and score"
  ilog.rules.dt = "eligibility.repayment and score"
  ilog.rules.package_name = "eligibility"
  status = "new"
  when {
    miniloan.Borrower() from $EngineData.this.borrower;
```



```

        evaluate ( $EngineData.this.borrower.yearlyIncome > 0);
    }
    match many {
        case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [0,30[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
            case ($EngineData.this.borrower != null &&
[0,200[.contains($EngineData.this.borrower.creditScore)) : then 1{
                $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
                $EngineData.this.loan.reject();
            }
        }
        case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [30,45[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
            case ($EngineData.this.borrower != null &&
[0,400[.contains($EngineData.this.borrower.creditScore)) : then 3{
                $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
                $EngineData.this.loan.reject();
            }
        }
        case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [45,50[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
            case ($EngineData.this.borrower != null &&
[0,600[.contains($EngineData.this.borrower.creditScore)) : then 5{
                $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
                $EngineData.this.loan.reject();
            }
        }
        case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && ($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome >= 50)) : if ($EngineData.this.borrower
!= null && [0,800[.contains($EngineData.this.borrower.creditScore)){
            then 7{
                $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
                $EngineData.this.loan.reject();
            }
        }
    }
}

```

Parent topic: [Advanced Rule Language \(ARL\)](#)

Rule Designer Messages

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRA	Messages pertaining to rule authoring
GBRE	Messages pertaining to the rule engine
GBRM	Messages pertaining to the model and the business object model (BOM)
GBRSB	Messages pertaining to the business object model
GBRSD	Messages pertaining to debug
GBRSE	Messages pertaining to rule execution
GBRSM00	Messages pertaining to the model (00 - 99)
GBRSM01	Messages pertaining to the model (100-199)
GBRSM02	Messages pertaining to the model (200-299)
GBRST	Messages pertaining to tools
GBRSU	Messages pertaining to core user interface
GBRT	Messages pertaining to testing and simulation

Rule Designer Messages - Messages pertaining to rule authoring

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRAB0001E	Bad index for token: {0} (line: {2}, column: {3}).
GBRAB0002E	Cannot add a child to token {0} (line: {2}, column: {3}).
GBRAB0003E	Reference to unknown operator "{0}" in grammar (type: {1}) at: {2}.
GBRAB0004E	Reference to unknown terminal "{0}" in grammar (type: {1}) at: {2}.
GBRAB0005E	Invalid numeric value.
GBRAB0006E	Empty numeric value.
GBRAB0007E	Empty value.
GBRAB0008E	The schema has no element.
GBRAB0009E	Element type not found: {0}.
GBRAB0010E	Grammar not set on this syntax tree.
GBRAB0011E	There is no root named "{0}" in grammar.
GBRAB0012E	Current grammar node is not a list: "{0}" (type: {1}) in path: {2}
GBRAB0013E	The grammar node "{0}" has not been found in path: {1}
GBRAB0014E	Bad index for syntax tree node {0} : {1}.
GBRAB0015E	Node not found in the abstract syntax tree: {0}.
GBRAD0001E	Exception raised while loading "{0}". See log file.
GBRAD0002E	Exception raised while checking "{0}". See log file.
GBRAL0000W	The search server cannot be started because it is not configured correctly. ({0})
GBRAL0001W	Could not connect to the search server. ({0})
GBRAL0002W	The search query failed to run. ({0})
GBRAL0003W	The search request cannot be processed. Ensure that the server is configured correctly. ({0})
GBRAL0500E	Bad configuration. Variable {0} is not set. Search service disabled.
GBRAL0501E	Bad configuration. Invalid value for {0}. Supported search service providers are: {1}.
GBRAL0502E	Bad configuration. Variable {0} must be set when search service {1} is enabled.
GBRAL0503E	Bad configuration. Invalid value {0} for variable {1}.
GBRAL0504E	Failed to start search service.
GBRAL0505I	Search service is currently unavailable due to bad configuration.

Rule Designer Messages - Messages pertaining to the rule engine

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBREA0001E	The syntactic node of class "{0}" is not well formed
GBREA0002E	The checking of a syntactic node of class "{0}" is not implemented
GBREA0003E	The type "{0}" is undefined
GBREA0004E	The primitive type "{0}" is undefined
GBREA0005E	The type "{0}" is not a valid annotation type
GBREA0006E	The type "{0}" is ambiguous, the possible meanings are "{1}"
GBREA0007E	The type "{0}" is not a valid component type
GBREA0008E	The type "{0}" is not a valid attribute type
GBREA0009E	The type "{0}" is not a valid indexer type
GBREA0010E	The type "{0}" is not a valid variable type
GBREA0011E	Only a single extension is supported
GBREA0012E	The type "{0}" is not a valid extension type
GBREA0013E	Interface extension is not supported
GBREA0014E	The type "{0}" is not a valid interface type
GBREA0015E	The declaration of type "{0}" is recursive. Check the declaration of type "{1}"
GBREA0016E	The type "{0}" cannot be used as the bound of a type parameter
GBREA0017E	The type "{0}" is not a valid type argument
GBREA0018E	The type "{0}" cannot be used as the bound of a type argument
GBREA0019E	The type "{0}" cannot be used as the bound of a type argument
GBREA0020E	The type "{0}" is not a valid exception type
GBREA0021E	A type variable named "{0}" already exists
GBREA0	A type named "{0}" already exists

022E	
GBREA0023E	The modifier "{0}" is unknown
GBREA0024E	The modifier "{0}" has already been specified
GBREA0025E	The modifier "{0}" cannot be used here
GBREA0026E	The modifier "{0}" is conflicting with the other modifiers
GBREA0027E	"{0}" cannot be assigned
GBREA0030E	An attribute named "{0}" already exists
GBREA0031E	Unable to find an attribute named "{0}" in type "{1}"
GBREA0032E	Attribute "{0}" is write-only
GBREA0033E	Attribute "{0}" is read-only
GBREA0034E	Attribute "{0}" is not static
GBREA0035E	Attribute "{0}" has "{1}" access
GBREA0036E	All the branches of the getter of the attribute named "{0}" should return an instance of "{1}"
GBREA0037E	The attribute "{0}" of type "{1}" cannot be initialized with a value of type "{2}"
GBREA0040E	The name of the constructor should be "{0}"
GBREA0041E	A constructor with a similar signature already exists
GBREA0042E	Unable to find a constructor of signature "{0}" in type "{1}"
GBREA0043E	Constructor "{0}" has "{1}" access
GBREA0044E	Recursive constructor call
GBREA0045E	The constructor named "{0}" should catch or declare exceptions of type "{1}"
GBREA0050E	A method with a similar signature already exists
GBREA0051E	This operator cannot be defined
GBREA0052E	Cannot find a method "{1}" in type "{0}"
GBREA0053E	Method "{0}" has "{1}" access
GBREA0054E	Method "{0}" is not static
GBREA0055E	All the branches of the body of the method named "{0}" should return an instance of "{1}"
GBREA0056E	The method named "{0}" should catch or declare exceptions of type "{1}"
GBREA0060E	An indexer with a similar signature already exists
GBREA0061E	An indexer must declare at least one parameter
GBREA0	Unable to find an indexer of signature "{0}" in type "{1}"

062E	
GBREA0063E	This indexer is not visible from here
GBREA0064E	Indexer "{0}" is not static
GBREA0065E	All the branches of the getter of the indexer named "{0}" should return an instance of "{1}"
GBREA0070E	A variable named "{0}" already exists
GBREA0071E	A variable of type "{0}" cannot be initialized with a value of type "{1}"
GBREA0072E	Unable to find a variable named "{0}"
GBREA0073E	Variable "{0}" has not been initialized
GBREA0074E	Variable "{0}" might not have been initialized
GBREA0080E	This value could not be checked
GBREA0081E	This value is ambiguous
GBREA0082E	A value of type "{0}" cannot be initialized with a value of type "{1}"
GBREA0083E	This literal value is undefined
GBREA0084E	The parsing of literal value "{0}" has failed
GBREA0085E	Cannot use a value of type "{0}" when type "{1}" is expected
GBREA0086E	A constant cannot be assigned a value
GBREA0090E	The unary operator "{0}" is undefined
GBREA0091E	Cannot find the operator "{0}" to apply to "{1}"
GBREA0092E	The binary operator "{0}" is undefined
GBREA0093E	Cannot find an operator "{0}" to apply to "{1}", "{2}"
GBREA0094E	Post operator "{0}" is not supported
GBREA0095E	The ternary operator "{0}" is undefined
GBREA0096E	Unable to find a "{0}" ternary operator that takes ("{1}", "{2}", "{3}")
GBREA0097E	The cast operation from type "{0}" to type "{1}" is undefined
GBREA0098E	The "as" cast operation from type "{0}" to type "{1}" is undefined because it is restricted to reference types
GBREA0099E	Literal "{0}" is unexpected here
GBREA0100E	Division by zero is not allowed "{0}"
GBREA0105E	An array dimension or an array initializer is expected
GBREA0106E	An array initializer may not be specified when an array dimension exists
GBREA0	Unable to find a constructor of signature "{0}" in array type "{1}"

107E	
GBREA0110E	An aggregate is unexpected here
GBREA0111E	At least one generator is required in an "aggregate"
GBREA0112E	At least one argument is required in an "aggregate"
GBREA0113E	Cannot find aggregation function called "{0}"
GBREA0114E	The type "{0}" of the value is not a valid collection type
GBREA0120E	A constant value is expected
GBREA0121E	A "default" already exists for this "switch"
GBREA0122E	A case with the same value already exists for this "switch"
GBREA0123E	A value is expected for this "switch" case
GBREA0124E	The type "{0}" of this "case" is incompatible with the expected type "{1}"
GBREA0125E	A default result is expected for this "switch"
GBREA0126E	A value of type "{0}" cannot be passed to a "switch"
GBREA0127E	A "default" already exists for this "switch"
GBREA0128E	A case of type "{0}" is not compatible with a switch on type "{1}"
GBREA0129E	A case with the same value already exists for this "switch"
GBREA0130E	A body is expected for this "switch" case
GBREA0140E	"return" is unexpected here
GBREA0141E	The type "{0}" of the value is incompatible with the specified return type "{1}"
GBREA0142E	"break" is unexpected here
GBREA0143E	"continue" is unexpected here
GBREA0144E	The type "{0}" is already caught
GBREA0145E	A value of type "{0}" cannot be thrown
GBREA0146E	The exception type "{0}" should be caught or declared as thrown
GBREA0147E	Unreachable statement "{0}"
GBREA0148E	Missing "return" statement
GBREA0149E	This value is not a statement
GBREA0150E	This value is not a test, its type is not "boolean"
GBREA0160E	The argument "{0}" of the annotation of class "{1}" is invalid
GBREA0	The type "{0}" can't be used as a base type in a restriction

161E	
GBREA0162E	A type named "{0}" already exists
GBREA0163E	Wrong number of type arguments for generic type "{0}". Expected "{1}", got "{2}"
GBREA0170E	"this" cannot be used in this scope
GBREA0171E	"super" cannot be used in this scope
GBREB0001E	The "{0}" exception has been thrown:
GBREB0002I	Location stack trace:
GBREB0003I	Caused by: {0}
GBREB0004I	at line {0}, column {1}
GBREB0005I	in file "{0}", at line {1}, column {2}
GBREB0006I	at offset {0}, with length {1}
GBREB0007E	Version read {0} expected version {1}
GBREB0008E	Not a valid decision engine jar file or stream
GBREB0009W	Generated with Java specification version {0}, used in version {1}
GBRED0001I	in the conditions of rule "{0}", between line {1}, column {2} and line {3}, column {4}
GBRED0002I	in the conditions of rule "{0}"
GBRED0003I	in the action "{1}" of rule "{0}", between line {2}, column {3} and line {4}, column {5}
GBRED0004I	in the action "{1}" of rule "{0}"
GBRED0005I	in rule "{0}", between line {1}, column {2} and line {3}, column {4}
GBRED0006I	in the action of the rule "{0}"
GBRED0007E	A ruleset named "{0}" already exists
GBRED0007I	in the conditions of rule "{0}", at offset {1}, and length {2}
GBRED0008E	The namespace "{0}" does not contain any rule
GBRED0008I	in rule "{0}", at offset {1}, and length {2}
GBRED0009E	Unable to find the rule named "{0}"
GBRED0010E	Could not load custom exception handler class "{0}"
GBRED0011E	The rule named "{0}" requires an environment of type "{1}" which is incompatible with an environment of type "{2}"
GBRED0011I	While integrating the custom exception handler "{0}"
GBRED0012E	Declared custom exception handler class "{0}" has to implement "{1}"
GBRED0	The custom exception handler class "{0}" has no public default constructor

013E	
GBRED0014E	A rule named "{0}" already exists
GBRED0015E	The return type "{0}" is not compliant with the expected return type "{1}"
GBRED0016E	A product condition is expected
GBRED0017E	A case with the same value already exists
GBRED0018E	A variable named "{0}" already exists
GBRED0019E	The type "{0}" is not a valid condition type
GBRED0020E	All the branches of this rule content should return an instance of "{0}"
GBRED0021E	The type "{0}" is not a valid engine data type
GBRED0022E	An object of the type "{0}" cannot be used as an argument to "stop"
GBRED0023E	A value of type "{0}" cannot be used as an "in" generator
GBRED0024E	Unexpected element "{1}" in the "{0}" stipulation
GBRED0025E	A ruleset property stipulation must be located in the body of the ruleset
GBRED0026E	The parameter "{0}" of the instruction "{1}" of the stipulation "{2}" is not correct
GBRED0027E	No matching condition template of namespace "{0}" and of name "{1}" declared in the ruleset
GBRED0028E	Duplicate declaration of the "{0}" condition template
GBRED0029E	Recursive declaration in the "{0}" condition template
GBRED0030E	No matching rule template of namespace "{0}" and of name "{1}" declared in the ruleset
GBRED0031E	Duplicate declaration of the "{0}" rule template
GBRED0032E	A rule property stipulation must be located in the body of the ruleset
GBRED0033E	Plug-in uniqueness error: two declared compilation plug-ins of class "{0}" and "{1}" share the same id "{2}"
GBRED0034E	Plug-in dependency cycle found involving the plug-in of class "{0}"
GBRED0035E	You declared too many references to the current enclosing class instance in a finder of the "{0}" stipulation
GBRED0036E	Invalid key in a finder of the "{0}" stipulation
GBRED0037E	No key found in a finder declaration of the "{0}" stipulation
GBRED0038E	Arity impossible to infer from signature in a finder declaration of the "{0}" stipulation
GBRED0039E	Must be an Iterable or an Array
GBREF1001E	Cannot find a main task named "{0}"
GBREF1002E	A task named "{0}" already exists
GBREF2	task not found

001E	
GBREF2002E	The rule {1} in task {0} has a dynamic priority, which is not allowed for SORTED tasks
GBREF3001W	A ruleset containing homogeneous rules is recommended for sequential task "{0}"
GBREF3002W	A ruleset with at least one rule is recommended for rule task "{0}"
GBREF3003W	A ruleset with no object modification in the action part is recommended for Fastpath and Sequential algorithms in rule task "{0}"
GBREF4001E	Engine not running
GBREF4002E	No main task set
GBREF4003E	Stopped by installed controller
GBREF4004E	Cannot use dynamic ordering with FIRINGKIND set to FIRST_ELIGIBLE_RULE
GBREF4005E	does not work on this kind of engine
GBREF4006E	The specified task "{0}" has not been defined
GBREF4007I	in task "{0}"
GBREF4008I	in task "{0}", between line {1}, column {2} and line {3}, column {4}
GBREM0008E	Cannot migrate postfix operation used as a value
GBREM0009E	Cannot migrate context value
GBREM0010E	Cannot migrate invocation of method {0}, no corresponding method found in type {1}
GBREM0011E	Cannot migrate invocation of instance method {0} as the instance could not be migrated
GBREM0012E	Cannot migrate access to the value of attribute {0}
GBREM0013E	Cannot migrate invocation of function {0}
GBREM0014E	Cannot migrate binding of null
GBREM0015E	Cannot migrate operator unknown
GBREM0016E	Cannot migrate usage of binary operator {0}
GBREM0017E	Cannot migrate usage of unary operator {0}
GBREM0018E	Cannot migrate assignment
GBREM0019E	Cannot migrate parameter/package variable {0}. The variable is of type {1} whereas the default value is of type {2}
GBREM0020E	Cannot migrate call to IlrContext.invokeFunction, function {0} cannot be found
GBREM0021E	Cannot migrate call to IlrContext.invokeFunction, function {0} has not been migrated
GBREM0022E	Cannot migrate call to IlrContext.invokeFunction with a non inline creation of an arguments array
GBREM0023E	Cannot migrate call to IlrContext.invokeFunction without an array as a parameter
GBREM0	Cannot migrate call IlrContext.invokeFunction when the function name is not constant

024E	
GBREM0025E	Cannot migrate call <code>llrContext.invokeFunction({0},...)</code> . Internal error the engine value is missing.
GBREM0026E	Cannot migrate access to package name.
GBREM0027E	Cannot migrate deprecated <code>{0}</code> statement
GBREM0028E	Rule instance access cannot be migrated in this scope
GBREM0029E	Events and time are not supported. Cannot migrate <code>{0}</code>
GBREM0030E	"collect *from* <source>" expression is not supported
GBREM0031E	"collect in <source> *where*" expression is not supported
GBREM0032E	An update refresh action has been detected. It is forbidden because the update refresh is not migrated.
GBREM0033E	Could not migrate rule property <code>{0}</code>
GBREM0034E	Could not migrate access to rule property <code>{0}</code>
GBREM0035E	Cannot migrate the hasher <code>{0}</code>
GBREM0036E	Events and time are not supported. Cannot migrate temporal conditions beginning with "event"
GBREM0037E	Only <code>llrDefaultCollector</code> is supported when migrating a collect condition
GBREM0038E	Cannot translate <code>llrContext.getCurrentTask</code> outside of task actions
GBREM0039E	Cannot translate <code>llrContext.currentTask</code> outside of task actions
GBREM0040E	Cannot translate dynamic select using rule array selections for task <code>{0}</code> .
GBREM0041E	Cannot migrate access to package name in dynamic select.
GBREM0042E	Cannot migrate access to rule name in dynamic select.
GBREM0043E	Internal error cannot migrate condition variable of type <code>{0}</code>
GBREM0044E	Internal error cannot migrate negation of <code>{0}</code>
GBREM0045E	Cannot migrate instanceof <code>{0}</code>
GBREM0046E	Cannot migrate match updown
GBREM0047E	stop task does not exist.
GBREM0048E	Internal migration error when trying to migrate <code>{0}</code>
GBREM0048W	Property " <code>{0}</code> " is not used by the decision engine
GBREM0049W	"ilrmain" function is not migrated
GBREM0050E	Internal error migration when trying migrate B2X. Got execution factory parser errors <code>{0}</code>
GBREM0051E	Cannot migrate variable " <code>{0}</code> "
GBREM0	Missing parameter to call super constructor " <code>{0}</code> " from constructor " <code>{1}</code> "

052E	
GBREM0053W	Don't know what to do with parameter "{0}" : there is no attribute with such a name in class "{1}", a throw statement has been generated in constructor
GBREM0054E	Cannot find execution class "{0}" for translating business class "{1}"
GBREM0055E	Error when parsing B2X IRL body : {0}
GBREM0056E	Error in B2X : {0}
GBREM0057W	Warning in B2X : {0}
GBREP1001E	Must be an Iterable, an Array, or an Enumeration
GBREP1002E	"exists" not supported in sequential mode without a generator
GBREP1003E	"not" not supported in sequential mode without a generator
GBREP1004E	"aggregate" not supported in sequential mode without a generator
GBREP1005E	"or" not supported in sequential mode
GBREP2005E	does not work on this kind of engine
GBRETO001E	Failed to transform method "{0}"
GBRETO002E	Failed to transform type "{0}"
GBRETO003E	Cannot find an operator "{0}" to apply to "{1}","{2}"
GBRETO004E	Failed to transform value of attribute "{0}"
GBRETO005E	Failed to transform assignment of attribute "{0}"
GBRETO006E	Failed to transform value of indexer "{0}"
GBRETO007E	Failed to transform assignment of indexer "{0}"
GBRETO008E	Failed to transform invocation of method "{0}"
GBRETO009E	Failed to transform usage of constructor "{0}"
GBRETO010E	Failed to retrieve access to type "{0}" in a body of type "{1}"
GBRETO011E	Cannot convert "{0}" to type "{1}"
GBRETO012E	Failed to create instance of abstract class "{0}"
GBRETO013E	Failed to transform indexer "{0}"
GBRETO030I	While transforming the implementation of method "{0}"
GBRETO031I	While transforming the implementations of the members of class "{0}"
GBRETO032I	While transforming the statement "{0}"
GBRETO033I	While transforming the implementation of constructor "{0}"
GBRETO	While transforming the implementation of the getter of attribute "{0}"

034I	
GBRETO035I	While transforming the implementation of the setter of attribute "{0}"
GBRETO036I	While transforming the value "{0}"
GBRETO037I	While transforming the initial value of attribute "{0}"
GBRETO038I	While applying model rewriter "{0}"
GBRETO039I	While transforming the declaration of constructor "{0}"
GBRETO040I	While transforming the declaration of method "{0}"
GBRETO041I	While transforming the declaration of attribute "{0}"
GBRETO042I	While transforming the declaration of indexer "{0}"
GBRETO043I	While transforming the declaration of the members of class "{0}"
GBRETO050E	Duplicated service installers for the service class "{0}"
GBRETO051E	The engine service "{0}" cannot be instantiated because it depends on another service "{1}" that is missing
GBRETO052E	Found a dependency loop related to the installer of the service class "{0}"
GBRETO053E	The engine service "{0}" is mandatory and was neither provided nor created
GBREW0027I	While applying business to execution (B2X) model mapping
GBREX0001E	Cannot find execution class "{0}" for translating business class "{1}"
GBREX0002E	Cannot find execution generic class "{0}" for translating business generic class "{1}"
GBREX0003E	Cannot map business class "{0}" to an execution class
GBREX0004E	Cannot find extender "{0}" for business class "{1}"
GBREX0005E	Extender method "{0}" must be static
GBREX0006E	Cannot find class "{0}" in business object model
GBREX0007E	Cannot find generic class "{0}" in business object model
GBREX0008E	Mismatch of static modifier between "{0}" and "{1}"
GBREX0009E	Mismatch of member type, type "{0}" is returned by member "{1}", whereas the type "{2}" or a subclass is expected.
GBREX0010W	Mismatch of member type, type "{0}" is returned by member "{1}", whereas type "{2}" or a subclass is expected. It might come from the use of covariance.
GBREX0011E	Cannot find method "{0}" in execution class "{1}"
GBREX0012I	In the B2X getter for attribute "{0}" of class "{1}"
GBREX0013I	In the B2X setter for attribute "{0}" of class "{1}"
GBREX0014I	In the B2X body for method "{0}" of class "{1}"
GBREX0	In the B2X body for constructor "{0}" of class "{1}"

015I	
GBREX0016I	In the B2X tester of class "{1}"
GBREX0017E	Business class "{0}" is marked as not translated
GBREX0018E	Business attribute "{0}" is marked as not translated
GBREX0019E	Business method "{0}" is marked as not translated
GBREX0020E	Business constructor "{0}" is marked as not translated
GBREX0021E	Cannot find attribute "{0}" in execution class "{1}"
GBREX0022E	Cannot find getter for attribute "{0}"
GBREX0023E	Cannot find setter for attribute "{0}"
GBREX0024E	Syntax error in pattern "{0}" : "{1}"
GBREX0025E	B2X cannot work without an engine data class concrete implementation
GBREX0026W	The attributes "{0}" and "{1}" in business enum class "{2}" are both mapped to the same value
GBREX0028E	Cannot generate the implementation of method "{0}" for type "{1}"
GBREX0029E	Attribute "{1}" of class "{0}" is a collection of type "{2}", which is not supported
GBREX0030E	Error "{0}" about "{1}" may be related to the usage of Classic Rule Engine API in the BOM
GBREX0031E	Attribute "{1}" of business class "{0}" is mapped to a readonly attribute
GBREX0032E	Attribute "{1}" of business class "{0}" is mapped to a writeonly attribute
GBREX0033E	Cannot load execution class "{0}" for translating business class "{1}", got "{2}"
GBREX0034E	Cannot load execution generic class "{0}" for translating business generic class "{1}", got "{2}"
GBREX0035E	Return found in setter for attribute "{0}"
GBREX0200E	in B2X body of {0}, at line {1}, column {2}
GBREX1000E	No importer registered for node "{0}"
GBREX1001E	Exception "{0}" occurred: "{1}" when importing the node "{2}"
GBREX1002E	Unexpected empty node "{0}"
GBREX1003E	Parse exception "{0}" occurred when reading "{1}"
GBREX1004E	Missing node "{0}" in the node "{1}"
GBREX1005E	Cannot load class "{0}" when reading the node "{1}"
GBREX1006E	Class "{0}" must have a default constructor or a constructor marked with "dataio.default"
GBREX1007W	Unknown attribute "{0}" when reading node "{1}"
GBREX1	Cannot set the value of read-only attribute "{0}"

008E	
GBREX1009E	Missing reference in enum value node "{0}"
GBREX1010E	Cannot find static attribute named "{0}" when reading "{1}" node
GBREX1011E	Cannot find component type "{0}"
GBREX1012E	Parameter "{0}" is missing for constructing an instance of class "{1}"
GBREX1013E	Component type is missing in array node
GBREX1022E	Object "{0}" of class "{1}" must be of expected type "{2}"
GBREX1023E	Method not accessible while writing or reading objects
GBREX1024W	Cannot instantiate collection class "{0}", the default "java.util.ArrayList" will be used
GBREX1025W	Cannot instantiate map class "{0}", the default "java.util.HashMap" will be used
GBREX1026I	While importing the value of attribute "{0}" of expected type "{1}"
GBREX1027I	While importing the value of a collection item at index "{0}"
GBREX1028I	While importing the content of an object of type "{0}"
GBREX1050E	No exporter registered for the class "{0}"
GBREX1051E	Cannot export attribute "{0}" with value of class "{1}"
GBREX1052E	Exception {0}:{1} occurred when exporting value of attribute "{2}"
GBREX1053E	Cannot find static attribute corresponding to value "{0}" when exporting node "{1}"
GBREX1080W	Cannot use constructor "{0}" for creating instance of class "{1}". No usable importer will be created for this class.
GBREX1081W	Cannot use constructor "{0}" for creating instance of class "{1}". No usable converter will be created for this class.
GBREX1082W	In the constructor tagged "dataio.default" of class "{0}", the parameter "{1}" of type "{2}" is not compatible with attribute "{1}" of type "{3}"
GBREX1083W	In the constructor tagged "dataio.default" of class "{0}", there is no attribute named as the parameter "{1}"
GBREX1084W	Business class "{0}" is mapped to "{1}" without a tester. All instances of "{1}" will be considered as being of business class "{0}" if property dataio.ignore is not true
GBREX1100E	Expected "{0}", got "{1}"
GBREX1101E	Unexpected end of document
GBREX1102I	At line {0} column {1}
GBREX1103E	Expected digit, got "{0}"
GBREX1104E	Expected hexadecimal number, got "{0}"
GBREX1105E	While reading unfinished string "{0}", got unexpected character "{1}" (in hexadecimal)
GBREX1106E	Invalid escape sequence, got "{0}"
GBREX1	Invalid beginning of JSON value, got unexpected character "{0}"

107E	
GBREX1 108E	Cannot retrieve value using JSONPATH "{0}"
GBREX1 109E	Type "{0}" should have a "value" element
GBREX1 110E	Unexpected element "{0}" in type "{0}"
GBREX1 111E	Enumerated type "{0}" should have a "\$enumref" element
GBREX1 112W	Ignoring unexpected element "{0}"
GBREX1 113E	Unexpected string "{0}" as a value of type "{1}"
GBREX1 114E	Expected end of document, got "{0}"
GBREX1 115E	Unexpected end of document while reading unfinished string "{0}"

Rule Designer Messages - Messages pertaining to the model and the business object model (BOM)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRMI0001E	Unknown attribute name
GBRMI0002E	unknown component property name
GBRMI0003E	Unknown type when creating the property {0}
GBRMI0004E	Parameter declaration is too late
GBRMI0005E	unknown method name
GBRMI0006E	Wrong domain definition
GBRMI0007E	Unexpected domain outside domain intersection or collection domain
GBRMI0008E	Unexpected modifier {0}
GBRMI0009E	invalid super class {0}
GBRMI0010E	invalid exception class {0}
GBRMI0011E	Type {0} cannot be changed: the metaclass is not an IlrMutableClass
GBRMI0012E	The type of attribute {0} is unknown when trying to read the default value
GBRMI0013E	Unknown type when creating the attribute {0}
GBRMI0014E	Unknown return type when creating the indexed property {0}
GBRMI0015E	Unknown return type when creating the method {0}
GBRMI0016E	Cannot change the type of the member {0} to {1}
GBRMI0017E	{0} errors found while parsing object model
GBRMI0018E	Method {0} not found for indexed component property {1}
GBRMI0019E	Method {0} not found for component property {1}
GBRMI0020W	Cannot find formatter {0}
GBRMI0021W	Cannot instantiate formatter {0}
GBRMI0022W	Class {0} already exists. The new declaration is ignored.
GBRMI0023W	Modifier {0} specified more than once
GBRMI0024E	Cannot not bind non generic type {0}
GBRMI0025E	Cannot not bind generic type {0} with parameters {1}
GBRMI0026E	Cannot bind {0} which is not a class
GBRMI0027E	Cannot include {0}: file not found
GBRMI0028E	Cannot include {0}: I/O Exception {1}
GBRMI0029E	Mismatch in properties
GBRMI0030E	Unresolved type {0}
GBRMI0031E	identifier expected, found "{0}"
GBRMI0032E	String literal expected, found "{0}"

GBRMI0033E	Word expected, found "{0}"
GBRMI0034E	End of file expected, found "{0}"
GBRMI0035E	integer value expected, found "{0}"
GBRMI0036E	Symbol "{0}" expected, found "{1}"
GBRMI0037E	Keyword "{0}" expected, found "{1}"
GBRMI0038E	Word "{0}" expected, found {1}
GBRMI0039E	Expected "class" or "interface" expected, found "{0}"
GBRMI0040E	Class name expected, found primitive type {0}
GBRMI0041E	Minimal cardinality has to be more than 0. It cannot be {0}
GBRMI0042E	invalid destructor declaration
GBRMI0043E	An operator has to be a method
GBRMM0001E	No BOM found
GBRMM0002E	Cannot migrate type {0}
GBRMM0003E	Cannot migrate attribute {0}
GBRMM0004E	Cannot migrate constructor {0}
GBRMM0005E	Cannot migrate method {0}
GBRMM0006E	Cannot migrate static reference "{0}": cannot find the corresponding attribute
GBRMM0007W	Enum "{0}" must not contain a public constructor
GBRMM0008W	Initial value "{0}" of type "{1}" is ignored as its type is not compatible with attribute "{2}"
GBRMO0001J	Object model
GBRMO0002E	Loop in inheritance tree
GBRMO0003E	LinkageError {1} while trying to load the class {0}
GBRMO0004E	ClassNotFoundException while trying to load the class {0}
GBRMO0005E	Cannot find attribute {0} referenced in domain
GBRMO0006E	Referenced attribute {0} must be static
GBRMO0007E	Referenced attribute is of type {0}, whereas type {1} is expected
GBRMO0008E	Actual value {0} is of type {1}, whereas type {2} is expected
GBRMO0009E	Actual value {0} is not well formatted: reading it back produces {1}
GBRMO0010W	Class {0} is deprecated. {1}
GBRMO0011W	Member {0} is deprecated. {1}
GBRMO0012W	Referenced type {0} is not defined
GBRMO0013E	Type is not known
GBRMO0014E	Error {1} when loading members of class {0}: {2}

GBRMO0015 E	Error {1} when loading members of class {0} : {2}
GBRMO0016 E	Error {1} when loading members of class {0} : {2}
GBRMO0017 E	There are two namespaces (probably a class and a package) with the same name {0}
GBRMO0018 E	Cannot merge class "{0}"
GBRMO0019 E	Cannot find referenced attribute ""{0}" from restricted attribute "{1}"
GBRMO0020 E	Cannot create package "{0}"
GBRMO0021 E	Member "{0}" has two parameters with the same name "{1}"
GBRMO0023 E	Enum class {0} cannot inherit from class {1} which has an incompatible domain
GBRMO0024 E	The domain of enum class {0} cannot contain the attribute {1} which is not in the domain of enum class {2}
GBRMO0025 E	The BOM element "{0}" has a legacy translation property "{1}". Translation properties are not supported anymore.

Rule Designer Messages - Messages pertaining to the business object model

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRS B000 1E	Failed to load XOM class: "{0}" for XOM path entry: "{1}"
GBRS B000 2E	Error processing "{0}" for element "{1}".
GBRS B000 3E	Unexpected exception in BOM plugins.
GBRS B000 4E	Unexpected exception in BOM plugins.
GBRS B000 5E	Unexpected exception in BOM plugins.
GBRS B000 6E	Unexpected exception in BOM plugins.
GBRS B000 7E	Unexpected exception in BOM plugins.
GBRS B000 8E	Unexpected exception in BOM plugins.
GBRS B000 9E	Unexpected exception in BOM plugins.
GBRS B001 0E	Unexpected exception in BOM plugins
GBRS B001 1E	Unexpected exception in BOM plugins.
GBRS B001 2E	Unexpected exception in BOM plugins.
GBRS B001 3E	Unexpected exception in BOM plugins
GBRS B001 4E	Unexpected exception in BOM plugins.
GBRS B001 5E	Unexpected exception in BOM plugins.
GBRS	Unexpected exception in BOM plugins.

B001 6E	
GBRS B001 7E	Unexpected exception in BOM plugins.
GBRS B001 8E	Unexpected exception in BOM plugins.
GBRS B001 9E	Unexpected exception in BOM plugins.
GBRS B002 0E	Unexpected exception in BOM plugins.
GBRS B002 1E	Unexpected exception in BOM plugins.
GBRS B002 2E	Unexpected exception in BOM plugins.
GBRS B002 3E	Unexpected exception in BOM plugins.
GBRS B002 4E	Unexpected exception in BOM plugins.
GBRS B002 5E	Unexpected exception in BOM plugins.
GBRS B002 6E	Unexpected exception in BOM plugins.
GBRS B002 7E	Unexpected exception in BOM plugins.
GBRS B002 8E	Unexpected exception in BOM wizards plugins
GBRS B002 9E	Unexpected exception in BOM wizards plugins
GBRS B003 0E	Unexpected exception in BOM wizards plugins
GBRS B003 1E	Unexpected exception in BOM wizards plugins
GBRS B003 2E	Unexpected exception in BOM wizards plugins
GBRS B003 3E	Unexpected exception in BOM wizards plugins
GBRS B003 4E	Unexpected exception in BOM wizards plugins
GBRS B003 5E	Unexpected exception in BOM wizards plugins
GBRS B003 6E	Unexpected exception in BOM wizards plugins
GBRS	Unexpected exception in BOM wizards plugins

B0037E	
GBRS B0038E	Unexpected exception in BOM wizards plugins
GBRS B0039E	Unexpected exception in BOM wizards plugins
GBRS B0040E	Unexpected exception in BOM wizards plugins
GBRS B0041E	Unexpected exception in BOM wizards plugins
GBRS B0042E	Unexpected exception in BOM wizards plugins
GBRS B0043E	Unexpected exception in BOM wizards plugins
GBRS B0044E	NoClassDefFoundError: "{0}"
GBRS B0045E	Could not load all the selected classes. These classes may be private or protected, or other classes that are required to load these classes may be missing. Please verify the classes can be loaded by the JDK using Class.forName(...).
GBRS B0046E	Errors raised loading classes:
GBRS B0047I	Warnings raised loading classes:
GBRS B0048E	Impossible to read "{0}" : "{1}"
GBRS B0049E	NoClassDefFoundError: "{0}"
GBRS B0050E	The BOM entry file contains invalid information. Fix the problems and reopen the editor. See the log file for details.
GBRS B0052E	Unexpected exception. See log file for details.
GBRS B0053E	Failed to find type in temporary object model: "{0}"
GBRS B0055E	Failed to create a renderer for a domain used in the BOM.
GBRS B0056E	The domain on the element "{0}" contains an invalid entry: "{1}".

Rule Designer Messages - Messages pertaining to debug

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSD0001E	Exception while retrieving underlying stack frame
GBRSD0002E	Exception while computing source support
GBRSD0003E	Exception while invoking method : "{0}"
GBRSD0004E	Exception while stepping out of rule stack frame
GBRSD0005E	Exception while stepping out of rule stack frame
GBRSD0006E	Exception while stepping over rule stack frame
GBRSD0007E	Exception while stepping over rule stack frame
GBRSD0008E	Exception while stepping into rule stack frame
GBRSD0009E	Exception while stepping into rule stack frame
GBRSD0010E	Exception while stepping into rule stack frame
GBRSD0011E	Exception while stepping into rule stack frame
GBRSD0012E	Exception while stepping into rule stack frame
GBRSD0013E	Exception while launching debug target
GBRSD0014E	Exception while removing breakpoint
GBRSD0015E	Exception while removing breakpoint (bad location)
GBRSD0016E	Exception while retrieving breakpoint markers
GBRSD0017E	Exception while adding object breakpoint
GBRSD0018E	Exception while computing source support
GBRSD0019E	Exception while computing source support
GBRSD0020E	Exception while initializing data connector tab
GBRSD0021E	Exception while updating extractor from configuration
GBRSD0022E	Exception while loading Java project

GBRSD0023E	Exception while updating project from configuration
GBRSD0024E	Exception while retrieving attribute from configuration
GBRSD0025E	Exception while retrieving attribute from configuration
GBRSD0026E	Exception while retrieving attribute from configuration
GBRSD0027E	Exception while initializing name
GBRSD0028E	Exception while retrieving attribute from configuration
GBRSD0029E	Exception while retrieving attribute from configuration
GBRSD0030E	Exception while retrieving attribute from configuration
GBRSD0031E	Exception while retrieving attribute from configuration
GBRSD0032E	Exception while retrieving attribute from configuration
GBRSD0033E	Exception while loading Java project
GBRSD0034E	Exception while retrieving attribute from configuration
GBRSD0035E	Exception while retrieving classpath from configuration
GBRSD0036E	Exception while retrieving classpath from configuration
GBRSD0037E	Exception while displaying default classpath
GBRSD0038E	Exception while retrieving attribute from configuration
GBRSD0039E	Exception while retrieving attribute from configuration
GBRSD0040E	Exception while retrieving attribute from configuration
GBRSD0041E	Exception while retrieving attribute from configuration
GBRSD0042E	Exception while retrieving attribute from configuration
GBRSD0043E	Exception while retrieving attribute from configuration
GBRSD0044E	Exception while initializing source lookup table
GBRSD0045E	Exception while retrieving attribute from configuration
GBRSD0046E	Exception while retrieving attribute from configuration
GBRSD0047E	Exception while retrieving attribute from configuration
GBRSD0048E	Exception while retrieving attribute from configuration
GBRSD0049E	Exception while retrieving attribute from configuration
GBRSD0050E	Exception while retrieving attribute from configuration
GBRSD0051E	Exception while initializing image path

GBRSD005 2E	Exception while creating image descriptor
GBRSD005 3E	Exception while creating image descriptor
GBRSD005 4E	Exception while creating image descriptor
GBRSD005 5E	Exception while creating image descriptor
GBRSD005 6E	Exception while retrieving model presentation
GBRSD005 7E	Exception while saving configuration
GBRSD005 8E	Exception while looking for source element
GBRSD005 9E	Exception while showing source page
GBRSD006 0E	Unexpected exception in launching plugins
GBRSD006 1E	Unexpected exception in launching plugins
GBRSD006 2E	Unexpected exception in launching plugins
GBRSD006 3E	Unexpected exception in launching plugins
GBRSD006 4E	Unexpected exception in launching plugins
GBRSD006 5E	Unexpected exception in launching plugins
GBRSD006 6E	Unexpected exception in launching plugins
GBRSD006 7E	Unexpected exception in launching plugins
GBRSD006 8E	Unexpected exception in launching plugins
GBRSD006 9E	Unexpected exception in launching plugins
GBRSD007 0E	Unexpected exception in launching plugins
GBRSD007 1E	Unexpected exception in launching plugins
GBRSD007 2E	Unexpected exception in launching plugins
GBRSD007 3E	Unexpected exception in launching plugins
GBRSD007 4E	Unexpected exception in launching plugins
GBRSD007 5E	Unexpected exception in launching plugins
GBRSD007 6E	Unexpected exception in launching plugins
GBRSD007 7E	Unexpected exception in launching plugins
GBRSD007 8E	Unexpected exception in launching plugins
GBRSD007 9E	Unexpected exception in launching plugins
GBRSD008 0E	Unexpected exception in launching plugins

GBRSD0081E	Unexpected exception in launching plugins
GBRSD0082E	VM not fully specified in launch configuration {0} - missing VM name. Reverting to default VM.
GBRSD0084E	Invalid launchBuildModeSupport declaration from plugin {0}.
GBRSD0085E	Failed to instantiate an IlrProjectLaunchBuildModeSupport for the {0} build mode. Was the declaring plugin unloaded?
GBRSD0086E	No registered launchBuildModeSupport extension for build mode {0}.
GBRSD0087E	An unexpected error occurred while retrieving the location of the jrules-engine.jar
GBRSD0088E	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".
GBRSD0089E	An unexpected error occurred while generating ruleset archive "{0}" with extractor "{1}".
GBRSD0090E	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".

Rule Designer Messages - Messages pertaining to rule execution

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSE0001E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0002E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0003E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0004E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0005E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0006E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0007E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0008E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0009E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0010E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0011E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0012E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0013E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0014E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0015E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0016E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0017E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0018E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0019E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0020E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0021E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0022E	Unexpected exception in Rule Execution Server plug-ins

GBRSE0023E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0024E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0025E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0026E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0027E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0028E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0029E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0030E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0031E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0032E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0033E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0034E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0035E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0036E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0037E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0038E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0039E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0040E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0041E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0042E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0043E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0044E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0045E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0046E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0047E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0048E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0049E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0050E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0051E	Unexpected exception in Rule Execution Server plug-ins

GBRSE0052E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0053E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0054E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0055E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0056E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0057E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0058E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0061E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0064E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0065E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0066E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0067E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0068E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0069E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0070E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0071E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0072E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0073E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0074E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0075E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0076E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0077E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0078E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0079E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0080E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0081E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0082E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0083E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0084E	Unexpected exception in Rule Execution Server plug-ins

GBRSE0085E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0086E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0087E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0088E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0089E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0090E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0091E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0092E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0093E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0094E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0095E	An unexpected error occurred while loading the RuleApp XML descriptor "{0}".
GBRSE0102E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0103E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0104E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0105E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0106E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0107E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0108E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0109E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0110E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0111E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0112E	Unexpected exception while retrieving the engine type for ruleset archive "{0}".
GBRSE0117E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0118E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0119E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0120E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0121E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0122E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0123E	Unexpected exception in Rule Execution Server plug-ins

GBRSE 0124E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0125E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0126E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0127E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0128E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0129E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0130E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0131E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0132E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0134E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0135E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0136E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0137E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0138E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0139E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0140E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0145E	Failed to instantiate a password encryption manager. Will fall back to a non-encryption policy.
GBRSE 0146E	Failed to encrypt password. Returning the password not encrypted.
GBRSE 0147E	Failed to decrypt password. Returning the empty password.
GBRSE 0148E	Failed to decrypt password. Returning the empty password.
GBRSE 0149E	Failed to decrypt password. Returning the empty password.
GBRSE 0150E	Failed to decrypt password. Returning the empty password.
GBRSE 0153E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0154E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0155E	The file which stored the login and password information could not be found. Please re-enter the login and password values for your Rule Execution Server Configurations.
GBRSE 0156E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0157E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0158E	Unexpected exception in Rule Execution Server plug-ins
GBRSE 0159E	Unexpected exception in Rule Execution Server plug-ins

GBRSE0160E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0161E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0162E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0163E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0164E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0165E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0166E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0167E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0168E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0169E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0170E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0171E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0172E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0173E	Unexpected exception in Rule Execution Server plug-ins
GBRSE0174E	"{0}" : "{1}"
GBRSE0175E	The ruleset archive cannot be found.
GBRSE0176E	Not a valid ruleset archive.
GBRSE0177E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0178E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0179E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0180E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0181E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0182E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0183E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0184E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0185E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0186E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0187E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0188E	Unexpected exception in Rule Execution Server UI plug-ins.

GBRSE0189E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0190E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0191E	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
GBRSE0192E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0193E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0194E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0195E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0196E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0197E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0198E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0199E	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
GBRSE0200E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0201E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0202E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0203E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0204E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0205E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0206E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0207E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0208E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0209E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0210E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0211E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0212E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0213E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0214E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0215E	Failed to connect to Rule Execution Server at "{0}".
GBRSE0216E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE0217E	Unexpected exception in Rule Execution Server UI plug-ins.

GBRSE 0218E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE 0219E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE 0220E	Unexpected exception in Rule Execution Server UI plug-ins.
GBRSE 0221E	Unexpected exception in RES UI client plugins
GBRSE 0222E	Unexpected exception in RES UI client plugins
GBRSE 0223E	Unexpected exception in RES UI client plugins
GBRSE 0224E	Unexpected exception in RES UI client plugins
GBRSE 0225E	Unexpected exception in RES UI client plugins
GBRSE 0226E	Unexpected exception in RES UI client plugins
GBRSE 0227E	Unexpected exception in RES UI client plugins
GBRSE 0228E	Unexpected exception in RES UI client plugins
GBRSE 0229E	Unexpected exception in RES UI client plugins
GBRSE 0230E	Unexpected exception in RES UI client plugins
GBRSE 0231E	Unexpected exception in RES UI client plugins
GBRSE 0232E	Unexpected exception in RES UI client plugins
GBRSE 0233E	Unexpected exception in RES UI client plugins
GBRSE 0234E	Unexpected exception in RES UI client plugins
GBRSE 0235E	Unexpected exception in RES UI client plugins
GBRSE 0236E	Unexpected exception in RES UI client plugins
GBRSE 0237E	Unexpected exception in RES UI client plugins
GBRSE 0240E	Unexpected exception in RES UI client plugins
GBRSE 0241E	Unexpected exception in RES UI client plugins
GBRSE 0242E	Unexpected exception in RES UI client plugins
GBRSE 0243E	Unexpected exception in RES UI client plugins
GBRSE 0244E	Unexpected exception in RES UI client plugins
GBRSE 0245E	Unexpected exception in RES UI client plugins
GBRSE 0262E	Unexpected exception in DVS plugins
GBRSE 0263E	Unexpected exception in DVS plugins
GBRSE 0264E	Unexpected exception in DVS plugins

GBRSE 0265E	Unexpected exception in DVS plugins
GBRSE 0266E	Unexpected exception in DVS plugins
GBRSE 0267E	Unexpected exception in DVS plugins
GBRSE 0268E	Unexpected exception in DVS plugins
GBRSE 0269E	Unexpected exception in DVS plugins
GBRSE 0270E	Unexpected exception in DVS plugins
GBRSE 0271E	Unexpected exception in DVS plugins
GBRSE 0272E	Unexpected exception in DVS plugins
GBRSE 0273E	Unexpected exception in DVS plugins
GBRSE 0274E	Unexpected exception in DVS plugins
GBRSE 0275E	Unexpected exception in DVS plugins
GBRSE 0276E	Unexpected exception in DVS plugins
GBRSE 0277E	Unexpected exception in DVS plugins
GBRSE 0278E	Unexpected exception in DVS plugins
GBRSE 0279E	Unexpected exception in DVS plugins
GBRSE 0280E	Unexpected exception in DVS plugins
GBRSE 0281E	Unexpected exception in DVS plugins
GBRSE 0282E	Unexpected exception in DVS plugins
GBRSE 0283E	Unexpected exception in DVS plugins
GBRSE 0284E	Unexpected exception in DVS plugins
GBRSE 0285E	Unexpected exception in DVS plugins
GBRSE 0286E	Unexpected exception in DVS plugins
GBRSE 0287E	Unexpected exception in DVS plugins
GBRSE 0288E	Unexpected exception in DVS plugins
GBRSE 0289E	Unexpected exception in DVS plugins
GBRSE 0290E	Unexpected exception in DVS plugins
GBRSE 0291E	Unexpected exception in DVS plugins
GBRSE 0292E	Unexpected exception in DVS plugins
GBRSE 0293E	Unexpected exception in DVS plugins

GBRSE0294E	Unexpected exception in DVS plugins
GBRSE0295E	Unexpected exception in DVS plugins
GBRSE0296E	Unexpected exception in DVS plugins
GBRSE0297E	Unexpected exception in DVS plugins
GBRSE0298E	Unexpected exception in DVS plugins
GBRSE0299E	Unexpected exception in DVS plugins
GBRSE0300E	Unexpected exception in DVS plugins
GBRSE0301E	Unexpected exception in DVS plugins
GBRSE0302E	Unexpected exception in DVS plugins
GBRSE0303E	Unexpected exception in DVS plugins
GBRSE0304E	Unexpected exception in DVS plugins
GBRSE0305E	Unexpected exception in DVS plugins
GBRSE0306E	Unexpected exception in DVS plugins
GBRSE0307E	Unexpected exception in DVS plugins
GBRSE0308E	Unexpected exception in DVS plugins
GBRSE0309E	Unexpected exception in DVS plugins
GBRSE0310E	Unexpected exception in DVS plugins
GBRSE0311E	Unexpected exception in DVS plugins
GBRSE0312E	Unexpected exception in DVS plugins
GBRSE0313E	Unexpected exception in DVS plugins
GBRSE0314E	Unexpected exception in DVS plugins
GBRSE0315E	Unexpected exception in DVS plugins
GBRSE0316E	Unexpected exception in DVS plugins
GBRSE0317E	Unexpected exception in DVS plugins
GBRSE0318E	Unexpected exception in DVS plugins
GBRSE0319E	Unexpected exception in DVS plugins
GBRSE0320E	Unexpected exception in DVS plugins
GBRSE0321E	Unexpected exception in DVS plugins
GBRSE0322E	Unexpected exception in DVS plugins

GBRSE0323E	Unexpected exception in DVS plugins
GBRSE0324E	Unexpected exception in DVS plugins
GBRSE0325E	Unexpected exception in DVS plugins
GBRSE0326E	Unexpected exception in DVS plugins
GBRSE0327E	Unexpected exception in DVS plugins
GBRSE0328E	Unexpected exception in DVS plugins
GBRSE0329E	Unexpected exception in DVS plugins
GBRSE0330E	Unexpected exception in DVS plugins
GBRSE0331E	Unexpected exception in DVS plugins
GBRSE0332E	Unexpected exception in DVS plugins
GBRSE0333E	Unexpected exception in DVS plugins
GBRSE0334E	Unexpected exception in DVS plugins
GBRSE0335E	Unexpected exception in DVS plugins
GBRSE0336E	Unexpected exception in DVS plugins
GBRSE0337E	Unexpected exception in DVS plugins
GBRSE0338E	Unexpected exception in DVS plugins
GBRSE0339E	Unexpected exception in DVS plugins
GBRSE0340E	Unexpected exception in DVS plugins
GBRSE0341E	Unexpected exception in DVS plugins
GBRSE0342E	Unexpected exception in DVS plugins
GBRSE0343E	Unexpected exception in DVS plugins
GBRSE0344E	Unexpected exception in DVS plugins
GBRSE0345E	Unexpected exception in DVS plugins
GBRSE0346E	Unexpected exception in DVS plugins
GBRSE0347E	Unexpected exception in DVS plugins
GBRSE0348E	Unexpected exception in DVS plugins
GBRSE0349E	Unexpected exception in DVS plugins
GBRSE0350E	Unexpected exception in DVS plugins
GBRSE0351E	Unexpected exception in DVS plugins

GBRSE0352E	Unexpected exception in DVS plugins
GBRSE0353E	Ruleset cannot be generated
GBRSE0354E	Directory cannot be created: "{0}"
GBRSE0355E	The ruleset archive cannot be parsed, see following errors.
GBRSE0358E	The ruleset archive cannot be parsed, verify the XOM path
GBRSE0359E	The referenced rule project seems to be closed.
GBRSE0360E	The file cannot be written: it is read-only.
GBRSE0365E	Unexpected exception in DVS UI plug-ins
GBRSE0366E	Unexpected exception in DVS UI plug-ins
GBRSE0367E	Unexpected exception in DVS UI plug-ins
GBRSE0368E	Unexpected exception in DVS UI plug-ins
GBRSE0369E	Unexpected exception in DVS UI plug-ins
GBRSE0370E	Unexpected exception in DVS UI plug-ins
GBRSE0371E	Unexpected exception in DVS UI plug-ins
GBRSE0372E	Unexpected exception in DVS UI plug-ins
GBRSE0373E	Unexpected exception in DVS UI plug-ins
GBRSE0374E	Unexpected exception in DVS UI plug-ins
GBRSE0375E	Unexpected exception in DVS UI plug-ins
GBRSE0376E	Unexpected exception in DVS UI plug-ins
GBRSE0377E	Unexpected exception in DVS UI plug-ins
GBRSE0378E	Unexpected exception in DVS UI plug-ins
GBRSE0379E	Unexpected exception in DVS UI plug-ins
GBRSE0380E	Unexpected exception in DVS UI plug-ins
GBRSE0381E	Unexpected exception in DVS UI plug-ins
GBRSE0382E	Unexpected exception in DVS UI plug-ins
GBRSE0383E	Unexpected exception in DVS UI plug-ins
GBRSE0384E	Unexpected exception in DVS UI plug-ins
GBRSE0385E	Unexpected exception in DVS UI plug-ins
GBRSE0386E	Unexpected exception in DVS UI plug-ins

GBRSE0387E	Unexpected exception in DVS UI plug-ins
GBRSE0388E	Unexpected exception in DVS UI plug-ins
GBRSE0389E	Unexpected exception in DVS UI plug-ins
GBRSE0390E	Unexpected exception in DVS UI plug-ins
GBRSE0391E	Unexpected exception in DVS UI plug-ins
GBRSE0392E	Unexpected exception in DVS UI plug-ins
GBRSE0393E	Unexpected exception in DVS UI plug-ins
GBRSE0394E	Unexpected exception in DVS UI plug-ins
GBRSE0395E	Unexpected exception in DVS UI plug-ins
GBRSE0396E	Unexpected exception in DVS UI plug-ins
GBRSE0397E	Unexpected exception in DVS UI plug-ins
GBRSE0398E	Unexpected exception in DVS UI plug-ins
GBRSE0399E	Unexpected exception in DVS UI plug-ins
GBRSE0400E	Unexpected exception in DVS UI plug-ins
GBRSE0401E	Unexpected exception in DVS UI plug-ins
GBRSE0402E	Unexpected exception in DVS UI plug-ins
GBRSE0403E	Unexpected exception in DVS UI plug-ins
GBRSE0404E	Unexpected exception in DVS UI plug-ins
GBRSE0405E	Unexpected exception in DVS UI plug-ins
GBRSE0406E	Unexpected exception in DVS UI plug-ins
GBRSE0407E	Unexpected exception in DVS UI plug-ins
GBRSE0408E	Unexpected exception in DVS UI plug-ins
GBRSE0409E	An unexpected exception occurred while accessing the project {0}.
GBRSE0410E	Unexpected exception in DVS UI plug-ins
GBRSE0411E	Unexpected exception in DVS UI plug-ins
GBRSE0412E	Unexpected exception in DVS UI plug-ins
GBRSE0413E	Unexpected exception in DVS UI plug-ins
GBRSE0414E	Unexpected exception in DVS UI plug-ins
GBRSE0415E	An unexpected error occurred while accessing the DVS project {0}.

GBRSE 0416E	Unexpected exception in DVS UI plug-ins
GBRSE 0417E	Unexpected exception in DVS UI plug-ins
GBRSE 0418E	Unexpected exception in DVS UI plug-ins
GBRSE 0419E	Unexpected exception in DVS UI plug-ins
GBRSE 0420E	Unexpected exception in DVS UI plug-ins
GBRSE 0422E	Unexpected exception in DVS UI plug-ins
GBRSE 0423E	Unexpected exception in DVS UI plug-ins
GBRSE 0424E	Unexpected exception in DVS UI plug-ins
GBRSE 0425E	Unexpected exception in DVS UI plug-ins
GBRSE 0426E	Unexpected exception in DVS UI plug-ins
GBRSE 0427E	Unexpected exception in DVS UI plug-ins
GBRSE 0428E	Unexpected exception in DVS UI plug-ins
GBRSE 0429E	Unexpected exception in DVS UI plug-ins
GBRSE 0430E	Unexpected exception in DVS UI plug-ins
GBRSE 0431E	Unexpected exception in DVS UI plug-ins
GBRSE 0432E	An exception occurred while checking decision operation "{0}" for compatibility with testing and simulation.
GBRSE 0434E	Failed to run command
GBRSE 0435E	Failed to run command
GBRSE 0436E	An error occurs during the repackaging. See the log file for more information.
GBRSE 0437E	An error occurred removing the project {0}. See log file for details.
GBRSE 0438E	An error occurred removing the project {0}. See log file for details.
GBRSE 0439E	An error occurred removing the project {0}. See log file for details.
GBRSE 0440E	An error occurred removing the project {0}. See log file for details.
GBRSE 0441E	Exception while generating deployment.xml file.
GBRSE 0442E	Unexpected exception while reading launch configuration
GBRSE 0443E	There is no entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
GBRSE 0444E	An unexpected error occurred while accessing flags for the IType element {0} in DVS project {1}.
GBRSE 0445E	An unexpected error occurred while retrieving the hierarchy of IType element {0} in DVS project {1}.
GBRSE 0446E	An unexpected error occurred while retrieving the IType element {0} in DVS project {1}.

GBRSE 0447E	An unexpected error occurred while retrieving the KPI Result classes compatible with the Decision Center renderer {0} in DVS project {1}.
GBRSE 0448E	Exception while making .zip file for XOM deployment.
GBRSE 0449E	The generation of the DVS KPI implementation classes was unexpectedly interrupted.
GBRSE 0450E	An unexpected error occurred during the generation of the DVS KPI implementation classes.
GBRSE 0451E	The generation of the DVS KPI Result Aggregator implementation classes was unexpectedly interrupted.
GBRSE 0452E	An unexpected error occurred during the generation of the DVS KPI Result Aggregator implementation classes.
GBRSE 0453E	The generation of the DVS Scenario Provider implementation classes was unexpectedly interrupted.
GBRSE 0454E	An unexpected error occurred during the generation of the Scenario Provider implementation classes.
GBRSE 0455E	An unexpected error occurred while applying the changes brought about by renaming the DVS Format file "{0}".
GBRSE 0457E	An unexpected error occurred while synchronizing the editor of the DVS Customization "{0}".
GBRSE 0458E	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Format file "{0}".
GBRSE 0459E	An unexpected error occurred while applying the changes brought about by the DVS Configuration "{0}".
GBRSE 0460E	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Configuration "{0}".
GBRSE 0461E	An unexpected error occurred while applying the changes brought about by the deletion of the Rule project "{0}" to "{1}".
GBRSE 0462E	An unexpected error occurred while checking the nature of the project "{0}".
GBRSE 0467E	Failed to connect to Rule Execution Server at "{0}".
GBRSE 0471E	An unexpected exception occurred while checking rule project "{0}" for compatibility with testing and simulation.
GBRSE 0477E	Errors when trying to detect the engine type of rulesets declared in RuleApp {0}.
GBRSE 0479E	There is no deployment.xml file for rule project "{0}". To generate this file, deploy the XOM from "{0}" to "{1}".
GBRSE 0480E	There is no deployed XOM entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
GBRSE 0481E	The rule project that is passed to the validator for XOM deployment descriptor files is undefined (null value).
GBRSE 0482E	An unexpected error occurred while setting up the test report data for DVS launch configuration "{0}".
GBRSE 0483E	An unexpected error occurred upon a workspace refresh during the execution of DVS launch configuration "{0}".
GBRSE 0484E	An unexpected error occurred while opening the DVS execution report "{0}".
GBRSE 0485E	An unexpected error occurred while refreshing the workspace resource "{0}".
GBRSE 0486E	An unexpected exception occurred while retrieving the signature of the ruleset archive that is located in "{0}". Please check the Error View for more information about the cause of this issue.
GBRSE 0487E	An unexpected exception occurred while looking for decision operations in rule project "{0}".
GBRSE 0488E	An exception occurred while checking rule project "{0}" for compatibility with testing and simulation.
GBRSE 0489E	The check for compatibility with testing and simulation was interrupted for rule project "{0}".

GBRSE0490E	The variable "{0}" was not found in rule package "{1}".
GBRSE0491E	The variable "{0}" that is referenced by decision operation "{1}" was not found in the default package.
GBRSE0492E	The variable set "{0}" that is referenced by variable "{1}" in decision operation "{2}" was not found.
GBRSE0493E	An unexpected error occurred while retrieving the JRE upon generation of Java project "{0}".
GBRSE0494E	An unexpected exception occurred while retrieving the Decision Service property for project "{0}".
GBRSE0495E	An unexpected exception occurred while retrieving attribute "{0}" of type "java.lang.String" for launch configuration "{0}".
GBRSE0496E	An unexpected exception occurred while retrieving attribute "{0}" of type "boolean" for launch configuration "{0}".
GBRSE0497E	An unexpected error occurred while checking file location "{0}", where the Excel file with DVS output values is stored, for write access.
GBRSE0498E	Unexpected exception in DVS plugins
GBRSE0499E	Unexpected exception in DVS plugins
GBRSE0500E	Unexpected exception in SDsExcelRunner.
GBRSE0501E	An unexpected exception occurred when retrieving DVS input and output parameters values.
GBRSE0502E	An unexpected exception occurred when generating the DVS output values Excel file.
GBRSE0503E	Unexpected exception in SDsExcelRunner.
GBRSE0504E	Unexpected exception in IlrExcelRunner.
GBRSE0505E	Unexpected exception in IlrExcelRunner.
GBRSE0506E	Unexpected exception in IlrDVSArchiveRunner.
GBRSE0507E	An unexpected error occurred while exporting the RuleApp "{0}".
GBRSE0508E	Failed to connect to Rule Execution Server at "{0}".
GBRSE0509E	An unexpected error occurred during XOM deployment.

Rule Designer Messages - Messages pertaining to the model (00 - 99)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSM0001E	Unexpected exception in model
GBRSM0002E	Unexpected exception in model
GBRSM0003E	Unexpected exception in model
GBRSM0004E	Unexpected exception in model
GBRSM0005E	Unexpected exception in model
GBRSM0006E	Unexpected exception in model
GBRSM0007E	Unexpected exception in model
GBRSM0008E	Unexpected exception in model
GBRSM0009E	Unexpected exception in model
GBRSM0010E	Unexpected exception in model
GBRSM0011E	Unexpected exception in model
GBRSM0012E	Unexpected exception in model
GBRSM0013E	Unexpected exception in model
GBRSM0014E	Unexpected exception in model
GBRSM0015E	Unexpected exception in model
GBRSM0016E	Unexpected exception in model
GBRSM0017E	Unexpected exception in model
GBRSM0018E	Unexpected exception in model
GBRSM0019E	Unexpected exception in model
GBRSM0020E	Unexpected exception in model
GBRSM0021E	Unexpected exception in model
GBRSM0022E	Unexpected exception in model
GBRSM0023E	Unexpected exception in model
GBRSM0024E	Unexpected exception in model
GBRSM0025E	Unexpected exception in model
GBRSM0026E	Unexpected exception in model
GBRSM0027E	Unexpected exception in model
GBRSM0028E	Unexpected exception in model
GBRSM0029E	Unexpected exception in model
GBRSM0030E	Unexpected exception in model
GBRSM0031E	Unexpected exception in model
GBRSM0032E	Unexpected exception in model
GBRSM0033E	Unexpected exception in model
GBRSM0034E	Unexpected exception in model

GBRSM0035E	Unexpected exception in model
GBRSM0036E	Unexpected exception in model
GBRSM0037E	Unexpected exception in model
GBRSM0038E	Unexpected exception in model
GBRSM0039E	Unexpected exception in model
GBRSM0040E	Unexpected exception in model
GBRSM0041E	Unexpected exception in model
GBRSM0042E	Unexpected exception in model
GBRSM0043E	Unexpected exception in model
GBRSM0044E	Unexpected exception in model
GBRSM0045E	Unexpected exception in model
GBRSM0046E	Exception raised while compiling "{0}"
GBRSM0047E	Unexpected exception in model
GBRSM0048E	Unexpected exception in model
GBRSM0049E	Unexpected exception in model
GBRSM0050E	Unexpected exception in model
GBRSM0051E	Unexpected exception in model
GBRSM0052E	Unexpected exception in model
GBRSM0053E	Unexpected exception in model
GBRSM0054E	Unexpected exception in model
GBRSM0055E	Unexpected exception in model
GBRSM0056E	Unexpected exception in model
GBRSM0057E	Unexpected exception in model
GBRSM0058E	Unexpected exception in model
GBRSM0059E	Unexpected exception in model
GBRSM0060E	Unexpected exception in model
GBRSM0061E	Unexpected exception in model
GBRSM0062E	Unexpected exception in model
GBRSM0063E	Unexpected exception in model
GBRSM0064E	Unexpected exception in model
GBRSM0065E	Unexpected exception in model
GBRSM0066E	Unexpected exception in model
GBRSM0067E	Unexpected exception in model
GBRSM0068E	Unexpected exception in model
GBRSM0069E	Unexpected exception in model
GBRSM0070E	Unexpected exception in model
GBRSM0071E	Unexpected exception in model
GBRSM0072E	Unexpected exception in model
GBRSM0073E	Unexpected exception in model
GBRSM0074E	Unexpected exception in model
GBRSM0075E	Unexpected exception in model
GBRSM0076E	Unexpected exception in model
GBRSM0077E	Unexpected exception in model
GBRSM0078E	Unexpected exception in model
GBRSM0079E	Unexpected exception in model
GBRSM0080E	Unexpected exception in model
GBRSM0081E	Unexpected exception in model

GBRSM0082E	Unexpected exception in model
GBRSM0083E	Unexpected exception in model
GBRSM0084E	Unexpected exception in model
GBRSM0085E	Unexpected exception in model
GBRSM0086E	Unexpected exception in model
GBRSM0087E	Unexpected exception in model
GBRSM0088E	Unexpected exception in model
GBRSM0089E	Unexpected exception in model
GBRSM0090E	Unexpected exception in model
GBRSM0091E	Unexpected exception in model
GBRSM0092E	Unexpected exception in model
GBRSM0093E	Unexpected exception in model
GBRSM0094E	Unexpected exception in model
GBRSM0095E	Unexpected exception in model
GBRSM0096E	Unexpected exception in model
GBRSM0097E	Unexpected exception in model
GBRSM0098E	Unexpected exception in model
GBRSM0099E	Unexpected exception in model

Rule Designer Messages - Messages pertaining to the model (100-199)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSM0100E	Unexpected exception in model
GBRSM0101E	Unexpected exception in model
GBRSM0102E	Unexpected exception in model
GBRSM0103E	Unexpected exception in model
GBRSM0104E	Unexpected exception in model
GBRSM0105E	Unexpected exception in model
GBRSM0106E	Unexpected exception in model
GBRSM0107E	Unexpected exception in model
GBRSM0108E	Unexpected exception in model
GBRSM0109E	Unexpected exception in model
GBRSM0110E	Unexpected exception in model
GBRSM0111E	Unexpected exception in model
GBRSM0112E	Unexpected exception in model
GBRSM0113E	Unexpected exception in model
GBRSM0114E	Unexpected exception in model
GBRSM0115E	Unexpected exception in model
GBRSM0116E	Unexpected exception in model
GBRSM0117E	Unexpected exception in model
GBRSM0118E	Unexpected exception in model
GBRSM0119E	Unexpected exception in model
GBRSM0120E	Unexpected exception in model

GBRSM0121E	Unexpected exception in model
GBRSM0122E	Failed to create error marker on .project for rule project "{0}"
GBRSM0124E	Unexpected exception in model
GBRSM0125E	Unexpected exception in model
GBRSM0126E	Unexpected exception in model
GBRSM0127E	Unexpected exception in model
GBRSM0128E	Unexpected exception in model
GBRSM0129E	Unexpected exception in model
GBRSM0130E	Unexpected exception in model
GBRSM0131E	Unexpected exception in model
GBRSM0132E	Unexpected exception in model
GBRSM0133E	Failed to initialize variable {0}.
GBRSM0134E	Unexpected exception in model
GBRSM0135E	Failed to save the build mode property of rule project {0} on disk.
GBRSM0136E	Unexpected exception in model
GBRSM0137E	Unexpected exception in model
GBRSM0138E	Unexpected exception in model
GBRSM0139E	Unexpected exception in model
GBRSM0140E	Unexpected exception in model
GBRSM0141E	Unexpected exception in model
GBRSM0142E	Unexpected exception in model
GBRSM0143E	Unexpected exception in model
GBRSM0144E	Unexpected exception in model
GBRSM0145E	Unexpected exception in model
GBRSM0146E	Cannot create impact file: "{0}"
GBRSM0147E	Error while registering impact for file: "{0}"
GBRSM0148E	Error while registering impact for file: "{0}"
GBRSM0149E	Error while retrieving impact for file: "{0}"
GBRSM0150E	Error while retrieving impact for file: "{0}"

GBRSM0151E	Error while resetting impact for file: "{0}"
GBRSM0152E	Error while retrieving impact for file: "{0}"
GBRSM0153E	An unexpected error occurred while retrieving the model elements impacted by {0} changes on rule project {1}.
GBRSM0154E	An unexpected error occurred while checking whether a {0} change took place on rule project: {1}.
GBRSM0155E	Error getting impacts for element: "{0}"
GBRSM0156E	Exception raised while compiling "{0}"
GBRSM0157E	Error while retrieving rule elements for group: {0}
GBRSM0158E	Error creating refactoring change for "{0}"
GBRSM0159E	Unexpected error while retrieving content of the folder "{0}".
GBRSM0160E	Failed to move element: "{0}" to index: "{1}". Current folder elements: "{2}"
GBRSM0161E	Failed to move element: "{0}" to index: "{1}". Current package elements: "{2}"
GBRSM0162E	Error during dependency cycle scan
GBRSM0163E	Failed to save project references: "{0}"
GBRSM0164E	Failed to remove error marker on .project.
GBRSM0165E	WARNING: Duplicate ID for extension point "{0}","{1}"
GBRSM0166E	Unexpected model structure : the BOM Entry "{0}" has a vocabulary element that is null or that is a proxy for the locale "{1}"
GBRSM0167E	Impossible to read "{0}" : "{1}"
GBRSM0168E	Rule Project without BOM Path
GBRSM0169E	Project not associated with a rule project
GBRSM0170E	Could not flush plugin preferences on disk after modification; platform may have more chance later.
GBRSM0171E	Attempt made to register a Refactoring Participant which does not implement IlrRefactoringParticipant: "{0}"
GBRSM0172E	Failed to load BOM from: "{0}"
GBRSM0173E	Class "{0}" not found
GBRSM0174E	Class "{0}" not found
GBRSM0175E	Cannot read engine configuration file
GBRSM0176E	No model folder type is associated with the element: {0}.
GBRSM0177E	Failed to resolve rule project for URL: "{0}"
GBRSM0178E	Failed to resolve rule project for URL: "{0}"
GBRSM0179E	Failed to resolve XOM Path entry for URL: "{0}"

GBRSM0180E	Failed to resolve XOM Path entry for URL: "{0}"
GBRSM0181E	Cannot read engine configuration file
GBRSM0182E	The engine.conf file "{0}" has not been found.
GBRSM0183E	Null value for "{0}" attribute in extension "{1}". Extension ignored.
GBRSM0184E	BOMDomainValueProvider must specify a value for classname or property file
GBRSM0185E	Failed to instantiate BOMDomainValueProvider "{0}"
GBRSM0186E	Registered BOMDomainValueProvider "{0}" must implement IlrBOMDomainValueProvider
GBRSM0187I	Rule project reference cache cleared for project: "{0}"
GBRSM0188I	Referenced rule projects for project: "{0}" are: "{1}"
GBRSM0189I	Referencing projects for: "{0}" : "{1}"
GBRSM0190I	Failed to load build options settings. Using default values: "{0}"
GBRSM0191I	Full build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0192I	Full build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0193I	Full build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0194I	Incremental build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0195I	Incremental build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0196I	Incremental build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0197I	Auto build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0198I	Auto build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
GBRSM0199I	Auto build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.

Rule Designer Messages - Messages pertaining to the model (200-299)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSM0200I	Clean build for Rule Project "{0}" was interrupted after: {1} ms.
GBRSM0201I	Clean build for Rule Project "{0}" was canceled after: {1} ms.
GBRSM0202I	Clean build for Rule Project "{0}" completed in: {1} ms.
GBRSM0203I	Loading file : "{0}"
GBRSM0204I	Loading file : "{0}"
GBRSM0205I	Unloading resource : "{0}"
GBRSM0206E	Uncaught static analysis exception
GBRSM0208E	Unexpected exception in shared plugin
GBRSM0209E	Unexpected exception in shared plugin
GBRSM0210E	Unexpected exception in shared plugin
GBRSM0211E	Unexpected exception in shared plugin
GBRSM0212E	Unexpected exception in shared plugin
GBRSM0213E	Unexpected exception in shared plugin
GBRSM0214E	Unexpected exception in shared plugin
GBRSM0215E	Unexpected exception in shared plugin
GBRSM0216E	Unexpected exception in shared plugin
GBRSM0217E	Unexpected exception in shared plugin
GBRSM0218E	Unexpected exception in shared plugin
GBRSM0219E	Unexpected exception in shared plugin
GBRSM0220E	Unexpected exception in shared plugin
GBRSM0221E	Invalid namespace for package name conversion: {0} Using default package name instead.

GBRSM0222E	Failed loading L&F:
GBRSM0226E	An unexpected error occurred. See log file for details.
GBRSM0227E	An error occurred while loading the XOM file {0}. See log file for details.
GBRSM0228E	Could not load RulePackage "{0}". The associated .rulepackage file might be corrupted.
GBRSM0229E	Failed to initialize classpath variable {0} from {1}, which is null.
GBRSM0230E	An error occurred while loading the XML schema {0}. See log file for details.
GBRSM0231E	Unexpected exception when trying to build a URL for the location {0}
GBRSM0232E	Importing a XOM from a WSDL file was deprecated in JRules 6 and removed in JRules 7.
GBRSM0233E	Unmanaged XOM type error for dynamic XOM entry: {0}.
GBRSM0234E	Unable to find resource file: "{0}"
GBRSM0235E	Failed to locate entry "{0}" in bundle {1}
GBRSM0236E	Failed to convert URL for entry "{0}" in bundle {1} into a "file:" URL
GBRSM0237E	A file was deleted before the classloader could open it or a file could not be read.
GBRSM0238E	
GBRSM0239E	An unexpected error occurred while saving the Ruleflow "{0}". Please consult the Error Log for more details.
GBRSM0240E	An error occurred when saving the Ruleflow impacted by moving the Rule package "{0}" to "{1}".
GBRSM0241E	Multiple errors occurred when saving the Ruleflows impacted by moving the Rule package "{0}" to "{1}".
GBRSM0242E	An unexpected error occurred when moving the Rule package "{0}" to "{1}". Please consult the Error Log for more details.
GBRSM0243E	An unexpected error occurred while initializing the class "{0}" for the "ilog.rules.studio.model.rulePackageMovedListeners" extension point.
GBRSM0244E	Failed to read a local copy of the output file for resource "{0}".
GBRSM0245E	Illegal "archiveKind" value "{0}" in declaration of a rulesetArchiveExporter in plugin {1}. Ignoring this configuration element.
GBRSM0246E	Failed to convert project element "{0}" into IRL.
GBRSM0247E	Encountered multiple main ruleflows while extracting a ruleset archive: "{0}", "{1}". Check the "main ruleflow" property of the ruleflows or the extractor used for this ruleset archive export.
GBRSM0248E	Unexpected error while compiling ruleflow {0}.
GBRSM0249E	Unexpected error: the decision engine migration did not produce a BuildContext but did not report any error either.
GBRSM0250E	Unexpected error: expected error objects of type "{0}" but found a "{1}"
GBRSM0251E	Code converter raised some errors for rule "{0}"
GBRSM0252E	Unknown hierarchical property name "{0}"
GBRSM0253E	Unknown element "{0}" in hierarchical property "{1}"

GBRSM 0254E	Code converter raised some errors for function "{0}"
GBRSM 0255E	Unsupported direction kind "{0}" for ruleset parameter "{1}"
GBRSM 0256E	Found multiple rules with the same fully-qualified name while browsing project "{0}" : "{1}"
GBRSM 0257E	Found multiple declarations of hierarchical property "{0}". one with root "{1}" another with root "{2}"
GBRSM 0258E	Found multiple declarations of hierarchical rule property type for property "{0}": "{1}" and "{2}"
GBRSM 0259E	Found multiple declarations of rule property type for property "{0}": "{1}" and "{2}"
GBRSM 0260E	Found multiple variables with same name and package in project "{0}" : "{1}" in package "{2}"
GBRSM 0261E	An unexpected error occurred while retrieving the local file for the URI "{0}".
GBRSM 0262E	Unexpected ruletask algorithm: {0}
GBRSM 0263E	Failed to get an XML serialization of ruleflow element.
GBRSM 0264E	Failed to read the .xom file "{0}" for XSD XOMs in the output directory of the rule project.
GBRSM 0265E	Error while saving B2X migration file.
GBRSM 0266E	Project references cycle for the element "{0}"
GBRSM 0267E	Failed to read rule compiled unit from "{0}". It was probably compiled by a previous version of the product. Please force a rebuild of your projects before retrying.
GBRSM 0268E	No decision engine rule compiler found for rule type {0}.
GBRSM 0269E	Extension {0} : Unknown type of rule artifact "{1}". Ignoring it.
GBRSM 0270E	Extension {0} : Missing name of rule artifact type. Ignoring it.
GBRSM 0271E	Extension {0} : Missing class name of rule compiler. Ignoring it.
GBRSM 0272E	Extension {0} : Error when trying to instantiate rule compiler. Has this plugin been unregistered ? Discarding this extension.
GBRSM 0273E	No decision engine rule compiler found for artifact type {0}
GBRSM 0274E	Clean request failed for some projects.
GBRSM 0275E	Unexpected exception in model
GBRSM 0276E	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".
GBRSM 0277E	Unknown ruleset extractor "{0}" on project "{1}"
GBRSM 0278E	Found multiple ruleEnginePolicy plug-in extension. Expected at most one. Reverting to the default built-in one.
GBRSM 0279E	Failed to instantiate ruleEnginePolicy extension. Skipping.
GBRSM 0280E	Workspace error when creating error marker on project "{0}"
GBRSM 0281E	Failed to create the temporary Ant Build file "{0}".
GBRSM 0282E	Failed to execute the temporary Ant Build file "{0}".

GBRSM0283E	Failed to delete temporary file "{0}".
GBRSM0284E	Failed to parse the file "{0}" containing DVS classpath of required archives.
GBRSM0285E	An unexpected error occurred while compiling technical rule "{0}". Compilation failed with no error message.

Rule Designer Messages - Messages pertaining to tools

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRST0001E	Completeness analysis HTML rendering failed
GBRST0002E	Rule Analysis: Could not set job icon
GBRST0003E	Rule Analysis Error
GBRST0004E	Rule Analysis View: cannot set source folder for rule wizard using rule project : "{0}"
GBRST0005E	Rule Analysis View: Could not render body section
GBRST0006E	Data space coverage item rendering failed: no valid result data
GBRST0007E	Consistency checking item rendering failed: no valid result data
GBRST0008E	Error while rendering SA View results:
GBRST0009E	Rule Analysis Error: add view action failed
GBRST0010E	Unexpected exception in BOM update plugins
GBRST0011E	Unexpected exception in BOM update plugins
GBRST0012E	Unexpected exception in BOM update plugins
GBRST0013E	Unexpected exception in BOM update plugins
GBRST0014E	Unexpected exception in BOM update plugins
GBRST0015E	Unexpected exception in BOM update plugins
GBRST0016E	Unexpected exception in BOM update plugins
GBRST0017E	Unexpected exception in BOM update plugins
GBRST0018E	Unexpected exception in BOM update plugins
GBRST0019E	Unexpected exception in BOM update plugins
GBRST0020E	Unexpected exception in BOM update plugins
GBRST0021E	Failed to get initial value for classpath variable {0} from the Java system property {1}.
GBRST0022E	Failed to initialize classpath variable {0}

GBRST0024 E	Unexpected exception in studio editor plugins
GBRST0025 E	Unexpected exception in studio editor plugins
GBRST0026 E	Unexpected exception in studio editor plugins
GBRST0027 E	Unexpected exception in studio editor plugins
GBRST0028 E	Unexpected exception in studio editor plugins
GBRST0029 E	Unexpected exception in studio editor plugins
GBRST0030 E	Unexpected exception in studio editor plugins
GBRST0031 E	Unexpected exception in studio editor plugins
GBRST0032 E	Unexpected exception in studio editor plugins
GBRST0033 E	Unexpected exception in studio editor plugins
GBRST0034 E	Unexpected exception in studio editor plugins
GBRST0035 E	Unexpected exception in studio editor plugins
GBRST0036 E	Unexpected exception in studio editor plugins
GBRST0037 E	Unexpected exception in studio editor plugins
GBRST0038 E	Exception in "{0}".getNewFile(): "{1}"
GBRST0039 E	Exception in "{0}".getNewFile(): "{1}"
GBRST0040 E	Unexpected exception in function editor plugins
GBRST0041 E	Unexpected exception in function editor plugins
GBRST0042 E	Unexpected exception in function editor plugins
GBRST0043 E	Unexpected exception in function editor plugins
GBRST0044 E	Unexpected exception in rule editor plugins
GBRST0045 E	Unexpected exception in technical editor plugins
GBRST0046 E	Unexpected exception in technical editor plugins
GBRST0047 E	Unexpected exception in technical editor plugins
GBRST0048 E	Unexpected exception in technical editor plugins
GBRST0049 E	Unexpected exception in irl editor plugins
GBRST0050 E	Unexpected exception in irl editor plugins
GBRST0051 E	Unexpected exception in irl editor plugins
GBRST0052 E	Unexpected exception in irl editor plugins

GBRST0053 E	Unexpected exception in irl editor plugins
GBRST0054 E	Unexpected exception in irl editor plugins
GBRST0055 E	Unexpected exception in irl editor plugins
GBRST0056 E	Unexpected exception in irl editor plugins
GBRST0103 E	Caught exception launching BIRT Web Viewer:
GBRST0104 E	Exception in "{0}".getNewFile(): "{1}"
GBRST0106 E	Background task searching for the "IRL keyword" topic in documentation failed.
GBRST0154 E	An exception has occurred during migration prerequisites check: {0}
GBRST0155 E	An exception has occurred during migration action creation: {0}
GBRST0159 E	An exception has occurred during action {0}: {1}
GBRST0165 E	An exception has occurred during migration perform action: {0}
GBRST0166 E	An exception has occurred while persisting project "{0}"
GBRST0167 E	The element is not an instance of IlrResourceElement
GBRST0168 E	The project is not an instance of IlrRuleProject
GBRST0169 E	An exception has occurred while persisting element "{0}"
GBRST0176 E	The project "{0}" referenced by "{1}" does not exist or is closed.
GBRST0177 E	The project "{0}" referenced by "{1}" is not in sync with the file system.
GBRST0178 E	The project "{0}" is in read-only mode.
GBRST0179 E	The project "{0}" has no source folder.
GBRST0180 E	The project "{0}" source folder does not exist.
GBRST0181 E	The project "{0}" source folder is in read-only mode.
GBRST0184 E	An exception "{0}" occurred during ruleset archive refresh.
GBRST0191 E	An exception occurred during RuleApp archive creation.
GBRST0193 E	An exception "{0}" occurred during RuleApp archive refresh.
GBRST0200 E	An exception has occurred during the Deployment Configuration checking.
GBRST0203 E	An exception has occurred during the build of a RuleApp.
GBRST0220 E	Exception while retrieving attribute from launch configuration
GBRST0221 E	A value must be specified
GBRST0222 E	Exception while retrieving mapped resources from launch configuration

GBRST0230E	No server definition found in the imported file: "{0}"
GBRST0231E	An unexpected error occurred while importing the server list from file "{0}".
GBRST0232E	An unexpected error occurred while exporting the server list to the file "{0}".
GBRST0240E	An unexpected error occurred while retrieving the credentials for target "{0}".
GBRST0241E	An unexpected error occurred while retrieving the credentials for target "{0}".
GBRST0242E	An unexpected error occurred while setting the credentials for target "{0}".
GBRST0243E	An unexpected error occurred while checking the connections to the selected targets
GBRST0250E	An unexpected error occurred during the inspection of the class path of Java project "{0}".
GBRST0251E	An unexpected error occurred while adding the entry "{0}" to the exported JAR file "{1}".
GBRST0252E	An unexpected error occurred while adding the output directory of the Java project "{0}" to its exported JAR file.
GBRST0253E	An unexpected error occurred while adding the entry "{0}" to the root of its exported ZIP file.
GBRST0254E	An unexpected error occurred while adding the entry "{0}" to the path "{1}" of its exported ZIP file.
GBRST0255E	An unexpected error occurred when creating the exported ZIP file "{0}" for the class folder "{1}".
GBRST0256E	An unexpected error occurred when creating an instance of class for the extension point "{0}".
GBRST0257E	The JAR export operation was interrupted.
GBRST0258E	An unexpected error occurred when running the JAR export operation.
GBRST0259E	An unexpected error occurred while updating the XOM URIs of rule project "{0}".
GBRST0260E	An unexpected error occurred while retrieving the first existing directory for the location "{0}".
GBRST0261E	An unexpected error occurred while checking the location "{0}" for write access.
GBRST0262E	An unexpected error occurred while checking the location "{0}": The value does not actually refer to an existing file.
GBRST0270E	An unexpected error occurred while creating the launch configuration for the decision operation "{0}" ({1}).
GBRST0271E	An unexpected error occurred while retrieving existing decision operation launch configurations.
GBRST0272E	Classic rule project "{0}" cannot reference decision service rule project "{1}".
GBRST0273E	Decision service rule project "{0}" cannot reference classic rule project "{1}".
GBRST0274E	A standard rule project cannot reference the main rule project "{0}".
GBRST0275E	An unexpected exception occurred while retrieving the message associated with the key "{0}" in bundle "{1}".
GBRST0276E	An unexpected exception occurred while initializing the feature "{0}".
GBRST0277E	An unexpected exception occurred while loading the server list from the preferences of the plug-in "{0}".
GBRST0278E	An unexpected exception occurred while saving the server list to the preferences of the plug-in "{0}".

GBRST0279 E	An unexpected exception occurred while saving the server list to the preferences of the plug-in "{0}".
GBRST0280 E	An unexpected exception occurred while retrieving the message associated with the key "{0}" in bundle "{1}".
GBRST0281 E	An unexpected error occurred while validating rule project "{0}" that is referenced by decision operation "{1}".
GBRST0282 E	An unexpected exception occurred while generating a RuleApp client ({0})
GBRST0283 E	An unexpected error occurred while setting the encoding type "{0}" to an instance of the class "{1}".
GBRST0284 E	An unexpected error occurred while parsing the command line.
GBRST0285 E	The following Rule Execution Server definitions were removed or overwritten: "{0}".
GBRST0286 E	An exception has occurred while retrieving the Resource URIs defined in Rule project "{0}" for target "{1}": {2}.
GBRST0287 E	An unexpected error occurred while running the Rule Designer headless deployment tool.
GBRST0288 E	An unexpected exception occurred while loading server definition file specified by the "{0}" option.
GBRST0289 E	Initialization of plug-in extension failed: {0}.
GBRST0290 E	An unexpected exception occurred while retrieving the rulesets for RuleApp {0} version {1}: {2}
GBRST0291 E	An unexpected exception occurred while retrieving the versions for RuleApp {0}: {1}
GBRST0292 E	An exception occured while connecting to a remote server "{0}"
GBRST0293 E	ARL view : failed to get local modification timestamp for element "{0}". Resource exists: "{1}"
GBRST0294 E	Exception raised while deserializing SemRule for element "{0}"
GBRST0295 E	Server list file contains multiple servers with the same name

Rule Designer Messages - Messages pertaining to core user interface

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
GBRSU0001E	Unexpected exception in studio ui plugins
GBRSU0002E	Unexpected exception in studio ui plugins
GBRSU0003E	Unexpected exception in studio ui plugins
GBRSU0004E	Unexpected exception in studio ui plugins
GBRSU0005E	Unexpected exception in studio ui plugins
GBRSU0006E	Unexpected exception in studio ui plugins
GBRSU0007E	Unexpected exception in studio ui plugins
GBRSU0008E	Unexpected exception in studio ui plugins
GBRSU0009E	Unexpected exception in studio ui plugins
GBRSU0010E	Unexpected exception in studio ui plugins
GBRSU0011E	Unexpected exception in studio ui plugins
GBRSU0012E	Fail to load template "{0}"
GBRSU0013E	Unexpected exception in studio ui plugins
GBRSU0014E	Unexpected exception in studio ui plugins
GBRSU0015E	Unexpected exception in studio ui plugins
GBRSU0016E	Unexpected exception in studio ui plugins
GBRSU0017E	Unexpected exception in studio ui plugins
GBRSU0018E	Unexpected exception in studio ui plugins
GBRSU0019E	Unexpected exception in studio ui plugins
GBRSU0020E	Unexpected exception in studio ui plugins
GBRSU0021E	Unexpected exception in studio ui plugins

GBRSU0022E	Unexpected exception in studio ui plugins
GBRSU0023E	Unexpected exception in studio ui plugins
GBRSU0024E	Unexpected exception in studio ui plugins
GBRSU0025E	Unexpected exception in studio ui plugins
GBRSU0026E	Unexpected exception in studio ui plugins
GBRSU0027E	Unexpected exception in studio ui plugins
GBRSU0028E	Unexpected exception in studio ui plugins
GBRSU0029E	Unexpected exception in studio ui plugins
GBRSU0030E	Unexpected exception in studio ui plugins
GBRSU0031E	Unexpected exception in studio ui plugins
GBRSU0032E	Unexpected exception in studio ui plugins
GBRSU0033E	Unexpected exception in studio ui plugins
GBRSU0034E	Unexpected exception in studio ui plugins
GBRSU0035E	Unexpected exception in studio ui plugins
GBRSU0036E	Unexpected exception in studio ui plugins
GBRSU0037E	Unexpected exception in studio ui plugins
GBRSU0038E	Unexpected exception in studio ui plugins
GBRSU0039E	Unexpected exception in studio ui plugins
GBRSU0040E	Unexpected exception in studio ui plugins
GBRSU0041E	Unexpected exception in studio ui plugins
GBRSU0042E	Unexpected exception in studio ui plugins
GBRSU0043E	Unexpected exception in studio ui plugins
GBRSU0044E	Unexpected exception in studio ui plugins
GBRSU0045E	Unexpected exception in studio ui plugins
GBRSU0046E	Unexpected exception in studio ui plugins
GBRSU0047E	Unexpected exception in studio ui plugins
GBRSU0048E	Unexpected exception in studio ui plugins
GBRSU0049E	Unexpected exception in studio ui plugins
GBRSU0050E	Unexpected exception in studio ui plugins

GBRSU0051 E	Unexpected exception in studio ui plugins
GBRSU0052 E	Unexpected exception in studio ui plugins
GBRSU0053 E	Unexpected exception in studio ui plugins
GBRSU0054 E	Unexpected exception in studio ui plugins
GBRSU0055 E	Unexpected exception in studio ui plugins
GBRSU0056 E	Error while creating parameters viewer
GBRSU0057 E	Unexpected exception in studio ui plugins
GBRSU0058 E	Unexpected exception in studio ui plugins
GBRSU0059 E	Unexpected exception in studio ui plugins
GBRSU0060 E	Unexpected exception in studio ui plugins
GBRSU0062 E	Unexpected exception in studio ui plugins
GBRSU0063 E	Unexpected exception in studio ui plugins
GBRSU0064 E	Unexpected exception in studio ui plugins
GBRSU0065 E	Unexpected exception in studio ui plugins
GBRSU0066 E	Unexpected exception in studio ui plugins
GBRSU0067 E	Unexpected exception in studio ui plugins
GBRSU0068 E	Unexpected exception in studio ui plugins
GBRSU0069 E	Unexpected exception in studio ui plugins
GBRSU0070 E	Unexpected exception in studio ui plugins
GBRSU0071 E	Unexpected exception in studio ui plugins
GBRSU0072 E	Unexpected exception in studio ui plugins
GBRSU0073 E	Unexpected exception in studio ui plugins
GBRSU0074 E	Unexpected exception in studio ui plugins
GBRSU0075 E	Unexpected exception in studio ui plugins
GBRSU0076 E	WARNING: Duplicate id for extension-point :
GBRSU0077 E	The default classpath cannot be computed. Check the Rule Designer variable.
GBRSU0078 E	WARNING: Duplicate id for extension-point :
GBRSU0079 E	WARNING: Duplicate ID for extension point {0}.{1}.
GBRSU0082 E	WARNING: Duplicate ID for extension point {0}.{1}.

GBRSU0083 E	WARNING: Duplicate id for extension-point :
GBRSU0084 E	WARNING: Duplicate id for extension-point :
GBRSU0085 E	WARNING: Duplicate id for extension-point :
GBRSU0086 E	Cannot read engine configuration file
GBRSU0087 E	WARNING: Duplicate id for extension-point :
GBRSU0088 E	WARNING: Duplicate id for extension-point :
GBRSU0089 E	Error committing refactoring for "{0}"
GBRSU0090 E	Couldn't flush plugin preferences on disk after modification. Platform may have more chance later.
GBRSU0091 E	Exception in "{0}".performCopy(): "{1}"
GBRSU0092 E	Unexpected exception in studio ui opensource plugins
GBRSU0093 E	Unexpected exception in studio ui opensource plugins
GBRSU0094 E	WARNING: Duplicate id for extension-point org.eclipse.jdt.ui.javaElementFilters
GBRSU0100 E	Exception in "{0}".getNewFile(): "{1}"
GBRSU0101 E	An internal error has occured. See log file for details.
GBRSU0102 E	A problem occured while opening "{0}".
GBRSU0103 E	An error has occurred during the execution of the command {0}. See log file for details.
GBRSU0104 E	An error occured while creating an URL for "{0}". See log file for details.
GBRSU0105 E	An error occured while attaching the source {0}. See log file for details.
GBRSU0106 E	An unexpected error occurred while persisting the remote help preferences.
GBRSU0107 E	An unexpected error occurred while accessing the remote help properties file.
GBRSU0108 E	The IBM_DS_HOME classpath variable is not defined. ODM Remote Help infocenter preferences will not be defined.
GBRSU0109 E	The infocenter.properties file was not found at the expected location {0}.
GBRSU0110 E	An exception occured while trying to access members of an IContainer {0}
GBRSU0111 E	An unexpected error occurred while updating UUIDs.
GBRSU0112 E	The UUIDs update operation was interrupted by the user.
GBRSU0113 E	An unexpected error occurred while loading the dynamic XOM path entry {0} or one of its dependencies.

Rule Designer Messages - Messages pertaining to testing and simulation

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
G BR TE 00 01 E	An error has been detected. Contact your Administrator as this error typically occurs when trying to retrieve the signature of the ruleset. The XOM might not be correctly deployed to the SSP or to Rule Execution Server. Check the error trace for more detailed information.
G BR TE 00 02 E	The simulation part definition contains an error. A scenario suite part is defined with a first index {0} and last index {1}. The first index must be a positive value and the last index must be greater than or equal to the first index. Contact your Administrator to solve the issue with the scenario provider implementation.
G BR TE 00 03 E	The simulation part definition contains an error. A balanced list of scenario part is defined using a total number of scenario of {0} and a number of parts of {1}. The total number of scenarios must be greater than 0 and the number of parts must be less than or equal to the total number of scenarios. Contact your Administrator to solve the issue with the scenario provider implementation.
G BR TE 00 04 E	SSP has stopped the execution of the scenario suite because no resources were available on the server. Contact your Administrator to add more resources to the work managers used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
G BR TE 00 05 E	An error has been detected in one of the work managers used by the SSP. Contact your Administrator to fix the problem.
G BR TE 00 06 E	The execution of the scenario suite has been stopped by the SSP, because there were no resources available on the server. Contact your Administrator to add more resources to the thread pool used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
G BR TE 00 07 I	The resource allocation policy manager has rejected the execution request.
G BR TE 00 08 I	The execution request has been accepted by the resource allocation policy manager. The resource allocation policy for this job is: Execution Priority {0}, Maximum number of threads to use for parallel execution {1}.
G BR TE	Decision Engine is not supported for the supplied scenario provider "{0}"

0009E	
G BR TE 02 01 E	Cannot convert string value {0} to an instance of {1}
G BR TE 02 02 E	Cannot parse XML string {0}
G BR TE 02 03 E	Cannot compute equality. There are too many possible matching values because too many expected objects have no value provided.
G BR TE 02 04 E	Cannot cast object {0} to an instance of {1}
G BR TE 02 05 E	Cannot parse the Gregorian calendar {0}. The <time> XML element is missing.
G BR TE 02 06 E	Cannot compare complex object {0} directly. This object should be compared field by field.
G BR TE 02 07 E	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
G BR TE 02 08 W	No converter defined for type: "{0}". The toString() method is used to compare the expected value "{1}" with the observed value "{2}".
G BR TE 02 09 W	The equals method is used to compare object {0}.
G BR TE 02 10 W	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
G BR TE 02 11 W	Ensure that the equals method is correctly implemented for this object.
G	Cannot compare object {0} using BOM serialization because all the corresponding expected fields are empty in the

BR TE 02 12 E	Excel scenario file.
G BR TE 02 13 W	Object {0} is not compared because all the corresponding expected fields are empty in the Excel scenario file.
G BR TE 02 14 E	Cannot serialize object {0}.
G BR TE 06 01 I	GET request handled...
G BR TE 06 02 I	Logout requested, invalidating the session
G BR TE 06 03 I	PING requested
G BR TE 06 04 I	POST request handled, deserializing the request object...
G BR TE 06 05 I	ServiceCall deserialized from input stream, starting the service...
G BR TE 06 06 I	Service method called
G BR TE 06 07 I	Writing result {0} to the response stream
G BR TE 06 08 I	OK, result sent, response stream flushed
G BR TE 06 09	SSP: No value defined in web.xml for parameter {0}: using default value {1}

W	
G BR TE 06 10 I	SSP: Using {0} rule sessions
G BR TE 06 11 E	SSP: Unable to initialize Trace DAO
G BR TE 06 12 I	SSP: Successfully initialized the trace DAO factory: {0}
G BR TE 06 13 E	SSP: Unable to instantiate job results store from class: {0}. The in-memory store is used instead.
G BR TE 06 14 I	SSP: Using the job results store: {0} instead.
G BR TE 06 15 W	Value of parameter {0} in web.xml is not a valid integer. Using default value {1}
G BR TE 06 16 I	SSP: Using {0} as the custom policy manager for allocation of resources
G BR TE 06 17 W	SSP: An exception occurred while creating a new instance of the custom policy manager {0} for allocation of scenario suite resources. The default allocation policy manager is used instead.
G BR TE 06 18 I	SSP: The default policy manager for resource allocation is used with a setting of {0} for the maximum number of threads per parallel simulation.
G BR TE 06 19 I	SSP pool size: {0}
G BR TE 06 20 I	SSP: High-priority work manager name : {0}
G BR TE	SSP: Normal-priority work manager name : {0}

06 21 J	
G BR TE 06 22 J	SSP: Low-priority work manager name : {0}
G BR TE 06 23 E	SSP: Cannot retrieve the high-priority work manager {0} in the JNDI directory
G BR TE 06 24 E	SSP: Cannot retrieve the normal-priority work manager {0} in the JNDI directory
G BR TE 06 25 E	SSP: Cannot retrieve the low-priority work manager {0} in the JNDI directory
G BR TG 00 01 E	Cannot find sheet {0} in Excel scenario file.
G BR TG 00 02 E	Index out of bounds: scenario {0} requested, while valid index are from {1} to {2} for the scenario file.
G BR TG 00 03 E	Class {0}, referenced in the headers of sheet {1}, is not defined in the business object model.
G BR TG 00 04 E	Missing property {0} that defines the name of the sheet for input objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
G BR TG 00 05 E	Object "{0}" referenced in cell {1} of sheet {2} cannot be found in sheet {3}.
G BR TG 00 06 E	Missing property {0} that defines the name of the sheet for scenarios. Please define the missing property in the Custom tab of the Excel file properties dialog.
G BR TG 00 07 E	Missing property {0} that defines the name of the sheet for output objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
G	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results

G BR TG 00 08 E	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results.
G BR TG 00 09 E	Incorrect direction ({0}) defined for parameter {1} in property {2} of the Excel sheet {3}. The list of correct directions is: {4}.
G BR TG 00 10 E	The Excel scenario provider does not support rulesets that define no input parameters.
G BR TG 00 11 E	Missing property {0} that defines the locale of the scenario file. Please define the missing property in the Custom tab of the Excel file properties dialog.
G BR TG 00 12 E	{0} is not a valid name for the scenario description column: there is an input parameter defined with the same name. Please choose a different name for the scenario description column.
G BR TG 00 13 E	Header rows are missing in the sheet {0}.
G BR TG 00 14 E	Invalid value {0} for property {1} in the Excel sheet {2} to define the type of parameter {3}.
G BR TG 00 15 E	Invalid value {0} for property {1} in the Custom tab of the Excel file properties dialog to define the ruleset output (direction OUT) parameters. Valid values are comma separated lists of PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION items.
G BR TG 00 16 E	Invalid value {0} in the headers of column {1} in the Excel sheet {2}.
G BR TG 00 17 E	Mandatory value missing in cell {1} of sheet {0}.
G BR TG 00 18 E	The BOM of the ruleset to test is not suitable for generating an Excel Scenario File Template. Please check additional messages.
G BR TG 00 19	Ruleset input parameter {0}, which is an array, is not supported in flat mode.

E	
G BR TG 00 20 E	The data for your scenarios is not suitable for generating a flat Excel Scenario File Template. However, you may use it to generate a Tabbed Excel Scenario File Template. Please talk to your developer for assistance if this format is not currently available.
G BR TG 00 21 E	The business object model of the ruleset to test is not suitable for generating a flat Excel Scenario File Template. However, you can use it to generate a Tabbed Excel Scenario File Template. Please check additional messages.
G BR TG 00 22 E	Invalid value {0} in cell {1} of the Excel sheet {2}.
G BR TG 00 23 E	Unable to create XML reader.
G BR TG 00 24 E	The provided data must be a String representation of the XML when using BOM model type.
G BR TG 00 25 E	The provided data must be a String representation of the XML when using XML model type.
G BR TG 00 26 E	Unable to translate XML into BOM XML.
G BR TG 00 27 E	Unable to find parameter in the signature.
G BR TG 00 28 E	Ruleset Path not found.
G BR TG 00 29 E	Unexpected exception during the BOM serialization.
G BR TG 00 30 E	Unable to read XML.
G BR TG	BOM type not found. Create a special converter for this type: "{0}".

0031E	
G BR TG 0032E	Unknown property: "{0}".
G BR TG 0033E	Unable to find conversion class for type "{0}": "{1}".
G BR TG 0034E	SecurityException on the constructor for conversion class for type "{0}": "{1}"
G BR TG 0035E	No default public constructor for conversion class for type "{0}": "{1}".
G BR TG 0036E	Exception during the construction of conversion class for type "{0}": "{1}".
G BR TG 0037E	The conversion class for type "{0}" does not implement IlrBOM2DVSCConverter interface: "{1}"
G BR TG 0038I	Additional converter loaded for type: "{0}".
G BR TG 0039E	Unable to read inline BOM type "{0}".
G BR TG 0040E	Missing default constructor for type "{0}".
G BR TG 0041E	Unsupported BOM type "{0}".
G BR TG 0042E	Field "{0}" in class "{1}" not found.

G BR TG 00 43 E	Ruleset input parameter {0}, which is a collection, is not supported in flat mode.
G BR TG 00 44 E	It is impossible to create scenario data because of cross references between the following objects defined in the scenario: {0}.
G BR TG 00 45 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a byte (integer between -128 and 127) was expected.
G BR TG 00 46 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a short (integer between -32,768 and 32,767) was expected.
G BR TG 00 47 E	Invalid value {0} in cell {1} of the Excel sheet {2}: an integer (between -2,147,483,648 and 2,147,483,647) was expected.
G BR TG 00 48 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a long (integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807) was expected.
G BR TG 00 49 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a single character was expected.
G BR TG 00 50 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a boolean value (TRUE or FALSE) was expected.
G BR TG 00 51 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a double (decimal) was expected.
G BR TG 00 52 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a float (decimal) was expected.
G BR TG 00 53 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a date was expected.
G BR TG 00	Too many domains are defined for the input data, they cannot all be stored in the scenario suite.

54 E	
G BR TG 00 55 E	Only domains with less than {0} entries can be stored in the Excel file.
G BR TG 00 56 W	BOM type not found. Create a special converter for this type: "{0}".
G BR TG 00 57 W	Unsupported business object model type "{0}".
G BR TG 00 58 E	Cannot read the Excel file: Try to open the file in Microsoft Excel. If you can open the file in Microsoft Excel but you still receive this error, please contact your Administrator.
G BR TG 00 59 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigDecimal (decimal) was expected.
G BR TG 00 60 E	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigInteger (integer) was expected.
G BR TG 00 61 E	The default Excel format is not compatible with the ruleset parameter "{1}" of type {2}: it cannot be instantiated because of a cross-reference issue that involves an attribute of type {0}.
G BR TG 00 62 E	An unexpected error occurred while checking whether the Excel Flat format is compatible with the ruleset parameter "{0}" of type {1}.
G BR TG 00 63 E	An unexpected error occurred while invoking method "{0}" on instance "{1}".
G BR TG 02 01 E	Exception {0} raised during the generation of a test rule: {1}.
G BR TG 02 02 W	It was not possible to find a navigation phrase in the vocabulary for member {0} in BOM class {1}: a default verbalization is used for the Excel Scenario File.
G	You have reached the maximum number of columns allowed in an Excel sheet.

BR TR 00 01 E	
G BR TR 00 02 E	Exception thrown during the business object model serialization process of the execution trace: {0}.
G BR TV 00 01 E	{0}: the construction of {1} instances is not supported by the Excel Scenario Provider for testing and simulation.
G BR TV 00 02 E	{0}: cannot find the type {1} in the business object model.
G BR TV 00 03 E	{0}: cannot find a testing and simulation constructor for the business object model class. If there are no constructors, create one. If there are several constructors in the business object model class, you must specify a constructor by selecting the "Testing and simulation constructor" option in the business object model editor.
G BR TV 00 04 E	{0}: It is not possible to create instances of the business object model class. There is a cross-reference problem with the testing and simulation constructor, because at least one argument references the business object model class directly or indirectly. Check the arguments of the testing and simulation constructor.
G BR TV 00 05 E	{0}: there are no constructor arguments or attributes that enable the creation of instances of the business object model class. Select a testing and simulation constructor with arguments, or create attributes that are nonstatic and writable.
G BR TV 00 06 W	{0}: the arguments of the constructor have generic names (arg1, arg2, etc.). Rename the arguments by reusing the names of the corresponding attributes or with your own meaningful names.
G BR TV 00 07 E	{0}: the Excel Format does not support multidimensional arrays.
G BR TV 00 08 E	{0}: cannot define the contents of the collection. Add a domain to the collection.
G BR TV 00 09 W	{0}: cannot use the default Excel format because of a cross-reference problem with an attribute. You can use the Excel Tabbed Format.
G BR TV 00 10	{0}: input parameters that are collections are not supported because the domain of the collection cannot be defined. Replace the input parameter with JavaBeans that hold the collection in a single attribute, and define the collection domain of this attribute.

E	
G BR TV 00 11 E	There are no input parameters. Add at least one ruleset parameter of direction IN or IN_OUT.
G BR TV 00 12 W	Generic collections as output ruleset parameters are not supported.
G BR TV 00 13 W	Generic collections as attributes of a BOM class are not supported.
G BR TV 00 14 W	No supported child attribute found for this ruleset parameter.
G BR TV 00 15 W	No supported child attribute found for this attribute.
G BR TV 00 16 W	Multidimensional arrays as ruleset parameters are not supported.
G BR TV 00 17 W	Multidimensionals arrays ruleset parameter attributes are not supported.
G BR TV 00 18 W	Ruleset parameters of type "{0}" are not supported.
G BR TV 00 19 W	Ruleset parameter attributes of type "{0}" are not supported.
G BR TV 00 21 E	{0}: argument "{1}" of the constructor does not correspond to any attribute in the {2} class or its superclasses. Rename the arguments by reusing the names of the corresponding attributes or create virtual attributes that correspond to the arguments in the business object model.
G BR TV 00 22 W	{0}: is not ignored because it is not optional.
G BR TV	Engine type "{0}" is not supported.

0023E	
G BR TV 00 24 E	{0}: cannot create an instance for BOM class {1} because it is an abstract class. Use a concrete class instead of an abstract class.
G BR TX 00 01 E	Column index is out of bounds. The allowable column range is [0..{0}].
G BR TX 00 02 E	Row index is out of bounds. The allowable row range is [0..{0}].
G BR TX 00 03 E	Could not create a Sheet in a readable Excel file.
G BR TX 00 04 E	Could not retrieve the drawing patriarch for the Excel Sheet.
G BR TX 00 05 E	Unable to deserialize this BOM type descriptor: {0}.
G BR TX 00 05 W	Cell at [{0},{1}] is unexpectedly out of range.
G BR TX 00 06 E	Unable to create a new XOM array instance for this BOM type: {0}.
G BR TX 00 06 W	Cell at [{0},{1}] is unexpectedly within range.
G BR TX 00 07 E	Unable to retrieve the XOM class mapping for this BOM type: {0}.
G BR TX 00 08 E	Unable to create the object instance for this BOM type: {0}.
G	Unable to cast the value at index {0} in the Input Record into {1}.

BR TX 00 09 E	
G BR TX 00 10 E	The Input Record does not contain a value at index: {0}.
G BR TX 00 11 E	BOM support has not been enabled or the BOM is missing for the ruleset {0}.
G BR TX 00 12 E	Unable to retrieve the type of the array elements for this BOM type: {0}.
G BR TX 00 13 E	Unable to determine whether the BOM type {0} is an array.
G BR TX 00 14 E	Unable to determine whether the BOM type {0} is a collection.
G BR TX 00 15 E	Unable to determine whether the BOM type {0} is a collection with a single factory parameter.
G BR TX 00 16 E	Unable to retrieve the BOM class that declares the field {0} of the BOM type {1}.
G BR TX 00 17 E	Unable to retrieve the ruleset archive from Rule Execution Server.
G BR TX 00 18 E	Unable to parse the XML descriptor of the ruleset signature.
G BR TX 00 19 E	Unable to extract the XML descriptor of the ruleset signature from the ruleset archive.
G BR TX 00 20	Unable to serialize the ruleset parameter {0}.

E	
G BR TX 00 21 E	Unable to extract the BOM from the ruleset archive.
G BR TX 00 22 E	The BOM type for the ruleset parameter {0} is missing.
G BR TX 00 23 E	Unable to retrieve the BusinessDataIEService instance for ruleset {0}.
G BR TX 00 24 E	Unable to convert BOM instance into XOM instance.
G BR TX 00 25 E	The BOM is missing for ruleset {0}.
G BR TX 00 26 E	Unable to transform the input map into a compliant map.
G BR TX 02 01 E	Cannot create a ruleset parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
G BR TX 02 02 E	Cannot extract the ruleset parameter name from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION with a string that is not empty after it has been trimmed.
G BR TX 02 03 E	Cannot extract the ruleset parameter type from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
G BR TX 02 04 E	Cannot extract the ruleset parameter direction from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
G BR TX 02 05 E	Cannot create an object factory parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
G BR	Cannot extract the name of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL with a string that is not empty after it has been

TX 02 06 E	trimmed.
G BR TX 02 07 E	Cannot extract the type of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
G BR TX 02 08 E	Cannot extract the direction of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
G BR TX 02 09 E	Cannot find BOM file <{0}> in the class path.
G BR TX 02 10 E	The created object instance for class {0} must not be null.
G BR TX 02 11 E	Value "{0}" is not valid for class {1}.
G BR TX 02 12 E	Mandatory parameter "{0}" is missing for building an instance of {1}.
G BR TX 02 13 E	There is no attribute "{0}" in the BOM class {1}.
G BR TX 02 14 E	Maps are not supported.
G BR TX 02 15 E	Value of mandatory parameter {0} for class {1} must be a string.
G BR TX 02 16 E	The object factory signature for class {0} already contains a parameter that is named {1}.
G BR TX 02 17 E	Class {0} is not considered a simple type.

G BR TX 02 18 E	The node kind for class {0} must not be null.
G BR TX 02 19 E	Cannot convert date {0} to a string format for class {1}.
G BR TX 02 20 E	The object factory signature for class {0} must contain only one parameter.
G BR TX 02 21 E	Cannot convert an instance of BOM class {0} into a XOM instance.
G BR TX 02 22 E	Cannot create a node instance for the XMLGregorianCalendar class.
G BR TX 02 23 E	The argument "{0}" of the constructor does not correspond to any attribute in the {1} class or its super classes. Rename the arguments by reusing the names of the corresponding attributes, or create virtual attributes corresponding to the arguments in the Business Object Model.
G BR TX 02 24 E	Cannot find class {0} in the business object model.
G BR TX 02 25 E	Cannot find the default constructor for class {0} in the business object model. If there are no constructors, create one. If there are several constructors in the class, you must specify a constructor by selecting the "DVS constructor" option in the business object model editor.
G BR TX 02 26 E	Cannot get metadata for object {0}.
G BR TX 02 27 E	Cannot get array type for object {0}.
G BR TX 02 28 E	Cannot get object value for object {0}.
G BR TX	Cannot create a node instance for the Calendar class using value {0}.

02 29 E	
G BR TX 02 30 E	{0} does not represent a valid Calendar value.
G BR TX 02 31 E	Cannot create an instance for BOM class {0} because it is an abstract class. Use a concrete class instead of an abstract class.

Decision Center reference

Refer to these topics for public API and messages.

[Decision Center REST API](#)

You can use the Decision Center REST API to build, test, and deploy decision services.

[Decision Center modeling language reference](#)

This section lists the business rule language elements and syntax that are available for decision modeling.

[Messages](#)

Each listed message provides a detailed description about the error message and some steps to try and resolve the error.

Decision Center REST API

You can use the Decision Center REST API to build, test, and deploy decision services.

[Overview](#)

Decision Center exposes a REST API that you can use to build, test, and deploy decision services. With this REST API, you can easily set up and enforce a continuous deployment process by using the programming language of your choice.

[REST API](#)

Explore, build, test and deploy decision services stored in Decision Center.

Parent topic: [Decision Center reference](#)

Overview

Decision Center exposes a REST API that you can use to build, test, and deploy decision services. With this REST API, you can easily set up and enforce a continuous deployment process by using the programming language of your choice.

Any user with the relevant permissions can use the end points provided by the REST API to do the following actions:

- Retrieve the decision services of your repository, their branches, deployment configurations, and test suites.
- Retrieve the list of servers available.
- Build, download, or deploy a RuleApp for a deployment configuration.
- Run a test suite.
- Import and export decision services.

For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api/v1/decisionservices --user username@company_name.com:password
```

This command returns a JSON object such as:

```
{
  "elements": [
    {
      "id": "77072920-2ec3-11db-bb3d-a34db576b043",
      "internalId": "brm.RuleProject:1:1",
      "name": "miniloan-rules",
      "buildMode": "DecisionEngine"
    },
    {
      "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
      "internalId": "brm.RuleProject:2:2",
      "name": "AutoQuote",
      "buildMode": "DecisionEngine"
    }
  ],
  "totalCount": 2,
  "number": 0,
  "size": 2
}
```


Accessing the REST API tool

You can access the Swagger user interface at <https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api>. This interface exposes a view of the available endpoints, their documentation, and a **Try it out** button to test each endpoint.

Authentication

All endpoints, except /about, require authentication. You can authenticate by using basic authentication, with your Decision Center user name and password. For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api/v1/decisionservices --user username@company_name.com:password
```

Note: In the Swagger user interface, you might see a  warning icon in some of the REST API methods. When you click this warning, a window opens asking for your login credentials. If you are authenticated to Operational Decision Manager on Cloud portal, you do not need to authenticate again, and you can disregard this warning.

As an alternative, you can use service credentials to authenticate to the application. For more information, see [Service credentials for client applications](#).

Attention: If you change your Operational Decision Manager on Cloud user role, or create a new set of service credentials, the Decision Center REST API is temporarily unavailable, and becomes available again after a maximum of 5 minutes.

Errors

Conventional HTTP response codes are used to indicate the success or failure of an API request. When an error occurs, the body of the HTTP response always contains a JSON error object with an error code, a reason, and a reference. For example:

```
{
  "error": "IlrConnectException",
  "reason": "Could not look up datasource named 'mydatasource'",
  "status": "BAD_REQUEST",
  "details": [
    "Could not look up datasource named 'mydatasource'",
    "Name [mydatasource] is not bound in this Context. Unable to find [mydatasource].",
  ]
}
```

Filtering and pagination

Explore methods that return collections of objects, such as `/decisioncenter-api/v1/decisionservices`, can be paginated. By default, all elements of the collection are returned, but you can specify a page size and a page number to retrieve only a subset of the collection. For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-
api/v1/decisionservices?page=2&size=20 --user
username@company_name.com:password
```

You can also filter returned objects by their property, by using the `q` parameter:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-
api/v1/decisionservices?q=name:AutoQuote --user
username@company_name.com:password
```

Parent topic: [Decision Center REST API](#)

Decision Center modeling language reference

This section lists the business rule language elements and syntax that are available for decision modeling.

You use constructs, operators, and literals to write rules. You use different language constructs depending on whether you are writing the definitions, conditions, or actions part of a rule. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

Definition part

The definition part starts with the `definitions` keyword and introduces local variables that you can use in the rest of the rule.

Condition part

The condition part of the rule starts with the `if` keyword.

Action part

The action part describes the actions that the rule completes when the conditions are met. This part might start with the `then` keyword and can contain an `else` construct.

Operators

You use operators in rule statements to perform arithmetic operations, associate or negate conditions, and compare expressions.

Literals

The business rule language supports number, string, date, and time data types.

Punctuation in rules

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules, mandatory punctuation is usually predicted in the completion menu.

Parent topic: [Decision Center reference](#)

Definition part

The definition part starts with the `definitions` keyword and introduces local variables that you can use in the rest of the rule.

definitions

This construct introduces the part of the rule where you define variables.

from (object)

This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.

in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

where (test)

This construct matches objects against tests.

Parent topic: [Decision Center modeling language reference](#)

definitions

This construct introduces the part of the rule where you define variables.

Purpose

You use this construct to define local variables that you can use elsewhere in the same rule. Use local variables to produce more concise and readable rules.

Syntax

```
definitions
  set <variable> to <definition> in <list> | from <object> [where <test>*]
;*
```

Description

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules. You define local variables using the [set \(variable\) to \(definition\)](#) construct.

Example

The following example declares the local variable 'preferred customer' in the definitions part of the rule and uses it in the if and then parts of the rule.

```
definitions
  set 'preferred customer' to a customer from 'the customer';
if
  the age of 'preferred customer' is less than 18
then
  print "apply a 10% discount to the shopping cart of this customer";
```

Parent topic: [Definition part](#)

from (object)

This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.

Purpose

You use this construct to expand the set of data that can be used by a rule.

Syntax

```
from <object>
```

Description

The `from` construct is used to retrieve one distinct object from another one, and creates a variable that you can use in the rule to refer to this data. You cannot use it to retrieve a collection of objects. To access a collection of objects, use the [in \(list\)](#) construct.

Example

The following rule declares a variable based on the relationship between a customer and the customer's preferred item.

```
definitions
  set 'preferred CD' to a CD
    from the preferred item of the customer;
if
  the price of 'preferred CD' is more than 25
then...
```

Here the decision node does not have access to "CD" objects from its input data. In this case, to write a rule that uses a CD object, you must first declare a variable of type CD that pulls in data from the preferred item object of the customer. This is possible only if the customer object (and therefore the customer's preferred item) is already known to the decision node.

Parent topic: [Definition part](#)

in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

Purpose

You use this construct in the definition or condition part of a rule to access objects from collections.

Syntax

```
in <list>
```

Description

The `in` construct is used to retrieve data from a collection of objects, and creates a variable that you can use in the rule to refer to this data. The `<list>` parameter of the `in` construct must be an expression that denotes a collection.

The `in` construct can be used in the definitions part of a rule and in count conditions specified in the `if` part of a rule.

Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
    set 'item' to a CD
        in the items of the shopping cart of the customer ;
if
    there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
    there is an item
        in the items of the shopping cart of the customer ;
then...
```

Parent topic: [Definition part](#)

Parent topic: [Condition part](#)

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

Purpose

You use this construct in any part of the rule to access a list of attribute values.

Syntax

```
<attribute> of <list>
```

Description

You use this construct with other constructs that work with lists. For example, you can use it with the `in <list>` construct in a condition, or with the `set` construct in an action. The list can be an expression or an embedded list.

Example

The following action sets the value of the variable to the list of the names of the banks of all the customers.

```
set decision to the bank accounts of 'the customers';
```

The following definition creates a variable that represents the list of all the cities from the addresses of all customers.

```
definitions
  set 'registered city' to the cities of the addresses of 'the customers';
```

In the following rule, the action executes only if there is one or more customers who live in France.

```
if
  the countries of the addresses of 'the customers' contain "France"
then
  set decision to "include EU countries";
```

Parent topic: [Definition part](#)

Parent topic: [Condition part](#)

Parent topic: [Action part](#)

set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

Purpose

You use this construct in the definitions part of a rule to declare a local variable.

Syntax

```
set <variable> to <definition> [in <list> | from <object>] [where <test>*] ;
```

Description

Variables produce more concise rules by replacing a value or the result of an expression with a short, convenient identifier. A local variable can be used anywhere within the rule in which it is defined, but is not available in other rules.

Enclose the name of each variable in single quotes. This is compulsory for variable names that contain spaces. While it is not essential to enclose one-word variable names in single quotes, it makes them easier to identify and reduces the risk of confusion.

Example

The following examples show how to define a variable as an object.

```
definitions
  set 'i' to an item;
  set 'house' to a house
    where the price of this house is more than 1000;
```

The following examples show how to define a variable as a literal, string, or collection.

```
definitions
  set 'category' to Gold ;
  set 's' to "a string";
  set 'expensive items' to all items
    in the items of the shopping cart of the customer
    where the price of each item is more than 200;
```

The following example shows how to define a variable as the result of an expression.

```
definitions
  set 'expr' to the price of the car of the customer + 100;
  set 'h2' to the house of the customer
    where the price of this house is more than 1000;
```

The following example shows how to define a variable as another variable.

```
definitions
  set 'expr2' to 'expr';
```

Parent topic: [Definition part](#)

where (test)

This construct matches objects against tests.

Purpose

Matches objects against tests.

Syntax

```
where <test>[,]*
```

Description

In the `definitions` part of a rule, you can use one or more `where` clauses with the `set` construct to test that an object meets one or more conditions in order to be a valid value for the variable being declared. In the `definitions` part of the rule, the `where` statement does not end with a comma. Instead, each `set` construct should end with a semi-colon (;).

In the `if` part of a rule, you can use one or more `where` clauses in count conditions (`there is more than`, `there is less than`, and so on) to test that an object meets one or more conditions before being counted.

In a `where` construct, an implicit variable that refers to the current instance of the object being tested is automatically declared. In the test, you can thus refer to potential instances of the variable as `this <object type>` (or `each <object type>` if they are part of a collection.)

Example

The following example shows the `definitions` part of a rule that defines the local variable `'expensive items'` as a collection of items whose price is greater than 100. The implicit variable `each item` is used in the `where` clause to refer to each item object in the collection.

```
definitions
  set 'expensive items' to all items
    where the price of each item is more than 100;
```

The following example shows the `where` clause used together with the `there is one` construct in the `if` part of a rule to test if there is a customer older than 100. The implicit variable `this customer` is used in the `where` clause to refer to the customer object being tested.

```
if
  there is one customer
    where the age of this customer is more than 100,
```

Parent topic: [Definition part](#)

Parent topic: [Condition part](#)

Condition part

The condition part of the rule starts with the `if` keyword.

all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

if

This construct introduces the part of the rule in which you define conditions.

in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

it is not true that (a condition)

This construct negates a condition statement.

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

none of the following conditions are true

This construct negates a group of conditions.

there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the specified collection.

there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the specified collection.

there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in a collection.

there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

there is at least one (object)

This construct tests whether there is at least one object of a given type in a collection.

there is at most one (object)

This construct tests whether there is at most one object of a given type in a collection.

there is no (object)

This construct tests whether a data set contains no objects of the given type.

there is one (object)

This construct tests whether a collection contains one and only one object of a given type.

where (test)

This construct matches objects against tests.

Parent topic: [Decision Center modeling language reference](#)

all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

Purpose

The construct `all of the following conditions are true` provides a more compact and convenient alternative to the logical operator `and` when you want to combine multiple conditions. The resulting statement is considered true only if each of the listed conditions is true.

Syntax

```
all of the following conditions are true:  
  - <condition>* [,]
```

Description

This is equivalent to writing `if <condition 1> and <condition 2> and ... <condition 2n>`. However, if you have a large number of conditions, using the `all of the following conditions are true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This separates those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is Gold junior member.

```
if  
  all the following conditions are true:  
    - the category of the customer is Gold  
    - the age of the customer is at most 15  
then...
```

Parent topic: [Condition part](#)

any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

Purpose

The construct `any of the following conditions are true` provides a more compact and convenient alternative to the logical operator `or` when you want to combine multiple conditions. The resulting statement is considered true if at least one of the listed conditions is true.

Syntax

```
any of the following conditions is true:  
  - <condition>* [,]
```

Description

This is equivalent to writing `if <condition 1> or <condition 2> or ... <condition n>`. However, if you have a large number of conditions, using the `any of the following conditions is true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Note:

Indentation in rules is for readability only; it does not influence the way the rule is processed.

Example

This example shows how to test if a customer is a Gold member or a senior customer.

```
if  
  any of the following conditions is true:  
    - the category of the customer is Gold  
    - the age of the customer is more than 60  
then...
```

The following more complex example shows the use of a comma to mark the end of a group of conditions. In this example, the order must be over \$50, the customer must be either a Gold customer or a senior customer, and the order must be shipping to NJ.

```
if  
  all of the following conditions are true:  
    - the order total is greater than $50  
    - any of the following conditions is true:  
      - the category of the customer is Gold  
      - the age of the customer is more than 60,  
    - the shipping address is in NJ  
then...
```

Without the comma to mark the end of the `any of the following conditions are true` block, the final condition (the shipping address is in NJ) would be considered part of the previous group, and the rule would be interpreted to mean that the order must be over \$50 and (either the customer is a Gold customer or a senior customer or the shipping address is in NJ).

Parent topic: [Condition part](#)

if

This construct introduces the part of the rule in which you define conditions.

Purpose

You use this construct in a rule to define one or more condition statements.

Syntax

```
if  
    <condition>*
```

Description

If the conditions in the `if` part are met, the actions in the `then` part of the rule are executed. The `if` part of a rule is optional. If there is no `if` part, all actions in the `then` part of the rule are performed each time the rule is executed.

If the conditions in the `if` part of the rule are not met, the actions in the `else` part of the rule are executed. If the conditions are not met and the rule does not have an `else` part, the rule does nothing.

Example

The following rule shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if  
    the age of the customer is less than 18  
then  
    print "apply a 10% discount to the shopping cart of this customer";
```

Parent topic: [Condition part](#)

it is not true that (a condition)

This construct negates a condition statement.

Purpose

You use this construct when there is no operator to express the opposite of a condition.

Syntax

```
it is not true that <condition>
```

Description

When there is no operator to express the opposite of a condition, you can insert the `it is not true that` construct in front of an existing condition to negate it.

This can be useful in cases where a boolean (true/false) attribute defined in your model is verbalized as an expression such as 'customer is a loyalty member', but where you want to test for the opposite of this expression ('customer is not a loyalty member'). You can write 'it is not true that customer is a loyalty member' to achieve the required result.

In general, the logic of a negative expression is more difficult to interpret, so it is advisable to avoid them where possible.

Example

The following example uses the `it is not true that` construct to check the age of the customer, assuming that the boolean property 'is a minor' has been verbalized in the vocabulary, but that the opposite of the expression 'is a major' has not been defined.

```
if
  it is not true that the Customer is a minor
```

The following example shows how to negate a set of conditions that have been grouped using the `all of the following conditions are true` construct. The resulting expression will return True except if the customer is both a gold member and over 60.

```
if
  it is not true that all of the following conditions are true:
    - the category of the Customer is gold
    - the age of the customer is more than 60
```

Parent topic: [Condition part](#)

none of the following conditions are true

This construct negates a group of conditions.

Purpose

You use this construct to group together a series of condition statements that you want to negate.

Syntax

```
none of the following conditions are true:  
  - <condition>* [,]
```

Description

The group evaluates to true only if none of the conditions listed are true.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is not a Gold senior member and not under 60.

```
if  
  none of the following conditions are true:  
    - the category of the customer is gold  
    - the age of the customer is least 60  
then...
```

Parent topic: [Condition part](#)

there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the specified collection.

Purpose

You use this construct to determine whether the collection contains exactly the specified number of occurrences of a particular object.

Syntax

```
there are <number> <objects> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests if the number of Gold customers is equal to 8.

```
if
    there are 8 Customers in the customers of 'the bank'
        where the category of each customer is Gold
then...
```

Parent topic: [Condition part](#)

there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the specified collection.

Purpose

You use this construct to determine whether the collection contains at least the specified number of occurrences of a particular object.

Syntax

```
there are at least <number> <objects> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are at least 10 Gold customers.

```
if
    there are at least 10 Customers in the customers of 'the bank'
        where the category of each customer is Gold
then...
```

Parent topic: [Condition part](#)

there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains no more than the specified number of occurrences of a particular object.

Syntax

```
there are at most <number> <objects> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than or equal to 3.

```
if
  there are at most 3 Customers in the customers of 'the bank'
  where the category of each customer is Gold
then...
```

Parent topic: [Condition part](#)

there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the specified collection contains fewer than the specified number of occurrences of a particular object.

Syntax

```
there are less than <number> <objects> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The *<number>* argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than 3.

```
if
    there are less than 3 Customers in the customers of 'the bank'
    where the category of each customer is Gold
then...
```

Parent topic: [Condition part](#)

there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the specified collection contains more than the specified number of occurrences of a particular object.

Syntax

```
there are more than <number> <objects> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are more than 10 customers with the Gold category.

```
if
    there are more than Customers in the customers of 'the bank'
        where the category of each customer is Gold,
then...
```

Parent topic: [Condition part](#)

there is at least one (object)

This construct tests whether there is at least one object of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains at least one occurrence of a particular object.

Syntax

```
there is at least one <object> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests whether or not a Customer object exists.

```
if
    there is at least one customer
        where...
then...
```

The following condition tests if a senior Gold customer exists.

```
definitions
    set 'gold customers' to all Customers in the customers of 'the bank'
        where the category of each customer is Gold;
if
    there is at least one customer in 'gold customers'
        where the age of this customer is at least 60,
then...
```

Parent topic: [Condition part](#)

there is at most one (object)

This construct tests whether there is at most one object of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains zero or one occurrence of a particular object.

Syntax

```
there is at most one <object> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

Example

The following condition tests if there is at most one Gold customer in the list of customers.

```
if
    there is at most one Customers in the customers of 'the bank'
        where the category of this customer is Gold,
then...
```

Parent topic: [Condition part](#)

there is no (object)

This construct tests whether a data set contains no objects of the given type.

Purpose

You use this construct to determine whether the specified collection contains no occurrences of a particular object.

Syntax

```
there is no <object> in <list> [where <tests>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the `if` part of a rule.

Example

The following condition tests that there is no object of type `Customer` with an age over 60 in the list of customers.

```
if
    there is no Customer in the customers of 'the bank' where the age of this
customer is at least 60
then...
```

Parent topic: [Condition part](#)

there is one (object)

This construct tests whether a collection contains one and only one object of a given type.

Purpose

You use this construct to determine whether the specified collection contains one and only one occurrence of a particular object.

Syntax

```
there is one <object> in <list> [where <test>,*]
```

Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the `if` part of a rule.

Example

The following condition tests if there is one Gold customer in the list of customers.

```
if
    there is one Customer in the customers of 'the bank'
        where the category of this customer is Gold
then...
```

Parent topic: [Condition part](#)

Action part

The action part describes the actions that the rule completes when the conditions are met. This part might start with the then keyword and can contain an else construct.

[add \(object\) to decision](#)

The add construct adds an item to a list of objects.

[else](#)

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

[for each \(object\) in \(list\)](#)

This construct specifies the actions that must be performed to every element of a collection.

[for each \(object\) called \(variable\), in \(list\)](#)

This construct specifies that the rest of the rule must be applied to each element of a collection.

[\(attribute\) of \(list\)](#)

You use this construct to access the list of values of a given attribute for a list of objects.

[print \(string\)](#)

This construct prints a string to the standard output.

[remove \(object\) from decision](#)

The remove construct removes an item from a list.

[set decision to \(value\)](#)

This construct specifies a value for the decision to make.

[then](#)

This construct introduces the part of the rule that defines the actions that are executed if the condition part of a rule is satisfied.

Parent topic: [Decision Center modeling language reference](#)

add (object) to decision

The add construct adds an item to a list of objects.

Purpose

You use this construct in the action part of a rule to add an object to a list of objects.

Syntax

```
add <object> to decision;
```

Description

The <object> specifies the object that you want to add. The list in which you want to include this object corresponds to the type of the decision node output, which must be a list.

Example

The following action adds a string with the name of a race to the decision, which is a list of strings.

```
add "New matching race: " + the name of 'the race' to decision;
```

Parent topic: [Action part](#)

else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

Purpose

You use this construct to declare actions that are executed if the `if` part of a rule has not been satisfied. These actions are ignored if all local variables are not correctly initialized due to preconditions in the definitions part failing to be satisfied.

Syntax

```
else
    <action>;*
```

Description

The `else` part of a rule is optional and allows you to specify one or more actions to perform if the conditions in the condition part of the rule are not met.

The actions in the `else` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

If one or more variables with preconditions are defined in the definition part of a rule, the rule is only processed if these preconditions are satisfied and all local variables are correctly initialized. This means that if these preconditions are not satisfied, the rule is not processed at all, and the actions in the `else` part of the rule are never executed, whether or not the conditions in the condition part of the rule are met.

Adding preconditions in the definition part of a rule rather than in the condition part of a rule can provide a small performance improvement, since it potentially reduces the number of conditions to be checked. However, since the logic of the rule is not the same in each case, the method to use depends on the result you want to obtain.

Example

Consider a rule that sets the decision to the value of a discount. A customer will get a 10% discount if the customer is in the Gold category. Otherwise, the customer gets a 5% discount.

```
definitions
    set 'valued customer' to a customer
if
    the category of 'valued customer' is Gold
then
    set decision to 10;
else
    set decision to 5;
```

In the following example, a second condition has been added in the condition part of the rule. Now, a customer receives a 10% discount only if the customer is in the Gold category and is aged over 20. All other customers receive a 5% discount.

```
definitions
    set 'valued customer' to a customer
if
    the category of 'valued customer' is Gold and 'valued customer' is older
    than 20
then
    set decision to 10;
else
    set decision to 5;
```

Now consider a third example. In the following rule, a minimum age requirement is added as a precondition in the definition part of the rule. This means that a customer must first be over 20 years old to be considered a valued customer. Otherwise, the `valued customer` variable is not initialized and the rest of the rule is not executed. With this rule, a customer under 20 receives no discount at all. A customer who is over 20 and in the Gold category receives a 10% discount. The `else` part of the rule is executed and a 5% discount applied only if the customer is over 20 but not in the Gold category.

```
definitions
    set 'valued customer' to a customer
    where this customer is older than 20;
```


```
if
  the category of 'valued customer' is Gold
then
  set decision to 10;
else
  set decision to 5;
```

Parent topic: [Action part](#)




IBM Operational Decision Manager on Cloud

Welcome to the IBM Knowledge Center for Operational Decision Manager on Cloud. Here you can find information about creating and deploying business rules for your applications through the cloud version of Operational Decision Manager. Updated: June 2019







Get started

- Overview
- First steps
- Tutorials
-  Videos
- Product legal notices
- Additional notices

Find support

-  Online Education
- Troubleshooting and support
-  Detailed system requirements
-  FAQ

Stay current

- What's new
-  Known limitations
-  Technotes
-  Recommended reading
-  Facebook
-  Twitter
-  Blog

Overview

IBM® Operational Decision Manager on Cloud is a web-based platform for making business rule solutions that automate the application of complex decisions.

[What's new](#)

Ongoing development ensures that Operational Decision Manager on Cloud continues to meet your needs for creating and implementing business rule applications.

[Introduction](#)

Operational Decision Manager on Cloud is a comprehensive decision automation platform for capturing, automating, and governing rule-based business decisions.

[Cloud workflow](#)

This example introduces you to the collaborative process for creating and deploying a rule application in Operational Decision Manager on Cloud.

[Operational Decision Manager](#)

Operational Decision Manager delivers a proven development system for capturing policies in solutions that automate the application of decisions.

[Key components](#)

Operational Decision Manager on Cloud includes the main components in Operational Decision Manager.

[Cloud environments and user roles](#)

The components for creating and using rule applications are kept in cloud environments. Access to the environments is controlled through user roles.

[Express](#)

IBM Operational Decision Manager on Cloud Express covers the entire lifecycle of a decision service in one environment.

[Hybrid cloud environments](#)

Complement your on-premises or cloud development system with Operational Decision Manager on Cloud.

[First steps](#)

Before you begin, you must gain access to the cloud portal. If you want to create decision services, you must install Rule Designer. When you finish setting up, you can go through the tutorials to learn how to use Operational Decision Manager on Cloud.

[Usage reports](#)

You can view the use of your decisions and artifacts through a web-based dashboard. It displays performance data in graphs that enable you to readily understand your rule application traffic.

Related information:

[Operational Decision Manager](#)

What's new

Ongoing development ensures that Operational Decision Manager on Cloud continues to meet your needs for creating and implementing business rule applications.

The latest release introduces a dashboard for monitoring the use of artifacts and decisions. It replaces the monthly usage reports.

Users of Express can also add an additional environment for integration testing or production, and Mac users can use Rule Designer on macOS Mojave.


To see all the new features, consult the sections below. You can skip ahead to a section by clicking its shortcut link:

Sections:


- [General](#)
- [Rule Designer](#)
- [Decision Center](#)

General


Usage dashboard

Operational Decision Manager on Cloud now includes a dashboard for monitoring artifacts in Decision Center and decisions in Rule Execution Servers. The dashboard is accessible to cloud administrators, and monitors usage over a selected period of time. The dashboard replaces the monthly usage reports.  [Learn more...](#)

Adding an environment to Express


Express comes with a single environment for developing and deploying decision services. Now, you can add an additional environment for integration testing or production. Taking Express all the way to three environment requires switching to the full version of Operational Decision Manager on Cloud.  [Learn more...](#)

Checking your user role


Now, you can check your user role in your portal profile. Your role determines your access to the various features, environments, and components in the portal. Check the role to see what is available to you. If you have more than one account with different roles, you can check the role in the profile to determine which account you are working in.  [Learn more...](#)

Rule Designer


MacOS Mojave support

You can now use Rule Designer on macOS Mojave (version 10.14). 

Testing rulesets in Rule Designer


Now you can verify that rulesets behave as expected in Rule Designer by defining scenarios and running a testing configuration. Rule Designer displays the results in its Console view.  [Learn more...](#)

Upgrading Rule Designer


Each release comes with its own version of Rule Designer. When you upgrade to the latest release, you must reinstall the rule editor. To see which release you are working with, look for its number in the bottom right corner of your instance of Operational Decision Manager on Cloud.  [Learn more...](#)

Decision Center


Configuration settings available in the Business console

You can set some configuration options to customize Decision Center behavior or display in the Business console. You must have the permission manager role to edit the configuration settings.  [Learn more...](#)


Exporting and importing test suites and simulation artifacts


Export test suites and simulation artifacts with a project from the Business console. The export function generates a compressed file that you can import back into the console later or use with a different instance of Decision Center. A few limitations apply, but you get a working, portable file that's ready for validation.  [Learn more...](#)

Grid columns for displaying properties

Select and order the columns used to display the properties of rules, decision tables, ruleflows, and variable sets in the Decision Artifacts and Queries tabs of decision service branches in the Business console.  [Learn more...](#)

Generating reports on your projects and decision services

You can now generate reports on decision services, projects, or queries. Displayed in the HTML format, the reports show the content and properties of deployed project elements. You can use the reports in auditing your repository or doing other resource management operations.  [Learn more...](#)

You can also use the new Decision Center REST API endpoints to see the contents of your repository.  [Learn more in Explore section...](#)


Creating and editing advanced rule artifacts in the Business console

Now you can create and edit advanced rule artifacts in the Business console. Developers can use technical rules to tap constructs and features such as loops and explicit ILOG Rule Language (IRL) mapping, and functions to share procedure code between elements in a ruleset. (**Note:** The Business Action Language (BAL) is not supported.)


 [Learn more about technical rules...](#)

 [Learn more about functions...](#)

Locking decision tables in the Business console

In Rule Designer, you can apply different types of locks to a decision table to prevent edition by others. Now, you can apply those locks to decision tables in the Business console when you publish your decision service.  [Learn more...](#)

Sharing projects in different decision services or branches

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in other decision services, or in other branches of the same decision service. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that you need to maintain only one BOM and one vocabulary, and changes made to the BOM and vocabulary are automatically propagated to all the projects that use them.  [Learn more...](#)

[Change history](#)

See what's new in past releases of Operational Decision Manager on Cloud.

[Deprecated features](#)

These features are deprecated or removed from Operational Decision Manager on Cloud.

Parent topic: [Overview](#)

Change history

See what's new in past releases of Operational Decision Manager on Cloud.

Releases:

- [2018.12](#)
- [2018.10](#)
- [2018.06](#)
- [2017.10](#)
- [2017.06](#)
- [2017.03](#)
- [2016.12](#)

2018.12

Features introduced in December 2018.

General

Subscription selection: Now you can switch between your instances of Operational Decision Manager on Cloud without logging out. The new feature enables you to move quickly among subscriptions. [!\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\) Learn more...](#)

Decision Center

Decision modeling in the Business console: The decision model service makes business experts autonomous in developing rule applications. They can create, maintain, and deploy decision services, and edit a model directly by using a relatively nontechnical language. A diagram-based development system clearly shows the flow of data and decisions through a decision service. [!\[\]\(aa53ad6fea213b8b2226d3077e30533a_img.jpg\) Learn more...](#)

Decision governance through REST API: The Decision Center REST API now includes decision service management through the decision governance framework. [!\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\) Learn more in the Govern section...](#)

Running diagnostics in the Business console: Administrators can now run diagnostics on their systems from the Business console. They can get information on versioning, data sources, extensions, verbalizers, and databases, and use it to troubleshoot issues. [!\[\]\(758ebdf4629c903da74c2e079717ae32_img.jpg\) Learn more...](#)

Webhook notification from Decision Center: You can use webhooks to send notifications from Decision Center to an external application of your choice. Get real-time notifications for events that are related to decision governance, RuleApp deployment, and changes to business rules. [!\[\]\(fe3aebe81acea8d45108cd2768939da7_img.jpg\) Learn more...](#)

Customer survey in the Business console: An in-application survey now solicits feedback from users of the Business console. A pop-up window opens on your screen, giving you an opportunity to rate the service by selecting a score. (Note: The survey opens in English when it does not support the user's locale.) The system is set to check in with you every 90 days. Your time is valuable, and you are under no obligation to participate. Even if you close the pop-up without responding, taking a couple of minutes to provide input directly to the Operational Decision Manager on Cloud team is appreciated. Your feedback is taken seriously. By taking advantage of this feature, you can contribute to improving Operational Decision Manager on Cloud. [!\[\]\(626ce8ac21792b9405bfddfea8e0c96a_img.jpg\) Learn more...](#)

Rule Designer

Move to Eclipse 4.7.3: Rule Designer now runs on Eclipse 4.7.3. The move brings changes to the component for creating and deploying rule applications. [!\[\]\(248b91fcdac4810ffd15cf33fb6aec6f_img.jpg\) Learn more...](#)

Rule Execution Server

REST API methods for Rule Execution Server diagnostics: You can use new methods in the REST API to get Rule Execution Server diagnostics. These methods are available in the **/utilities** tab in the REST API test tool in the Rule Execution Server console. [!\[\]\(c1168d6a8b365d11e842ece304635fa7_img.jpg\) Learn more...](#)

Ruleset selection for RuleApp archives: When you package a RuleApp archive for downloading from the Rule Execution Server console, you can now choose to include only selected rulesets. Previously, all rulesets in the RuleApp were included in the package downloaded. [!\[\]\(cbd8541a32dfc32f356f5c6c994b0a21_img.jpg\) Learn more...](#)

Tutorials

Getting started with decision modeling: This tutorial shows the basics of creating a decision model service. You model and author a decision service from the Decision Center Business console. [!\[\]\(40770d9ed6ed4f1222ebf89a1396e8b2_img.jpg\) Learn more...](#)

2018.10

Features introduced in October 2018.

General

Operational Decision Manager on Cloud Express®: The new Operational Decision Manager on Cloud

Express covers the entire lifecycle of a decision service in one environment. It includes all the components for creating a decision service, developing it collaboratively, and running it for a client application. Unlike the complete Operational Decision Manager on Cloud, Express does not include separate environments that are dedicated to integration testing and production applications. [!\[\]\(f4912148590488019602cab6e009e597_img.jpg\) Learn more...](#)

Updated password policy: The constraints for composing passwords have been updated. Also, the duration for passwords is now 90 days. [!\[\]\(8af806fb1314382d09bc5ec5b767526c_img.jpg\) Learn more...](#)

2018.06

Features introduced in June 2018.

General

Customer-defined cloud portals: Now you can configure the environments in your cloud portal to better match your development process. New customers determine the makeup of their cloud environments during the initial provisioning. Existing customer can buy new environments to change their current setup. The provisioning of additional environments is done by IBM®. [!\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\) Learn more...](#)

Usage reports for decisions and managed artifacts: Reports are now sent on the usage of decisions and managed artifacts. You can use the reports to track your services and manage your subscription compliance. The reports are sent monthly, and cover daily usage. You can request to receive your reports more frequently. For information on metrics: [!\[\]\(0aff635c4179ba9e710b00f4b01d3b20_img.jpg\) Learn more...](#)

New password rules and duration: The constraints for composing passwords have been updated to remain in compliance with rules set by the US Federal Risk and Authorization Management Program (FedRAMP). The rules also include a shorter duration for cloud portal passwords, which now expire in 60 days instead of 90 days. When a password created under the old 90-day expiration period is changed, the new password becomes subject to the new expiration period. [!\[\]\(830769b31eeeaca920791081939ff8ba_img.jpg\) Learn more...](#)

Decision Center

Using Decision Center with multiple subscriptions: Now you can attach the Decision Center of one Operational Decision Manager on Cloud subscription to the execution runtime environments of another subscription of Operational Decision Manager on Cloud. This can increase the redundancy of execution environments in cases where higher availability is required, and a choice has been made to have an instance deployed in the same or multiple data centers.

You can use a similar setup in a hybrid environment where the runtime environment is running on premises or another cloud. The Rule Execution Server console in the cloud environment must be securely exposed on the public Internet or through an outbound VPN. Ask your IBM representative for more information. For information on hybrid environments: [!\[\]\(6bb0e4f14c4133b37d2887cb37e67ddd_img.jpg\) Learn more...](#)

Offline updating of decision tables from the Business console: Now you can work on a decision table offline, and then import it into the Business console. You start by exporting the decision table from the Business console as an Excel spreadsheet. After making changes, you import the spreadsheet back into the Business console to update the corresponding decision table. [!\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\) Learn more...](#)

Managing servers in the Business console: As a permission manager, you can now use the Business console to manage the list of servers that are available through Decision Center for deployments, tests, and simulations. You can also restrict access to specific groups. [!\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\) Learn more...](#)

Tutorials

Getting started in Operational Decision Manager on Cloud: This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal. [!\[\]\(799877f5c2f906134441300079881630_img.jpg\) Learn more...](#)

Debugging a decision service in Rule Designer: This tutorial provides a methodology for debugging a decision service in Rule Designer. [!\[\]\(41aea2746216b27a6939d696d8e035da_img.jpg\) Learn more...](#)

2017.10

Features introduced in October 2017.

General


Customer survey: An in-application survey now solicits feedback from users of Operational Decision Manager on Cloud. A pop-up window opens on your screen, giving you an opportunity to rate the service by selecting a score. (Note: The survey opens in English when it does not support the user's locale.) The system is set to check in with you every 90 days. Your time is valuable, and you are under no obligation to participate. Even if you close the pop-up without responding, taking a couple of minutes to provide input directly to the Operational Decision Manager on Cloud team is appreciated. Your feedback is taken seriously. By taking advantage of this feature, you can contribute to improving Operational Decision Manager on Cloud. [!\[\]\(179f167ede0522ebb4ea025b3ad78ca7_img.jpg\) Learn more...](#)

Password restrictions: The restrictions for composing passwords for the cloud portal have been updated to conform with the current guidelines from the Federal Risk and Authorization Management Program (FedRAMP). These restrictions do not apply to SAML accounts. [!\[\]\(4a7b4ce770af8456e11a71f9565c8c2b_img.jpg\) Learn more...](#)


Rule Designer

Automatic exception handling in rule conditions: The behavior of the following BAL expressions is now the same as the other expressions in rule conditions:

- If there is no ... then ...
- If there is at least one ... then ...


The objects are counted only when the overall where clause is true. False and unknown objects are ignored.  [Learn more...](#)


Rule Execution Server console


Deleting resources and libraries cleanly: When you remove a Java™ XOM resource or library, it might be referenced by other artifacts or it might reference other artifacts. Now you can remove all the links that point to the resource or library, as well as all the links that point from the library, in the Rule Execution Server console in just one go, giving you a clean delete.  [Learn more...](#)


Decision Center

Decision Center REST API: You can use the Decision Center REST API to build, test, and deploy decision services. You can set up automated continuous deployment that uses the programming language of your choice.  [Learn more...](#)

Exporting and importing projects in the Business console: Administrators and configuration managers can now export the working branch of a decision service to a compressed file, and import a project into the Decision Center Business console from a compressed file. The working branch can contain more than one project.  [Learn more...](#)

Multipage display in the Business console: The Business console now shows the contents of its tabs (Decision Artifacts, Queries, Tests, Simulations, Deployment, and Snapshots) on one or more pages. The new feature improves the browsing of projects, especially when you have numerous artifacts.  [Learn more...](#)


Deleting decision services: Administrators can now delete decision services directly through the Business console. This operation used to be limited to the Enterprise console in Decision Center.  [Learn more...](#)

Embedded managed Java XOM in Decision Center: You can now use the Decision Center REST API to generate a RuleApp archive that includes an embedded managed Java XOM.  [Learn more...](#)


2017.06

Features introduced in June 2017.

Cloud portal

Service credentials for client applications: You use service credentials to authenticate a client application when it calls a decision service that is running in the cloud.  [Learn more...](#)

Decision Center tutorial

Merging branches in the Business console: This tutorial shows you how to merge changes between branches in a decision service in the Decision Center Business console. You can develop rules in one branch, and then transfer your changes the same rules in other branches.  [Learn more...](#)


Troubleshooting and support

Support for Decision Center: Find out how to get support for Operational Decision Manager on Cloud. Customer support is available for users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.  [Learn more...](#)


2017.03


Features introduced in March 2017.

Rule Execution Server

Running on Java 8: Rule Designer runs on Java 8. You must use JDK 8 and Eclipse 4.4 to run Rule Designer on an existing instance of Eclipse.  [Learn more...](#)

Decision Center Business console

Using custom test data providers: You can generate scenario files for custom test data providers in the Business console. You select the tests and expected execution details, and then download a custom data provider template. You can then use the template with various data sources, such as XML files or a database.  [Learn more...](#)

Comparing versions of a variable set: You can see the differences between versions of a variable set by comparing snapshots of the versions in the Business console.  [Learn more...](#)

2016.12

Features introduced in December 2016.

IBM Cloud™ Product Insights

Enablement for IBM Cloud Product Insights: *(No longer available.)* Decision Center is now enabled for IBM Cloud Product Insights. This Bluemix® service replaces the experimental service IBM Cloud Hybrid Connect. IBM Cloud Product Insights serves as a window onto your running inventory and runtime usage metrics.

Rule Execution Server

OpenAPI support for running decision services: Generating OpenAPI descriptions, or Swagger, is part of the Decision Connect capability. It makes it easier to integrate decision service APIs into calling applications. The descriptions are compatible with IBM API Connect® and other API management products. [!\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\) Learn more...](#)

Including resources and libraries in the deployed RuleApp archive: When you deploy a RuleApp archive by using the Rule Execution Server console, resources and libraries can be included in the deployed archive. [!\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\) Learn more...](#)

Decision Center Business console

Comparing and merging branches: You can now use the compare and merge function in the Business console to manually merge the contents of branches, releases, and activities in a decision service. [!\[\]\(3211b5d1d968fc1665909b34f9f16010_img.jpg\) Learn more...](#)

Managing resources in decision service projects: Upload files of any type to a decision service project in the Decision Center repository through the Business console. You can then find the files in the Resources folder of the project, where you can view, edit, create, or download files. [!\[\]\(6059a5aa8b4ca7bb793408023d6c6e42_img.jpg\) Learn more...](#)

Updating dynamic domains: You can now update dynamic domains in the Business console. If a domain is defined in an Excel file, you can download and upload the file from the Model tab in the Business console. [!\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d_img.jpg\) Learn more...](#)

Filtering columns in decision tables: Searching through a long decision table for a specific value can be slow, tedious work. The new filtering feature for decision tables can shorten your search time by showing only the rows that contain your filter values. [!\[\]\(6a9b39b98eb945faa14c645ec99e4eaa_img.jpg\) Learn more...](#)

Direct links to console content: Any URL in the Business console is accessible externally. You can bookmark these URLs in your browser, or reference them from an external source, and navigate directly to a specified branch, release, activity, or project element. You can now see the differences between versions of a variable set by comparing snapshots in the Business console. [!\[\]\(9c2e8d1b5bd77cb5c9f83b7a9cff79fd_img.jpg\) Learn more...](#)

Decision Center tutorials

Creating ruleflows: You set the order for firing rules in one or more ruleflows, and in doing so, list the rules that are run in your decision service. This tutorial shows you how to create and implement ruleflows. [!\[\]\(f60b7a900783ac3fd531bfd9c111be6d_img.jpg\) Learn more...](#)

Creating simulations: Use the simulator in the Business console to validate your rules with real or representative data. The simulator produces reports that you can use to refine decision services. Follow this tutorial to learn how to make simulations. [!\[\]\(f1c5da15572e3e09d343161be98f508d_img.jpg\) Learn more...](#)

Creating decision tables: *(No longer available)* Don't write the same rule over and over again to apply different condition and action values. Instead, create a decision table that forms a series of rules by applying the same rule statement to a table of values. Learn how in this tutorial on decision tables.

Parent topic: [What's new](#)

Deprecated features

These features are deprecated or removed from Operational Decision Manager on Cloud.

- Two-factor authentication
- Templates for action rules and decision tables
- Decision tree artifacts
- Ruleset completeness analysis

Parent topic: [What's new](#)

Introduction

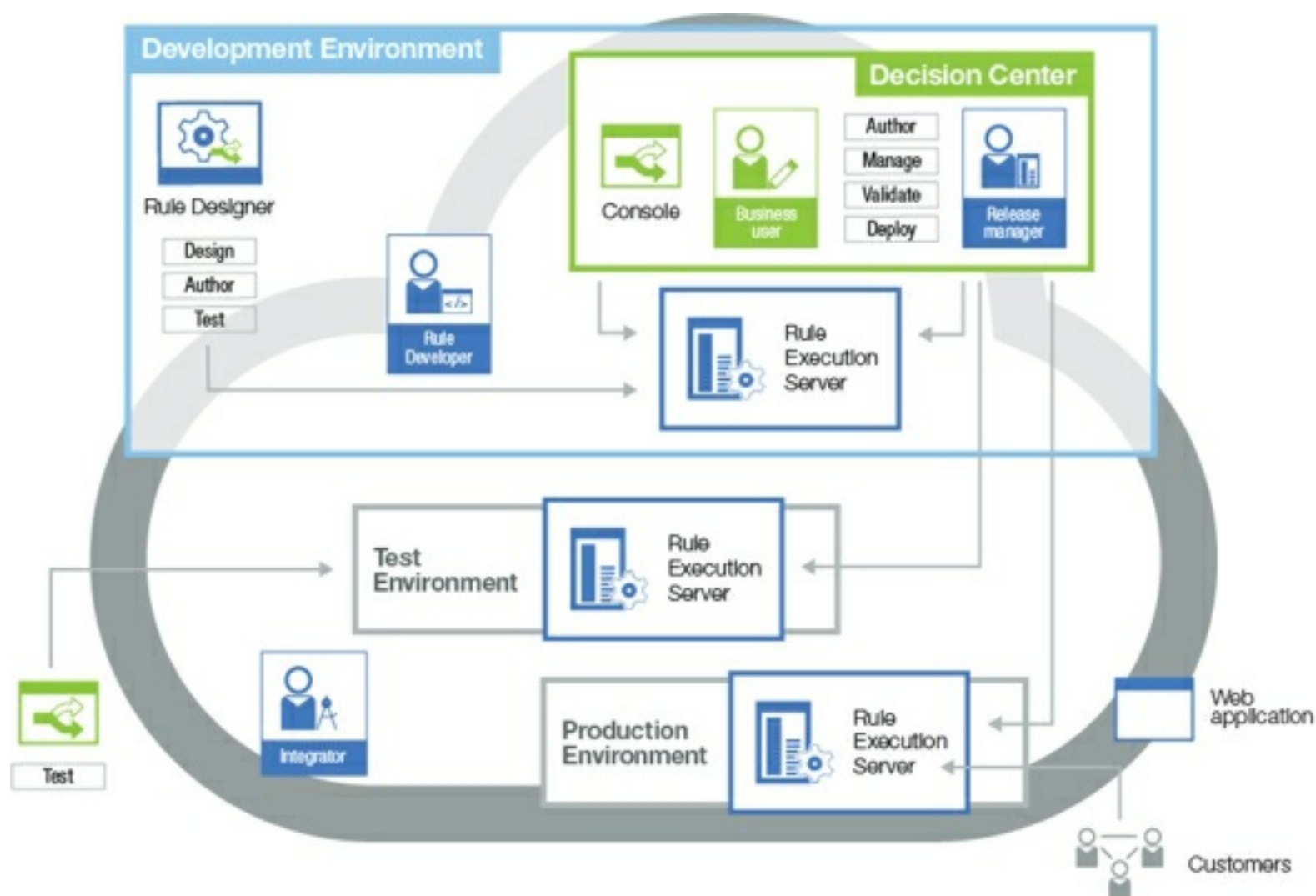
Operational Decision Manager on Cloud is a comprehensive decision automation platform for capturing, automating, and governing rule-based business decisions.

Both technical and business users can use its components to create solutions for key operations such as authorizing loans, applying promotions, detecting cross-sell opportunities, and processing insurance. You can implement decisions immediately to keep your applications continuously up to date and aligned with the changing business objectives of your organization.

Your instance of Operational Decision Manager on Cloud is ready to use when you log in to it for the first time. You get a dedicated version of Operational Decision Manager that is set up, configured, and provisioned for you. You have immediate access to tools for designing, developing, and deploying business rule applications.

Collaborative development

As the following diagram shows, you capture the policies of your organization in business rules through a collaborative service that supports developers, analysts, and business users.



You start by creating the business rules in Rule Designer, an Eclipse-based component, and then further develop and maintain the rules in an online platform that supports both business and technical users. Governance features control user access and impose a workflow to ensure the proper development of your rules. When your rules are ready, you deploy them to an execution server for use with client applications.

Two solutions to choose from

Operational Decision Manager on Cloud is available in the full version and Express. The full version gives you three environments – development, test and production – for a complete rule lifecycle solution. The Express version delivers one environment that delivers many of the features in the full version. You can add a test or production environment to an instance of Express, but you must get the full version to use all three environments together. Whichever solution you use, you can obtain additional execution servers for running your rule applications.

Maintained by IBM

IBM does all the maintenance on Operational Decision Manager on Cloud. It provides regular upgrades to ensure that you get the latest features, and it performs migration tasks to keep your data in sync with the platform. If you need an additional environment or execution server, IBM adds it for you.

Cloud workflow

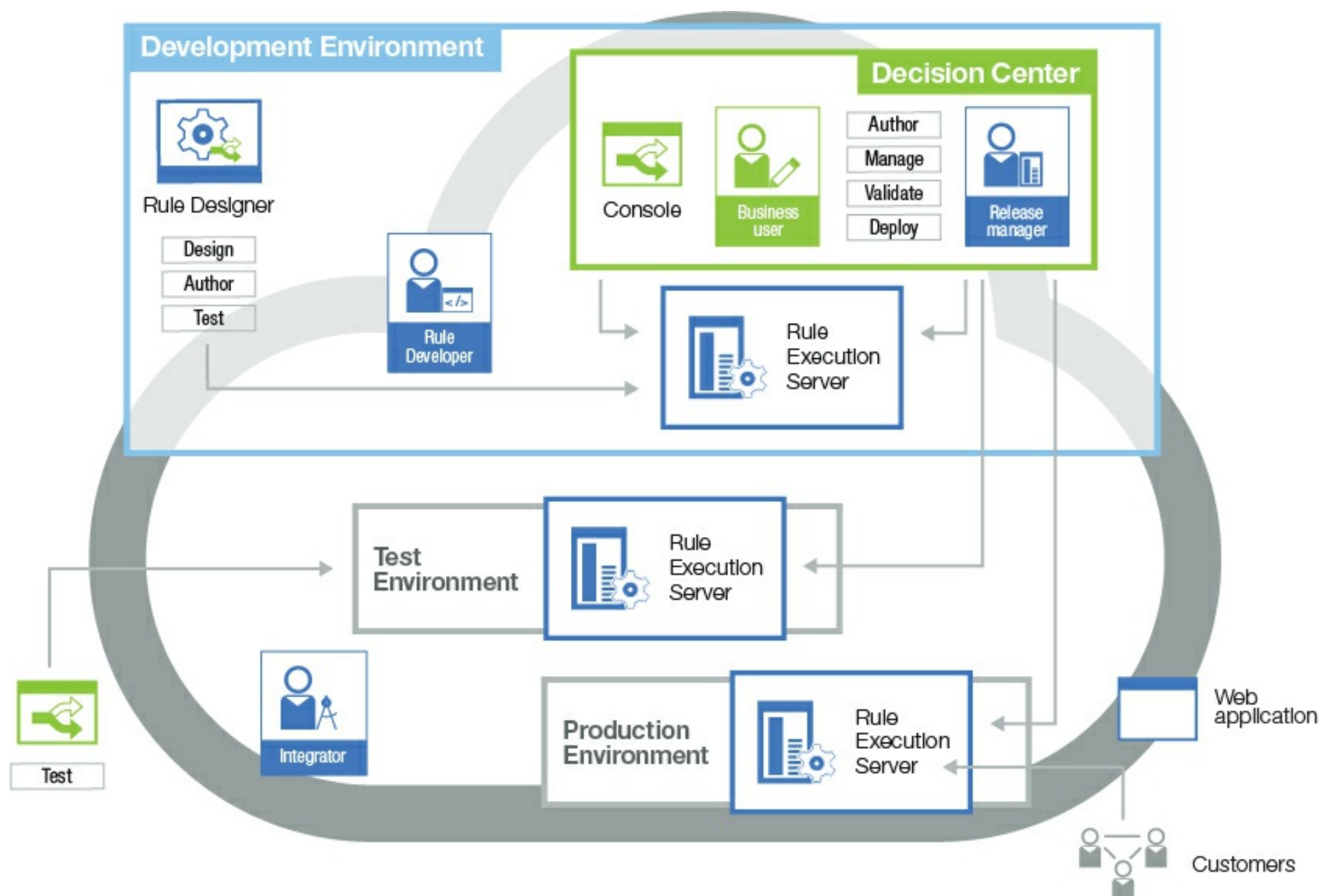
Follow the collaborative process for creating and deploying a rule application in Operational Decision Manager on Cloud.

This example takes you through the cloud workflow. A fictitious car rental company uses Operational Decision Manager on Cloud to implement rental decisions that are based on policies and pricing. A group of people collaborate to design the business rules, create and test a decision service, and deploy rule applications to runtime environments.

The example is organized into three sections:

- [Setting up the user roles](#)
- [Collaborating on the decision service](#)
- [Testing and promoting the decision service](#)

The following diagram illustrates how predefined user roles work with the product components during the lifecycle of a business rule application. The cycle includes rule development and validation, testing, and production. The collaborators work in the different cloud environments.



Setting up the user roles

In the table below, John is shown as the information technology (IT) administrator at the car rental company. He is responsible for administering user accounts in the Operational Decision Manager on Cloud portal. As the first person invited to the portal, he has the role of cloud administrator. Though he might not use the cloud components himself, he must invite other people to the portal and assign them roles so that they can use the components.

Table 1. Team members and user roles

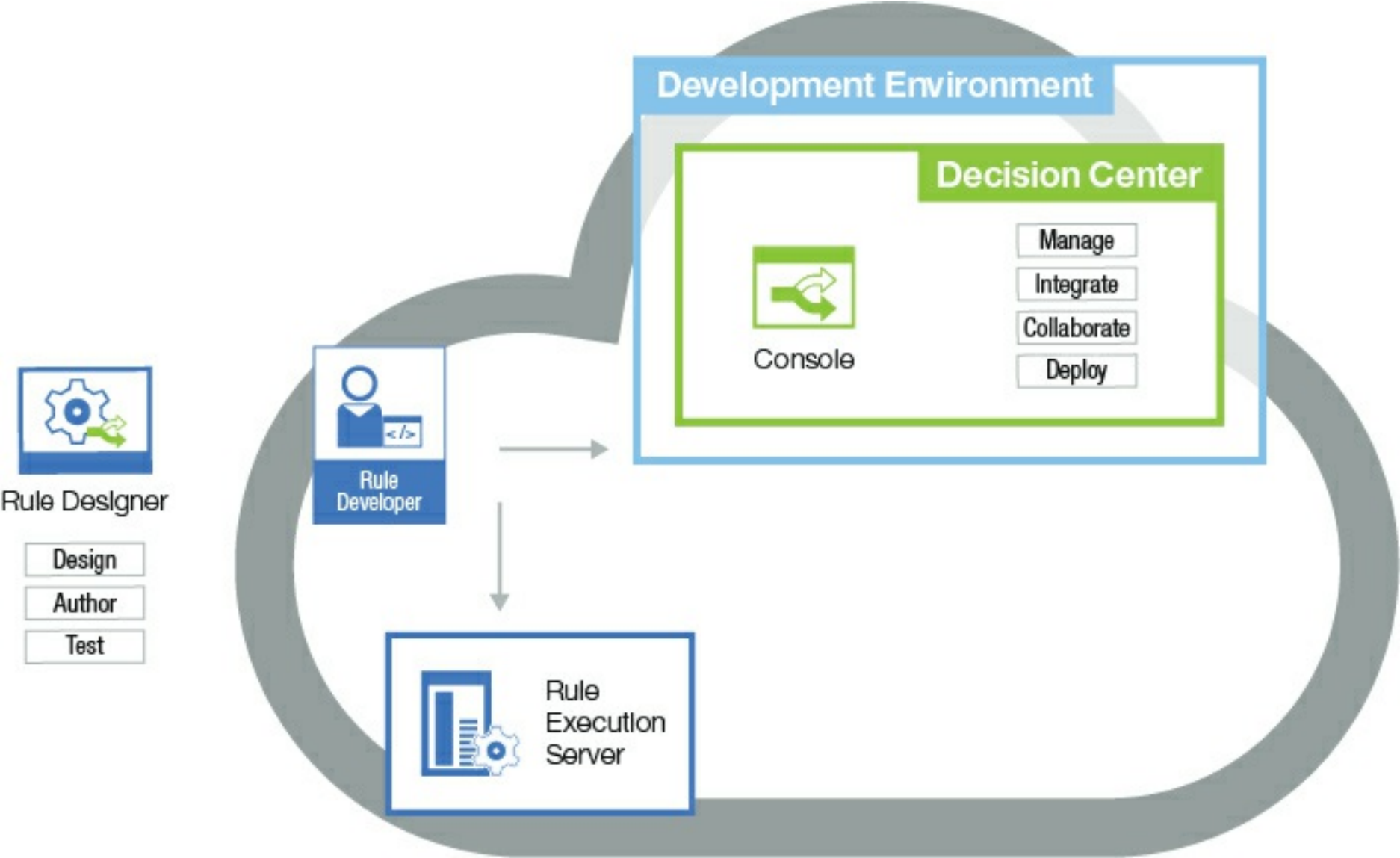
Team member	Job	User role	Component and environment access
John	IT administrator	Cloud administrator (admin) Permission manager	Operational Decision Manager on Cloud portal and Decision Center console, user accounts, groups and roles
Kim	Application developer	Rule developer	Rule Designer and Decision Center, deploys to the development and test environments
Gary	Marketing strategist	Business user	Decision Center Business console, deploys to the development environment

Frank	Analyst	Release manager	Decision Center Business and Enterprise consoles, including administrative functions, deploys to all environments
Arjun	IT operations specialist	Integrator	Decision Center Business console, deploys to development and test environments and performs decision service benchmark testing

Collaborating on the decision service

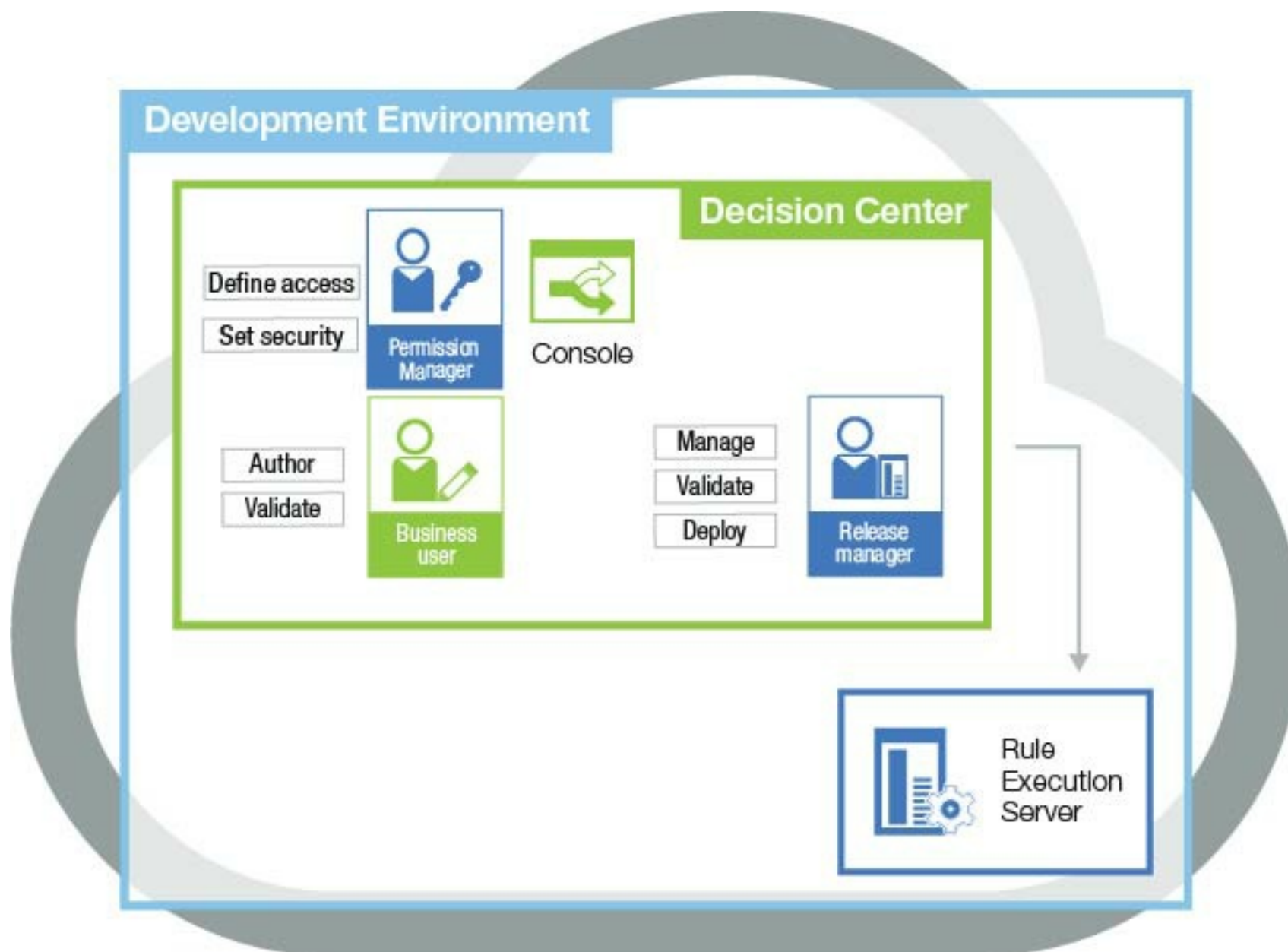
As the rule developer, Kim uses Rule Designer to create a decision service, deploy it to a runtime environment, and publish it to Decision Center. She knows Java™ and understands the company's object model. She works with software architects to develop the first version of the decision service, defining its vocabulary and writing the business rules that model the logic of the car rental business.

The following diagram shows Kim's activities. First, she creates the decision service and associated artifacts in Rule Designer. In working on the decision service, she tests it by deploying it to Rule Execution Server in the development environment. When she completes the initial version of the decision service, she publishes it to Decision Center.



As the release manager, Frank uses the decision governance framework in the Decision Center Business console. He creates a release for the decision service, and sets up change and validation activities in the release. When the team has completed the validation activities, and the rules are ready to be tested, Frank deploys the release to Rule Execution Server in the development environment.

In the following diagram, the business user, Gary, updates the decision service in Decision Center, and Frank deploys the decision service from Decision Center to Rule Execution Server.

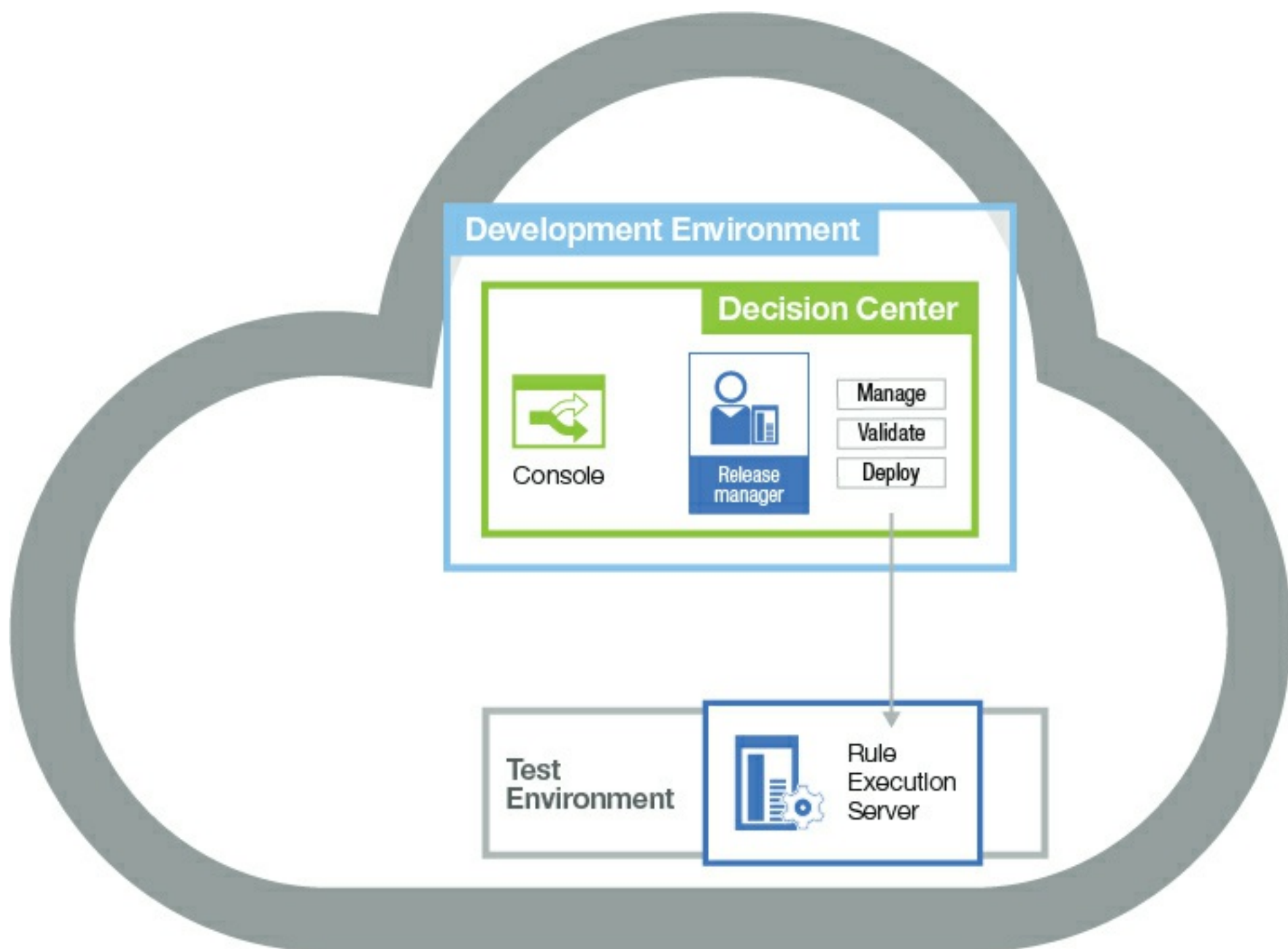


Because Gary works in the marketing department, he wants to review the pricing policies that are expressed in the rules. Gary views and edits the rules through the Decision Center Business console. Gary works in an activity branch that tracks his changes to the rules.

In addition to setting up the user accounts in the cloud portal, John also works as the permission manager in the Decision Center Business console. He sets the security and user access parameters in the administration tab. He makes sure that Gary, along with the other users who need access to the release and activity branches in the decision service, are assigned to the correct user groups.

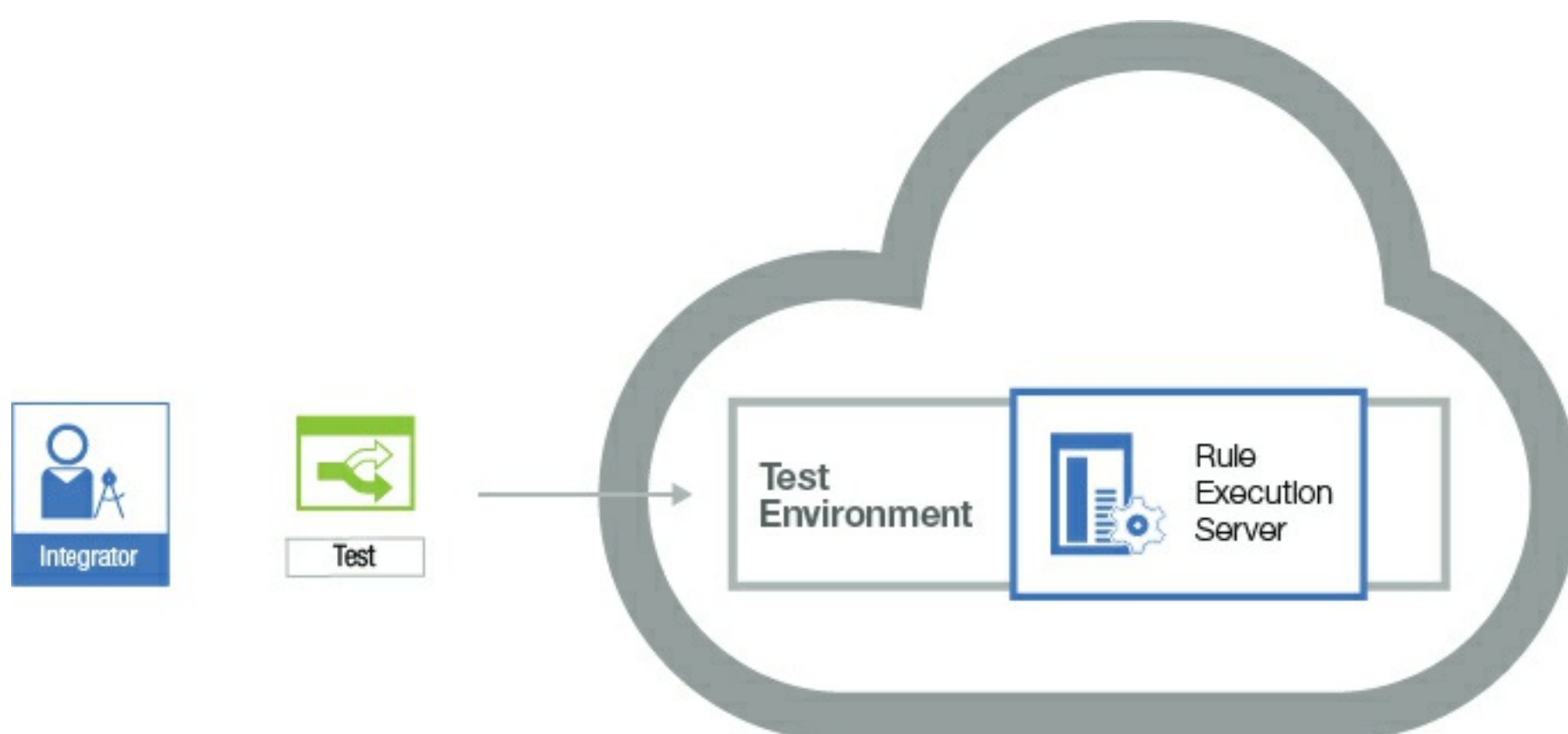
Testing and promoting the decision service

As the release manager, Frank has access to all three cloud environments. He can perform testing throughout the development lifecycle of the decision service. In the following diagram, when the team finishes the decision service in the development environment, Frank deploys it to the test environment.



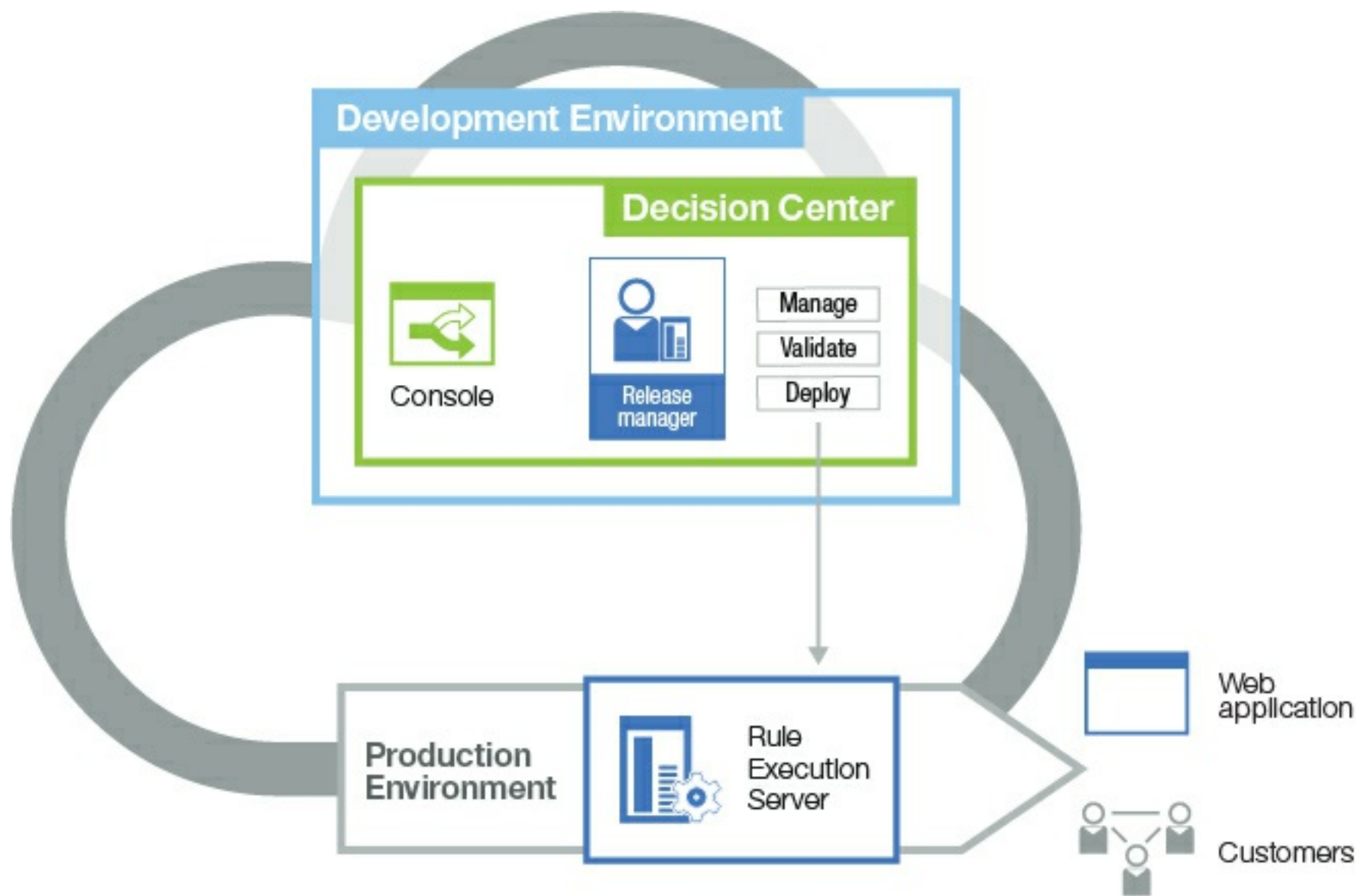
Frank works closely with Arjun, the integrator. To determine whether the decision service is running as expected, Arjun works in the Decision Center Business console and all three cloud environments. He monitors and evaluates the behavior of the decision service when it is called by a client application.

To evaluate the performance of the decision service in the test environment, Frank asks Arjun to run benchmarks that use standard software development validation procedures. Arjun adds code to a car rental web application to enable it to call the new decision service. In the following diagram, the car rental web application runs on the company application server and calls the decision service from the Operational Decision Manager on Cloud test environment.



When development is complete and Frank is satisfied with the benchmark results in the test environment, he promotes the decision service to Rule Execution Server in the production environment as the final step in the decision service lifecycle. As the release manager, Frank is the only user who can deploy a decision service to production.

The following diagram shows the promotion of the decision service to the production environment. When customers use the web application, it calls the decision service.



Gary can also tell his marketing colleagues that he can implement a new promotional pricing campaign next quarter because he can use Operational Decision Manager on Cloud to revise the pricing rules in the decision service.

Operational Decision Manager

Operational Decision Manager delivers a proven development system for capturing policies in solutions that automate the application of decisions.

Introduction

Operational Decision Manager provides cognitive technology that enables a business to respond to real-time data with intelligent, automated decisions. With Operational Decision Manager, IT and business users alike can manage the business decision logic that is used by operational systems within an organization.

Accessibility

Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with Operational Decision Manager.

Introduction

Operational Decision Manager provides cognitive technology that enables a business to respond to real-time data with intelligent, automated decisions. With Operational Decision Manager, IT and business users alike can manage the business decision logic that is used by operational systems within an organization.

Decision automation

Business agility depends on responsive, intelligent decision automation. Operational Decision Manager helps manage decisions separately from business applications, and with more flexibility and responsiveness to the changing needs of the business.

Decision services

Decision Server provides tools for designing, developing, and deploying decision services. Rule Designer is an Eclipse-based development environment in which you can develop and integrate decision services. Rule Execution Server provides the runtime environment for running and monitoring decision services.

Decision management

Decision Center is a scalable decision management application and repository with collaborative web environments for authoring, managing, validating, and deploying decision services.

Roles and activities

Each step of the rule development lifecycle requires specific skills that are held by different users. The roles of the users must be assigned at the outset of a project.

Parent topic: [Operational Decision Manager](#)

Decision automation

Business agility depends on responsive, intelligent decision automation. Operational Decision Manager helps manage decisions separately from business applications, and with more flexibility and responsiveness to the changing needs of the business.

The ability to deal with change in operational systems is directly related to the decisions that they are able to make. Every transaction, order, customer interaction, or process depends on decisions, which are in turn dependent on particular internal or external requirements and contexts. Every change therefore affects decisions, many of which are handled automatically within business systems.

Extracting decisions from your application code

Business policies are statements that are used to make decisions. These decisions can determine pricing for insurance or loan underwriting, eligibility approval for social or health services, or product recommendations for online purchases. Business policies are typically found inside application code, in the form of if-then statements. However, they can also be stored elsewhere for documentation purposes, such as in procedural manuals and other documents.



A business policy can be expressed as several business rules. Here is an example of the kind of business policy that might be familiar:

Customers who spend a lot of money in a single transaction need an upgrade.

The process of capturing rules happens in two steps. The first step consists in formalizing the vocabulary that is required to express the policy as a conceptual object model. The second step consists in representing the logic of the business policy as if-then statements.

After the vocabulary is created, the business policy can be implemented with the following business rule:

```
if
    the customer's category is Gold
    and the value of the customer's shopping cart is more than $1500
then
    change the customer's category to Platinum
```

When a business policy also has an IT policy or security policy that is embedded in it, you can combine business rule management with capabilities to handle the business policy aspects. For example, the following business policies can be handled as rules: *customers who spend a lot of money should be routed to a preferential service* or *customers who spend a lot of money require additional security on their transactions*.

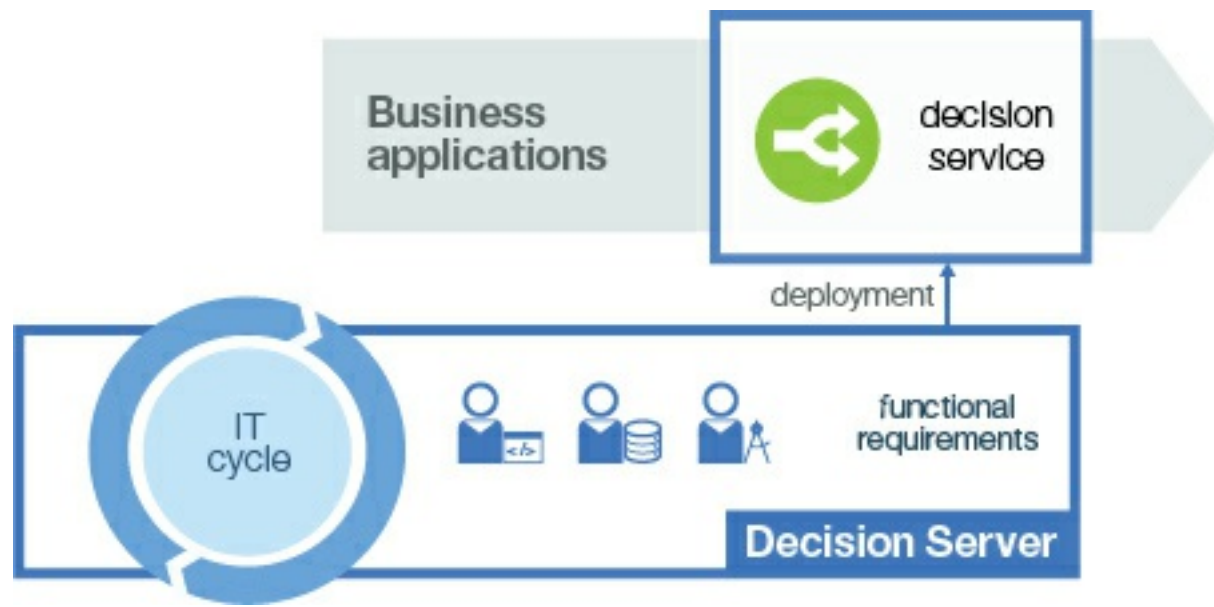
In the form of business rules, the business logic can be packaged and called from the application code as a decision service. Therefore, changes to the business policy do not require changes to the application or process code.

Managing decisions

When decision management is separate from application code, business experts can define and manage the business logic. Decision management reduces the amount of time and effort that is required to update the business logic in production systems, and increases the ability of an organization to respond to changes in the business environment.

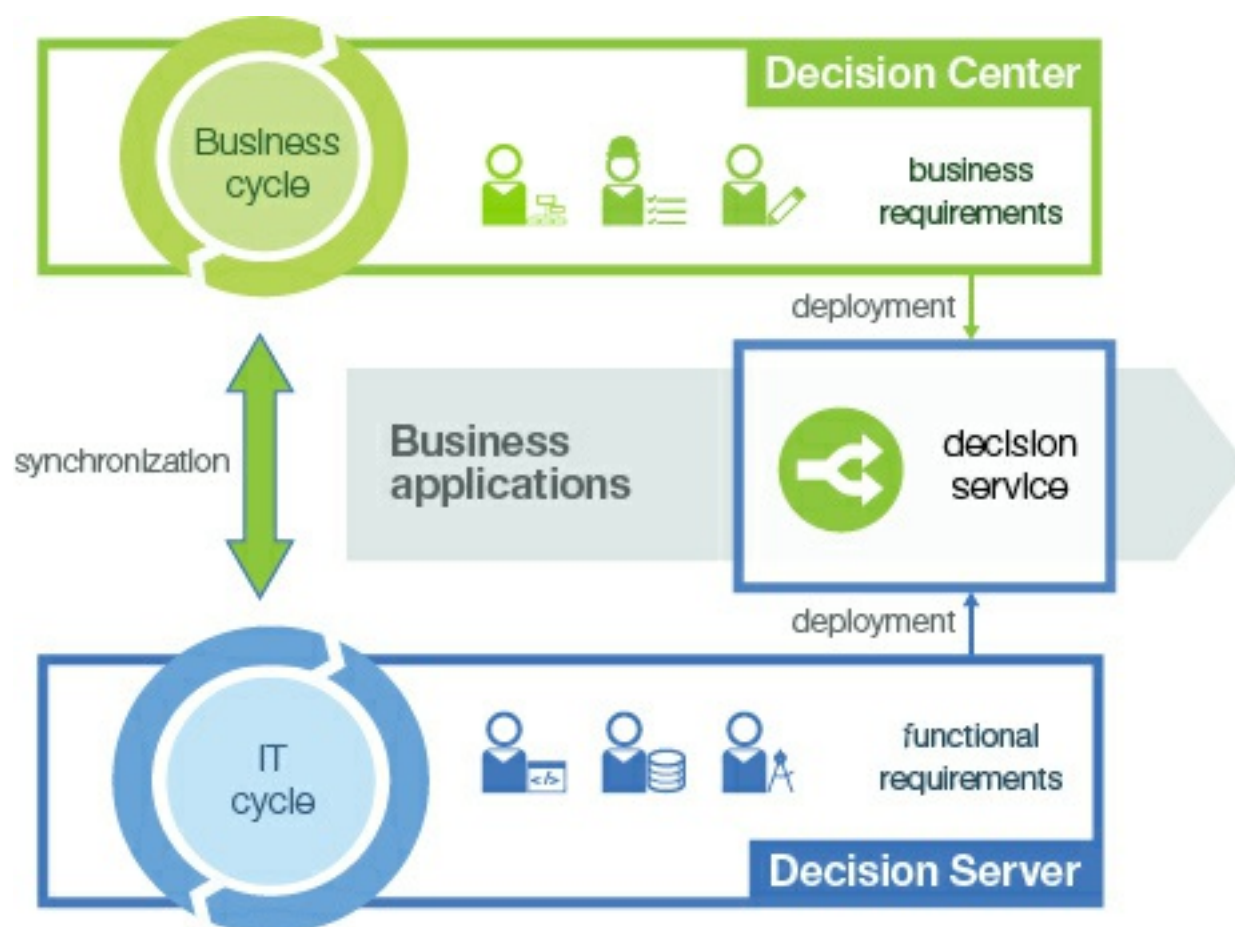
Operational Decision Manager provides an environment for designing, developing, and deploying decision services. The IT cycle consists of developing and maintaining this infrastructure. After the infrastructure is set up, distributed business teams can start collaborating through a web-based environment to create and maintain the decision logic.

Decision Server provides the runtime and development components to automate the response of highly variable decisions that are based on the specific context of a process, transaction, or interaction.



Putting decision management in the hands of business users

With Decision Center, business users can manage decisions that are directly based on organizational knowledge, with limited dependence on the IT department. Business users have a varying degree of dependence, which can range from a limited review to complete control over the specification, creation, testing, and deployment of the business logic. Business and IT functions can work in collaboration. The entire organization can align in the implementation of automated decisions. The maintenance lifecycle accelerates based on new external and internal requirements.



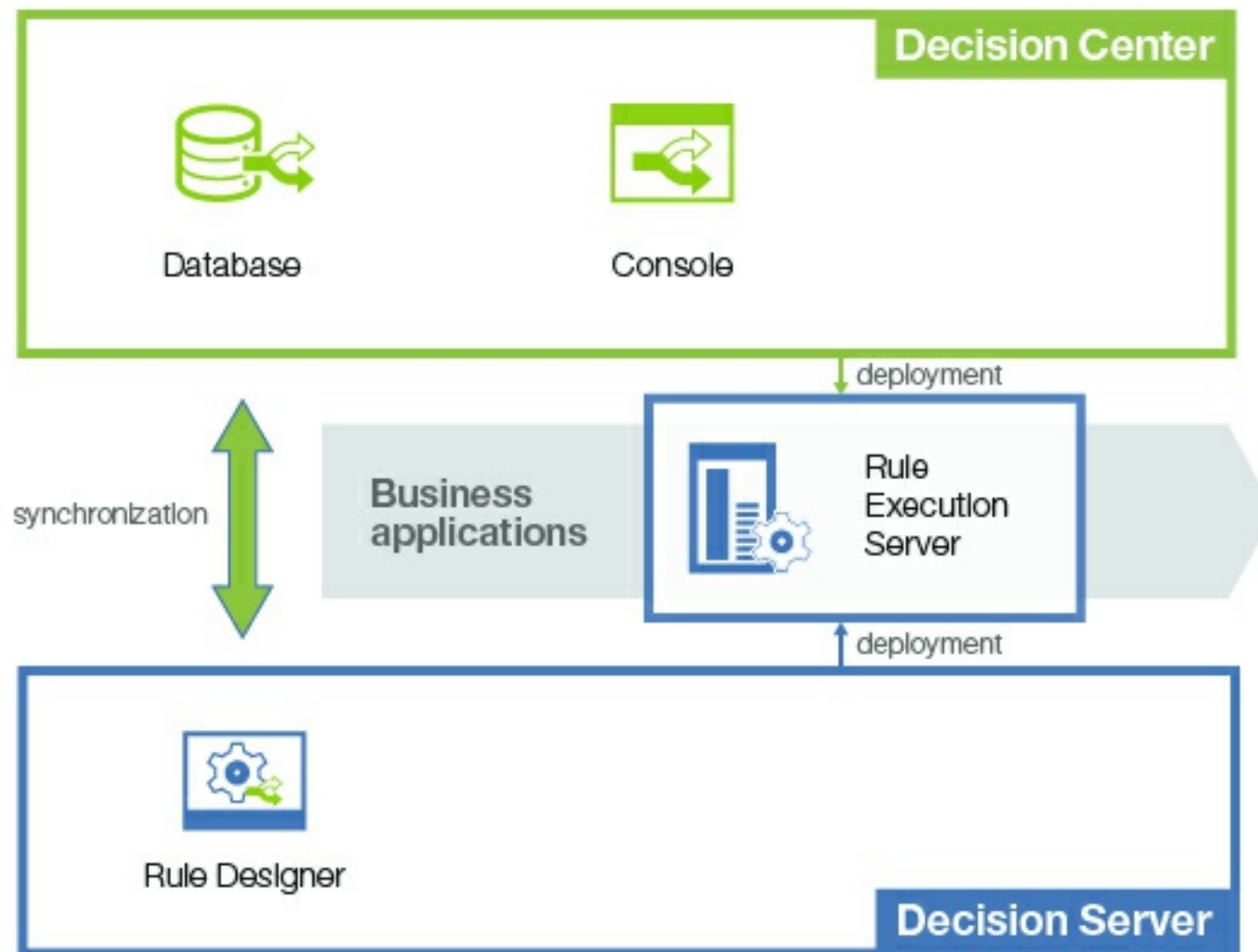
The lifecycles of decision service development and management can evolve in parallel. Decisions can evolve as required by the business context, without putting an extra load on the development of the business rule application. Each time the business rule application evolves, the decision management environment synchronizes with the development environment.

With this separation, decisions and application architecture can be managed asynchronously. For example, developers can develop a new version of a decision service in response to changing application infrastructure and core business requirements. At the same time, policy managers can work on new decisions that are delivered in response to an evolving market or a changing regulatory environment.

Operational Decision Manager components for decision management

The following figure shows the components that Operational Decision Manager provides for decision service development, rule management and authoring, and the execution environment.

IBM Operational Decision Manager



In addition to working on different timelines, developers and business users also expect to work with different tools, reflecting their different skill sets and views of the application. For example, developers are accustomed to the Java™ world. They use source-code management systems to work simultaneously on separate copies of a project without interfering with each other.

Business users do not concern themselves with the details of application development, but are interested in testing and managing decisions. Therefore, they need tools that can help them author, organize, and search for rules in the context of the overall policy.

With developers and business users that work in their own environments at their own pace, the work of these two groups must be synchronized and merged.

Finally, both developers and business users require access to a rule execution environment to deploy rules to enable testing, validation, and rollout to production of new and changed business logic.

Parent topic: [Introduction](#)

Related concepts:

[Decision services](#)

[Decision management](#)

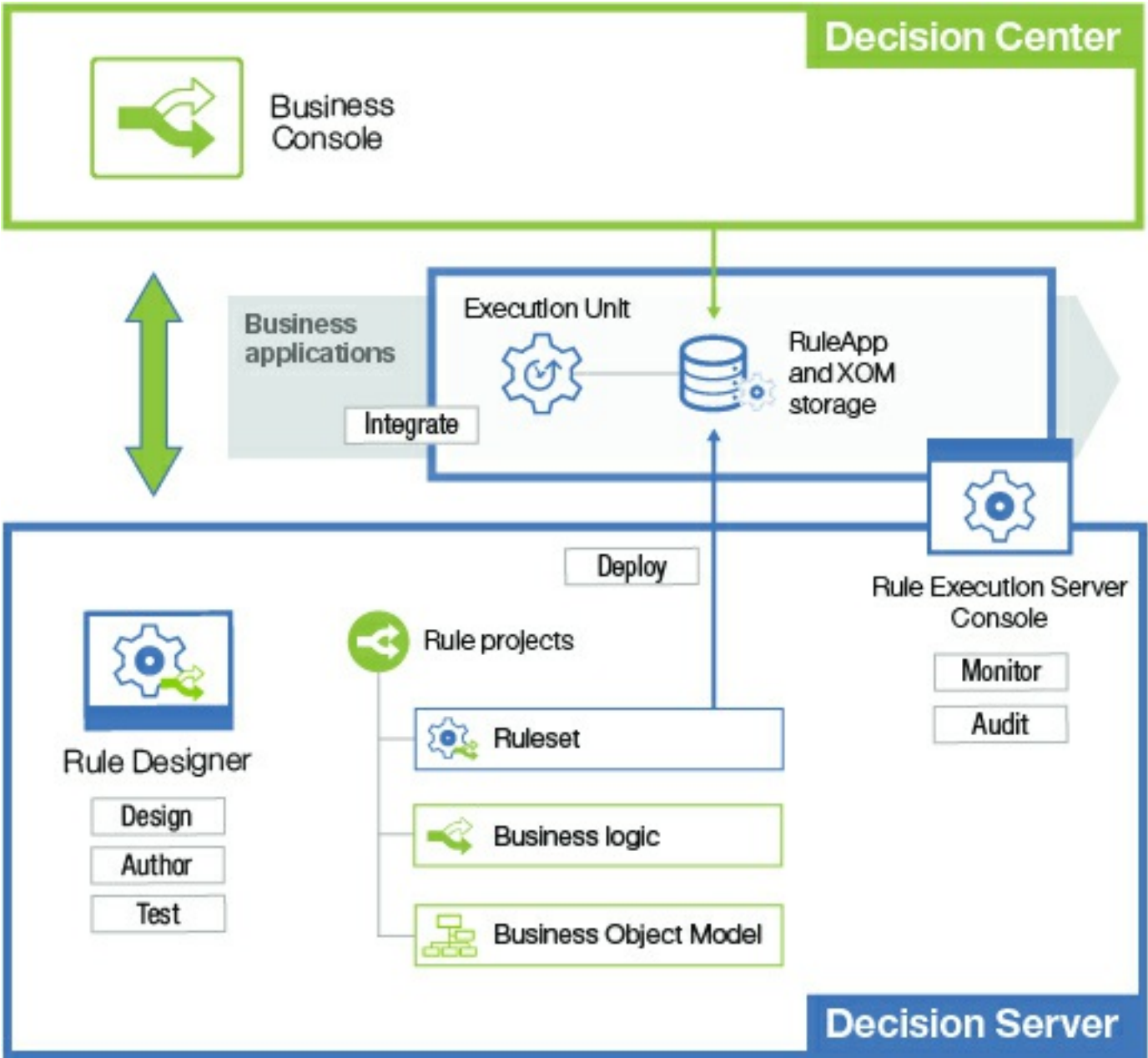
Related information:

[Roles and activities](#)

Decision services

Decision Server provides tools for designing, developing, and deploying decision services. Rule Designer is an Eclipse-based development environment in which you can develop and integrate decision services. Rule Execution Server provides the runtime environment for running and monitoring decision services.

The following diagram illustrates the different tools that you use to develop and deploy a decision service as a RuleApp, and the tasks you must do for this development.



Activity	Learn more
Get started To get started with the development tools and processes for business rules, follow the First Steps tutorial. To learn more about the architecture of business rule applications, read an overview.	Tutorials Introduction
Design In Rule Designer, you design the granularity of your decision services and the contract between client applications and the decision service. You also design the model and vocabulary for authoring business rules.	Designing projects for rule authoring
Author You create an initial set of business rules, and set up some tools to facilitate rule authoring for business users.	Authoring rules
Monitor You administer and monitor business rule applications by using the tools that are provided with Rule Execution Server.	Managing rule execution in Rule Execution Server

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision management](#)

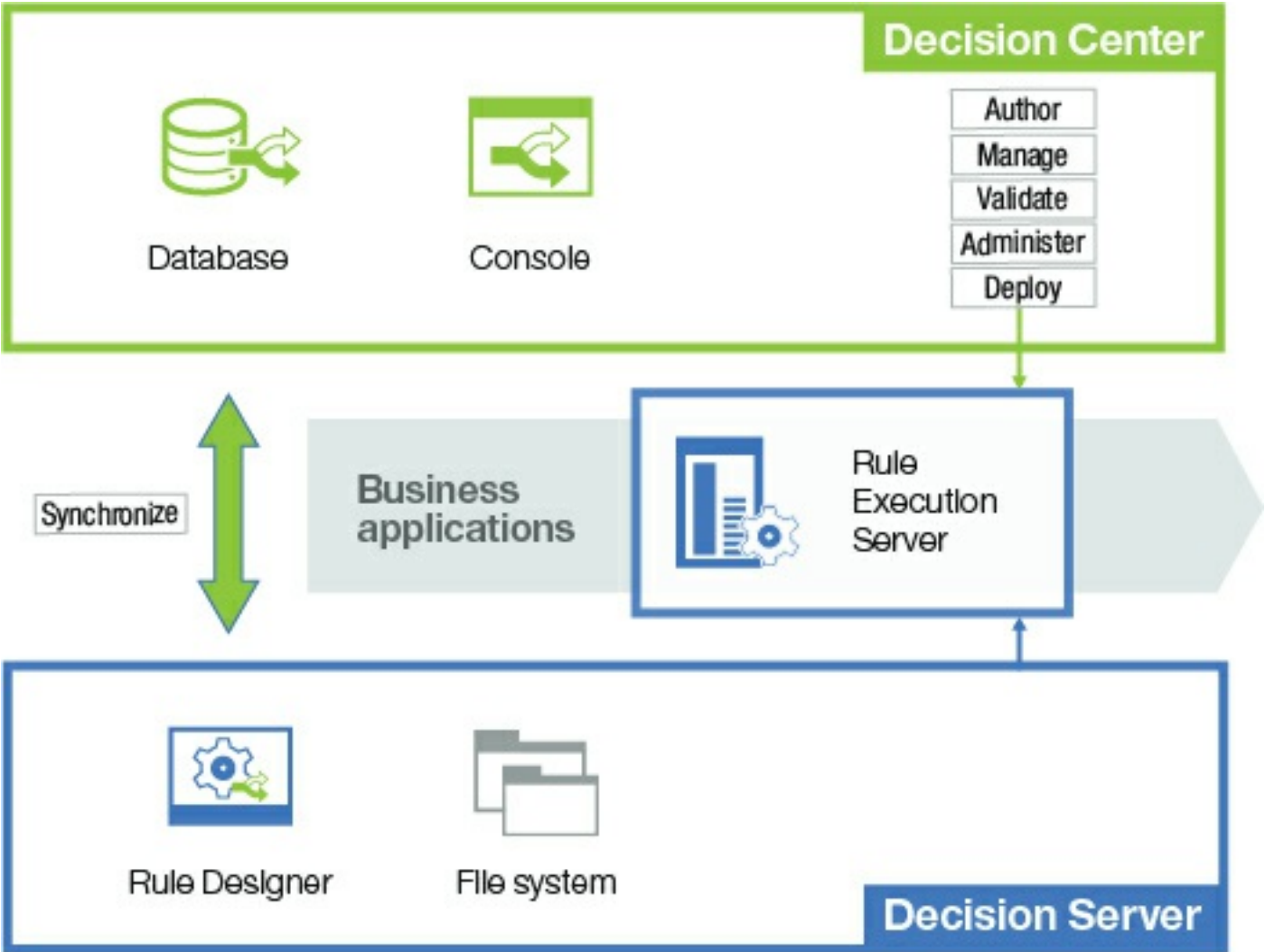
Related information:
[Roles and activities](#)

Decision management

Decision Center is a scalable decision management application and repository with collaborative web environments for authoring, managing, validating, and deploying decision services.

For decision services that are not too complex, business users can also create the underlying model on which decisions are authored.

The following diagram illustrates the main tools and tasks for decision management in Decision Center.



In the Business console, business users can work with a ready-to-use approach to change management and governance, which is based on releases and change activities. This approach is called decision governance.

Activity	Learn more
Get started To learn more about decision management in Decision Center, read the overview.	Decision Center
Configure For business users to manage automated decisions, you configure the projects and work environment. You can configure and customize the model and vocabulary that is used in the rules, and you can define permissions for different types of users. You also set up the deployment configurations. Part of the configuration can be done in Rule Designer, or in Decision Center for users with administrator privileges.	Designing projects for rule authoring
Synchronize Synchronization is the link between the IT cycle and the business cycle. When the projects are ready on the IT side, you can publish them to Decision Center. What you publish and the options you set when publishing have an impact on how rules are managed in the business side. If you publish a decision service, you can choose whether the decision service is managed with regular branches, or with the decision governance framework in the Business console.	Synchronizing and storing rules Managing changes

	nges
Manage To manage decision services, you can use the decision governance framework, a predefined release workflow that is based on change and validation activities. You can also choose to use regular branches for a customized change management workflow. The decision governance framework is only available in the Business console.	Overview: Identifying a set of rules Managing changes
Author Business users can author action rules and decision tables, and Decision Center tracks versions of rules. With the decision governance framework, you must create a change activity within a release to modify rules. Policy managers and rule authors can author business rules in the Decision Center Business and Enterprise consoles. You can integrate business rule authoring and management extensions that are developed in Rule Designer into the Decision Center consoles.	Working with rules
Validate Decision Center includes testing and simulation features that enable business users to validate the behavior of rules. Business users must be confident that business rules are written correctly and that any update does not break the business logic encapsulated in the ruleset. Business users validate the business logic against well-defined usage scenarios, by running tests and simulations against their rules.	Testing sets of rules Simulating business application results Decision Center RES T API

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision services](#)

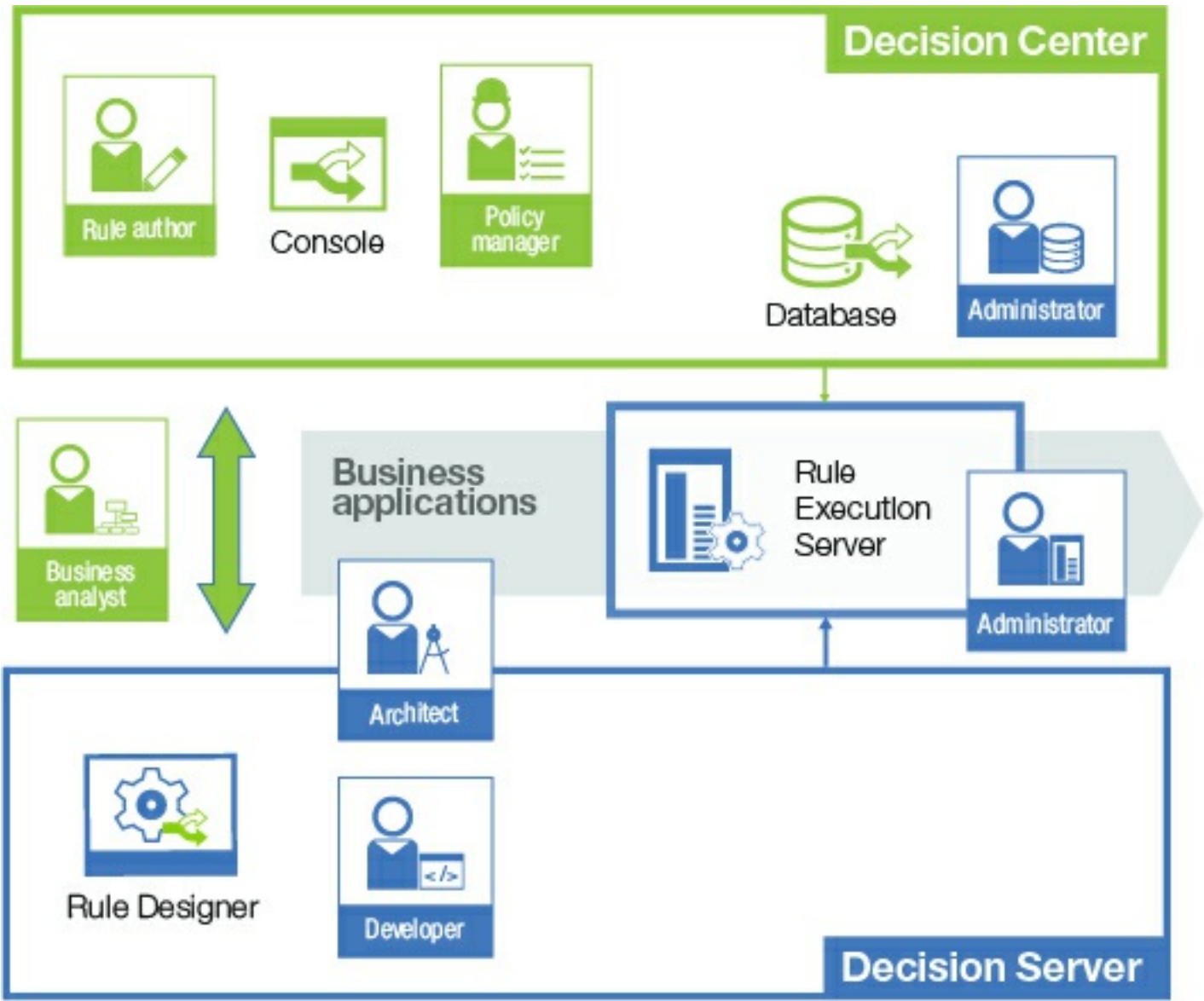
Related information:
[Roles and activities](#)

Roles and activities

Each step of the rule development lifecycle requires specific skills that are held by different users. The roles of the users must be assigned at the outset of a project.

Decision Server Rules and Decision Center

Together, Decision Server Rules and Decision Center provide a comprehensive decision management environment. The following figure shows the various development roles and their places in the management environment.



The roles can be broken down into two categories:

IT users



The architects, developers, and administrators who develop and maintain the rule application.



Business users


The business analysts, policy managers, and rule authors who develop and maintain the decision logic.

The following table lists the different types of IT and business users.

Table 1. User roles for the development of business rule applications

Role	Activities	Description
	<ul style="list-style-type: none">DesignIntegrateDeploy	<p>Architects work mainly in Decision Server to do the following tasks:</p> <ul style="list-style-type: none">Define the decision services and their interface with the calling applications.Define the project organization for the developers and business users. For example, they set up the data model for the rule vocabulary.Optimize rule execution.
	<ul style="list-style-type: none">Design	<p>Developers know the object models, APIs, and the development environment. They work mainly in Decision Server to do the following tasks:</p>

	<div>g n</div> <ul style="list-style-type: none">• A u t h o r• T e s t• I n t e g r a t e• D e p l o y	<ul style="list-style-type: none">• Develop, test, debug, and deploy decision services. They contribute to the design of the rules.• Define the client execution code to integrate decision services in to business applications.
	<div>D e s i g n</div> <ul style="list-style-type: none">• A u t h o r• T e s t• S y n c h r o n i z e• M a n a g e• V a l i d a t e	<p>Business analysts work with business and IT departments, from design to integration of a software application. They work in Decision Server and Decision Center to do the following tasks:</p> <ul style="list-style-type: none">• Design a formal specification for the rules, with validation from both developers and policy managers.• Define the vocabulary for the rules.• Identify candidate business rules.• Write and organize business rules so that rule authors can maintain them.• Validate rules to make sure they produce the expected results. <p>Business analysts can share tasks with developers, but typically they do not write code.</p>
	<div>M a n a g e</div> <ul style="list-style-type: none">• V a l i d a t e• A u t	<p>Policy managers own the decisions within an organization. They work mainly in Decision Center to do the following tasks:</p> <ul style="list-style-type: none">• Participate in the design of a formal specification for the rules.• Define vocabulary elements with the help of business analysts.• Manage releases and validation activities in the context of decision governance.• Create and update rules.• Review how the running of rules is orchestrated.• Report on the status of the decisions.• Test rules to ensure that they provide the intended business outcome. <ul style="list-style-type: none">• Run simulations to assess the potential impact of changes to rules.

	h o r	
	• A u t h o r	<p>Rule authors work in Decision Center to do the following tasks:</p> <ul style="list-style-type: none">• Update and sometimes create rules.• Review business rules.• Create change activities in the context of decision governance.

Parent topic: [Introduction](#)

Related concepts:
[Decision automation](#)
[Decision services](#)
[Decision management](#)

Accessibility

Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with Operational Decision Manager.

This product supports the following accessibility features:

Keyboard shortcuts

You can navigate through the different Operational Decision Manager environments and their documentation using keyboard shortcuts. The rule management environments, such as Eclipse, provide documentation on their accessibility features. You can also find topics that cover keyboard shortcuts in the documentation for Operational Decision Manager.

Magnification

You can use the settings of display systems to magnify development interfaces and supporting documentation.

Screen reading

You can use a screen reader with a digital speech synthesizer to read aloud what is displayed on your screen. Consult the documentation with your assistive technology for details on using it with this product suite and its documentation.

Technical documentation

The IBM® product documentation is available in an online environment that you can access by using a web browser. See [IBM Knowledge Center release notes](#).

Keyboard shortcuts

The Operational Decision Manager modules provide keyboard shortcuts. You can find information on the keyboard shortcuts in other topics and the product interfaces.

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see [IBM Accessibility](#).

Parent topic: [Operational Decision Manager](#)

Key components

Operational Decision Manager on Cloud includes the main components in Operational Decision Manager.

Decision Server

Decision Server provides development and production components for creating and running rule applications.

Decision Center

Decision Center supports collaborative development and management of decision services.

Version differences

Operational Decision Manager on Cloud has fewer features than the on-premises version of Operational Decision Manager.

Parent topic: [Overview](#)

Decision Server

Decision Server provides development and production components for creating and running rule applications.

The following Decision Server components are in Operational Decision Manager on Cloud:

- Rule Designer: An Eclipse-based development component for designing, authoring, testing, and deploying rule applications.
- Rule Execution Server: The execution component for running and monitoring rule applications.

In a rule-based solution, a client application requests a decision from a rule application. The client application can contain many decision points that use the rule application. At each decision point, business rules in rulesets express policies for decisions. A rule application is deployed to Rule Execution Server as a RuleApp. Each RuleApp contains one or more rulesets, and each ruleset corresponds to a decision.

Decision Server also interacts with Decision Center, which includes a rule repository and collaborative web consoles for business and technical users to author, manage, validate, and deploy rules.

The design and development phases of rule applications are managed through rule projects. In Operational Decision Manager on Cloud, you work primarily in rule projects called decision services. A decision service can support complex decisions that involve several rule projects. Synchronization, branching, and change management are applied to all the rule projects in a decision service hierarchy, allowing the decision service behavior to be governed and deployed consistently.

Creating

In Rule Designer, you create a rule application and set up its connection with the client application. You also create the model and vocabulary for authoring business rules.

To create a rule application, you must put in place the necessary infrastructure for editing rules and producing one or more rulesets:

- You define the business rules that make up an executable decision unit, or ruleset. The ruleset uses input and output parameters to pass data to and from the client application. You give each ruleset a unique signature of input and output parameters. In decision services, you create decision operations that define the contents and signatures of rulesets. Because you can use the same rule projects and packages in different decision operations, you can easily change a decision service to add new decision points that require different signatures.
- You define the vocabulary that is used in the business rules. In Rule Designer, you develop the business object model (BOM) that defines the elements and relationships in the vocabulary. You can define the vocabulary by mapping the BOM to the execution object model (XOM). You can also create the vocabulary by generating the BOM from the XOM, and then configuring the business vocabulary from the BOM.
- You set up a rule project hierarchy. A rule project is a type of Eclipse project that is dedicated to the development of rule applications. In a decision service, a main rule project serves as the top-level project so that the decision service behavior can be governed and deployed consistently.
- You organize rule packages in rule projects to store business rules and define a ruleflow to specify the order for executing rules.
- You set up business user validation tools by configuring and customizing tests and simulations.

Authoring

If you are responsible for creating and managing the rules, you might author most of the business rules in a project. If business users are responsible for creating and managing the rules, you set up tools to facilitate rule authoring for them. You can create the following types of business rules:

- Action rules
- Decision tables

These business rules use the Business Action Language (BAL), which is designed to have a natural language syntax. You can also create technical rules, which are based on the ILOG® Rule Language (IRL) and require programming skills.

Authoring might require you to do the following operations:

- Create business rule templates to facilitate the creation of rules that use a common pattern.
- Define vocabulary categories to filter the vocabulary elements that are available when you author business rules.

Debugging and testing

You can debug a ruleset in Rule Designer, and test that the decision service or a project implements the expected business logic:

- You debug the ruleset by using a rule engine to manage rule execution.
- You analyze rules by using constrained semantic queries, which check the consistency and completeness of individual rules and the ruleset as a whole.
- You run test scenarios on rules. You can run these tests locally in Rule Designer, without the entire Rule Execution Server component.

Integrating

You integrate the client application from Rule Designer.

After the ruleset connection is defined, you host the ruleset on Rule Execution Server and call it from the client application. Rule Execution Server is the execution component for deployed business rules. It handles the creation, pooling, and management of ruleset instances so that client applications can call the decisions as easily as possible.

To call the ruleset from the client application, you post the ruleset as a hosted transparent decision service (HTDS) that is called through web service protocols.

Deploying

A RuleApp holds a group of rulesets that are deployed together. The RuleApp also contains the input and output parameters that define the interaction with the client application, and the ruleset path needed by the client application to identify rulesets and their versions.

When a decision service is fully validated and approved, you can deploy it to Rule Execution Server for use in production. You deploy the decision service by using a deployment configuration that defines what to assemble in a RuleApp and where to deploy it. A deployment configuration can reference one or more decision operations. A decision operation includes all the settings needed to produce a ruleset, including input and output parameters, main ruleflow, and any rule extraction. The deployment configuration can be synchronized with Decision Center, and subject to change management and governance in the Business console.

Parent topic: [Key components](#)

Decision Center

Decision Center supports collaborative development and management of decision services.

Business users can work in Decision Center to apply organizational knowledge and best practices in decision services, with little dependence on developers. Decision Center includes a rule repository and web consoles for authoring, managing, validating, and deploying rules.

Rule editing

Business users work primarily in the Decision Center Business console. They use rule editors that ensure that they use the correct business vocabulary and comply with proper business rule syntax.

There are two consoles in Decision Center:

- Business console: The primary environment for change management and deployment of decision services (see [Governing rules with the Business console](#))
- Enterprise console: The environment in which administrative users do certain tasks (see [Managing business rules with the Enterprise console](#))

Synchronization

Developers and business users work on rules in different environments, and save the rules to different locations. Developers typically work in source code control, and business users save to the Decision Center repository.

To enable collaboration between the development and business cycles of rules, you must use the synchronization tool in Rule Designer to synchronize the work that is done by the different users (see [Synchronizing and storing rules](#)).

Deploying rules

When you complete the development and testing of rules, you can deploy them to Rule Execution Server, which is the runtime execution component. In Decision Center, you deploy from the Business console. You can also deploy rules by using the [Decision Center REST API](#).

Administrators can create and edit deployment configurations in decision services. These deployment configurations can then be used to deploy releases, and change activities and regular branches in the decision services. When you deploy decision services, the XOM is also deployed.

Decision governance framework

Decision Center offers a ready-to-use approach to change management and governance that is based on decision services, releases, and activities. It encourages collaboration, and enables more advanced users to check the work of other users.

The decision governance framework is the preferred way of working and is presented as such to the business users. You should be familiar with the following sections:

- [Change management](#)
- [Governance principles](#)

Validating rules

You can validate the behavior of rules or assess the effects of rule changes. The Decision Center Business console can run tests and simulations:

- Tests: Compare expected results with the actual results from applying rules to usage scenarios.
- Simulations: Run rules against representative data to determine how changes to the rules affect key performance indicators.

You can also run tests with the [Decision Center REST API](#).

Parent topic: [Key components](#)

Version differences

Operational Decision Manager on Cloud has fewer features than the on-premises version of Operational Decision Manager.

Both the cloud and on-premises versions of Operational Decision Manager include collaborative components. However, the following on-premises features are not available in the cloud version (note that some of these features are also no longer available in the on-premises version):

- Decision Server Insights
- Classic rule projects
- Classic rule engine
- Rule Solutions for Office
- Decision Warehouse
- Decision Validation Services (DVS)
- Customization of Decision Center and Decision Server
- Java™ API
- Ant scripts
- .NET platform
- Sample server and samples console
- Operational Decision Manager tutorials
- B2X or XOM methods that make outbound calls to a database or a service
- COBOL and PL/I support
- Access to the server operating system
- Access to the application server console
- Repackaging of EARs (no customization)
- Rule model extension
- Custom value editor
- Custom data provider for dynamic domains
- Custom data provider for testing and simulation
- Monitored Transparent Decision Service

Parent topic: [Key components](#)

Cloud environments and user roles

The components for creating and using rule applications are kept in cloud environments. Access to the environments is controlled through user roles.

Each Operational Decision Manager on Cloud portal contains cloud environments. The standard cloud environments are development, test, and production.

Each environment covers part of the decision service lifecycle, and has the appropriate Operational Decision Manager components for the work. For example, you create decision services in the development environment, which contains Rule Designer, Rule Execution Server, and Decision Center.

Access to the environments and their components is controlled through user roles, which are assigned based on the type of work. The role of business user, for example, can limit a user to modifying decision services in the Decision Center Business console.

The following table associates the user roles with the different cloud environments. The first environment is for creating decision services, the second is for testing decision services, and the third is for making decision services available to client applications.

Table 1. Table shows how the access of cloud user roles can be limited to environments and components.

User role	Development environment			Test environment	Production environment
	Rule Designer	Decision Center	Rule Execution Server	Rule Execution Server	Rule Execution Server
Rule developer	X	X	X	-	-
Release manager	X	X	X	X	X
Business user	-	X	X	X	-
Integrator	-	X	X	X	X
Permission manager	-	X	-	-	-

Cloud environments

Operational Decision Manager on Cloud comes with cloud environments for developing, testing, and calling decision services.

User roles

Access to the environments and components is controlled through user roles.

Cloud environments

Operational Decision Manager on Cloud comes with cloud environments for developing, testing, and calling decision services.

Development Environment(s)

Rule developers, release managers, business users, integrators, and permission managers use this workspace to collaborate on the development of decision services. It includes Decision Center, the main component for collaborative development and governance. The workspace also includes at least one instance of Rule Execution Server, the execution environment for running rulesets deployed from decision services. These activities can include functional tests and simulations in Decision Center, or a sample application that calls a ruleset. A development version of a production application can be used to obtain feedback from domain users during the development phase. Rule developers create decision services in Rule Designer, and then publish the decision services to Decision Center.

Test Environment(s)

Business users and integrators use one or more instances of Rule Execution Server in this workspace to test rulesets from decision services during or after development, and within the governance framework that is defined by the release manager. Tests can include nonfunctional tests for performance, system integration, or other user acceptance requirements. For example, a nonproduction web application can be used as a production application to run a ruleset on simulated data to validate the results. In this workspace, you can use a load testing harness to stress a ruleset, and measure its performance and scalability.

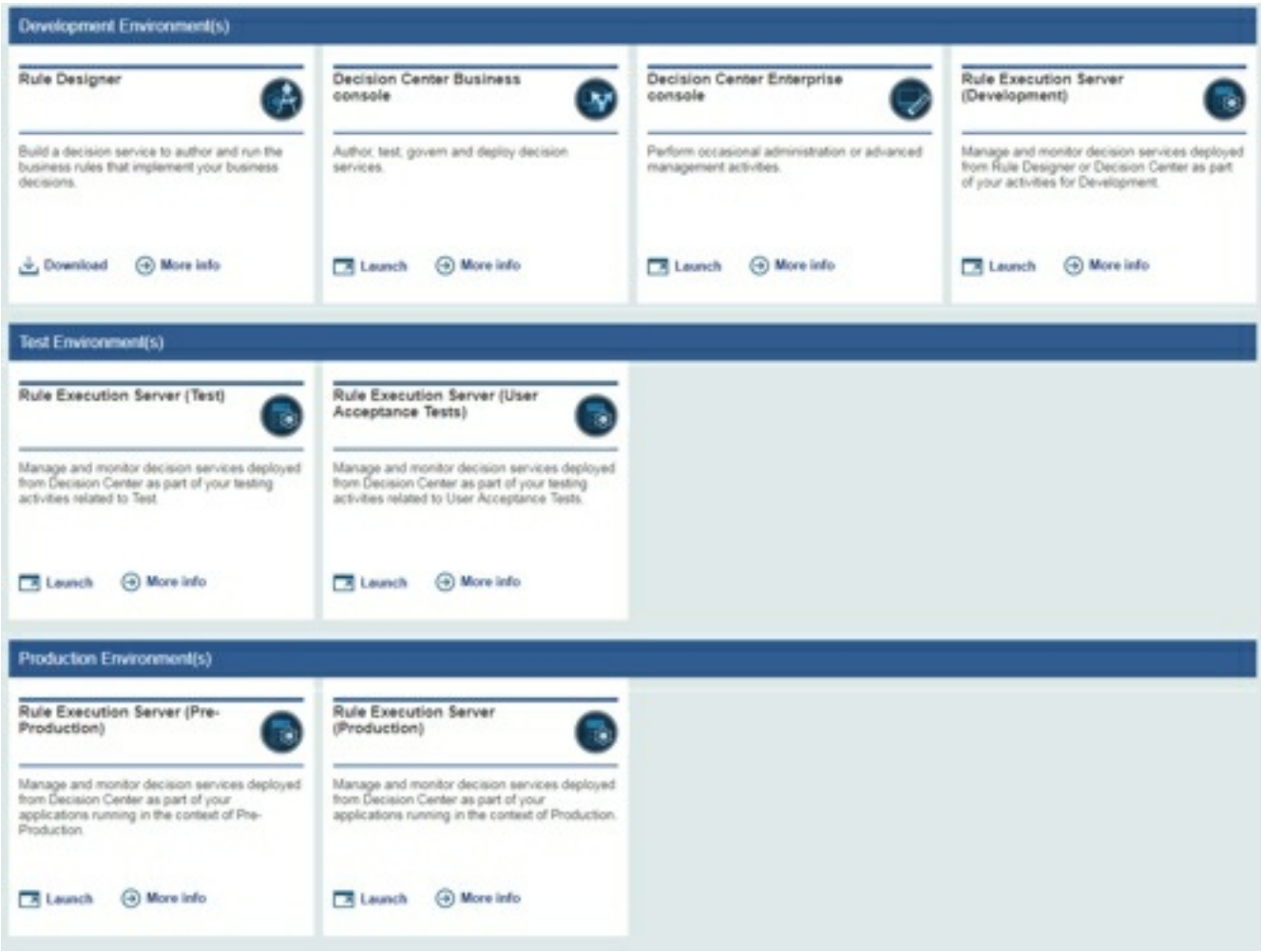
Production Environment(s)

Release managers use one or more instances of Rule Execution Server in this workspace to run rulesets from decision services for production applications. This workspace enables an application to call a ruleset to implement a decision-based business function. This workspace is not for development or tests. It is restricted to release managers, who are ultimately responsible for validating, completing and deploying a ruleset to production. Some user roles can view what is deployed, but some roles cannot access this workspace.

Adding execution environments

You can add additional instances of Rule Execution Server to your environments. During the provisioning of your cloud portal, you determine how many execution servers are needed for your work. You can also purchase more execution servers for an existing portal by contacting your IBM® representative.

The following image shows a cloud portal that has five instances of Rule Execution Server:



Parent topic: [Cloud environments and user roles](#)

User roles

Access to the environments and components is controlled through user roles.

Operational Decision Manager on Cloud has two groups of users.

Cloud group

At the cloud level, there is only one user role:

Administrator

The administrator manages the list of users and assigns roles in the Operational Decision Manager on Cloud portal. The administrator also creates service credentials for authenticating client applications. Each cloud portal must have at least one administrator, and only an administrator can remove another administrator. Most users do not have this role.

ODM group

The following user roles collaborate on modeling, authoring, governing, deploying, and integrating decision services:

Rule developer

- Works primarily in Rule Designer and the development environment.
- Creates the model of a decision service.
- Uses the Rule Designer component to convert the knowledge from the business domain into IBM® ODM artifacts.
- Makes the initial version of the business rule artifacts, including action rules, decision tables, and ruleflows.
- Runs the decision service locally or in the cloud development environment until the expected results are achieved.
- Publishes the decision service from Rule Designer to Decision Center.
- Collaborates with the release manager and business users in authoring and governance activities.
- Collaborates with the integrator to integrate the decision service into an application.
- Creates deployment configurations in Rule Designer for the development, test, and production environments.
- Can deploy a decision service to the development environment, but cannot deploy to the test or production environment.

Release manager

- Works primarily in the Decision Center Business console component.
- Orchestrates the lifecycle of a decision service, and is responsible for the deployment of a decision service release to production.
- Follows a staged progression from development to production.
- Creates development branches or releases.
- Defines change and validation activities for rule developers, business users, and integrators.
- Assigns ownership for work, reviews, and approvals.
- Can create deployment configurations in the Business console for the development, test, and production environments.

Business user

- Works primarily in the Decision Center Business console component.
- Implements and maintains some or all of the business rule artifacts that are in a decision service.
- Runs functional tests and simulations in the development environment to validate the changes that are made for a release.
- Can deploy a decision service to the test environment to validate changes in a test application. Can also deploy to the development environment.
- Can participate in the review or approval process for other business users or integrators.

Integrator

- Works primarily in the Decision Center Business console component, and uses other development, integration, and test tools.
- Builds the client applications that call a decision service in the development, test, and production environments.
- Sets up performance and reliability test applications.
- Can be involved in the validation activities that are defined by the release manager.
- Can deploy decision services to the development and test environments.
- Can view and use the decision services that are deployed in the production environment.

Permission manager

- Works primarily in the Decision Center Business console.
- Implements security on decision services.
- Creates groups, sets the permissions, adds users to the groups, and sets the groups on decision services.
- Can create deployment configurations in the Business console for the development, test, and production environments.
- Can deploy decision services from any cloud environment.

- Can hold all the ODM roles when, for example, a team is small.

The following table associates the ODM roles with some of the key activities.

The last column uses the following abbreviations:

- Dev: development environment
- Test: test environment
- Prod: production environment

Table 1. Table summarizes the user roles.

ODM role	Creates branches	Edits rules	Edits vocabulary (XOM/BOM)	Deploys to cloud environments (Dev/Test/Prod)
Rule developer	X	X	X	Dev
Release manager	X	X	X	Dev/Test/Prod
Business user	X	X	-	Dev/Test
Integrator	X	X	X	Dev/Test/Prod
Permission manager	X	X	X	Dev/Test/Prod

Parent topic: [Cloud environments and user roles](#)

Related concepts:
[Governance principles](#)

Related information:
[Roles and activities](#)

Express

Operational Decision Manager on Cloud Express® covers the entire lifecycle of a decision service in one environment.

It includes all the components for creating a decision service, developing it collaboratively, and running it for a client application. Unlike the complete Operational Decision Manager on Cloud, Express does not include separate environments that are dedicated to integration testing and production applications.

Tip: You can add a test or production environment to your Express instance. To add both environments, you must purchase the full IBM® Operational Decision Manager on Cloud. The contents of an Express instance can be migrated to the full version.

Introduction

Express is intended for non-mission critical rule applications. These applications have fewer than 500 artifacts and do fewer than a million executions per month. They address simple problems such as input validation and project assignment. These applications do not do the complex, load-heavy decision-making found in, for example, loan processing or order management.

The Express environment is similar to the development environment in the complete Operational Decision Manager on Cloud. The functionality of the components is the same as the functionality of the components in the complete version.

Express includes the user roles and permissions of the full version. However, every user has all the permissions in Express. The roles and permissions are included to facilitate migration to the full version, where they control user access.

Client applications can use the execution server in the Express environment. However, it is better to keep development and production in separate environments. The execution servers in the complete Operational Decision Manager on Cloud meet the specific development, testing, and production demands of rule applications.

Express is fully provisioned by IBM, and includes the following components:

Rule Designer

You use Rule Designer to create decision services. Based on Eclipse, Rule Designer comes with features for designing and assembling decision services. You start in Rule Designer, and typically publish decision services to Decision Center for final development.

Decision Center Business console

You work with colleagues to fully develop and maintain decision services in the Business console. This component includes testing and simulation features to validate rules, and the decision governance framework for managing projects.

Decision Center Enterprise console

Express includes the Enterprise console primarily for file management. It includes development features for advanced users, but development work is usually done in the Business console.

Rule Execution Server console

You test and run rulesets from decision services in Rule Execution Server as part of the development process. When completed, the rulesets can be used by production client applications. Both development and production share this execution server.

Comparison

Express is the small-solution option to the full IBM Operational Decision Manager on Cloud. If your development needs change, you can upgrade easily from Express to the full version with minimal changes to your existing projects.

The following table compares the two offerings by feature:

Table 1. Comparison table

Feature	Express	IBM Operational Decision Manager on Cloud
Decision service creation and development	Yes	Yes
Online collaboration	Yes	Yes
User access security	Yes	Yes
Number of users	Limited (5 concurrent users)	Depends on the conditions of the subscription
Workflow governance	Yes	Yes
System/environments	1	3+
Role-based access permissions	No	Yes

permissions		
Production workload	Limited (1 million executions per month)	Very high (licensed in units of 1 million executions)
Number of rules	Performance reduced above 100	High
Amount of data	Limited (500 artifacts)	High
High availability	No	Yes
Service availability guaranty	None	99.93%

Cloud Services terms

For the Cloud Services terms for Express, see [IBM Operational Decision Manager on Cloud Express](#).

Parent topic: [Overview](#)

Hybrid cloud environments

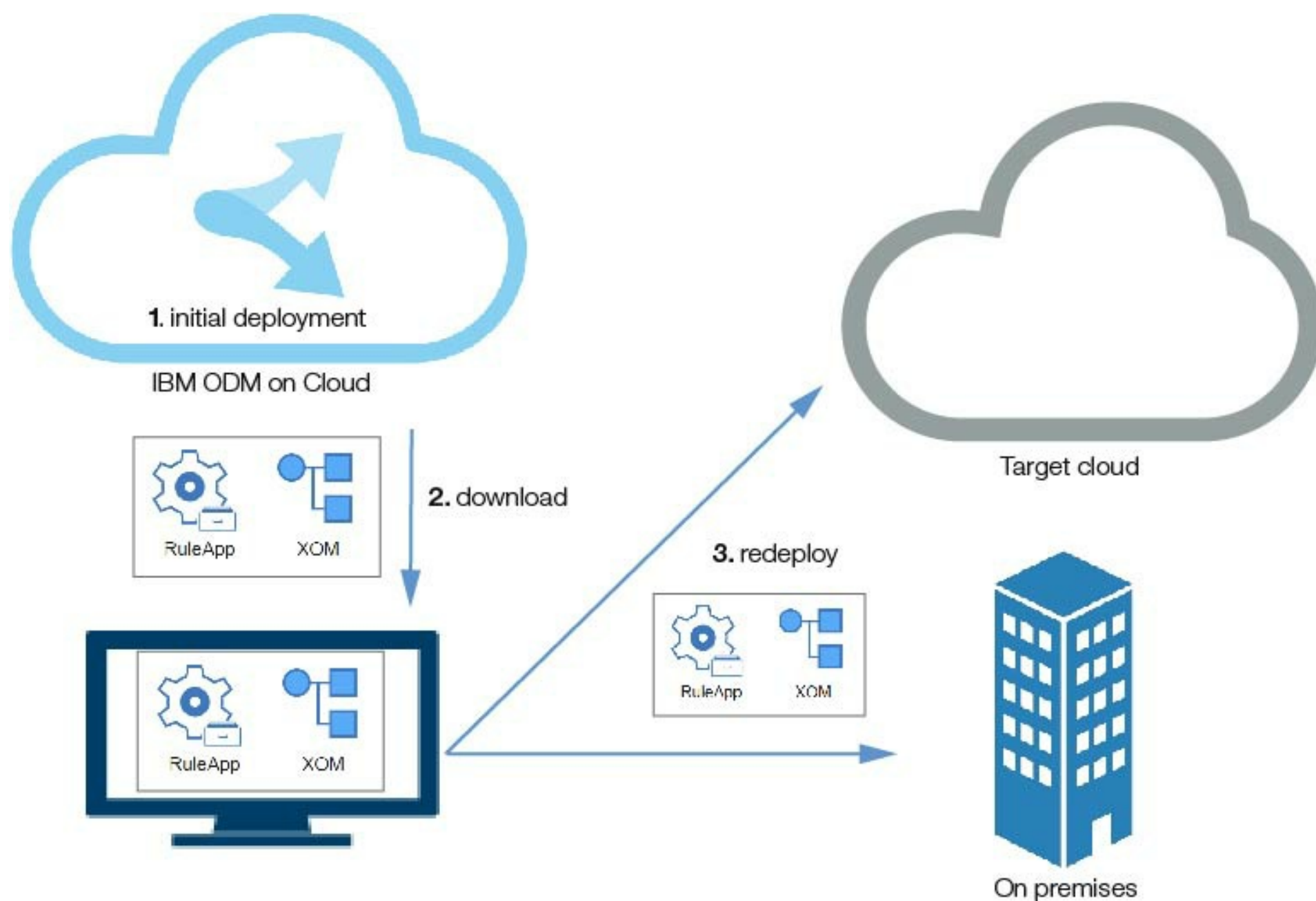
Complement your on-premises or cloud development system with Operational Decision Manager on Cloud.

Hybrid cloud environments bring together complementary on-premises and cloud systems. These hybrid solutions offer a flexible allocation of systems that delivers high levels of availability, performance, and security.

Operational Decision Manager on Cloud supports hybrid cloud environments that bring Operational Decision Manager on Cloud components together with compatible Operational Decision Manager components that are outside the Operational Decision Manager on Cloud environment, for example, on a private cloud or an enterprise on-premises system.

In a typical hybrid scenario, you create a decision service project in Rule Designer on your computer, and publish or deploy it to Operational Decision Manager on Cloud, where it can be further developed and tested. Then, you download and redeploy the executable assets (RuleApp and XOM) to an instance of Decision Server that is running on premises or in a cloud. You can redeploy the asset through a REST API or the Rule Execution Server console.

The following diagram shows the process for developing a decision service in a hybrid cloud environment:



Parent topic: [Overview](#)

Related tasks:
[Deploying to a hybrid cloud environment](#)

Related reference:
[IBM Operational Decision Manager on Cloud Compatibility](#)

First steps

Before you begin, you must gain access to the cloud portal. If you want to create decision services, you must install Rule Designer. When you finish setting up, you can go through the tutorials to learn how to use Operational Decision Manager on Cloud.

[Accessing your cloud portal](#)

Your cloud administrator invites users to your portal. You must initialize your account to work in the portal. The portal provides two types of user authentication, and service credentials for client application authentication.

[Preparing to create decision services](#)

You use Rule Designer to create decision services for Operational Decision Manager on Cloud. You must install the component in an existing version of Eclipse.

Parent topic: [Overview](#)

Accessing your cloud portal

Your cloud administrator invites users to your portal. You must initialize your account to work in the portal. The portal provides two types of user authentication, and service credentials for client application authentication.

Selecting subscriptions

You can switch between instances of Operational Decision Manager on Cloud without logging out.

Inviting users to the portal

The cloud administrator invites users to the cloud portal, and assigns them roles.

Removing accounts from the portal

The cloud administrator can remove accounts from the portal.

Managing your portal account

You must activate your account to use the cloud portal. You provide a password for your account, and update it regularly.

User authentication

To access the cloud portal and its components, you use basic or SAML authentication.

Client application authentication

You use service credentials to authenticate a client application when it calls a decision service that is running in Operational Decision Manager on Cloud.

Parent topic: [First steps](#)

Selecting subscriptions

You can switch between instances of Operational Decision Manager on Cloud without logging out.

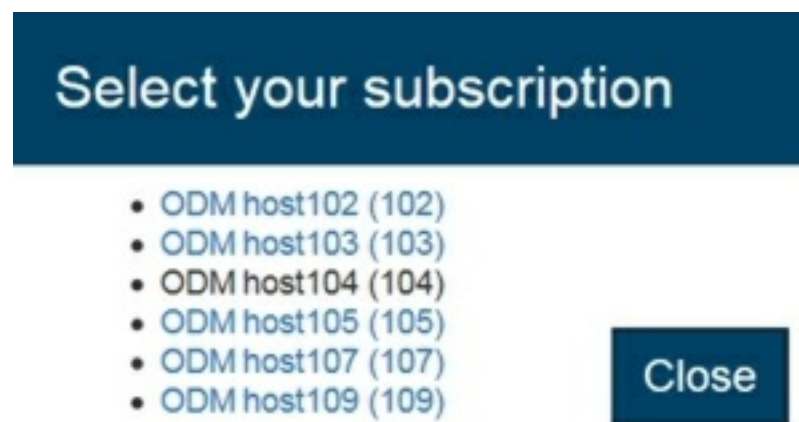
About this task

If your organization has more than one subscription for IBM® Digital Business Automation offerings on the cloud, you can switch between the instances through the cloud portal. This feature can enable you to check the consistency of your assets across the subscriptions. For example, you can check that the latest version of a decision service deployed in Operational Decision Manager on Cloud is referred from a process that you defined in Business Process Manager on Cloud, or that the same version of a decision service is deployed on all your Operational Decision Manager on Cloud subscriptions.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`. The variable `<vhostname>` is the name of the virtual host machine that holds your cloud portal.
2. Click your account name at the top of the home page.
3. Click **Your Subscriptions**. A list shows the instances that you can access. The open subscription is not hyperlinked.

The following image is an example. Your list probably does not contain the subscriptions shown in this example.



4. Click the link of the subscription that you want to open. The cloud portal of the linked subscription opens in your browser.

Parent topic: [Accessing your cloud portal](#)

Inviting users to the portal

The cloud administrator invites users to the cloud portal, and assigns them roles.

About this task

The cloud administrator makes the cloud portal available to other users. The administrator sends an invite to each user, and assigns roles to the users. The roles limit the users' access to cloud environments and components.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`. The variable `<vhostname>` is the name of the virtual host machine that holds your cloud portal.
2. Click **Admin** to open the user management page.
3. Click **Invite New Users** to open the invite dialog.
4. Enter the email addresses of the people you want to invite. Enter the addresses in the following format: `user@example.com`. Separate the addresses by pressing the Comma, Space, or Enter key.
5. Click **Send Invite** to send the invitations. The user management page displays the invited users.

Tip: If the user management page does not display the invited users automatically, refresh the web page.

6. Assign roles to the users by hovering over their names on the user management page and using the drop-down menus to select roles. You can select from the following roles:
 - Cloud role:
 - Administrator
 - Operational Decision Manager component roles:
 - Rule developer
 - Release manager
 - Business user
 - Integrator
 - Permission manager

Results

Your users now have accounts and roles that allow them to work in the cloud portal.

Parent topic: [Accessing your cloud portal](#)

Removing accounts from the portal

The cloud administrator can remove accounts from the portal.

About this task

You can remove user or service accounts from your instance of the cloud portal. You might do this to remove people who no longer use the portal, to clear out temporary accounts, or to replace an existing account.

Procedure

1. On the Operational Decision Manager on Cloud main page, click **Admin**.
2. Hover over the name of a user to be removed until you see a minus sign.
3. Click the minus sign to remove the user.

Results

The user can no longer log in to the portal.

Parent topic: [Accessing your cloud portal](#)

Managing your portal account

You must activate your account to use the cloud portal. You provide a password for your account, and update it regularly.

[Activating your account](#)

You must activate your account to access your cloud portal.

[Updating your profile](#)

Use your user profile to provide your name, and to select your preferred language.

[Resetting your password](#)

You can reset your password through the cloud portal or an expiration notice. You can also reset a forgotten password.

Parent topic: [Accessing your cloud portal](#)

Activating your account

You must activate your account to access your cloud portal.

Before you begin

Check your email for an invitation from `noreply@odm.ibmcloud.com`. The invitation contains a link and instructions to activate your account. If the invitation is not in your main inbox, check your spam mail.

About this task

You create a password when you activate your account. Your email address and the password serve as your login credentials. To activate your account, complete the following steps.

Procedure

1. Click the link in the email invitation. The Operational Decision Manager on Cloud welcome page opens in your default browser.

Tip: To check that you are using a compatible browser, see System requirements .

2. Enter your name and create a password.
3. Click **Activate**.

Results

The main page of the cloud portal opens. It displays the **Applications** tab, which shows the cloud environments that are available to your user role. If you are the cloud administrator, you can also see the **Admin** tab for managing user roles. If you are not the cloud administrator, you must contact your administrator to change your assigned role.

Your password expires in 60 days. You receive a reminder by email to renew the password before it expires. The expiration notice includes a link to change your password. You can also change your password through the main page.

Parent topic: [Managing your portal account](#)

Updating your profile

Use your user profile to provide your name, and to select your preferred language.

Procedure

1. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`.
2. Click your name, and then click **User profile**. The profile editor opens. It shows your name, default language, and user role:



3. Update your profile. You can change your name and the language.

Important: The profile editor does not show your login name, which is your email address. To change your login name, contact your cloud administrator, who must create another account for you that uses the new login name.

4. Click **Update** to save your changes.

Tip: If your user profile language is set to **Browser language (default)** and you change the language of your browser, you must log back in to the portal for the change to take effect.

Parent topic: [Managing your portal account](#)

Resetting your password

You can reset your password through the cloud portal or an expiration notice. You can also reset a forgotten password.

About this task

These password services are available for user accounts that use basic authentication in the cloud portal. You cannot use them with SAML authentication or service credentials.

Changes to the password policies are applied to existing accounts when the passwords are changed. For example, an existing password becomes subject to a new expiry period when the password is changed or it expires under its original expiry period.

The following restrictions apply to the composition of your password:

- Minimum number of characters: 8
- Minimum number of alphabetic characters: 2 (at least 1 uppercase and 1 lowercase)
- Minimum number of numeric characters: 1
- Minimum number of special characters: 1 (special characters: _-|@.,?V!~#\$%^&*(){}[]=)
- Validity: maximum 90 days
- History of used passwords: 10

Example password: MyNewPa\$sw0rd

Login failure

You are locked out of your cloud instance for 30 minutes if you exceed five failed login attempts. After a password expires, you get three attempts to log in to reset your password. If after three attempts you still cannot log in, you can click **Forgotten your password?** to reset your password.

Procedure

To reset your password, do one of the following procedures:

- In the cloud portal, click your name and **Change password**. A dialog opens to confirm your current password and to enter a new one.
- Reset your password through the expiration notice that is sent to you when your password is about to expire. The notice contains a link that takes you to a dialog to change your password.

For a forgotten password, follow these steps:

1. In the login window of Operational Decision Manager on Cloud, enter your user name and click **Continue**. Your user name is your email address.
2. Click **Forgot your password?** A message directs you to check your email for information on how to reset your password.
3. Click the link in the email message to change your password. The link remains valid for one hour.
4. Enter a new password, and click **Set Password**. The main portal page opens.

Parent topic: [Managing your portal account](#)

User authentication

To access the cloud portal and its components, you use basic or SAML authentication.

When you provision your cloud portal, you select a type of user authentication:

Basic authentication

The user logs in to the cloud portal directly. The user has a user account in the portal, and logs in with an email address and a password that meets the security requirements of the portal.

SAML authentication

The user signs in to the cloud portal through a non-portal login service. Security Assertion Markup Language (SAML) is an XML standard for exchanging single sign-on information. SAML delegates authentication to a third party. When a customer subscribes to Operational Decision Manager on Cloud through SAML, authentication is typically delegated to the customer's organization. The organization stores and manages its own user credentials. There is no duplication of credentials between IBM® and the customer.

Important:

If you log in to the cloud portal by using SAML authentication, you cannot run decision services or download archive files for a hybrid cloud environment. You need basic authentication for these tasks.

Using SAML

To use SAML, you must submit metadata for your login service. IBM uses the metadata to delegate authentication to your service. SAML support is configured per portal instance for the users of the instance, and for their email domain.

When the cloud administrator first logs in to the cloud portal:

1. The administrator enters an email address.
2. Operational Decision Manager on Cloud determines the type of authentication:
 - Basic authentication: The account administrator is prompted for a password.
 - SAML authentication: The administrator is redirected to another login page.
3. If the authentication is successful, the account administrator is logged in to Operational Decision Manager on Cloud.

Authorizing users

The cloud administrator invites users to Operational Decision Manager on Cloud, and assigns them roles. For SAML authentication, Operational Decision Manager on Cloud stores only the user roles and not the user passwords, which stay in users' login service.

Authenticating client applications

You must use basic authentication to connect client applications to the cloud portal. You cannot use SAML to authenticate a client application. Service credentials are specifically designed for authenticating client applications (see [Service credentials for client applications](#)).

Parent topic: [Accessing your cloud portal](#)

Client application authentication

You use service credentials to authenticate a client application when it calls a decision service that is running in Operational Decision Manager on Cloud.

Operational Decision Manager on Cloud provides two options for authentication:

- **User accounts:** These accounts provide credentials that users enter to sign in to the cloud portal and its components. While client applications can also login credentials of user accounts, this practice is not recommended because user passwords and any applications that use them must be updated regularly.
- **Service credentials:** These accounts provide credentials for authenticating calling applications. They include an ID that is associated with a function, and a highly secure password. Because of its high security, the password does not require regular updates and is well suited for authenticating client applications.

Because service credentials are linked to functions, and not to real users, they do not have to be changed when a user leaves a project or a company. SAML users, who log in to the cloud portal through the login systems of their organizations (see [Authentication](#)), must use service credentials for applications that need to authenticate with the cloud.

Tip: If you are using the trial version of Operational Decision Manager on Cloud, you can use the login credentials of your user account in a client application to call rules from the cloud portal. You can also request service credentials for this operation.

Service credential basics

Service credentials comply with basic authentication. They use a functional ID for the user name, and a long, machine-generated password to foil brute force attacks by hackers, for example:

- Functional ID: `custval.fid@t100`
- Password: `8xcFS90S60EGcvj0coppPDH9+/iBx9aDrjhD8zwn`

When you create a set of service credentials, you enter an alias (for example, `custval`). The cloud service generates a functional ID from the alias by adding an extension that stands for functional ID (`fid`) and your instance of the cloud portal (`t100`).

Tip: For the functional ID, use an alias that associates the service credentials with a decision service or a client application. For example, you can use the alias `custval` for a decision service that validates customers.

Important points:

- Service credentials are only used to authenticate calling applications. They have no cloud role, and cannot be used by a user or Rule Designer to log in to the cloud portal. If you try to log in by using service credentials, you get an error message: A functional user is not allowed to perform this operation.
- Only cloud portal administrators can create service credentials. The administrators give the service credentials to the developers of the client applications.
- You cannot update an existing set of service credentials. You must replace the set with another set.
- You cannot use the same alias in more than one set of service credentials. You must delete the first set before you can create another set that uses the same alias.
- When you delete a set of service credentials, the cloud portal no longer recognizes it. The client application needs a new set to connect to the cloud portal.

[Getting a set of service credentials](#)

You create service credentials for a client application.

[Deleting a set of service credentials](#)

You delete service credentials so that a client application can no longer use them to connect to the cloud.

Parent topic: [Accessing your cloud portal](#)

Getting a set of service credentials

You create service credentials for a client application.

About this task

This procedure shows you how to create service credentials, and share them with the developers of a client application.

Important: For the trial version, you can get service credentials by sending an email message to supportodmoncloud@us.ibm.com. Include your name, the name of your organization, the URL of your cloud instance, and your login name (email address).

Procedure

1. Log in to the Operational Decision Manager on Cloud portal as an administrator. You must be a cloud administrator to do this procedure.
2. Click **Admin > Service Credentials**. A table displays the service credentials in your instance of Operational Decision Manager on Cloud.
3. Click the **Create Credentials** button to open the Create service credentials dialog.
4. Enter a functional ID alias, for example, custval. You do not have to enter a description, but it can help later in determining the purpose of the credentials.

Functional ID alias (required)
custval
Description (optional)
Functional ID for client application

Tip: For the functional ID, use an alias that associates the service credentials with a decision service or a client application. For example, you can use the alias custval for a decision service that validates customers.

5. Click the **Create** button. The Your service credentials dialog opens. It shows the functional ID and password that your instance of Operational Decision Manager on Cloud can now recognize when a client application uses the credentials to call a decision service that is running in Operational Decision Manager on Cloud:

Functional ID: custval.fid@t100
Password: 8xcFS9OS60EGcvj0coppPDH9+/iBx9aDrjhD8zwn
Description: Functional ID for client application

6. Click the **Copy to Clipboard** button. After a moment, the following message appears when the operation is successful:

The credentials were copied to the clipboard.

Important: You must copy the credentials before you close the dialog. When you close the dialog, the table of service credentials shows the functional ID, but the password is hidden. You cannot reopen the dialog to obtain the same password. If you do not save the credentials, you must generate a new set of service credentials.

7. Paste the credentials into a text file. The credentials look similar to the following example:

```
Functional ID: custval.fid@t100
Password: 8xcFS9OS60EGcvj0coppPDH9+/iBx9aDrjhD8zwn
Description: Functional ID for client application
```

Results

You can now give the service credentials to the developers of the client application. The developers must integrate the credentials into the application to enable it to call a decision service.

Parent topic: [Client application authentication](#)

Deleting a set of service credentials


You delete service credentials so that a client application can no longer use them to connect to the cloud.

About this task

You learn how to delete a set of service credentials. When you no longer need a set of service credentials or you want to delete it for security reasons, you can remove it from the cloud portal. Also, when you want to change the password in a set of service credentials, you must first delete the set, and then generate a new set with the same functional ID.

Important: After you remove a set of service credentials, the cloud no longer recognizes the credentials. The client application can no longer use the credentials to authenticate with the cloud when the application calls a decision service.

Procedure

1. Log in to the Operational Decision Manager on Cloud portal as an administrator. You must be a cloud administrator to do this procedure.
2. Click **Admin** > **Service Credentials** to open the table of service credentials.
3. Hover over the row that contains the set of service credentials, and click the **Delete** button  to open the delete dialog:



4. Ensure that the functional ID in the dialog box matches the ID in the set of service credentials that you want to delete.
5. Click the **Delete** button.

Results

When the table refreshes, it no longer displays the deleted set of service credentials.

Parent topic: [Client application authentication](#)

Preparing to create decision services

You use Rule Designer to create decision services for Operational Decision Manager on Cloud. You must install the component in an existing version of Eclipse.

Installing Rule Designer

You add Rule Designer to an existing Eclipse environment.

Connecting to a proxy server

You can configure Rule Designer to connect to your Operational Decision Manager on Cloud portal through an HTTP proxy server.

Inbound and outbound connectivity

Operational Decision Manager on Cloud supports authorized inbound and outbound connectivity. Except for Rule Designer, connectivity is based on API.

Parent topic: [First steps](#)

Installing Rule Designer

You add Rule Designer to an existing Eclipse environment.

Before you begin

You use Rule Designer to create decision services. To install Rule Designer, you must have the appropriate JDK and Eclipse for your operating environment already installed on your computer. Consult the Rule Designer download dialogue box in the development environment of your instance of Operational Decision Manager on Cloud. It provides links to resources for downloading Java, Eclipse, and sample projects.

Important: Each release comes with its own version of Rule Designer. You must reinstall the rule editor when you upgrade to the latest release. Starting with Operational Decision Manager on Cloud 2019.06, you can find the release number in the bottom right corner of the portal homepage.

Check the Java™ version of your Eclipse installation, and change it if necessary. In Eclipse, click **Help > Installation Details**. On the Configuration tab, check the version of Java in the `java.version` system property. You must use the SDK listed in the Rule Designer download dialogue box.

To change your Eclipse to the correct version of Java:

1. Close Eclipse.
2. Open `eclipse/eclipse.ini`, and add or edit the following two lines at the beginning of the file:

```
-vm  
<SDK_InstallDir>/bin
```

<SDK_InstallDir> is the location of the Java instance on your computer.

3. Restart Eclipse.

Make sure Eclipse and Java both use the same bit value, either 32 or 64 bits.

Tip: If you get an error while installing Rule Designer, install the Eclipse Modeling Tools.

About this task

In your Eclipse installation, you configure a repository for Operational Decision Manager on Cloud and use it to install Rule Designer.

Procedure

1. In Eclipse, click **Help > Install New Software**.
2. In the Install dialog, click **Add**.
3. In the Add Repository dialog, enter the repository details that are available from the Operational Decision Manager on Cloud portal.

To obtain the repository details:

- a. Log in to Operational Decision Manager on Cloud at `https://<vhostname>.bpm.ibmcloud.com`.
- b. On the **Applications** tab, in the Rule Designer box, click **Download**.
- c. From the Rule Designer download window, copy the Eclipse update site URL and paste it in the **Location** field of the Add Repository dialog of your Eclipse. Click **OK**.

4. Select the package **IBM ODM on Cloud**, and click **Next**.
5. Click **Next** again.
6. Accept the license terms.
7. Click **Finish**.
8. Click **OK** to accept unsigned content.
9. Click **Yes** to restart Eclipse. Make sure your workspace is selected, and then click **OK**.
10. In the wizard that opens, enter the URL for your Operational Decision Manager on Cloud portal in the format `https://<vhostname>.bpm.ibmcloud.com/`. You can test the connection to the cloud portal by clicking **Connect** and providing your Operational Decision Manager on Cloud credentials.
11. Click **Finish**. The wizard configures the connections to your Decision Center and Rule Execution Server instances on the cloud.

Tip: You can run the connection configuration wizard anytime from the Rule Designer menu bar by clicking **File > Configure IBM ODM on Cloud connection**. For example, you might want to connect as a different user to Operational Decision Manager on Cloud or to a different cloud instance.

12. When Rule Designer opens, switch to the Rule perspective if it does not open by default.

Parent topic: [Preparing to create decision services](#)

Connecting to a proxy server

You can configure Rule Designer to connect to your Operational Decision Manager on Cloud portal through an HTTP proxy server.

About this task

You enable an HTTP proxy server for Rule Designer by adding the server to your operating system internet options and to your Eclipse preferences for Rule Designer.

Procedure

1. Specify the proxy server in your operating system's internet options. For instance, on Windows 7:
 - a. Open Internet Explorer.
 - b. Open **Tools > Internet options > Connections**. For your type of network, do as follows:
 - Dial-up or private network: Select your configuration and click **Settings**.
 - Local area network: Click **LAN**.
 - c. Select **Use a proxy server**.
 - d. Enter your proxy server address in Address, and your proxy port in Port.
 - e. Save your changes.
2. Add the proxy server to your Eclipse preferences.
 - a. Start Rule Designer.
 - b. Click **Preferences > General > Network Connection**.
 - c. Set Active Provider to **Manual**.
 - d. Edit the HTTP and HTTPS proxy entries so that they correspond to your proxy server.
 - e. Save your changes.

Results

Rule Designer is now configured to connect to your proxy server.

Parent topic: [Preparing to create decision services](#)

Inbound and outbound connectivity

Operational Decision Manager on Cloud supports authorized inbound and outbound connectivity. Except for Rule Designer, connectivity is based on API.

The the types of connectivity use the following security protocols:

- Basic authentication: cloud user name and password
- Service credentials: a highly secure, multicharacter ID for connecting client applications
- SAML security: deferred login to your enterprise security system

Theses sections cover the authorized connectivity for Operational Decision Manager on Cloud.

Inbound connectivity

Rule Designer

- Use a version of Rule Designer that is entitled to connect to Operational Decision Manager on Cloud. Use basic authentication or SAML security.
- Rule Designer can synchronize projects to Decision Center in Operational Decision Manager on Cloud.
- Rule Designer can deploy to Rule Execution Server in Operational Decision Manager on Cloud.

Deploying to Rule Execution Server

- Deploy from Decision Center in an on-premises Operational Decision Manager installation to Rule Execution Server in Operational Decision Manager on Cloud.
- Deploy from Decision Center in another cloud service to Rule Execution Server in Operational Decision Manager on Cloud.

Invocation of decision services

The invocation of decision services deployed in Operational Decision Manager on Cloud requires basic authentication or service credentials.

Invocation of Decision Center API

The invocation of the Decision Center API deployed in Operational Decision Manager on Cloud requires basic authentication or service credentials.

IP whitelisting

For connectivity from known IP addresses and to avoid unauthorized connections, you can use IP whitelisting on a per-tenant basis. For a client tenant, you provide an IP address range or IP addresses that are authorized to connect to the tenant.

Outbound connectivity

Deploying to another Rule Execution Server

- From Decision Center in Operational Decision Manager on Cloud, you can deploy to an external Rule Execution Server by providing the required authentication credentials for the target server.
- From the Decision Center in Operational Decision Manager on Cloud, you can deploy to any Rule Execution Server outside the cloud by providing the required basic authentication.
- In the Operational Decision Manager eXecutable Object Model (XOM) code, you can use transport layer security to create connections to any external server and invoke any methods. You must ensure that the server is accessible, and that it has the required credentials. The server should have a CA-signed certificate to avoid certificate management. If the server's certificate is self-signed, your XOM code needs to set up a truststore with that certificate.

Parent topic: [Preparing to create decision services](#)

Usage reports

You can view the use of your artifacts and decisions through a web-based dashboard. It displays data in graphs that enable you to readily understand your rule application traffic.

Dashboard

Directly monitor the traffic of your artifacts and decisions.

Monthly reports

Sent reports provide data on the use of artifacts and decisions.

Dashboard

Directly monitor your usage statistics on Operational Decision Manager on Cloud.

Operational Decision Manager on Cloud includes a dashboard for monitoring artifacts in Decision Center and decisions in your Rule Execution Servers. The dashboard is accessible to cloud administrators, and displays usage data for a chosen period of time.

The dashboard is included in new instances of Operational Decision Manager on Cloud, and becomes available in existing instances when they are upgraded to the latest release. Until an existing instance is upgraded, you continue to receive monthly usage reports for it (see [Monthly reports](#)).

Metrics

The dashboard displays information on two types of metrics:

Decisions

This metric shows the number of calls to a decision service in Rule Execution Server during the selected time period. For example, if an application calls a decision service 10,000 times per hour in one day, the metric shows 240,000 calls for that day.

Artifacts

This metric shows the number of unique versioned artifacts in Decision Center. The artifacts are counted only once across all branches, releases and activities, in a given instance. If you use duplicates of artifacts in different instances, the artifacts are counted separately for each instance. The metering service does not see the duplication.

Types of artifacts:

- Business artifacts (BAL rules, decision tables, decision trees, technical rules, functions)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- Business object models (BOMs)
- Vocabulary
- Business-to-Exchange (B2X) artifacts
- Resources
- Simulation artifacts (metrics, key performance indicators, report formats, input data, and simulations)
- Test suites
- Decision validation service (DVS) artifacts (scenario suites for testing and simulation)
- Business Action Language (BAL) templates
- Smart views

Time window

You set the period of time that is monitored by the dashboard. You select the dates to start and end a monitoring period. The dashboard collects the data for the metrics from Operational Decision Manager on Cloud.

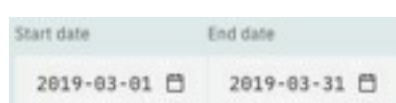
Using the dashboard

You access the dashboard through the Operational Decision Manager on Cloud homepage.

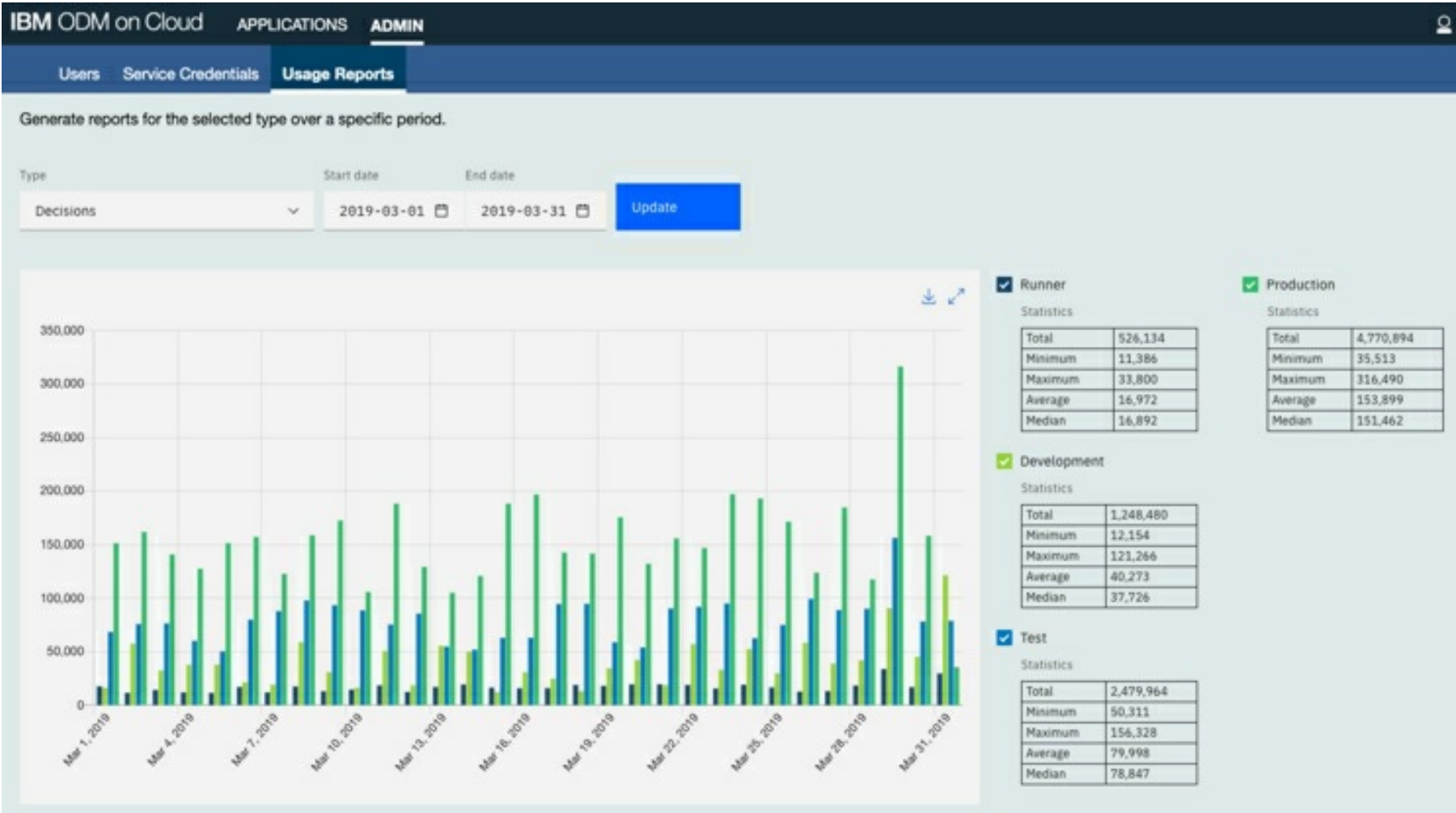
1. Open your instance of Operational Decision Manager on Cloud as a cloud administrator. The homepage displays the Applications window with the different environments and their components.
2. Click **Admin** to display the cloud administrative features.
3. Click **Usage Reports** to open the dashboard. You are presented with an empty report page.
4. Under Type, click **Select report type** to open its drop-down menu, and select a metric, for example, **Decisions**:



5. Use the **Start date** and **End date** calendars to define the reporting period similarly to the following image:

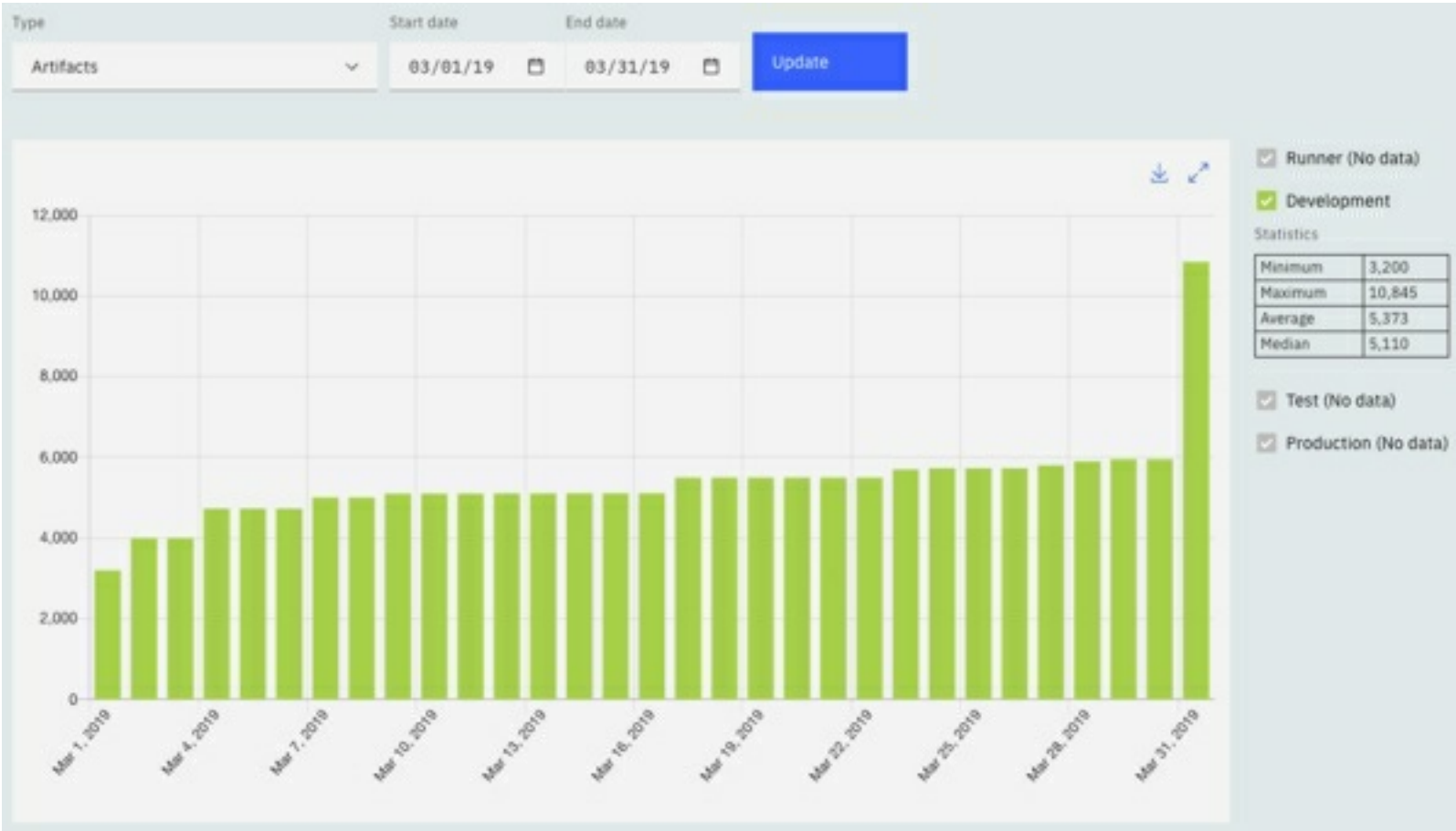




6. Click **Show** to produce a report. Similar to the following image, the dashboard displays the results in a bar chart, and includes data from the decision runner (Runner) and the Rule Execution Server environments:



The overall statistics are displayed to the right of the dashboard. By default, all the environments are shown. To hide data for a metric, click the box next to its name to deselect it.

7. Change Type to **Artifacts** and click **Update**. Similar to the following image, the dashboard displays data on the artifacts in Decision Center:



8. To see the chart in full screen, click the **Expand** button . The dashboard enlarges the graph, and hides the display parameters and overall statistics.
9. To download the report, click the **Download** button . The download dialog of your browser opens. Use the dialog to download a Comma Separated Values (CSV) file of the report to your computer.

Tip: A CSV file uses a simple format for storing tabular data. You can open it by using a program that stores data in tables.

10. To refresh the data in the chart, click **Update**. The dashboard adds any new data for the metrics and the chosen time period.

Attention: The dashboard does not retain data when you navigate away from it in your web browser. You must reset its display parameters when you return to it.

Monthly reports

Sent reports provide data on the use of artifacts and decisions.

New instances of Operational Decision Manager on Cloud come with the usage dashboard (see [Dashboard](#)). For existing instances, IBM continues to send monthly usage reports until the instances are upgraded to the latest release of Operational Decision Manager on Cloud.

The report shows the traffic of rule applications for the previous month. Typically, the report goes to the administrators on record for the instance, but it can be directed to a different recipient. Like the dashboard, the reports provides usage data on artifacts and decisions.

The following image is an example of a usage report. The example is for the full version of Operational Decision Manager on Cloud, and shows usage data for the artifacts, decision runner, and the development (dev), production (prod), and test (test) environments. The dates are formatted day-month-year, and the total shown for artifacts is the largest number for the month. The last column shows the sum total of decisions for each day in the month, and the total at the base of each column of decisions is the sum total for the month for a specific environment.

	A	B	C	D	E	F	G
1	Day	Artifacts	Decision Runner Decisions	Decision Server (dev) Decisions	Decision Server (prod) Decisions	Decision Server (test) Decisions	Total Decisions
2	01/03/2019	3000	11323	37388	108857	78125	238693
3	02/03/2019	3200	17525	16250	151407	68414	256796
4	03/03/2019	4000	11469	57444	161828	75679	310420
5	04/03/2019	4000	14230	32517	140828	76327	267902
6	05/03/2019	4729	11902	37726	127523	60104	241984
7	06/03/2019	4729	11386	37880	151462	50311	255768
8	07/03/2019	4729	17030	21391	157219	79866	280235
9	08/03/2019	5007	11933	19207	122763	87888	246798
10	09/03/2019	5007	17397	58924	158692	97851	337871
11	10/03/2019	5100	12967	30821	172595	93454	314937
12	11/03/2019	5100	14558	16271	105765	88584	230278
13	12/03/2019	5100	18498	50821	188264	75240	337923
14	13/03/2019	5105	12355	18584	129202	85543	250789
15	14/03/2019	5105	16892	55501	104865	54764	237127
16	15/03/2019	5110	19374	50148	120793	51868	247293
17	16/03/2019	5110	16394	12154	188265	62900	284823
18	17/03/2019	5110	15876	30764	196803	62892	311445
19	18/03/2019	5500	15914	24850	142611	94598	283473
20	19/03/2019	5500	18883	13106	141546	94827	273862
21	20/03/2019	5500	18034	34542	175612	58763	292451
22	21/03/2019	5500	19474	42084	132061	53864	252983
23	22/03/2019	5500	19555	18694	155874	90227	289850
24	23/03/2019	5500	18808	57160	146960	91914	320342
25	24/03/2019	5700	15566	32998	197179	95137	346580
26	25/03/2019	5728	19074	52504	193058	62577	332941
27	26/03/2019	5728	16573	29739	171433	74952	298425
28	27/03/2019	5728	12684	58401	123897	99170	299880
29	28/03/2019	5800	13101	38886	184685	88878	331350
30	29/03/2019	5900	18338	42007	117456	90033	273734
31	30/03/2019	5955	16900	45280	158245	78164	304544
32	31/03/2019	5955	16848	47984	187587	77845	336219
33	Total	5955	490861	1122026	4715335	2400759	8887716

Tutorials

The Operational Decision Manager on Cloud tutorials cover the key features in the development components. They demonstrate how to create decision services and deploy them to a production environment.

Note: Additional projects can be found in the [ODMDEV Repositories](#).

Introduction to Operational Decision Manager on Cloud

Get a head start in using the cloud portal.

[Getting started in Operational Decision Manager on Cloud](#)

This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.

Rule Designer tutorials

You use Rule Designer to create decision services and publish them to the Operational Decision Manager on Cloud portal for further development and deployment.

[Creating a decision service in Rule Designer](#)

This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.

[Debugging a decision service in Rule Designer](#)

This tutorial provides a methodology for debugging a decision service in Rule Designer.

Decision Center Business console tutorials

You use the Business console to develop and maintain decision services. You can create new rule artifacts, update existing ones, and deploy decision services to Rule Execution Server.

[Getting started with decision modeling](#)

This tutorial shows the basics of creating a decision model service, in which you model and author a decision service, all from the Business console.

[Creating a ruleflow in the Business console](#)

This tutorial shows you how to create a ruleflow in a decision service in the Business console.

[Creating a simulation in the Business console](#)

This tutorial shows you how to create a simulation in a decision service in the Business console.

[Merging branches in the Business console](#)

This tutorial shows you how to merge changes between branches in a decision service in the Business console.

Miniloan Service

Most of the tutorials are based on the Miniloan Service decision service. Setting it up is essential to using the tutorials.

[Preparing and removing the tutorial project](#)

You use the Miniloan Service decision service in most of the tutorials. You import it into Decision Center, and when you no longer need it, you delete it.

[Next >](#)

Getting started in Operational Decision Manager on Cloud

This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.

Operational Decision Manager on Cloud is a complete system for creating and using rule applications. It enables you to express your organization's decisions and policies in business rules, and to make the rules available to your production applications. Users can work together in the portal to develop and maintain the rules in decision services.

In this tutorial, you work in the cloud portal. You update the rules in a decision service, and then test the decision service. After the decision service passes the test, you deploy the rules from the decision service to different execution servers. Finally, you look at a sample application that uses the rules that were deployed from the decision service.

Important:

- You can do this tutorial in Operational Decision Manager on Cloud Express. However, only the parts of the tutorial on the development environment apply to Express. The parts that refer to the test and production environments only work in the full version of Operational Decision Manager on Cloud.
- You do not create a decision service in this tutorial. To learn how to create a decision service, see [Creating a decision service in Rule Designer](#).
- This tutorial does not cover the decision governance framework, the system for coordinating and tracking work on projects in Decision Center. For more information about the framework, see [Governance principles](#) and the video [Using the Decision Governance Framework in the Decision Center Business Console](#).

Learning objectives

You do the following tasks:

- Import a decision service.
- Create a branch in a decision service.
- Edit a decision table and create an action rule.
- Test a decision service.
- Deploy a RuleApp from a decision service.
- Explore the contents of a RuleApp in an execution server console.
- Call a ruleset from a client application by using REST.

Time required

1 hour

[Next >](#)

Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work on the Miniloan Service decision service, which is used by a fictitious loan company to determine whether borrowers are eligible for loans. When the company updates its loan policies, you reflect the changes in the business rules in the decision service.

The rules in the decision service are kept in two folders:

- Eligibility: Contains the rules that determine whether a loan can be approved.
- Validation: Contains the rules that make preliminary checks to determine whether data is rejected immediately.

You update the decision service in the cloud portal, which has three different work environments. You modify and test the decision service in the development environment, and deploy the rules from the decision service to execution servers in the development, test, and production environments. Finally, you make the rules available to a sample client application, which uses the rules to process data.

Note: This tutorial is also available on GitHub. You can find it at [Getting started in IBM Operational Decision Manager on Cloud](#).

Audience

The tutorial is for users who want to work on decision services in the cloud portal of Operational Decision Manager on Cloud.

Prerequisites

You need the following items to do this tutorial:

- Access to a cloud portal instance of Operational Decision Manager on Cloud.
- Access to ODMDEV on GitHub: <https://github.com/ODMDev>

Lessons in this tutorial

[Task 1: Preparing to do the tutorial](#)

You download the project files for the tutorial, and gain access to the Operational Decision Manager on Cloud portal. Then, you import the Miniloan Service decision service, and create a branch in the service for your changes.

[Task 2: Touring the decision service](#)

You look at some of the artifacts in the Miniloan Service decision service.

[Task 3: Creating and editing rules](#)

You update a decision table, and create an action rule to implement policy changes.

[Task 4: Testing and deploying the decision service](#)

You test the changes to the decision service, and then deploy the rules in decision service to an execution server.

[Task 5: Testing in the execution server](#)

You test the rules from the decision service in the execution server. When you finish, you deploy the rules to the test environment for integration testing, and then deploy them to the production environment for use with client applications.

[Task 6: Calling the rules from a client application](#)

You have deployed the rules from the decision service. Now, you can set up a client application to call the rules from the execution server in the development environment.

Task 1: Preparing to do the tutorial

You download the project files for the tutorial, and gain access to the Operational Decision Manager on Cloud portal. Then, you import the Miniloan Service decision service, and create a branch in the service for your changes.

About this task

You update the Miniloan Service decision service in the Decision Center Business console in the development environment in the cloud portal. The decision service holds the business rules and related artifacts for developing and deploying a rule application that is used by a client application. The Business console contains editors for creating and editing rules, and tools for testing decision services and deploying their rules to the execution servers in the development, test, and production environments in the cloud portal.

To start the tutorial, you must first download a group of projects from GitHub. Step 1 explains how to download the files.

To use the Business console, you must have access to the cloud portal. Step 2 explains how to obtain access to the cloud portal if you do not have it.

When you have access to the cloud portal, you can work on the Miniloan Service decision service. The decision service must be imported into Decision Center through the Business console. You look for the decision service in the Business console in step 3, and if you need to import it, you follow the instructions in step 4.

Finally, when Miniloan Service is in the Business console, you create a branch for your work in the decision service. Your branch isolates your changes from the other branches in the decision service.

Step 1: Downloading the tutorial files

You download the source files for the tutorial from Github and install them on your computer. These files are required to do the tutorial.

Procedure

1. Go to the GitHub repository for the tutorial: [Getting started in IBM® Operational Decision Manager on Cloud](#)
2. Download the contents of the repository to a directory on your computer. The files are downloaded in a compressed file that is named `odm-cloud-getting-started-master.zip`.
3. Open the downloaded file `<InstallDir>/odm-cloud-getting-started-master.zip`. `InstallDir` is used throughout the tutorial to refer to your directory for the GitHub files.
4. Extract the contents of the compressed file to the `InstallDir` directory. You get a new directory, named `odm-cloud-getting-started-master`, which contains the following items:
 - `miniloan.zip`: This compressed file contains the Miniloan Service decision service and the `miniloan-xom` Java object model. You import this file into Decision Center through the Business console.
 - `Miniloan Service`: This decision service contains the rule artifacts for approving loans. The decision service was created in Rule Designer.
 - `miniloan-xom`: This Java object model describes the classes that are used in the decision service.
 - `miniloan-server`: This client application uses REST to call the rules that are deployed from the decision services.
 - Documentation for the tutorial

Step 2: Gaining access to the cloud portal

Each instance of the Operational Decision Manager on Cloud portal has an administrator, who invites people to the portal and assigns user roles.

Before you begin

You need a user role that gives you access to the cloud portal and the Business console. You can do this tutorial with the business user role or greater. Check with your administrator. If you already have access to the cloud portal, you can skip this step and go to step 3.

About this task

The administrator gives people access to the portal by sending them invitations from the portal. The invitations enable people to connect to the portal for the first time.

The administrator also assigns user roles, which determine which components can be accessed by the users. For more information, see [Cloud environments and user roles](#).

For this tutorial, it is recommended that you use one of the following user roles:

- **Business user**: Works primarily in the Business console. Can update and create rules, but cannot deploy them.

- Release manager: Can create new releases, update and test rules, and deploy decision services.

To gain access to the cloud, follow the instructions for your type of portal:

Client portal

If your organization has an instance of the cloud portal, you must contact the administrator of the portal to gain access. For more information, see [Accessing your cloud portal](#).

Sample portal

If you want to use the trial version of the cloud portal, you must contact IBM Support to be invited to the portal. For more information, see [IBM ODM on Cloud Free Trial](#).

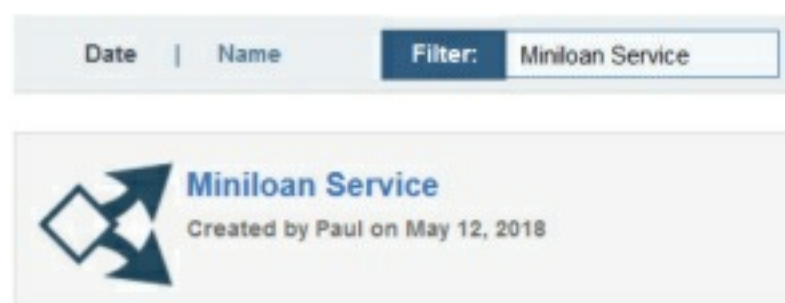
Step 3: Checking the availability of the decision service

You determine whether the Miniloan Service decision service is already in the Business console of the development environment in the cloud portal.

Procedure

1. Sign in to the cloud portal. The portal displays the resources that are available to your cloud user role.
2. Launch the Decision Center Business console in the development environment. The console opens to its home page.
3. Click **LIBRARY** to open the list of decision services that are currently in Decision Center.
4. Enter Miniloan Service in the filter to look for the decision service.

Decision Services




If the Miniloan Service decision service is in the library, skip step 4 and go to step 5.

Step 4: Importing the decision service

You import the decision service into Decision Center.

Procedure

1. In the Business console **LIBRARY** tab, click the **Import Decision Service** button .
2. Click **Choose**, and navigate to the miniloan.zip file in the directory <InstallDir>/odm-cloud-getting-started-master, which you created in step 1.
3. Select the file, and click **Open**.

Note: Do not select **Use Decision Governance Framework**. You do not use this feature in this tutorial.

4. Click **Import**. The decision service is added to the library in the Business console.


Step 5: Creating a branch

You create a branch to isolate your changes to the decision service.

About this task

The main branch in the decision service contains the rule artifacts in their original form. To update the decision service, you create a branch based on the main branch, and work in the new branch.

Procedure

1. Click Miniloan Service to open the decision service.
2. Open the **Branches** tab, and expand the main branch. Look at the names of the existing branches. When you name your branch, do not reuse the name of an existing branch.
3. Click the **New Branch** button .
4. Enter a name for your branch, for example, My Branch. Remember not to reuse the name of an existing branch. You can personalize your branch by using the initials of your name.

You can also enter a goal for your branch, for example:

Make policy changes in getting started tutorial.

5. Select **main** as the parent branch, and then click **Create**. The Business console duplicates the artifacts of the main branch in a new branch that you can modify.

Note: In the rest of the tutorial, My Branch is used as the name of the branch.

What to do next

In the next task, you explore the contents of the decision service before making changes.

[< Previous](#) | [Next >](#)

Task 2: Touring the decision service

You look at some of the artifacts in the Miniloan Service decision service.

About this task

Companies set policies for their operations. These policies can address such matters as pricing, eligibility, and product configuration. You use Operational Decision Manager on Cloud to develop business rules that express these policies. The rules state the conditions for applying the policies, and the resulting actions. Client applications then call the rules to implement the policies.

In this task, you look at some of the artifacts in your branch of the Miniloan Service decision service.

Step 1: Opening your branch

You open your branch and display its decision artifacts.

Procedure

1. Click **My Branch** in the Branches tab of the Miniloan Service decision service.
2. Click **Decision Artifacts** to see the rule artifacts. By default, only the rules are displayed.
3. Hover over **Types(1/5) X** and click the X to display all the artifacts in the decision service. The labels indicate the types of artifacts, and some show the number of artifacts:



Step 2: Exploring the ruleflow

You open the ruleflow to determine how it works.

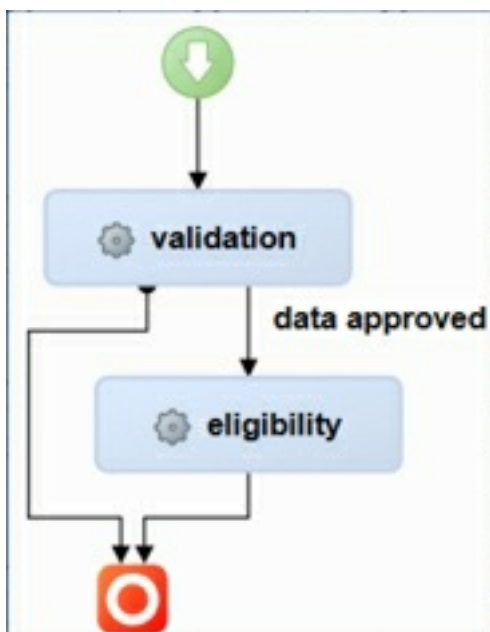
Procedure

1. Hover over **miniloan** in the list of decision artifacts.
2. Click the ruleflow to open it in the preview window.

Results

A decision service keeps its decision points in projects and folders. The Miniloan Service decision service contains one project, and two folders for different decision points. The rule in the validation folder determines whether incoming loan data is valid, and the rules in the eligibility folder apply various conditions for eligibility.

The ruleflow states the sequence for applying the rules:



The ruleflow starts by running the rule in the validation folder. If the data from a loan request passes the rule, the ruleflow directs the data to the eligibility rules. However, if the validation decision point does not approve the data, the ruleflow skips the eligibility decision point, and the decision service stops processing the loan request. For more information, see [Editing ruleflows](#).

Step 3: Exploring an action rule

You open an action rule to determine how it works.

About this task

There are two types of rules in the decision service:

- Action rules: An action rule states a policy in a natural language. It uses conditions and actions to check the data and apply a response.
- Decision tables: A decision table holds several action rules that share the same policy statement but use different variables. The variables are listed in condition and action columns, and each row of the table forms a complete action rule.

You start by looking at an action rule.

Procedure

1. In the list of decision artifacts, click **eligibility** to open the folder in the preview window.
2. Click **minimum credit score** to open the action rule in the preview window.

Results

An action rule associates one or more actions with one or more conditions. When the conditions are met, the actions are triggered.

The rule statement in the minimum credit score rule looks as follows:

```
if
  the credit score of 'the borrower' is less than 200
then
  add "Credit score below 200" to the messages of 'the loan' ;
  reject 'the loan' ;
```

As data for the borrower enters the decision service, the action rule tests the credit score variable. If the variable is less than 200, the prescribed action is to reject the loan and work stops on the loan. If the variable is equal to or more than 200, the loan is further processed. For more information, see [Action rules](#).

Step 4: Exploring a decision table

You open a decision table to determine how it works.

Procedure

1. In the list of decision artifacts, click **eligibility** to open the folder in the preview window.
2. Click **repayment and score** to open the decision table in the preview window.

Results

A decision table provides a way to view and manage sets of similar rules. Each row in a decision table represents a complete rule. The rules in the table share the same rule statement, but the condition and action values vary. The first two columns (debt to income and credit score) contain the condition variables, and the next two columns (message and rejected) contain the action variables.

The repayment and score decision table looks as follows:



	debt to income		credit score		message	loan approve
	min	max	min	max		
1	0 %	30 %	0	200	debt-to-income too high compared to credit score	<input type="checkbox"/>
2	0 %	30 %	200	800		<input checked="" type="checkbox"/>
3	30 %	45 %	0	400	debt-to-income too high compared to credit score	<input type="checkbox"/>
4	30 %	45 %	400	800		<input checked="" type="checkbox"/>
5	45 %	50 %	0	600	debt-to-income too high compared to credit score	<input type="checkbox"/>
6	45 %	50 %	600	800		<input checked="" type="checkbox"/>
7	≥ 50 %		0	800	debt-to-income too high compared to credit score	<input type="checkbox"/>

When the decision table checks a loan request, it rejects the request if the borrower must pay back an annual amount that is too high for the borrower's credit score. Otherwise, the table does not reject the loan. For more information, see [Decision tables](#).

Step 5: Running a test suite

You add a test suite and run it to verify that the decision service works correctly before changes are made.

Procedure

1. Click **Tests > Test Suites** to open the subtab.
2. Click the **New Test Suite** button . The dialog for creating a new test suite opens.
3. Click **Choose**, and navigate to <InstallDir>/odm-cloud-getting-started-master/Miniloan Service. InstallDir is your directory for the extracted files from the GitHub repository, as shown in task1, step2.
4. Select miniloan-test.xlsx, and then click **Open**.
5. Make sure that the selected server is **Testing And Simulation Runtime**, which is the default server for running tests and simulations.
6. Click the **Save and Run** button .
7. Enter the information that is shown in the following image:

Create New Version

A new version of this element will be created.

You can add a comment to associate with this version which can be viewed in the timeline.

Testing the summer release.

Create New Version

8. Click **Create New Version**, and then click **OK** in the Run Test Suite dialog. The console runs the test and switches to the Reports tab. A new report indicates a successful test by displaying a check mark in the Status column:

<input type="checkbox"/>	Name	Operation	Status	Run Date	Test Suite
<input checked="" type="checkbox"/>	Report - 2018-05-23_02-24-56-632	Miniloan ServiceOperation		5/23/18, 4:24 PM	Test Suite (latest)

9. Click the report under Recent Report. The report opens, and displays details that include the following summary:



10. Close the report.

What to do next

In the next task, you search for the decision table, update it, and create an action rule.

[< Previous](#) | [Next >](#)

Task 3: Creating and editing rules

You update a decision table, and create an action rule to implement policy changes.

About this task

The loan company wants the following changes to be implemented in the decision service:

- Increase the minimum credit score from 200 to 300 for all new loan requests.
- Reject loans that are shorter than six months.

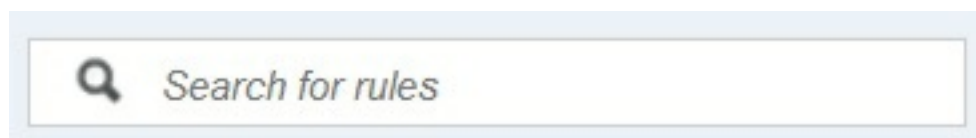
To implement the changes, you use the search function to locate the table that applies the minimum credit score. After you change the score, you create an action rule to implement the minimum loan duration.

Step 1: Searching for rules

To make the first change, you use the search function to find all the rules that change the credit score.

Procedure

1. In your branch, click inside the search field to be able to enter a search string:



2. Type score, and press Enter. The results show all the action rules and decision tables in which the word score occurs. You look at the rules, and conclude that you must edit the repayment and score decision table.
3. Click **repayment and score** in the list of search results. The decision table opens in the preview window.

Step 2: Modifying the decision table

You update the decision table. For more information, see [Decision table editor](#).

Procedure


1. Click **Edit**. The decision table editor opens.
2. Double-click 200 in row 1 of the **credit score** column, and replace it with 300. Do the same in row 2.
3. Click **Save**.
4. Enter the following comment, and then click **Create New Version**.

Changed credit score to 300

Step 3: Creating an action rule

You introduce the second policy change by creating an action rule. For more information, see [Building rules using the Intellirule editor](#).

Procedure

1. In **My Branch**, click **Decision Artifacts**, and then click **validation** to open the folder.
2. Click the **Create** button , and click **New Rule**.
3. Enter check duration as the name of the rule, and click **Create**.
4. Enter the following rule statement in the rule editor. If your web browser supports it, you can copy the rule to the rule editor. Otherwise, type in the rule or build it with the completion menu.

```
if
  the duration of 'the loan' is less than 6
then
  add "The duration of the loan is too short" to the messages of 'the loan';
  reject 'the loan';
```

The rule checks the duration of a loan. If the duration is shorter than six months, the rule adds a rejection message to the loan and rejects the loan.

5. Click **Save**.
6. Enter the following message, and then click **Create New Version**.

Added rule for summer policy

What to do next

In the next task, you test and deploy the decision service.

[< Previous](#) | [Next >](#)

Task 4: Testing and deploying the decision service

You test the changes to the decision service, and then deploy the rules in decision service to an execution server.


About this task

After you update the decision service, you want to make sure that it still works correctly. In this task, you run the decision service against a test scenario file. When the decision service produces the expected results, you deploy its rules to an execution server.

Step 1: Running a test

You run a test to see whether the decision service generates expected results. For more information, see [Testing sets of rules in the Business console](#).

Procedure

1. In **My Branch**, click **Tests** > **Test Suites** to open the subtab for running tests.
2. Hover over the Test Suite name that you added in task 2 and click the **Run** button .
3. Click the generated report when the run finishes and the status shows a check mark. The report shows a 100% success rate. The results produced by the decision service match the expected results.



4. Click **Close** to close the report.


Step 2: Deploying the decision service rules


You deploy the rules in the decision service to the execution server in the development environment to further test the service. For more information, see [Deploying from the Business console](#)

Procedure

1. Click the **Deployments** tab to see its contents. The tab contains the deployment configurations that are available to your user role.

The cloud portal has three standard environments: development, test, and production. You can only see the environments that are available to your user role. For example, the business user has access to only the development and test environments, and can only see the deployment configurations for these two environments.

2. Click the **Development** deployment configuration to open it. Go through the tabs in the configuration:
 - **General**: Provides an overview of the deployment configuration, including the name, type, RuleApp name, and base version number.
 - **Operations**: Lists the decision operations to deploy. Here, **Miniloan Service Operation** is selected for deployment. Decision operations define how the rules are used in specific rulesets for deployment. Hover over the decision operation name to see its content.
 - **Targets**: Lists where the rules can be deployed. The environment for this deployment is the development environment.
 - **Ruleset Properties**: Defines the versioning policy for each deployment. You use the default settings.
 - **Groups**: Lets the administrator choose which groups can deploy using this deployment configuration.
3. Click the **Deploy** button  in the upper right corner of the window. A dialog box opens with a summary of the deployment configuration. The Target list gives you the option of deploying to the sample server or creating a RuleApp archive.
4. Leave **Server Development Environment** selected as the target, and click **Deploy**. A message opens with the status of the deployment.
5. Click **OK** in the deployment status message. The Reports subtab opens in the Deployments tab.

6. Click the report for the deployment. The report opens and shows a summary of the deployment. It shows the target server, the configuration name, the ruleset with the rules, the deployment time, the version of the ruleset, and a link to the deployment snapshot, which can be redeployed.
7. Click the **Close** button  in the upper right corner of the window to close the report.

What to do next

In the next task, you test the rules the execution server, and then deploy the rules to the test and production environments.

[< Previous](#) | [Next >](#)

Task 5: Testing in the execution server

You test the rules from the decision service in the execution server. When you finish, you deploy the rules to the test environment for integration testing, and then deploy them to the production environment for use with client applications.

About this task

When you deployed the rules from the decision service, they were sent as a *ruleset* in a RuleApp to Rule Execution Server in the development environment. Now, you test the ruleset in the execution server. When the ruleset passes the test, you deploy it to the test environment for integration testing, and then to the production environment for use with client applications. For more information, see [Deploying decision services](#).

Note: The information in the Rule Execution Server consoles in your cloud portal might vary from the information shown in this tutorial. For example, your consoles might display more RuleApps and different version numbers.

Step 1: Viewing the deployed RuleApp

You find the RuleApp from the decision service in the execution server, and look at its contents.

Procedure

1. Return to the cloud portal, and launch the Rule Execution Server (Development) console in the development environment.
2. Click the **Explorer** tab in the console.
3. In the Navigator, expand RuleApps, and then /Miniloan/1.0. Your RuleApp contains the ruleset Miniloan_ServiceRuleset:



The RuleApp might contain more than one ruleset. You can find yours by looking at the creation dates.

4. Click **Miniloan_ServiceRuleset** to open the ruleset in the Ruleset View. The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties:



Step 2: Testing the ruleset by using the REST API

You test the ruleset to determine whether it executes correctly. You use the REST API to run the test on manually entered data.

Procedure

1. Click **Retrieve HTDS Description File** in the Ruleset View. The file retrieval page opens.
2. Select **REST** as the service protocol type, and **OpenAPI-JSON** as the format.
3. Click **Test**. The Hosted Transparent Decision Service (HTDS) opens.
4. Enter the following data as the run request:

```
{
  "loan": {
    "amount": 500000,
    "duration": 240,
```

```

    "yearlyInterestRate": 0.05
  },
  "__DecisionID__": "Test",
  "borrower": {
    "name": "Joe",
    "creditScore": 600,
    "yearlyIncome": 80000
  }
}

```

Your run request should look as follows:

```

1 {
2   "loan": {
3     "amount": 500000,
4     "duration": 240,
5     "yearlyInterestRate": 0.05
6   },
7   "__DecisionID__": "Test",
8   "borrower": {
9     "name": "Joe",
10    "creditScore": 600,
11    "yearlyIncome": 80000
12  }
13 }

```

5. Click **Execute Request**. You get the following server response:

```

{
  "__DecisionID__": "Test",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}

```

The rule application produces the expected results. The loan is rejected, and a message explains that the debt-to-income ratio is too big.

6. Close the HTDS window.

Step 3: Deploying to the test environment


Satisfied that the rules work as expected, you return to the Business console to deploy the rules to the test environment. The integrator runs integration tests on the rules in the test environment. These tests determine the stability and reliability of the rules under load before they are promoted to the production environment for use with client applications.

Procedure

Note:

If you are a business user, you cannot view the test and production environments, and cannot do steps 3 and 4.

1. Return to the Business console, and open your branch in the Miniloan Service decision service.
2. Open the **Deployment** tab. In Configurations, click **Test**.
3. Open the **Targets** tab. The Test configuration deploys the rules in the decision service to Rule Execution Server in the test environment.

- Click the **Deploy** button . The deployment dialog opens. It shows the Server Test Environment as the target of the deployment.
- Click **Deploy**, and click **OK** in the deployment status box. The list of reports opens in the Deployment tab. The new report shows that the rules have been deployed successfully:

<input type="checkbox"/>	Name	Status	Run By	Run Date	Configuration
<input type="checkbox"/>	 Report 2018-04-12_08-27-20-137		userodmcloud@mail.com	4/12/18, 10:27 AM	Development
<input type="checkbox"/>	 Report 2018-04-12_10-00-14-588		userodmcloud@mail.com	4/12/18, 12:00 PM	Test

Report 2018-04-12_10-00-14-588

- Return to the cloud portal, and launch the Rule Execution Server (Test) console in the test environment.
- Open the **Explorer** tab in the console.
- In the Navigator, expand RuleApps, and then /Miniloan/1.0. Your RuleApp contains the ruleset Miniloan_ServiceRuleset:



- Click **Miniloan_ServiceRuleset** to open the ruleset in the Ruleset View. The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties:

	/Miniloan/1.0/Miniloan_ServiceRuleset/1.0
Name	Miniloan_ServiceRuleset
Version	1.0
Creation Date	Apr 3, 2018, 4:22:29 PM GMT+2
Display Name	Minloan Service Operation
Description	
Rule engine	Decision Engine - 1.40.6
Status	 enabled
Debug	 disabled

The integrator can now do nonfunctional and integration testing on the rules before they are deployed to the production environment.

Tip: The test shown in step 2 works in Rule Execution Server in the test environment.

Step 4: Deploying to the production environment

When the integrator finishes testing the rules from the decision service, the rules can be deployed to the production environment. Client applications then call the rules from this environment to process data.

About this task

You must have the release manager role to deploy the rules to the production environment. If you have this role, you can see the deployment configuration for the production environment in the Deployments tab. You can use it as you did the deployment configuration for the test environment. However, there is no need to deploy the Miniloan RuleApp to the production environment. The deployment configuration for the production environment is included to provide a complete list of configurations for release managers.

For more information on calling the rules from this environment, see [Integrating decision services](#).

What to do next

In the next task, you set up a client application to call the rules from the execution server in the development environment.

[< Previous](#)

Task 6: Calling the rules from a client application

You have deployed the rules from the decision service. Now, you can set up a client application to call the rules from the execution server in the development environment.

Before you begin

This task describes how an application uses the rules from the decision service. It also explains how to set up and run the application. Setting up the application is optional because it requires some technical knowledge. To set up the application, you must be familiar with Maven, and Java application servers such as IBM WebSphere Liberty and Apache Tomcat.

About this task

When you deployed the rules to Rule Execution Server, you sent a ruleset in a RuleApp to the execution server. The Miniloan client application can now call the ruleset from the server.

When the application opens, it displays default information:

The screenshot shows a web application interface with two main sections: 'Borrower' and 'Loan'. The 'Borrower' section has three input fields: 'Name' (Joe), 'Yearly Income' (80000), and 'Credit Score' (600). The 'Loan' section has three input fields: 'Amount' (500000), 'Duration' (240), and 'Yearly Interest Rate' (0.05). Below these sections is an 'Execution' section with a 'Validate Loan' button.

In the Execution section, the parameters are set to call the ruleset from Rule Execution Server in the development environment:

The screenshot shows the 'Execution' section of the application. It contains four input fields: 'Server' (vhost1010.stage.bpm.ibmcloud.com), 'User' (paulreleasemanager@gmail.com), 'Ruleset' (Miniloan/Miniloan_ServiceRuleset), and 'Password' (masked with asterisks). There is also a 'Show trace' checkbox and three radio buttons for environment selection: 'Dev' (selected), 'Test', and 'Prod'.

The application uses the REST API with JSON for input and output. When the **Validate Loan** button is clicked, the application processes the default data and produces the following results:

```
Rejected

{
  "__DecisionID__": "dc22f36e-77db-4d08-b692-b9ff565845fc0",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}
```

Optional: Installing the client application on a custom server

The following steps are for any integrator or other user who wants to set up a web application to use the rules from the decision service.

Before you begin

You need the following items to do the installation:

- WebSphere Liberty application server from [Download the latest stable WebSphere Liberty runtime](#)
- Source files from the odm-cloud-getting-started GitHub repository
- Maven from [Apache Maven Project](#)

Step 1: Creating the server

You create a server to run the Miniloan Service sample application.

Procedure

1. Download the Liberty application server.
2. Decompress the Liberty file to a directory on your computer. The directory is referred to as <WLP_Home> in the following steps.
3. From an administrator command prompt, create your server by running the following command:

```
<WLP_HOME>\bin>server create testGettingStarted
```

The command creates the folder <WLP_HOME>\usr\servers\testGettingStarted, which contains the server configuration and steps.

4. To create the profile, launch the server by using the following command:

```
<WLP_HOME>\bin>server start testGettingStarted
```

You can access the started server at <http://localhost:9080>.

5. To stop the server, use the following command:

```
<WLP_HOME>\bin>server stop testGettingStarted
```

Step 2: Setting up the sample application

You use Maven to build the sample web application, and then you add the application to the application server.

Procedure

1. Go to the <InstallDir>/odm-cloud-getting-started-master/miniloan-server directory. InstallDir is your directory for the extracted files from the GitHub repository that is listed in the prerequisites.
2. In a command prompt, call `mvn clean install`. The command builds the miniloan-webapp.war file in the target directory.
3. Copy <InstallDir>/odm-cloud-getting-started-master/miniloan-server/target/zip to <WLP_HOME>/usr/servers/testGettingStarted/apps. The folder contains the .war file for the sample application: miniloan-webapp.war
4. Declare the sample application by adding the following line in <WLP_HOME>/usr/servers/testGettingStarted/server.xml:

```
<!-- Miniloan application -->  
<webApplication id="miniloan-webapp"  
  location="miniloan-webapp.war"  
  name="miniloan-webapp"/>  
</application>
```

5. Save the file.

Step 3: Running the sample application

You run the sample application.

Procedure

1. Launch the server by using the following command:

```
<WLP_HOME>\bin>server start testGettingStarted
```

A message tells you when the build is complete. The build might take a few minutes to finish.

2. Enter the following URL in your web browser to open the application:

```
http://localhost:9080/miniloan-webapp/
```

The Miniloan application opens in your browser, and displays the default values.

- 3. Click **Execution** to configure the connection with your decision service. Enter the information specific to your cloud portal, for example:

Borrower

Name:

Joe

Yearly Income:

80000

Credit Score:

600

Loan

Amount:

500000

Duration:

240

Yearly Interest Rate

0.05

Execution

Server:

vhost1010.stage.bpm.ibmcloud.com

User

paulreleasemanager@gmail.com

☐ Show trace

Ruleset:

Miniloan/Miniloan_ServiceRuleset

Password

☒ Dev ☐ Test ☐ Prod

✓ Validate Loan

- 4. Click **Validate Loan** to process the default values. The application processes the request and returns the following message:

Rejected

```
{
  "__DecisionID__": "dc22f36e-77db-4d08-b692-b9ff565845fc0",
  "loan": {
    "amount": 500000,
    "duration": 240,
    "yearlyInterestRate": 0.05,
    "yearlyRepayment": 39597,
    "approved": false,
    "messages": [
      "Too big Debt-To-Income ratio"
    ]
  }
}
```

You can find additional information by selecting **Show trace** in the Execution section and validating the data again. The additional information shows that one rule, `eligibility.minimum_income`, is executed.

- 5. When you finish using the sample application, close the application in your browser and stop the Liberty application server by using the following command as shown in step 1:

```
<WLP_HOME>\bin>server stop testGettingStarted
```

Results

You have completed the getting started tutorial for Operational Decision Manager on Cloud. The tutorial showed you the artifacts a decision service branch, how to update and test rules, and how to deploy the rules from the decision service to an execution server. You also looked at using the rules with a client application.

[< Previous](#)

[< Previous](#) | [Next >](#)

Creating a decision service in Rule Designer

This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.

Learning objectives

You do the following tasks:

- Design a decision service.
- Orchestrate rules.
- Create an action rule.
- Update a decision table.
- Test and debug rules.
- Deploy a decision service to Rule Execution Server.
- Publish a decision service to Decision Center.
- Delete a decision service from Decision Center.

The tutorial does not cover collaborative development and the decision governance framework in the Decision Center Business console. To simplify the tutorial, you work in an ungoverned branch of the decision service in the console.

Best practices

This tutorial includes the following best practices for creating a decision service in Rule Designer:

- Start with an empty workspace when you create a decision service. [Example...](#)
- Follow the Rule Project Map to create a decision service. [Example...](#)
- Begin by defining a sequence in which to run the rules. [Example...](#)

Time required

1 hour

[< Previous](#) | [Next >](#)

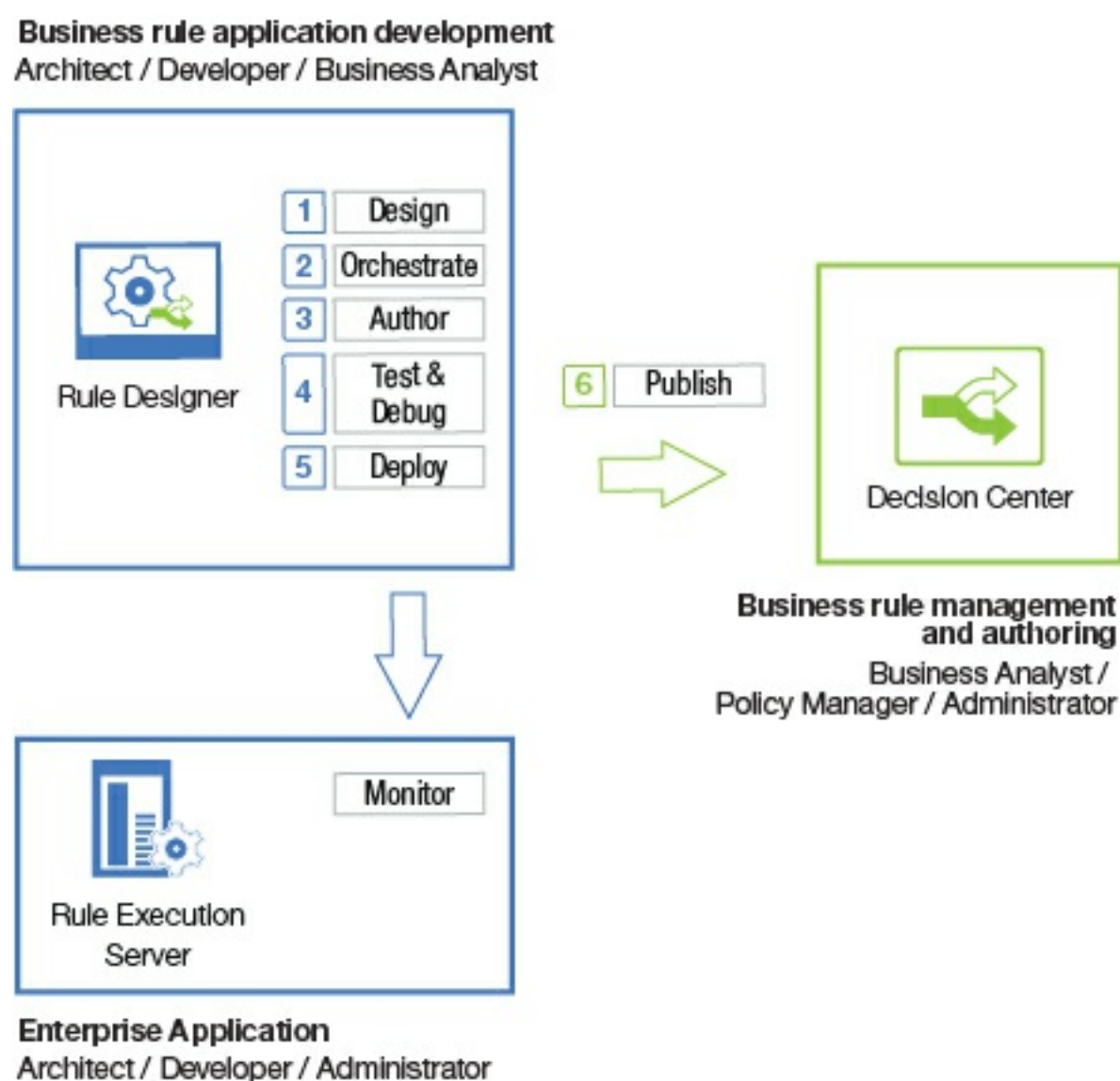
Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a decision service that determines whether a borrower is eligible for a loan. The decision service applies rule conditions for the amount of the loan, the annual income of the borrower, and the duration of the loan.

The tutorial shows you how to create and debug the decision service in Rule Designer, deploy it to Rule Execution Server, and publish it to Decision Center. Rule Designer is the editing environment for creating decision services and authoring rules, Rule Execution Server monitors and runs rules, and Decision Center is the cloud environment for collaborative rule authoring, management, and validation.

The following diagram shows the workflow for making a decision service. It goes from design to publication, and shows the users of the different components. The numbers in the diagram correspond to the tasks in the tutorial:



Audience

The tutorial introduces you to decision service development. The tutorial is primarily for release managers, and rule developers.

Prerequisites

You need the following items to do the tutorial:

- Operational Decision Manager on Cloud portal access: You must be invited to the portal and activate your account (see [Accessing your cloud portal](#)).
- Rule Designer: You must download this component from the portal and install it on your computer (see [Preparing to develop rule applications](#)). The size of the download file is about 800 MB.
- miniloanservice-projects.zip: Along with Rule Designer, download the sample project from the portal to the same directory, and extract its contents to the directory. The tutorial later refers to the directory as *<InstallDir>/miniloanservice-projects*. The size of the download file is about 13 KB.
- Rule Execution Server: You use this component in the Operational Decision Manager on Cloud portal.
- Decision Center Business and Enterprise consoles: You use these components in the Operational Decision Manager on Cloud portal.

Additional information:

- The product overview on the main goals of the product, its modules, and rule-oriented terminology (see [Introduction to Operational Decision Manager on Cloud](#))
- A knowledge of Java
- A knowledge of the Eclipse workspaces, perspectives, and views

Lessons in this tutorial

Task 1: Designing the main rule project for the decision service

You create a vocabulary so that business users can write business rules in a natural language. You use Rule Designer to create the vocabulary from a Java object model.

Task 2: Orchestrating rule running

You create a ruleflow to set the sequence in which the rules run.

Task 3: Authoring business rules

You write an action rule and import other rules for your rule project.

Task 4: Testing and debugging a ruleset

You run the ruleset by entering input data in a decision operation configuration, and then test and debug your ruleset.

Task 5: Deploying your decision service

You deploy your decision service to Rule Execution Server.

Task 6: Publishing to Decision Center

You publish your decision service to Decision Center in the Operational Decision Manager on Cloud portal, where you view your files in the Business console.

[< Previous](#) | [Next >](#)

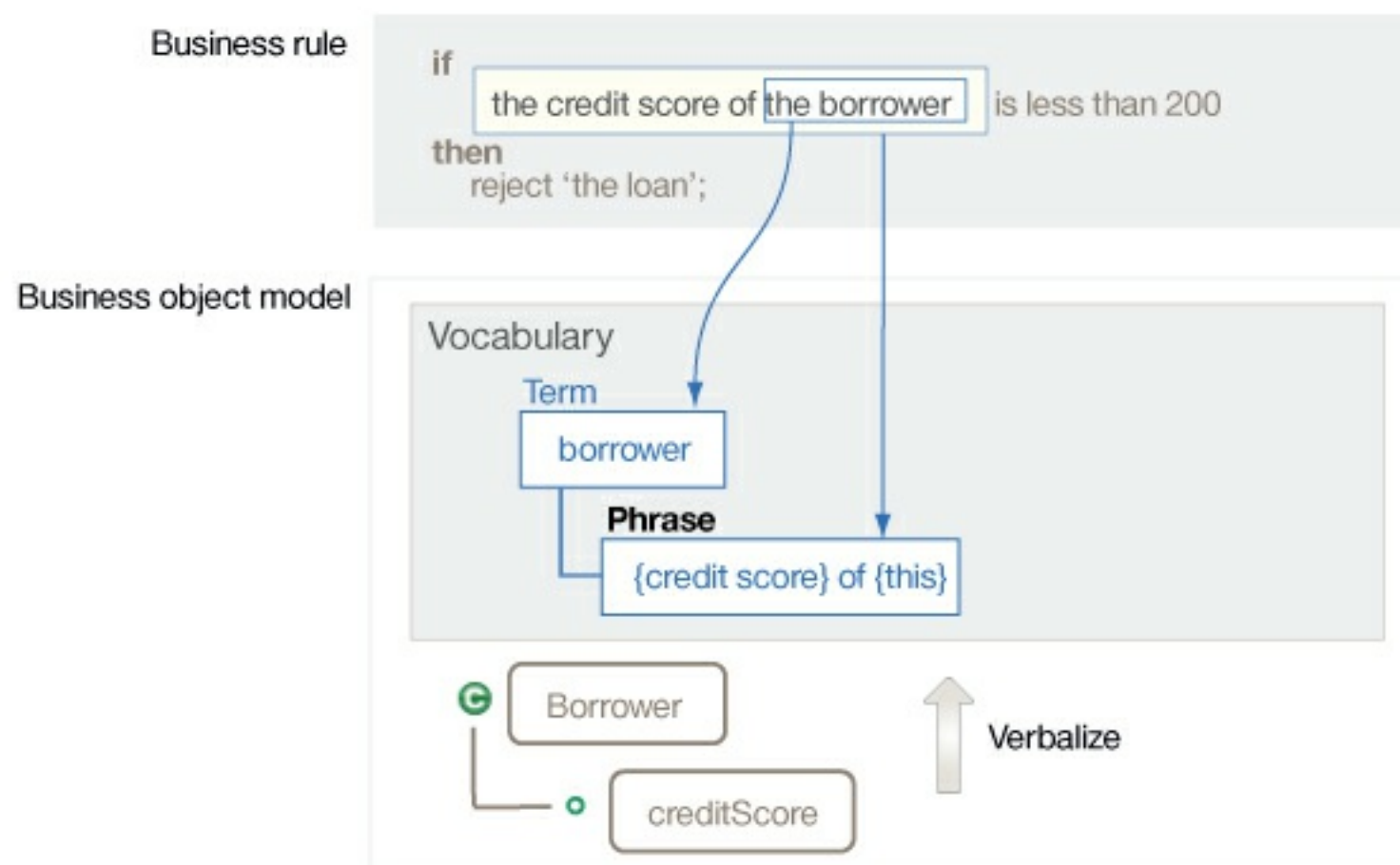
Task 1: Designing the main rule project for the decision service

You create a vocabulary so that business users can write business rules in a natural language. You use Rule Designer to create the vocabulary from a Java object model.

About this task

As a developer, you want to create a business rule vocabulary that business users can use to write and edit rules. The vocabulary must consist of terms that are familiar to the business users. The process of creating a vocabulary is called *verbalization*.

In this task, you create a business object model (BOM) that is based on an object model that is defined in a Java™ project. The following diagram shows how the classes and members of the BOM relate to the vocabulary that is familiar to the business users.



Step 1: Starting Rule Designer and importing the Java object model

Rule Designer is an Eclipse development environment for business rule applications. You can develop both rule projects and Java projects in this environment.

Before you begin

The file from which you copy rules for this tutorial is available in only American English (en_US). To use Rule Designer in American English, add the following lines to `<InstallDir>/eclipse/eclipse.ini` before `-vmargs`, where `<InstallDir>` is the installation directory for your instance of Rule Designer:

```
-nl  
en_US
```

Procedure

1. Start Rule Designer (see [Preparing to develop rule applications](#)).
2. Select a workspace. When the Workspace Launcher window opens, it shows your default workspace. If the workspace is not empty, switch to a workspace that is empty. If you want to create a workspace, change the name of the workspace in the Workspace Launcher, and click **OK**.

Tip: If you do not see the Workspace Launcher window when you start Rule Designer, you can change the workspace by clicking **File > Switch Workspace**.

3. Optional: If the connection wizard opens, connect to the cloud portal:
 - a. Enter the URL for your Operational Decision Manager on Cloud in the following format:
`https://<vhostname>.bpm.ibmcloud.com/`
 - b. Click **Connect** and enter your cloud credentials.
 - c. Click **Finish**.
4. Close the Eclipse Welcome page to see the Rule perspective. If the Rule perspective is not open, click **Window > Perspective > Open Perspective > Other**, select **Rule**, and click **Open**.

5. Click **File > Import**. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
6. Choose **Select root directory**, browse to `<InstallDir>/odm-cloud-getting-started-master`, and click **OK**. InstallDir is your directory for the files downloaded from GitHub (see [Importing and deleting the tutorial project](#)).
7. Make sure that `miniloan-xom` is selected, and deselect `Miniloan Service`.
8. Click **Finish** to import the project.

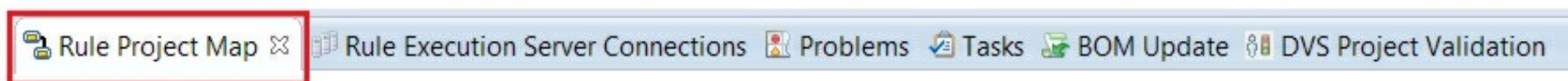
Results

The Rule perspective contains various views that you learn about in this tutorial, such as the Rule Explorer and the Decision Service Map.

The Rule Explorer shows `miniloan-xom`, which contains the execution object model (XOM) that you use in the tutorial. The XOM is the model against which you run rules. It references the application objects and data, and is the base implementation of the BOM (see [Overview: BOM and execution object model \(XOM\)](#)). The `miniloan` package holds the `Borrower` and `Loan` Java classes.

Notice the Rule Project Map tab. When you work on a decision service, the Rule Project Map tab shows the Decision Service Map, which contains the different steps for making and deploying a decision service. You can use the Decision Service Map as a guide to the creation process for the decision service. If the Decision Service Map is not displayed, click **Window > Show View > Rule Project Map** to open it.

The following image shows the Rule Project Map in the tab bar.



Step 2: Creating a rule project for a decision service

In Rule Designer, you store the business logic of your application in a decision service. A decision service can consist of a main rule project and optional standard rule projects to split the logic of your business application into different parts. Each rule project can contain rule artifacts, a BOM, a vocabulary, and a reference to the XOM. In a rule project, you can manage, build, and debug the items that comprise the business logic of your application. In this tutorial, you create only a main rule project to simplify the logic of the application.

Procedure

1. Ensure that you are in the Rule perspective, and the Rule Project Map is in the bottom tab bar.
2. In the Decision Service Map in the Rule Project Map tab, click **Create main rule project**.
3. Ensure that **Main Rule Project** is selected under Decision Service Rule Projects, and click **Next**.
4. In the **Project name** field, type `my decision service`. Use the default location.

The name `my decision service` is used throughout the tutorial. You can use a different name to avoid conflicts with other projects when you publish the decision service. For example, you can prefix `my decision service` with your initials.

5. Click **Finish**.

The Rule Explorer displays `my decision service`.

The rule project contains empty folders. You use the `rules` and `bom` folders to store your rules and BOM.

Step 3: Importing the XOM into your rule project

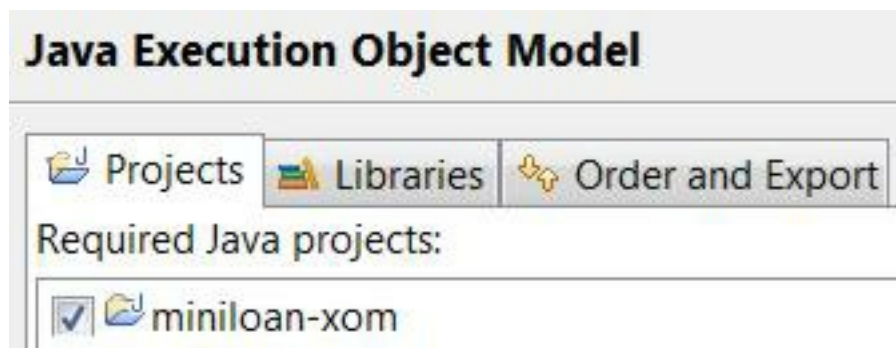
You import the XOM from the `Miniloan` Java project.

Procedure

1. In the Rule Explorer, click `my decision service`.

The Decision Service Map displays the steps to follow to design your rule project.

2. In the Decision Service Map, click **Import XOM**.
3. In the Import XOM dialog, ensure that **Java execution object model** is selected, and click **OK**.
4. In the **Projects** tab, select `miniloan-xom` and click **Apply and Close**:



Step 4: Creating the business object model

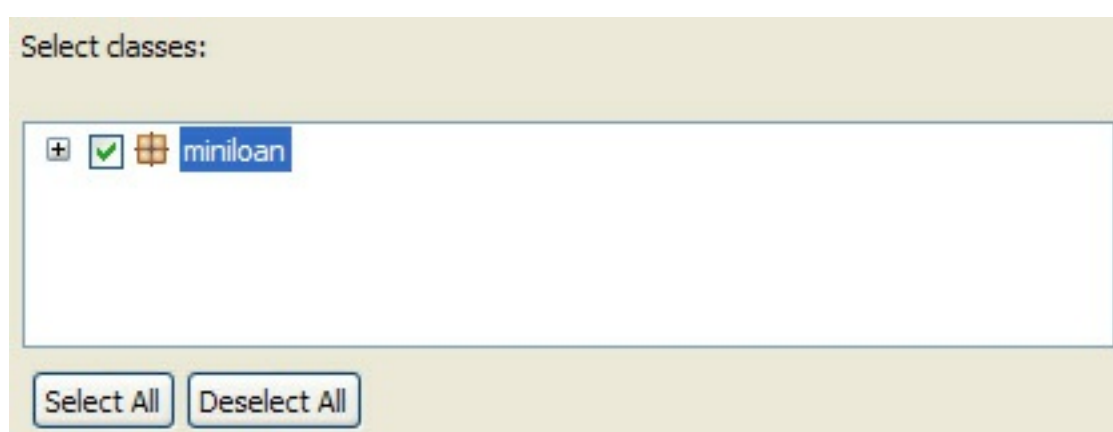
Before you can create and edit business rules, you must create a BOM. You can create a BOM manually or automatically by parsing your XOM. You use Rule Designer to parse the Java classes in the XOM, and create the BOM from their methods and properties. Then, you can write rules that use the verbalized terms that are contained in the BOM.

Procedure

1. In the Decision Service Map, click **Create BOM**.

Tip: You can also right-click the bom folder in the Rule Explorer and click **New > BOM Entry**.

2. In the New BOM Entry wizard, in the **Name** field, type miniloan.
3. Ensure that **Create a BOM entry from a XOM** is selected, and then click **Next**.
4. In the **Choose a XOM entry** field, click **Browse XOM**, select platform:/miniloan-xom, and then click **OK**.
5. Under **Select classes**, select the miniloan package. Selecting the package selects all the classes in the package.



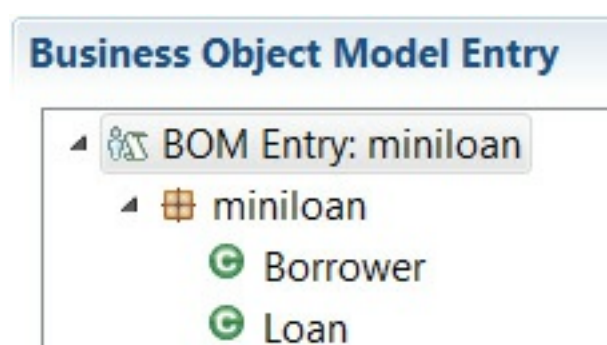
The miniloan package contains the Borrower and Loan classes.

6. Click **Next**.
7. On the BOM Verbalization page, you must select the **All Methods** check box. The **All Methods** check box ensures that all the methods are verbalized in addition to the elements already selected.



8. Click **Finish**.
9. In the Rule Explorer, double-click **bom > miniloan** to open the BOM Editor, and take a moment to look at the BOM.

In the BOM Editor, expand the miniloan entry:



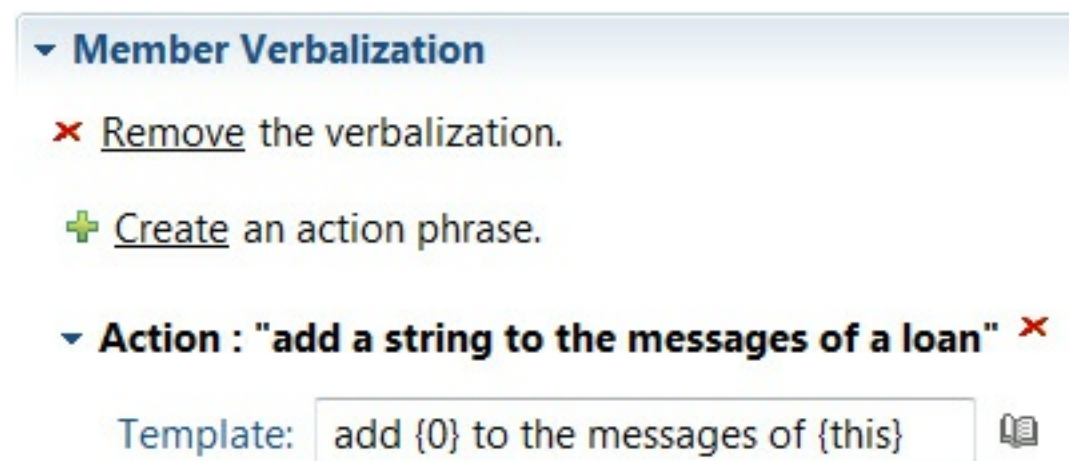
In your BOM, you now have two classes that are equivalent to the XOM classes Borrower and Loan.

10. Double-click the Loan class to open it in the BOM editor.

All the Java members and methods are converted and assigned a default verbalization.

11. In the Members section, double-click the `addToMessages(String)` method.

The BOM editor switches to the Member tab. In the Member Verbalization section, you can see that the verbalization of this method is add a string to the messages of a loan:



The verbalization is also used in the Rule Editor.

12. Close the miniloan tab to close the BOM Editor:

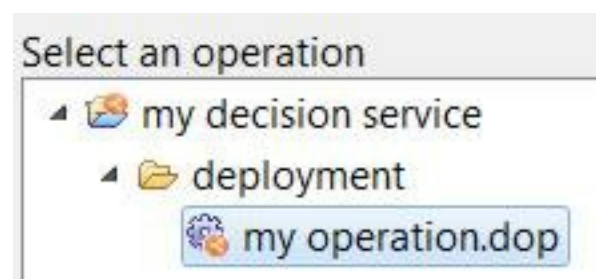


Step 5: Defining a decision operation

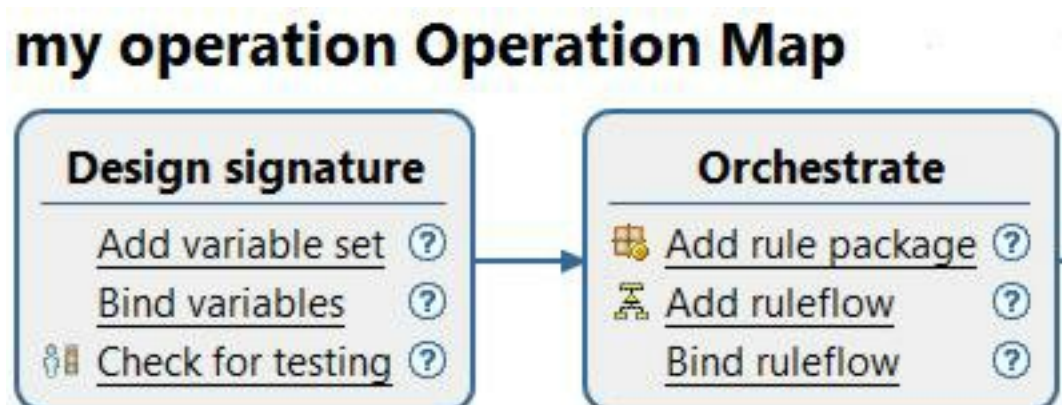
A ruleset is a runnable package that includes rule artifacts and other elements. It contains a set of rules that can be run by the rule engine. You must define the contents of the ruleset and the parameters that allow the client application and the ruleset to exchange information. A decision operation includes all the settings that are needed to define the contents of the ruleset and its parameters.

Procedure

1. In the Decision Service Map, click **Add decision operation**.
2. In the New Decision Operation wizard, in the **Name** field, type `my operation`.
3. Click **Next**. The Source Rule Project window opens.
4. Ensure that `my decision service` is selected, and then click **Finish**.
5. In the Decision Service Map, click **Go to operation map**.
6. Select **my decision service > deployment > my operation.dop**, and click **OK**.



The operation map opens and shows the different actions that you can do. The links in the map open different editors.



Step 6: Designing the operation signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation. They are equivalent to Java method parameters. They are references that you can use when you write rules.

Procedure

To enable a decision to be made on the status of a loan, you create ruleset parameters for the borrower and the loan:

- The borrower is an IN parameter. The value of the IN parameter is provided as input to the ruleset when

run.

- The loan is an IN_OUT parameter. The value of the IN_OUT parameter is provided as input to the ruleset when run, and can be modified by the ruleset and provided as output when the run completes.

1. In the operation map, click **Add variable set**.
2. In the New Variable Set wizard, in the **Name** field, type my parameters.
3. Click **Finish**.
4. Define the borrower parameter:
 - a. Click **Add**. A new row is displayed with default values.
 - b. In the **Name** column, type borrower.
 - c. In the **Type** column, click the ... button, and then double-click the **Borrower** type in the Matching types box.

The miniloan.Borrower type is displayed in the **Type** column.

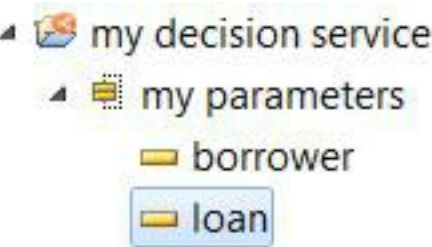
- d. In the **Verbalization** column, type the borrower.
5. Repeat step 4 to define the loan parameter:
 - Name: loan
 - Type: **Loan**
 - Verbalization: the loan

Your variable set is shown as follows:

Variable Set: my parameters



Name	Type	Verbalization	Initial Value
borrower	miniloan.Borrower	the borrower	
loan	miniloan.Loan	the loan	

6. Click the Eclipse **Save** button to save your work, and close the my parameters variable set editor.
7. In the Operation Map, click **Bind variables**.
8. Expand my parameters in the Eligible variables part of the Decision Operation Signature editor:



- a. Drag borrower to the input parameters.
- b. Drag loan to the input-output parameters.

The decision operation signature is displayed as follows:

Input Parameters		
Define the parameters required to call the execution.		
Parameter name	Verbalization	Type
 borrower	the borrower	miniloan.Borrower
Input - Output Parameters		
Define the parameters that are required, modified, and then returned by the execution.		
Parameter name	Verbalization	Type
 loan	the loan	miniloan.Loan

9. Save your work, and close my operation.

What to do next

In the next task, you create a ruleflow to orchestrate how the rules run.

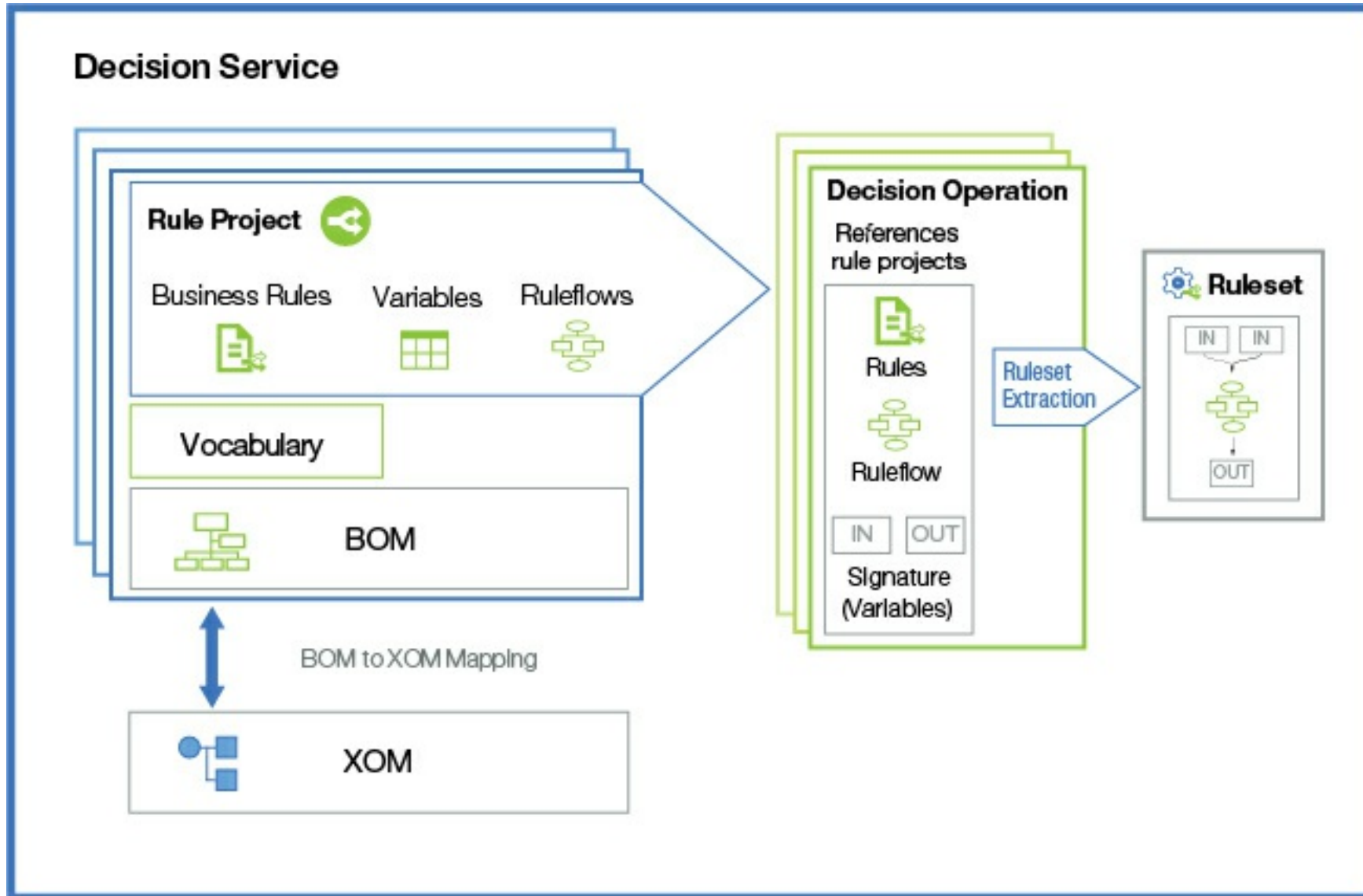
[< Previous](#) | [Next >](#)

Task 2: Orchestrating rule running

You create a ruleflow to set the sequence in which the rules run.

About this task

In Rule Designer, you use a ruleflow to orchestrate rules. The ruleflow defines the order in which the rule engine processes the rules.



Step 1: Creating rule packages

Before you define a ruleflow, you must organize your rules into packages that contain related rules. In this step, you create two rule packages, and then define the ruleflow for the packages.

Procedure

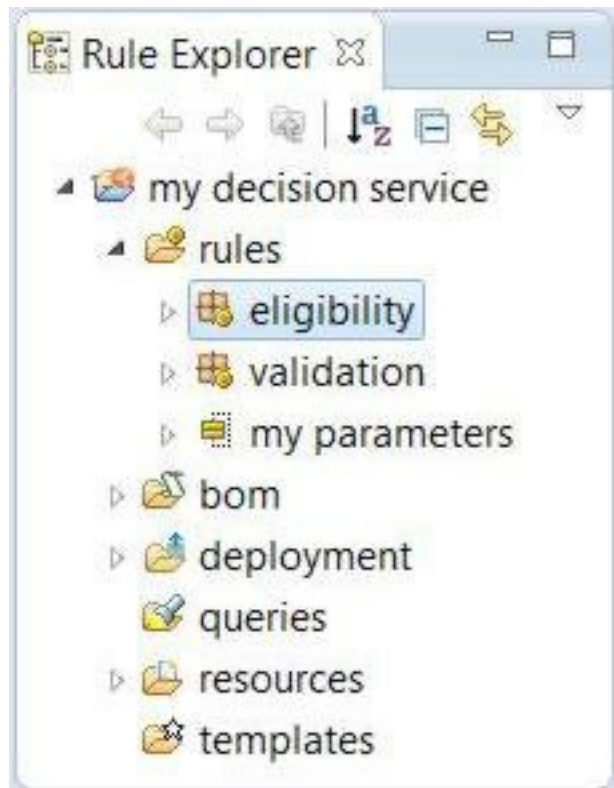
1. In the Operation Map, click **Add rule package**.

Tip: You can also right-click the my decision service/rules folder in the Rule Explorer and click **New > Rule Package**.

2. In the New Rule Package wizard, enter validation in the **Package** field, and then click **Finish**.

The validation rule package opens in the Rule Explorer.

3. Repeat steps 1 and 2 to create an eligibility rule package. Your rule project now contains two rule packages in which you can store rules.



Step 2: Creating the ruleflow diagram

You now design a ruleflow that creates a logical connection between the validation and eligibility rule packages. A ruleflow is a diagram that consists of several rule tasks that are connected by logical links. A ruleflow shows the sequence for running business rules.

Procedure

1. In the Operation Map, click **Add ruleflow**.

Tip: You can also right-click the my decision service/rules folder in the Rule Explorer and click **New > Ruleflow**.




2. In the New Ruleflow wizard, ensure that the **Source folder** field is set to /my decision service/rules, and that the **Package** field is empty.
3. In the **Name** field, enter miniloan.
4. Click **Finish**. The ruleflow editor opens.

You now construct a ruleflow in which you specify how tasks are related, that is, how, when, and under what conditions rules are run.

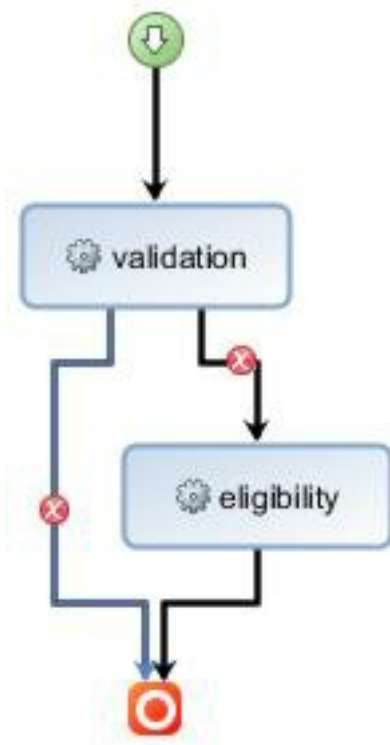
Step 3: Defining rule tasks

You add the start and end nodes, select the rule packages to include in the ruleflow diagram, and then you create the transitions between the packages.

Procedure

1. Click **Create a start node** , and then click inside the ruleflow editing window to add the node.
2. Click **Create an end node** , and then click inside the ruleflow editing window to add the node.
3. Drag the validation rule package from the Rule Explorer to the ruleflow editing window. The rule package becomes a rule task in the ruleflow.
4. Drag the eligibility rule package from the Rule Explorer to the ruleflow editing window.
5. Click **Create a transition**  and create the following transitions (shown as arrows) by clicking the first item and then clicking the second item:
 - a. Connect the start node to the validation task.
 - b. Connect the validation task to the eligibility task.
 - c. Connect the eligibility task to the end node.
 - d. Connect the validation task to the end node.

The ruleflow diagram shows errors on transitions to indicate that conditions are missing:



6. Click **Create a transition**  again to deselect the transition tool.
7. Click **Layout All Nodes**  to format the ruleflow diagram.
8. Save your work.

Step 4: Defining the main transition

You set a transition condition so that the rules in the eligibility package are run only when data is validated.

Procedure

1. Click the transition from validation to eligibility.

The Properties view shows the condition for this transition.

Tip: If you cannot see the Properties view, click **Window > Show View > Properties** to open the view.

2. In the Properties view, enter data approved in the **Label** field.
3. Ensure that **Use BAL for transition condition** is selected. The Business Action Language (BAL) is the language in which you write your conditions.
4. Click in the blank text area beneath **Use BAL for transition condition**, and then press Space to display the Content Assist box. Double-click the items to form the following statement: 'the loan' is approved.

Tip: You can also enter the statement directly in the text area.

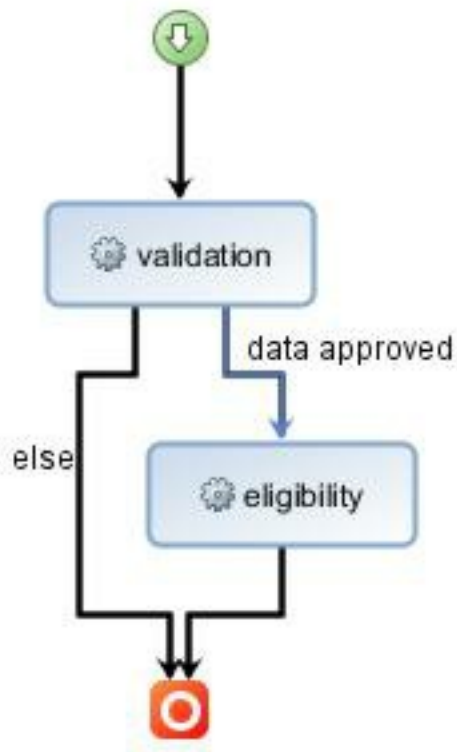
The Properties view looks as follows:



The transition from validation to the end node is automatically set to else.

5. Save your work.

Your ruleflow now looks like the following diagram:



Step 5: Defining the final action

You now have transitions between your rule packages. You can also define a final action to display a message in the console that indicates the status of the loan at the end of a rule run.

Procedure

1. Click the end node.
2. In the Properties view, in the **Final Action** section, ensure **Use BAL for action** is selected.
3. Click in the text area, and press Space to display the Content Assist box. Enter the following final action:

```
print the approval status of 'the loan' ;
```

You must add a semicolon (;) to the end of the action statement.
4. Save your work and close the ruleflow editor.

Step 6: Binding the ruleflow

To use the ruleflow at run time, you must bind the ruleflow to the decision operation.

Procedure

1. In the Operation Map, click **Bind ruleflow**.
2. In Decision Operation Overview, select **Use main ruleflow** in the Ruleflow section.
3. Click **<choose a ruleflow>** and select my decision service/rules/miniloan.
4. Click **OK**.
5. Save your work and close the decision operation editor.

What to do next

In the next task, you write an action rule.

[< Previous](#) | [Next >](#)

Task 3: Authoring business rules

You write an action rule and import other rules for your rule project.

About this task

As a developer, you write the initial business rules and design the rule templates. You want to enable business users to write and edit business rules in a web environment.

In this task, you create an action rule, and then you import the rest of the rules for your project. An action rule states the actions to take under certain conditions. You write the following rule, which is based partly on the vocabulary that you created earlier in this tutorial:

```
if
  the amount of 'the loan' is more than 1,000,000
then
  add "The loan cannot exceed 1,000,000" to the messages of 'the loan';
  reject 'the loan' ;
```

Step 1: Creating an action rule

You create an action rule that rejects any loan request that has an amount that is greater than 1,000,000.

Procedure

1. In Rule Designer, in the Author part of the Operation Map, click **Add action rule**.

Tip: You can also right-click the validation package in the Rule Explorer and click **New > Action Rule**.

2. In the **Package** field, type validation, and in the **Name** field, type maximum amount.

Source folder:

Package:

Name:

Template:

Type:

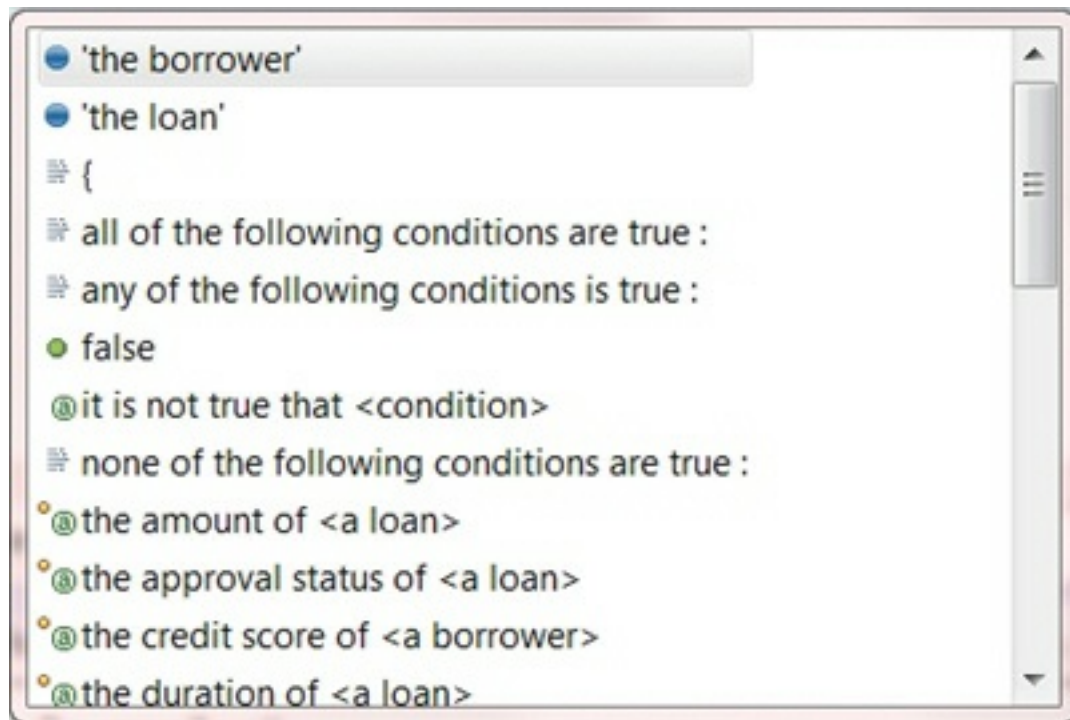
3. Click **Finish**. The Intellirule editor opens.

Step 2: Completing the action rule

You use the code completion menu in Intellirule to complete the action rule. You must create the parts of the rule, and use the completion mechanism to select phrases from a list of rule fragments to author the rule.

Procedure

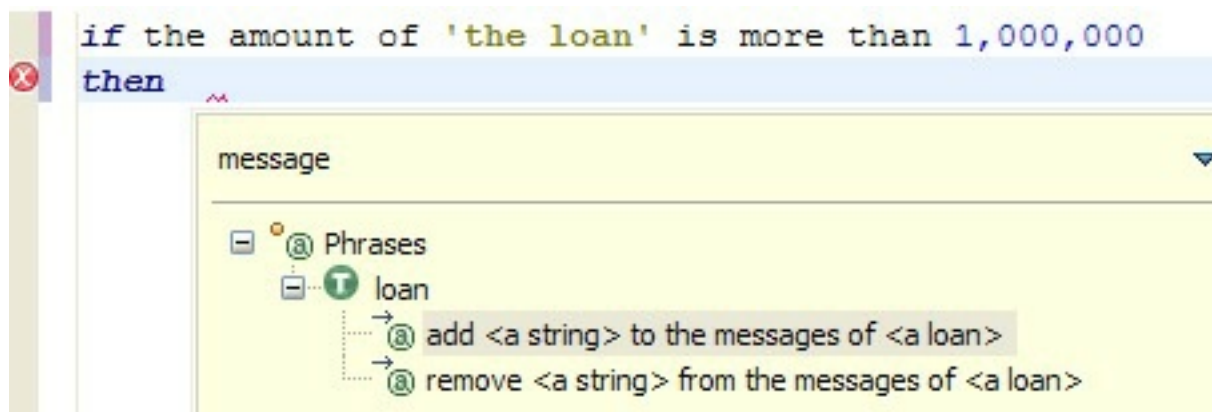
1. In the Intellirule editor, type `if`, and then press Space. The Content Assist box opens:



Select terms and phrases from the menu to compose the following expression:

```
if the amount of 'the loan' is more than 1,000,000
```

- On the next line, type then, press Space, and then press Ctrl+Shift+Space to activate the tree view of the completion menu. The tree view displays possible phrases to select for your rules, and a blinking prompt to enter text in a field.
- Type message in the text field. The tree view displays terms and phrases about messages:



- Double-click add <a string> to the messages of <a loan> to add the phrase to the rule. When the Content Assist box opens, use it to complete the phrase to create the following expression:

```
add "The loan cannot exceed 1,000,000" to the messages of 'the loan' ;
```

Important: You must add a semicolon (;) to the end of the line.

- If you are still in the Content Assist box, press Esc.
- Press Enter to create a new line, type a space, and then write the following statement. You can use the Content Assist box:

```
reject 'the loan' ;
```

- Press Ctrl+Shift+F to format the rule.
- Save your work and close the Intellirule editor.

Step 3: Importing the remaining rules

You must now add the rules that determine whether a borrower is eligible for a loan. In this step, you import the eligibility rules into your rule project.

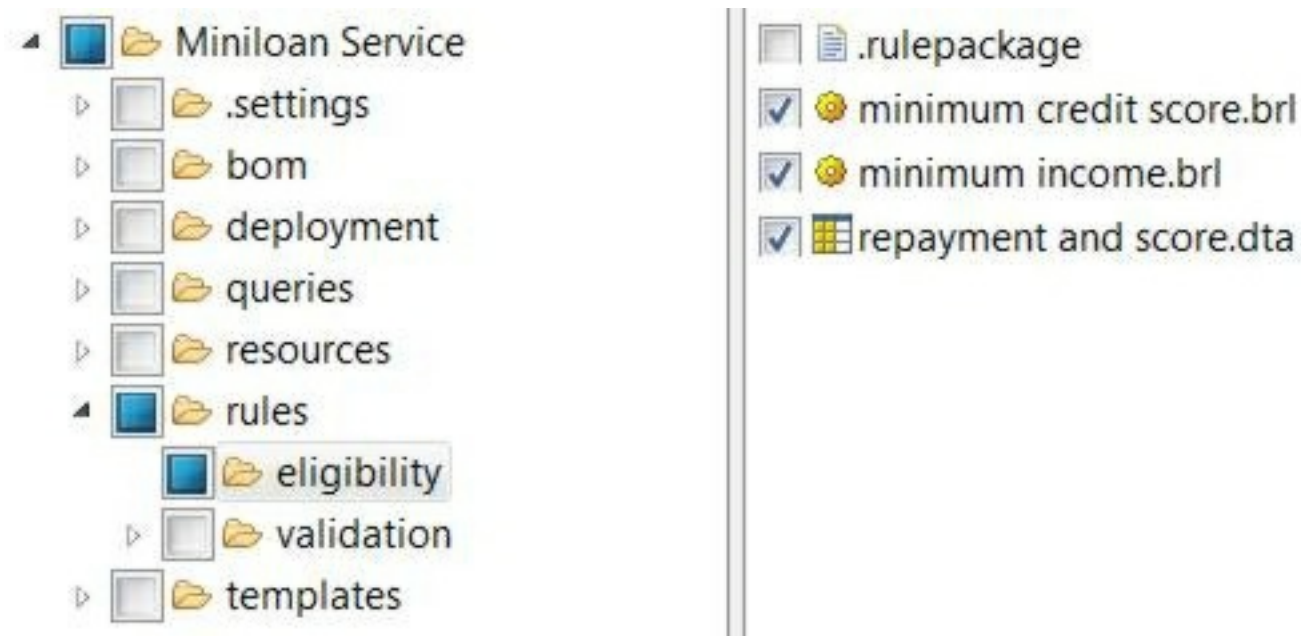
Procedure

- Click **File > Import > General > File System**, and then click **Next**.
- Next to the **From directory** field, click **Browse**. Navigate to <InstallDir>/odm-cloud-getting-started-master, select Miniloan Service, and then click **OK**.

Note: Ignore the message There are no resources currently selected for import. The Eclipse message prompts you to do the following action.

- Expand Miniloan Service/rules in the Import wizard.
- Select the check box for eligibility. You might have to click **Deselect All** before selecting eligibility.
- Click the name eligibility to display the contents of the folder.

6. Clear the .rulepackage check box, which you do not need for the tutorial. The Import wizard looks as follows:



7. In **Into folder**, ensure that my decision service is shown. If not, click **Browse**, select my decision service, and then click **OK**.
8. Under Options, select the **Overwrite existing resources without warning** option to prevent duplication:



9. Click **Finish**.

Step 4: Viewing the imported rules

You can view the imported rules in your project.

Procedure

1. In the Rule Explorer, expand eligibility and double-click the rules to review them:

minimum credit score

Rejects the loan if the credit score is too low.

minimum income

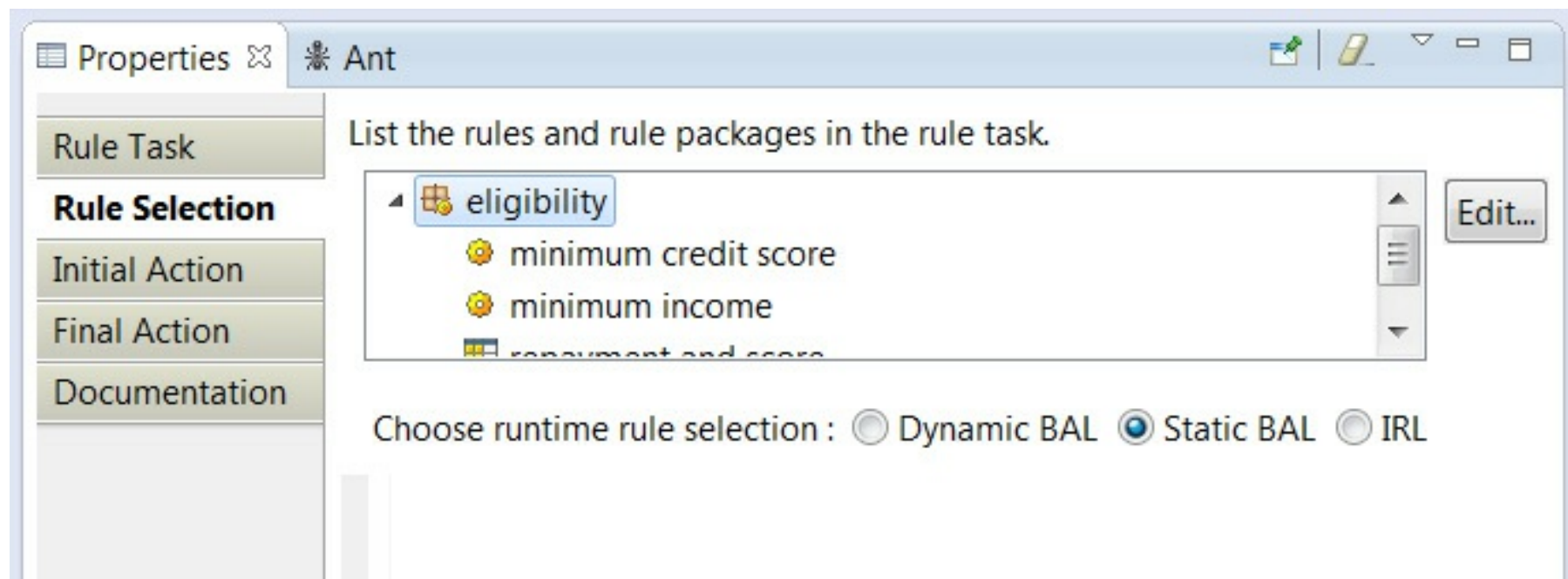
Rejects the loan if the income is too low for the yearly repayments.

repayment and score

Rejects the loan based on various values of the debt-to-income ratio and credit score. As a decision table, it represents rules that share conditions and actions. Each row in the table represents a rule. Place your cursor over the number of a row to view the corresponding rule as hover help.

Note: Rows with no actions display a warning about invalid or incomplete rows. These rows fill gaps in the table, and they are ignored when you run the project.

2. Double-click the miniloan ruleflow to open it.
3. In the Ruleflow Editor, double-click the eligibility task.
4. In the Properties view, open the Rule Selection tab, and expand the eligibility package.



By default, all the rules are used when the task is run by the rule engine.

5. Close the Ruleflow Editor.

What to do next

In the next task, you test and debug your rules.

[< Previous](#) | [Next >](#)

Task 4: Testing and debugging a ruleset

You run the ruleset by entering input data in a decision operation configuration, and then test and debug your ruleset.

About this task

You use Rule Designer to test and debug your rule project. You create a run configuration to run the decision operation in Rule Designer. Then, you insert a breakpoint that stops the run at a specific point in the ruleflow, and run the decision operation by using a debug configuration.

Step 1: Creating a run configuration

To test whether the ruleset can run, you run the decision operation in Rule Designer.

Procedure

1. In the Eclipse menu bar, click **Run > Run Configurations**.
2. Right-click **Decision Operation** and click **New**.
3. In the **Name** field, enter `Miniloan Test` for the name of the launch configuration.
4. To set the decision operation, browse to `my decision service/my operation`, and click **OK**.
5. On the Parameters & Arguments tab, click `borrower` and **Edit Value**.
6. In the Edit Parameter Value view, ensure that the Expression value is selected and enter the following text:

```
new miniloan.Borrower("Joe", 600, 8000)
```

7. Click **OK**.
 8. Repeat steps 5 to 7 to set `loan` to the following value:
- ```
new miniloan.Loan(50000, 240, 0.05)
```
9. Click **Apply**, and then click **Run**. In the Console view, you see the results:

```
false [Too big Debt-To-Income ratio]
```

#### Tip:

To open the Console view, on the **Window** menu, click **Show View > Other > General > Console**, and then click **OK**.

### Step 2: Inserting a breakpoint

You set a breakpoint in Rule Designer to debug rules.

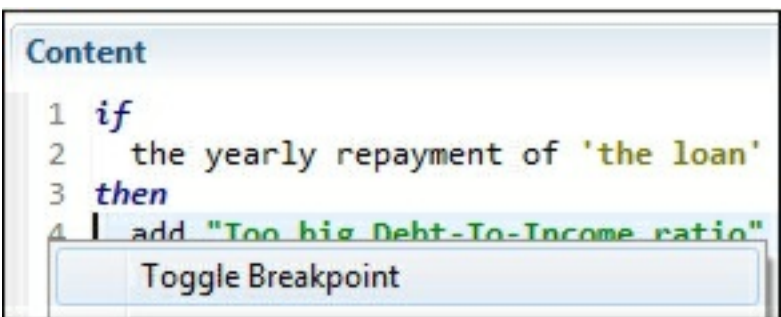
#### Procedure

1. In the Rule Explorer, double-click the `miniloan` ruleflow to display it in the Ruleflow Editor.
2. Select and right-click the `eligibility` task in the ruleflow diagram, and then click **Toggle Breakpoint**.

A breakpoint marker is added to the `eligibility` task.



3. In the Rule Explorer, open `eligibility` and double-click the `minimum income` rule to open it in the Intellirule editor.
4. Right-click the fourth line of the rule in the margin, and select **Toggle Breakpoint** to add a breakpoint to the first action in the rule:




### Step 3: Running the debugger

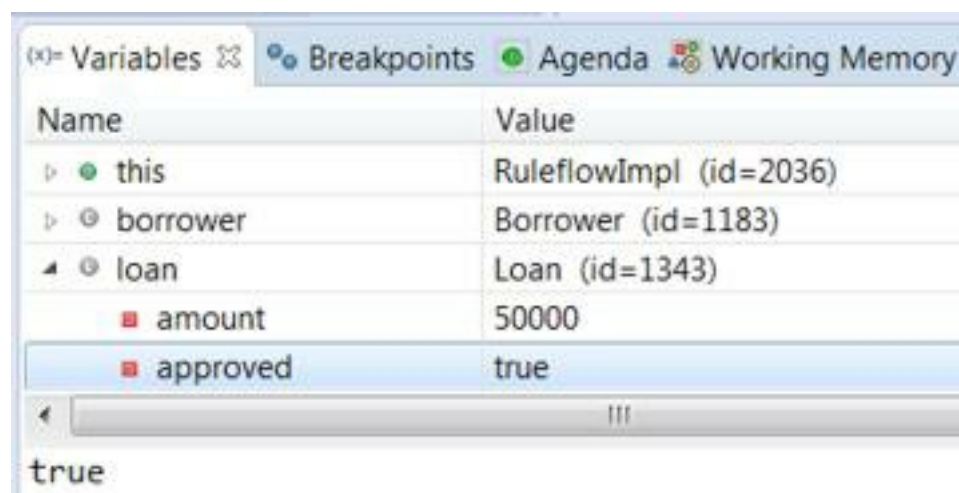


With the breakpoint in place, you can debug the run.

## Procedure

1. Start the debugger:
  - a. Click **Debug > Debug Configurations**.
  - b. Select **Decision Operation > Miniloan Test**.
  - c. Click **Debug**.
  - d. When the Confirm Perspective Switch dialog opens, click **Yes** to open the Debug perspective. The debugging commands are available from the Debug view and from the **Run** menu.

Debugging stops at the beginning of the eligibility task, where you inserted the breakpoint.
2. Step through the rule code:
  - a. Click **Resume** . Debugging stops at the first action of the minimum income rule at the second breakpoint that you inserted.
  - b. In the Variables view, expand the loan object. The value of the approved attribute is true:



3. Click **Resume**  to complete the run.

When the run ends, the Console view shows the following message:

```
false [Too big Debt-To-Income ratio]
```

4. Return to the Rule perspective, and close the ruleflow.

## What to do next

In the next task, you deploy a RuleApp from your decision service, and open it in the Rule Execution Server console.

[< Previous](#) | [Next >](#)

## Task 5: Deploying your decision service

You deploy your decision service to Rule Execution Server.



### About this task

Rule Execution Server is the running environment for business rule applications. It provides the management, performance, security, and logging capabilities that are associated with the running of rules. In Rule Designer, you create a deployment configuration in your decision service and use it to deploy a RuleApp to Rule Execution Server. You open the RuleApp to view its contents, and complete the task by removing the RuleApp.

### Step 1: Creating a deployment configuration

Before you can deploy your decision service, you must create a deployment configuration that references the decision operation and lists a target server.

#### Procedure

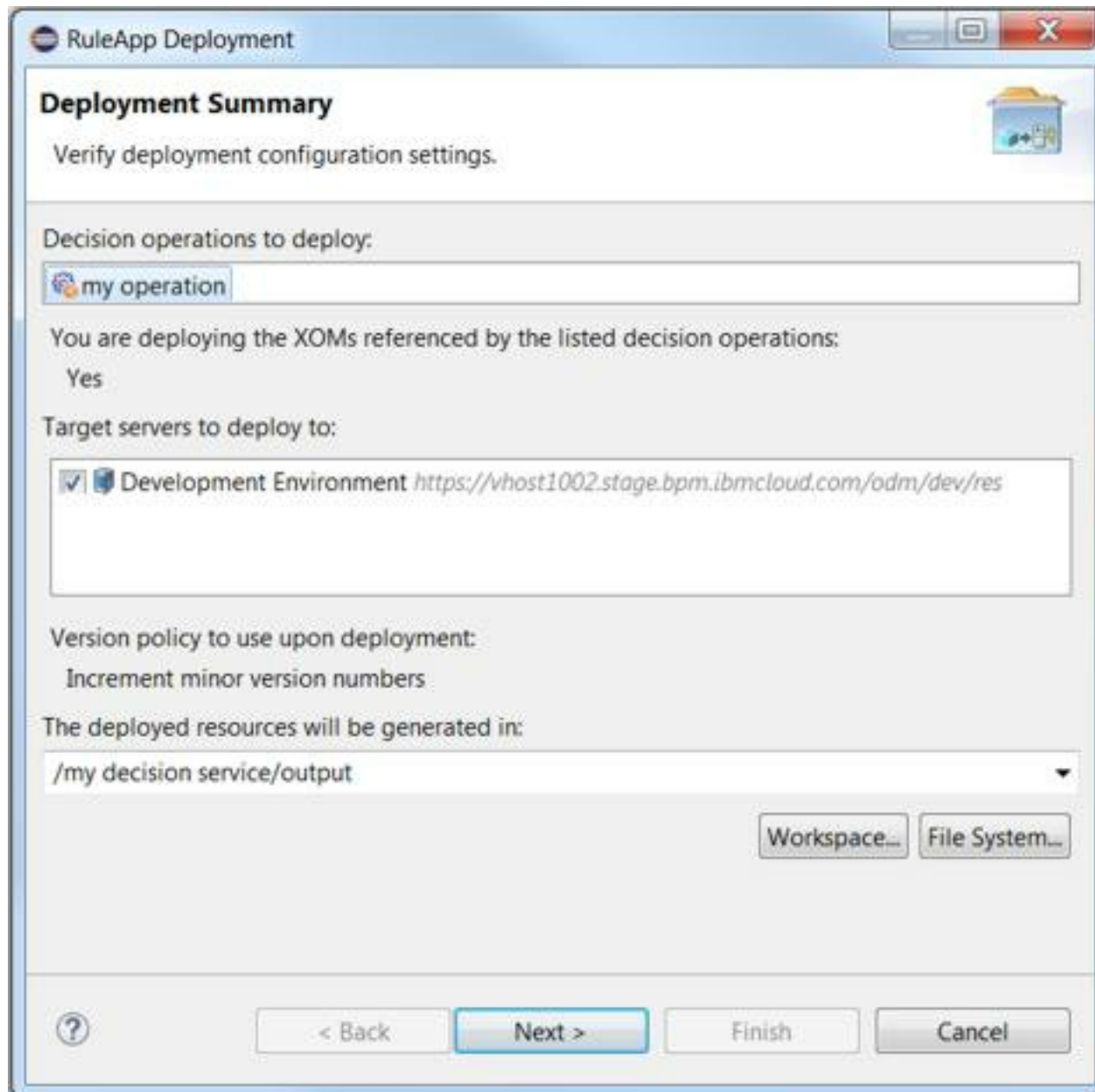
1. Right-click `my decision service`, and click **New > Deployment Configuration**.
2. Enter `my deployment` as the name, and make sure `/my decision service/deployment` is the deployment folder.  
  
The name `my deployment` is used in this task. You can use a different name to avoid conflicts with other projects when you deploy the decision service. For example, you can prefix `my deployment` with your initials.
3. Click **Finish**. The deployment overview opens.
4. Make sure **Nonproduction** is selected for the type of deployment. You use **Nonproduction** because the decision service is destined for the cloud development environment. **Production** is used to deploy a decision service for use by client applications. From Rule Designer, you can deploy to only the cloud development environment, and not to the test or production environment.
5. Check that `my_deployment` is the RuleApp name, and that `1.0` is the version number. Rule Designer added the underscore to the RuleApp name.
6. Click **Include decision operations** to select a decision operation.
7. Click **Add**  in Configured Decision Operations.
8. Click **Select existing decision operations**, select `my decision service`, and then click **Finish**.
9. In the **Target Servers** tab, click **Add** , select **Development Environment**, and click **Finish**. Check that the URL for the Development Environment matches the address of the Rule Execution Server in your cloud development environment. The address starts with the URL that you use to connect to the cloud portal, and ends with `/odm/dev/res`, for example, `https://<vhostname>.bpm.ibmcloud.com/odm/dev/res`.
10. Save your work.

### Step 2: Deploying the decision service

You deploy the RuleApp from Rule Designer to Rule Execution Server. The RuleApp contains a ruleset that holds the rules in your decision service.

#### Procedure

1. Right-click `my decision service` in the Rule Explorer.
2. Click **Rule Execution Server > Deploy**. The RuleApp Deployment wizard opens. It shows `my operation` as the decision operation, the `Development Environment` as the target server, and `/my decision operation/output` as the deployed resources.



3. Click **Next**. If you are not already authenticated, click **Connect** to authenticate your cloud connection. You must enter your cloud credentials when you authenticate your connection.
4. Click **Next**. Verify the RuleApp and ruleset versions, and then click **Finish**.

When the RuleApp is deployed, a deployment report is displayed in Rule Designer.



### Step 3: Viewing the deployed RuleApp

You view the RuleApp in Rule Execution Server.

#### Procedure

1. Sign in to your instance of the Operational Decision Manager on Cloud portal.
2. Click **Launch** in the Rule Execution Server console section to open the component.
3. Click the **Explorer** tab.
4. In the Navigator, expand **RuleApps**, and then **/my\_deployment/1.0**.

You see that Rule Execution Server contains version 1.0 of my\_deployment, which contains version 1.0 of the my\_operation ruleset:



5. Click **/my\_operation/1.0** to view the details of the ruleset in the Ruleset View.

The Ruleset View shows information about the ruleset, including its name, version number, and creation date, and provides commands for viewing performance information and adding elements such as properties. It also shows the version of the decision engine, and that the status of the ruleset is enabled,

indicating that it can be run.

|                                                                                                                       |                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|  /my_deployment/1.0/my_operation/1.0 |                                                                                            |
| Name                                                                                                                  | my_operation                                                                               |
| Version                                                                                                               | 1.0                                                                                        |
| Creation Date                                                                                                         | Mar 24, 2016 3:09:32 PM GMT+01:00                                                          |
| Display Name                                                                                                          | my operation                                                                               |
| Description                                                                                                           |                                                                                            |
| Rule engine                                                                                                           | Decision Engine - 1.40.0                                                                   |
| Status                                                                                                                |  enabled  |
| Debug                                                                                                                 |  disabled |

## Step 4: Testing the ruleset by using the REST API

Before you publish your decision service to Decision Center, you run it in Rule Execution Server to see whether it produces the expected results.

### Procedure

1. Click **Retrieve HTDS Description File** in the Ruleset View. The file retrieval page opens.
2. Select **REST**, and click **Test**. The Hosted Transparent Decision Service opens.
3. Change the run request:
  - creditScore: 100
  - yearlyIncome: 70000
  - amount: 8000
  - duration: 12
  - yearlyInterestRate: 2.5
  - Remove lines 16-19

Your run request looks as follows:

```
1 <par:Request xmlns:par="http://<...>
2 <!--Optional:-->
3 <par:DecisionID>string</par:DecisionID>
4 <!--Optional:-->
5 <par:borrower>
6 <!--Optional:-->
7 <name>string</name>
8 <creditScore>100</creditScore>
9 <yearlyIncome>70000</yearlyIncome>
10 </par:borrower>
11 <!--Optional:-->
12 <par:loan>
13 <amount>8000</amount>
14 <duration>12</duration>
15 <yearlyInterestRate>2.5</yearlyInterestRate>
16 </par:loan>
17 </par:Request>
```

4. Click **Execute Request**. You get the following server response:



```
<?xml version="1.0" encoding="UTF-8"?><par:Response xmlns:par="http://<...>
 <par:DecisionID>string</par:DecisionID>
 <par:loan>
 <amount>8000</amount>
 <duration>12</duration>
 <yearlyInterestRate>2.5</yearlyInterestRate>
 <yearlyRepayment>22301</yearlyRepayment>
 <approved>false</approved>
 <messages>Credit score below 200</messages>
 <messages>Too big Debt-To-Income ratio</messages>
 <messages>debt-to-income too high compared to credit score</messages>
 </par:loan>
</par:Response>
```

The rule application produces the expected results. The loan is rejected, and messages explain that the debt-to-income ratio is too high when compared to the credit score.

## Step 5: Removing the RuleApp

After you test the RuleApp, you no longer need it in Rule Execution Server and remove it.

### Procedure

1. Click **RuleApps** in the Navigator.
2. Select the check box for /my\_deployment/1.0.
3. Click **Remove**. Rule Execution Server deletes the RuleApp.

### What to do next

In the next task, you share your rules with other business users by publishing your project to Decision Center.

[< Previous](#) | [Next >](#)

[< Previous](#)

## Task 6: Publishing to Decision Center

You publish your decision service to Decision Center in the Operational Decision Manager on Cloud portal, where you view your files in the Business console.

### About this task

You now make your decision service available in Decision Center, a cloud environment where users view, create, and modify rules. You publish your decision service from Rule Designer to Decision Center, and can periodically synchronize the work of the business users with your Rule Designer copy.

This task is optional for learning how to make rules in Rule Designer. To do this task, you must have access to Decision Center in the cloud portal.

#### Important:

Before publishing your decision service, make sure the Decision Center Business console does not contain a decision service with the same name. If so, remove the other decision service from the console or change the name of your decision service in Rule Designer before you publish it.

To change the name of your decision service:

1. Right-click your decision service.
2. Click **Refactor** > **Rename**.
3. Enter a new name or modify the existing name.
4. Click OK. Rule Designer refactors the decision service.

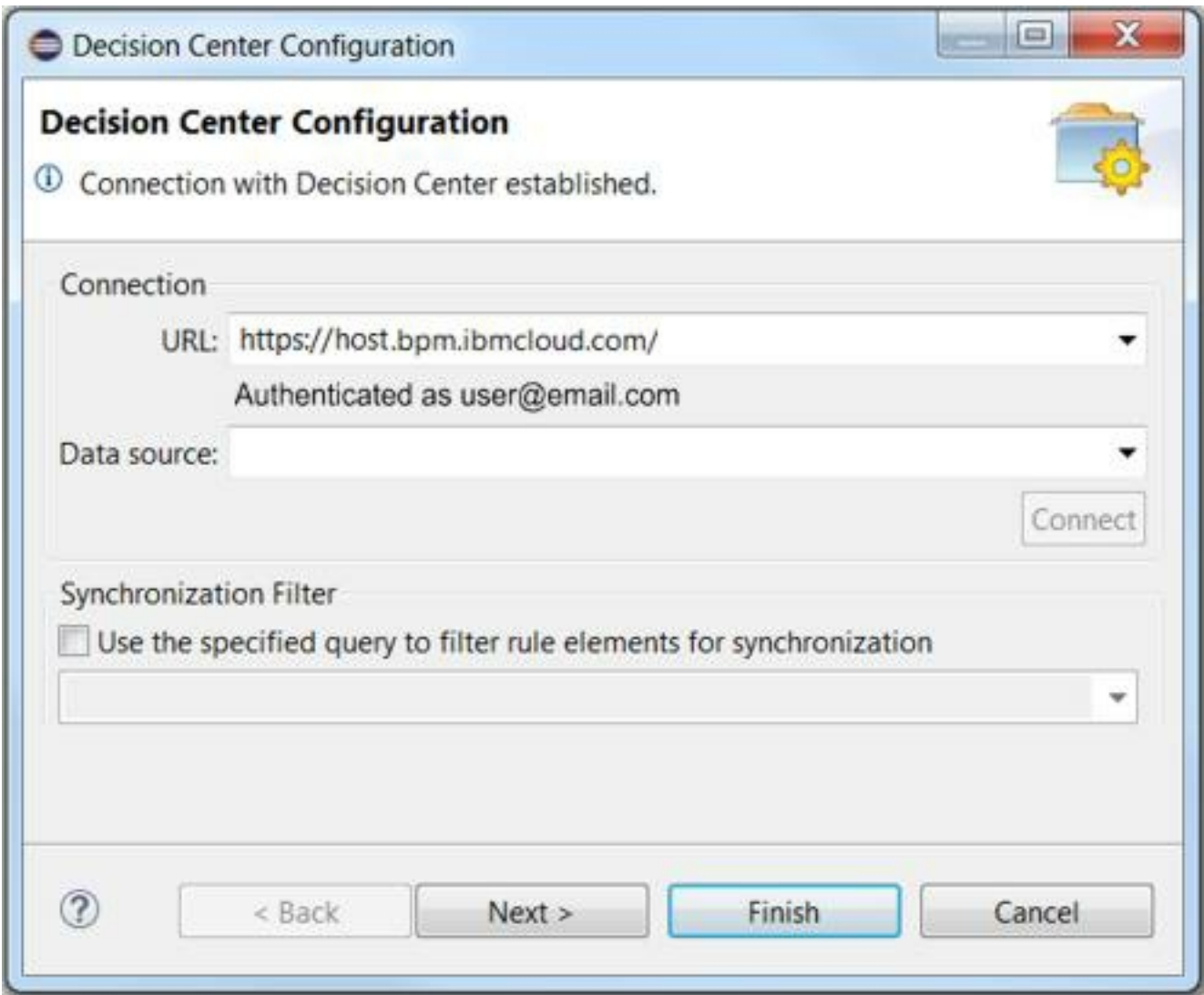
### Step 1: Publishing the decision service to Decision Center

Ensure that you have user access to the cloud portal. You connect Rule Designer to Decision Center, and then publish your decision service to Decision Center.

#### Procedure

1. In Rule Designer, right-click your decision service (my decision service or the renamed version), and then click **Decision Center** > **Connect**.
2. In the connection wizard, enter the URL for your instance of the cloud portal if it is not already predefined. Use the following format: `https://<myodmcloud>.bpm.ibmcloud.com/`
3. Click **Connect**, and enter your user name and password for your cloud instance.

When the connection is established, the following message is displayed in the connection wizard: Connection with Decision Center established.



4. Click **Finish** to publish the decision service to Decision Center.

**Tip:** If the wizard shows that you are authenticated but you cannot click **Finish**, click **Connect** to

reestablish your connection.

5. Click **OK** to close the Synchronize Complete dialog.
6. Click **Yes** in the a dialog to change to the Team Synchronizing perspective.

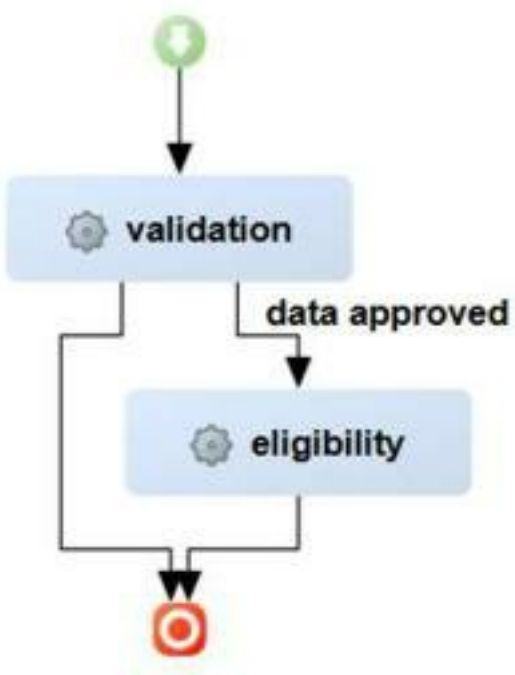
An empty Synchronize view opens. It shows that there are no changes in the project. Your rules are now published to Decision Center.

Step 2: Exploring the rule project in Decision Center

You open the Decision Center Business console to see your rules in the web-based editing environment.

Procedure

1. Sign in to the cloud portal, and open the Decision Center Business console.
2. Click **Library**. The library opens on the decision services by default.
3. Click your decision service (my decision service or the renamed version). The decision service shows tabs for releases and branches.
4. Click **Branches > main > Decision Artifacts** to view your rule files.
5. Click **All Project**, select **All Types**, and click **Apply** to display all the decision artifacts.
6. Click miniloan to see the ruleflow:



7. In the validation package, click maximum amount to see the contents of the rule:

```
if
 the amount of 'the loan' is more than 1000000
then
 add "The loan cannot exceed 1,000,000" to the messages of 'the loan' ;
 reject 'the loan' ;
```

8. In the eligibility package, click repayment and score to see the contents of the decision table:

|   | debt to income |      | credit score |     | message                                          | loan approved                       |
|---|----------------|------|--------------|-----|--------------------------------------------------|-------------------------------------|
|   | min            | max  | min          | max |                                                  |                                     |
| 1 | 0 %            | 30 % | 0            | 200 | debt-to-income too high compared to credit score | <input type="checkbox"/>            |
| 2 | 0 %            | 30 % | 200          | 800 |                                                  | <input checked="" type="checkbox"/> |
| 3 | 30 %           | 45 % | 0            | 400 | debt-to-income too high compared to credit score | <input type="checkbox"/>            |
| 4 | 30 %           | 45 % | 400          | 800 |                                                  | <input checked="" type="checkbox"/> |
| 5 | 45 %           | 50 % | 0            | 600 | debt-to-income too high compared to credit score | <input type="checkbox"/>            |
| 6 | 45 %           | 50 % | 600          | 800 |                                                  | <input checked="" type="checkbox"/> |
| 7 | ≥ 50 %         |      | 0            | 800 | debt-to-income too high compared to credit score | <input type="checkbox"/>            |

The warning icons indicate empty cells. These cells fill gaps in the table, and they are ignored at run time.

Results

You have completed the tutorial. Your business rules are now available in Decision Center, and can be maintained by business users.

Deleting your decision service

If you no longer need the decision service, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the](#)

[tutorial project.](#)

[< Previous](#)



[< Previous](#) | [Next >](#)

## Debugging a decision service in Rule Designer

This tutorial provides a methodology for debugging a decision service in Rule Designer.

### Learning objectives

You use debugging features to find and fix errors in a decision service. You do the following tasks:

- Use automatic exception handling.
- Set breakpoints in a ruleflow, action rule, and decision table.
- Run a debugging session.
- Identify and fix errors.

### Best practices

This tutorial includes the following best practices for debugging decision services:

- Carefully read the error messages. They tell you where to look for errors. [Example...](#)
- Try automatic exception handling when an exception occurs in a rule condition. [Example...](#)
- Check the stack trace to get the names of ruleflows and rule tasks. [Example...](#)
- Switch to the debug mode to get more precise error information. [Example...](#)
- Use the search function to find rule artifacts. [Example...](#)
- Use queries to search for rules by their content. [Example...](#)
- Check the values in the Variables view at each breakpoint during debugging. [Example...](#)
- Check the Agenda view to see which rules are going to be run. [Example...](#)

### Time required

40 minutes

[< Previous](#) | [Next >](#)

## Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work as a developer for a fictitious loan company. You are currently working on a decision service that validates loan requests. Called the Loan Validation Service, the decision service applies several criteria to determine the eligibility of potential borrowers, and contractual constraints for approved loans.

A business user is reviewing the results of the decision service. The user contacts you to tell you that the decision service rejects low-risk loans that actually qualify for approval. You must debug the decision service to determine why it rejects the loans.

In Rule Designer, you tag rule artifacts with breakpoints that stop the running of the decision service at key points. You can then examine the behavior of the rules in a step-by-step debugging process. You determine where the errors take place, and track them to specific rules. Then, you modify the rules to fix the errors.

**Note:** This tutorial is also available on GitHub. You can find it at [Tutorial: Debugging a decision service in Rule Designer](#).

### Audience

This tutorial is for users who make decision services in Rule Designer.

### Prerequisites

You need the following component, projects, and information:

- Rule Designer: Download this component from the Operational Decision Manager on Cloud portal and install it as directed.
- The following project files in the [Tutorial: Debugging a decision service in Rule Designer](#) GitHub repository:
  - answer: Contains the completed version of the decision service. You can run it to see the expected results.
  - start: Contains the faulty version of the decision service. You look for errors in it during the debugging process.

In the GitHub repository, click the **Clone or download** button to download the contents of the repository to a directory on your computer. Expand the downloaded file in the directory. In the tutorial, the directory is referred to as InstallDir.

- Knowledge of business rule programming principles, Java™, and the Eclipse environment

## Lessons in this tutorial

### [Running the completed decision service](#)

You run the completed version of the decision service to see the expected results.

### [Task 1: Using automatic exception handling](#)

You import and run the faulty decision service. You find an exception error in a rule, and try to fix the problem by using automatic exception handling.

### [Task 2: Setting breakpoints](#)

You set breakpoints to determine where a null pointer exception occurs.

### [Task 3: Debugging a rule](#)

You set breakpoints in a rule and run a debug session. When you find an error, you make changes to correct it.

### [Task 4: Debugging a decision table](#)

You add breakpoints to an action rule and a decision table, and then run a debugging session to find and fix an error.

### [Task 5: Debugging a ruleflow](#)

You debug a ruleflow to determine why the decision table does not approve the loan. When you find the cause, you reorganize the ruleflow to achieve the correct results.

## Running the completed decision service

You run the completed version of the decision service to see the expected results.

### About this task

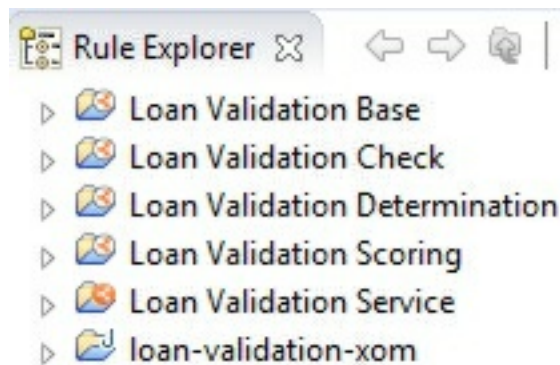
This version of the decision service contains the changes that you make in the tutorial. It produces the results that you can expect to achieve at the end of the tutorial.

**Tip:** If you want to refer to the completed decision service while you are going through the tutorial, use the answer and start projects in different Eclipse workspaces.

### Procedure

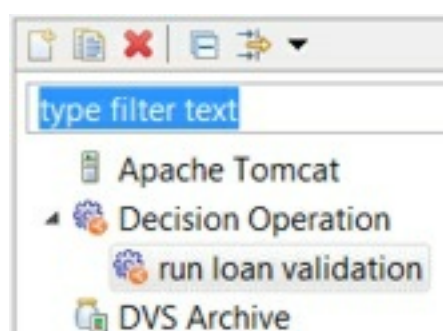
1. Start Rule Designer on your computer in the en\_US (American English) locale (see [Installing Rule Designer](#)).
2. Close the Eclipse welcome page if it is open.
3. Open the Rule perspective if it is not open. Click **Window > Perspective > Open Perspective > Other > Rule** to open the perspective.
4. Import the answer project that you downloaded from GitHub:
  - a. In the Rule perspective, click **File > Import**.
  - b. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
  - c. Choose Select root directory, and browse to `<InstallDir>/answer`. InstallDir is the name of the directory to which you downloaded the answer project from GitHub.
  - d. Click **Select All**, and in Options, click **Copy projects into workspace**.
  - e. Click **Finish**.

The Rule Explorer displays six projects:



The Java™ project loan-validation-xom defines an execution object model (XOM) for rule execution. The main decision service, Loan Validation Service, references the other projects.

5. In the **Run** menu, select **Run Configurations**.
6. Expand **Decision Operation**, and click the run loan validation configuration:



7. Click **Run**. The Console view displays the following results, which show that the input data is valid and the loan is approved:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
 - Yearly income 20,000
 - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved true
 - Rate 0.055
 - Monthly repayment 570.83
 - Insurance required true Insurance rate 2%
 - Message Low risk loan
Congratulations! Your loan has been approved
```

## What to do next

In the task, you run the faulty version of the decision service and see the errors in the output.

[< Previous](#) | [Next >](#)

## Task 1: Using automatic exception handling

You import and run the faulty decision service. You find an exception error in a rule, and try to fix the problem by using automatic exception handling.

### About this task

Before you make any changes to the faulty decision service, you want to see the results that it produces. You import the decision service into Rule Designer, and then run it. The results show a null pointer exception in a rule condition. Before going further, you activate the automatic exception handling feature to see whether it can correct the exception.

### Step 1: Running the decision service

You import the start projects for the tutorial, and run the decision service.

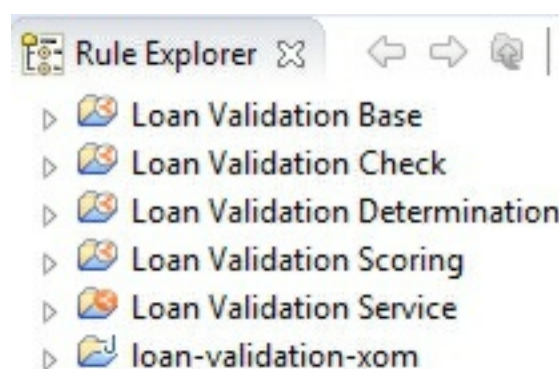
#### Before you begin

If you have imported the answer projects to run the completed decision service, delete them before importing the start projects into the same Eclipse workspace. Otherwise, use the start projects in a different Eclipse workspace.

#### Procedure

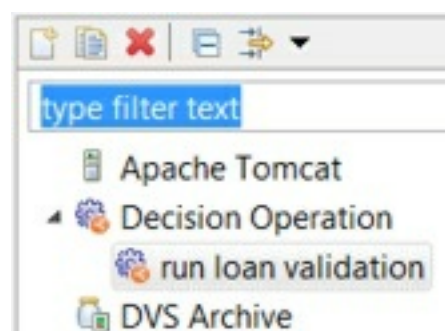
1. Start Rule Designer on your computer in the en\_US (American English) locale.
2. Close the Eclipse welcome page if it open.
3. Open the Rule perspective if it is not open. Click **Window > Open Perspective > Other > Rule** to open the perspective.
4. Import the start project that you downloaded from GitHub:
  - a. In the Rule perspective, click **File > Import**.
  - b. In the Import wizard, expand **General > Existing Projects into Workspace**, and click **Next**.
  - c. Choose Select root directory, and browse to `<InstallDir>/start`. InstallDir is the name of the directory to which you downloaded the start project from GitHub.
  - d. Click **Select All**, and in Options, click **Copy projects into workspace**.
  - e. Click **Finish**.

The Rule Explorer displays six projects:



The Java™ project loan-validation-xom defines an execution object model (XOM) for rule execution. The main decision service, Loan Validation Service, references the other projects.

5. From the **Run** menu, select **Run Configurations**.
6. Expand **Decision Operation**, and select the run loan validation configuration:



7. Click **Run**.

The Console view shows the results, which include the following exception error:

```
Error while executing ruleset: java.lang.NullPointerException
com.ibm.rules.engine.util.EngineExecutionException: java.lang.NullPointerException
 at com.ibm.rules.engine.fastpath.runtime.AbstractFastEngineServices.handleExceptionRaisedInCondition(AbstractFastEngineServices.java:69)
```

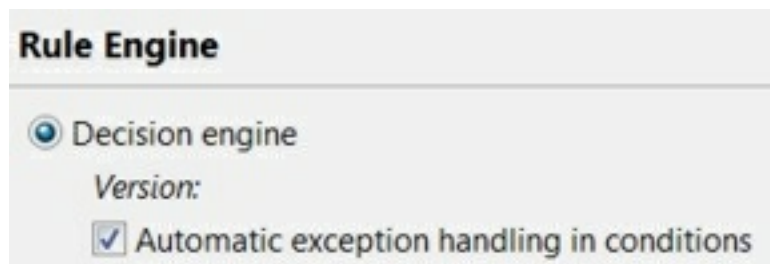
### Step 2: Applying automatic exception handling

Because the exception occurs in a rule condition, you decide to try to fix it by using automatic exception handling.



## Procedure

1. In the Rule Explorer, right-click Loan Validation Service.
2. In the pop-up menu, click **Properties**, and select **Rule Engine** in the left column of the **Properties** dialog.
3. Select **Automatic exception handling in conditions**, and then click **OK**:



4. Run the run configuration that is shown in step 1 of this task. The results no longer show an exception error. The decision engine automatically handles the exception. To determine where the exception takes place, you add more logging information to the results.

## Step 3: Adding exception handing logging

You add additional logging information to determine where the decision engine automatically handles the exception error. First, you look at the Loan Validation Service/logging.properties file, and set it as the logging configuration file. Then, you set the Java VM parameter of the run configuration to declare the path to the logging configuration file.

## Procedure

1. In the Rule Explorer, double-click Loan Validation Service/logging.properties. The file sets the Console as a logging handler:

```
handlers = java.util.logging.ConsoleHandler
```

It also sets the log level for the automatic exception handling to FINE.

2. Open **Run > Run Configurations**, and select the run loan validation configuration.
3. Open the **Parameters and Arguments** tab, and add the following VM argument:

```
-Djava.util.logging.config.file="${workspace_loc}/Loan Validation
Service/logging.properties"
```

4. Click **Apply**, and then click **Run**. The Console shows the results, which include a log message for the automatic exception handling:

```
Jul 27, 2017 12:28:25 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

The results tell you that the exception takes place in a condition in the checkZipcode rule.

## What to do next

In the next task, you add breakpoints to a ruleflow to determine where the exception handling occurs.

[< Previous](#) | [Next >](#)

## Task 2: Setting breakpoints

You set breakpoints to determine where a null pointer exception occurs.

### About this task

You look at the automatic exception handling log message:

```
Jul 27, 2017 12:28:25 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

It lists the loanvalidation ruleflow and its validation rule task as follows:

```
ruleflow.loanvalidation$003evaluation
```

You decide to track the processes in the task by placing a breakpoint on it. Then, you search for the rule that produces the error, add more tracing, and then run a debug configuration.

### Step 1: Setting a breakpoint

You run the decision service, search for the source of an error, and add a breakpoint.

#### Procedure

1. Run the decision service as shown in [Task 1: Using automatic exception handling](#). The results contain the following error code:

```
ruleflow.loanvalidation$003evaluation
```

You understand the code as follows:


- Ruleflow name: loanvalidation
- Rule task: validation

You want to find the validation task in the loanvalidation ruleflow.

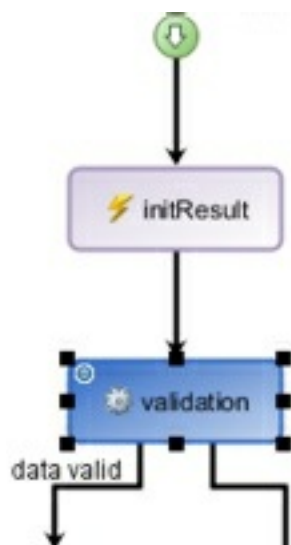
2. Click **Search** > **Search** in the toolbar.
3. Open the **Rule Search** tab.
4. Enter loanvalidation in the search field.

**Tip:** You can copy the name of the ruleflow in the error message, and paste it in the search field.

5. Select **Ruleflow**, and then click **Search**. The loanvalidation ruleflow is displayed in the Search view:

 loanvalidation - [Loan Validation Service] - /Loan Validation Service/rules

6. Double-click the ruleflow in the Search view to open it.
7. Click the validation task to select it.
8. Right-click the task and click **Toggle Breakpoint** at the bottom of the pop-up menu to tag the task with a breakpoint. The task now shows a dot to indicate the addition of the breakpoint:



### Step 2: Preparing the debug session

Because the checkZipcode action rule shows an exception, you look for the rule and add trace information to it.

#### Before you begin

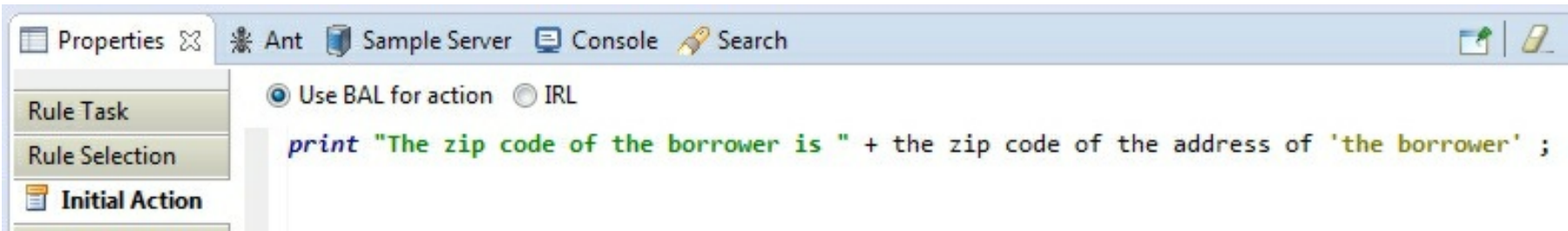
In this step, you navigate to the action rule through the **Rule Selection** tab of the properties of the ruleflow task. Optionally, you can do a rule search in **Search** to find the action rule by name.

Procedure

- 1. Look at the automatic exception handling message. You see that the null pointer exception occurs in the checkZipcode rule condition:
- 2. Double-click the validation task in the loanvalidation ruleflow.
- 3. In the Properties tab, click **Rule Selection** and expand the validation package.
- 4. Expand borrower, and double-click the checkZipcode action rule to open it in the rule editor. You see that the condition in the rule uses the length of the zip code. You want to know the value of the zip code, so you add trace information to print the value.
- 5. Go back to the loanvalidation ruleflow in the ruleflow editor.
- 6. Double-click the validation task.
- 7. In the Properties view, select **Initial action**. Keep **Use BAL for action** selected.
- 8. Add the following code to display the borrower zip code:

```
print "The zip code of the borrower is " + the zip code of the address of 'the borrower' ;
```

The code looks as follows:



- 9. Save your work.

Step 3: Running the debugger on a ruleflow

You run a debug configuration to check the values of the borrower zip code. You are looking for null values to determine where the null pointer exception occurs.

Procedure

- 1. Click **Run > Debug Configurations** in the toolbar.
- 2. Open the run loan validation configuration and click **Debug**.
- 3. If the Save and Launch dialog opens, click **OK** to save your changes before running the debugger.
- 4. Click **Yes** in the Confirm Perspective Switch dialog box to open the debug perspective. The debugger stops at the beginning of the validation rule task, at the breakpoint that you set in step 1 of this task.
- 5. Expand borrower in the **Variables** tab. Notice the null values for latestBankruptcy, spouse, and zipCode:

|                  |                             |
|------------------|-----------------------------|
| borrower         | Borrower (id=1550)          |
| birth            | GregorianCalendar (id=1967) |
| creditScore      | 200                         |
| firstName        | "Smith" (id=1403)           |
| lastName         | "John" (id=1790)            |
| latestBankruptcy | null                        |
| spouse           | null                        |
| SSN              | Borrower\$SSN (id=1828)     |
| yearlyIncome     | 20000                       |
| zipCode          | null                        |

- 6. Click the **Resume** button. The results in the Console view begin with the following message:

```
The zip code of the borrower is null
Jul 27, 2017 12:48:39 PM com.ibm.rules.generated.ruleflow.loanvalidation$003evaluation.FASTEngine exceptionLogger
FINE: AEH_LOG: java.lang.NullPointerException: In 'checkZipcode' rule condition, at offset 47, and length 87
```

The first line is the trace that you added to the initial action of the validation task. You see that the automatic exception handling catches the null pointer exception because of the null zip code in the checkZipcode rule. The condition part has an unknown status, so the rule is ignored. You tell the business user that the borrower in the configuration has a null zip code, and you are told that it is not a problem.

The automatic exception handling correctly addresses the problem. You can remove the trace for the logging information that comes from the automatic exception handling. You keep the trace on the initial action of the validation task to get the zip code value.



**Note:** The Console shows the AEH\_LOG message twice. Because the checkZipcode rule contains a then action and an else action, its condition part is evaluated twice, producing two AEH\_LOG messages.

7. Switch to the Rule perspective.
8. Click **Run > Run Configurations**, and open the run loan validation configuration.
9. Delete the VM arguments from the **Parameters and Arguments** tab.
10. Click **Apply**, and then **Run**. The automatic exception handling trace information is no longer shown. However, the loan is still not approved, so there are more errors to fix:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
 - Yearly income 20,000
 - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved false
 - Rate 0.055
 - Monthly repayment ∞
 - Message Very risky loan
Too big Debt/Income ratio: ∞
We are sorry. Your loan has not been approved
```

### What to do next

You see in the output that the monthly repayment value has an infinite value, which is not normal. In the next task, you look for the rule that generates the value.

[< Previous](#) | [Next >](#)

## Task 3: Debugging a rule

You set breakpoints in a rule and run a debug session. When you find an error, you make changes to correct it.

### About this task

When you run the decision service, it does not generate the correct monthly repayment rate. In the results, the rate is prefaced with the variable name `monthly repayment`. You create a query to find the rule that uses the variable in an action. Then, you place a breakpoint on the action in the rule, and another breakpoint on the code that computes the monthly repayment rate. You run a debugging session, which leads you to an error in the Java code, which you try to fix.

### Step 1: Setting a breakpoint in a rule

You create a query to find the action rule that uses the variable `monthly repayment`, and then add a breakpoint to track the rule.

#### Procedure

1. Run the decision service as shown in [Task 1: Using automatic exception handling](#). In the results, you see that the report does not show the correct monthly repayment value. You want to find the source of this value to determine how to fix it.
2. In the Rule Explorer, expand the `Loan Validation Service` project.
3. Right-click the queries folder, and click **New > Query**.
4. Enter `Monthly` as the name in the New Query dialog, and click **Finish**.
5. Click **<enter a condition>** and add search parameters to form the following query:

```
Content
1 Find all business rules
2 such that the definition of each business rule contains "monthly"
```

**Tip:** The search function is case sensitive. Enter the name of the variable as you expect to find it in the action rule.

6. Click **Run query**, and click **Yes** in the Save Resource dialog. The Search view shows the repayment action rule because it contains the word `monthly`:

```
Monthly - 1 matches
🔍 repayment - [Loan Validation Scoring] - /Loan Validation Scoring/rules/computation
```

7. Double-click the action rule in the Search view to open it in the rule editor. You see that the first action, line 7, sets the value of the `monthly repayment` variable.
8. Open the **ARL** tab. It shows the Advanced Rule Language that is applied by the action rule.

The rule has three main parts: the definition of the variables, the condition beginning with `when`, and the action beginning with `then`. In the action part, you see that the rule calls the `loan.LoanUtil.getMonthlyRepayment` method to calculate the monthly repayment rate. You must step into this method.

9. Return to the rule editor, right-click line 7, and click **Toggle Breakpoint** in the pop-up menu. A breakpoint is displayed next to the line:


```
• 7 set the monthly repayment of 'the loan report' to
```


In the next step, you debug the action rule.

### Step 2: Debugging the action rule

You run a debugging session. When it stops at breakpoints, you look for reasons for the `monthly repayment` error.

#### Procedure

1. Run the `run loan validation` configuration in the Debug Configurations.
2. Click **OK** in the Save and Launch dialog if it opens, and click **Yes** in the Confirm Perspective Switch dialog. The Debug perspective opens.
3. The debugger stops at the `validation` task in the `loanvalidation` ruleflow. Right-click the task, and click **Toggle Breakpoint** to remove the breakpoint. You do not need this breakpoint for this task.
4. Click the **Resume** button . The Content view now shows the repayment action rule.
5. Open the **Variables** tab and expand `report`. The `monthlyRepayment` variable shows `0.0`.

- Click the **Step Into** button . The debugger stops at `LoanUtil.java`. The Java code shows the following method at the breakpoint:

```
32 public static double getMonthlyRepayment(double amount, int numberOfMonth,
33 double yearlyRate) {
34 double i = yearlyRate / MonthsInYear;
35 double p = i * amount / Math.pow(1 + i, -numberOfMonth);
```

The function computes a monthly repayment rate by dividing `yearlyRate` by `MonthsInYear`. In the Variables view, you see that the value of `MonthsInYear` is 0. The error is that the function is dividing by 0, when it should divide by 12.

- Stop the debugging session by clicking the **Terminate** button .

### Step 3: Fixing the error in the rule

You change the value of the `MonthsInYear` variable to 12, and run the decision service to check the results.

#### Procedure


- Switch to `LoanUtil.java` in the Rule perspective.
- Change the value of the `MonthsInYear` variable to 12:

```
20 public class LoanUtil implements Serializable {
21 private static final long serialVersionUID = 7764736178720624835L;
22 private static final short MonthsInYear= 12;
```

- Save your changes, and run the decision service normally by selecting `run loan validation` in the Run Configurations. The Console now shows the correct monthly repayment rate of 570.83:

```
Report: Valid data true Approved false
- Rate 0.055
- Monthly repayment 570.83
- Message Low risk loan
We are sorry. Your loan has not been approved
```

However, the decision service still does not approve the loan, so you have more work to do.

- Remove all the breakpoints because you no longer need them:
  - Open **Window > Show view > Other**.
  - In the filter field, type `breakpoints`, select **Breakpoints** in the list, and click **OK**.
  - Click the **Remove All Breakpoints** button .
- Save your work.

#### What to do next

You have corrected the monthly repayment error. In the next task, you debug a decision table to try to fix the approval error.

[< Previous](#) | [Next >](#)



## Task 4: Debugging a decision table

You add breakpoints to an action rule and a decision table, and then run a debugging session to find and fix an error.

### About this task

The decision service still doesn't approve the loan. You carefully read through the log in the Console, and it leads you to a rule that determines whether a loan is approved under certain conditions. You add a breakpoint to the rule, and look for the source of the decision in the condition. This search takes you to a decision table, where you insert breakpoints. Using the debugger, you trace the error to a value in the decision table, and then fix it.

### Step 1: Setting breakpoints in an action rule

You run a query to find the rule that contains the message Your loan has not been approved. Then, you add breakpoints to the rule.

#### Procedure

1. Click **Run > Run Configurations**, open the run loan validation configuration, and click **Run**.

The results from running the decision service still show an error. The output indicates that the loan is low risk, but the decision service still does not approve the loan:

```
Report: Valid data true Approved false
- Rate 0.055
- Monthly repayment 570.83
- Message Low risk loan
We are sorry. Your loan has not been approved
```

You decide to search for the rule that makes the rejection message by running a query for the message.

2. In the Rule Explorer, expand the Loan Validation Service file.
3. Right-click the queries folder, and click **New > Query**. You create a query to find the rule that uses the following rejection message:

Your loan has not been approved.

**Tip:** If you do not see the **Query** command in the **New** menu, make sure that you are in the Rule perspective.

4. Enter Approval as the name in the New Query dialog, and click **Finish**.
5. Click **<enter a condition>** and add search parameters to create the following query to look for the rejection message:

```
1 Find all business rules
2 such that the definition of each business rule contains "Your loan has not been approved"
```

6. Click **Run query**, and click **Yes** in the Save Resource dialog. The Search view shows the approval action rule:

```
Approval - 1 matches
approval - [Loan Validation Determination] - /Loan Validation Determination/rules/eligibility
```

7. Double-click the action rule in the Search view to open it in the rule editor. The action depends on the grade that is given to the loan. If the grade does not equal A, B or C, the loan is rejected:

```
if
 'the grade' is one of { "A" , "B" , "C" }
then
 in 'the loan report', accept the loan with the message
 "Congratulations! Your loan has been approved" ;
else
 in 'the loan report', refuse the loan with the message "We are sorry.
 Your loan has not been approved" ;
```

8. Place breakpoints on both actions (lines 4 and 6) in the approval action rule:

```
3 then
4 in 'the loan report',
5 else
6 in 'the loan report',
```

Now you search for the rule that computes the grade condition for the approval rule.

### Step 2: Setting breakpoints in a decision table

The approval action rule applies a decision that is based on the grade that is given to the loan. You decide to locate the rule that assigns grades to the loans. You create a query that finds a decision table, and then you add breakpoints to the decision table to observe it in a debugging session.

#### Procedure

1. Right-click the queries folder, and click **New > Query**.
2. Name the query gradeSet, and define the query by entering the following search parameters:

```
1 Find all business rules
2 such that each business rule may lead to a state where ['the grade' is not one of { "A", "B", "C" }]
```

The query looks for a rule that can assign a grade that is not A, B or C, and therefore, causes the decision service to reject the loan.

3. Run the query. The Search view shows the grade decision table, and lists the rows that do not contain grade A, B or C:





4. Double-click the decision table or one of the rows in the list to open the table in the decision table editor.
5. Select the Grade column by clicking the column header cell. Right-click the A cell in row 1, and click **Toggle Breakpoint** in the pop-up menu. Every Grade cell now has a breakpoint:

| Grade |   |
|-------|---|
| •     | A |
| •     | A |
| •     | D |
| •     | A |
| •     | B |
| •     | C |
| •     | B |
| •     | C |
| •     | D |
| •     | C |
| •     | D |
| •     | E |

### Step 3: Debugging the decision table

You use a debugging session to find an error in the decision table.

#### Procedure

1. Run the run loan validation configuration in the Debug Configurations. The debugger stops at row 3, which shows grade D and the message Low risk loan.
2. Click the **Resume** button . The debugger stops at the second breakpoint in the approval action rule. The loan is rejected because the grade is D. The error is coming from row 3 of the grade decision table.
3. Click the **Terminate** button  to stop the debugging session.

You contact the business user, who tells you to change the grade in row 3 to B. You update the decision table to correct the error.

4. Open the grade decision table in the Rule perspective, and change the grade in row 3 to B.
5. Save your changes, and run the decision service normally by selecting run loan validation in the Run Configurations. The results show that the loan is low risk, but the decision service still rejects the loan.

#### What to do next

You have corrected the error in the decision table, but you are still not getting the expected results. In the next task, you debug a ruleflow to fix the last error.

## Task 5: Debugging a ruleflow

You debug a ruleflow to determine why the decision table does not approve the loan. When you find the cause, you reorganize the ruleflow to achieve the correct results.

### About this task

The decision service still rejects the loan. The rules run correctly now, so you decide to test the flow of decisions among the rules. You start by running a debugging session that associates the error with two rules. You determine that the rules reside in the same rule task in a ruleflow. This arrangement prevents one rule from processing information from the other rule. You reorganize the ruleflow by adding a rule task to make sure that the condition part of one rule takes into account the values that are computed in the action part of the other rule.

### Step 1: Debugging the ruleflow

You run a debugging session to determine the source of the error.

#### Procedure

1. Run the `run loan validation` configuration in the Debug Configurations. The debugger stops at row 3 of the grade decision table.
2. Open the **Agenda** tab in the Debug perspective. The tab shows two rules that are in the same eligibility rule task. The first entry points to the rule that is formed by row 3 in the grade decision table, and the second entry points to the approval action rule:

| Name                     | Value                    |
|--------------------------|--------------------------|
| ▶ ➡ eligibility.grade_3  | AgendaRuleInfo (id=1971) |
| ▶ ● eligibility.approval | AgendaRuleInfo (id=1039) |

In the grade decision table, you see that an arrow points to the grade cell in row 3. The grade is B, and the message for the row is Low risk loan:

| Grade | Message       |
|-------|---------------|
| ➡ B   | Low risk loan |

3. Click the **Resume** button . The debugger stops on the `else` line in the approval action rule, which rejects the loan:

```
5 else
6 in 'the loan report', refuse the loan with the message "We are sorry. Your loan has not been approved" ;
```

You open the Variables view and look for the grade value. You see that it is B as set in the grade decision table, but the action rule still goes to the `else` condition, which is for any grade other than A, B or C. The grade value in the approval rule is not set to B. The grade value is set in the decision table, but it is not seen by the action rule. This disconnect between the rules indicates a ruleflow problem. Both rules are probably in the same rule task. You look to see which algorithm is used by the rule task.

4. Click the **Terminate** button to stop the debugging session.

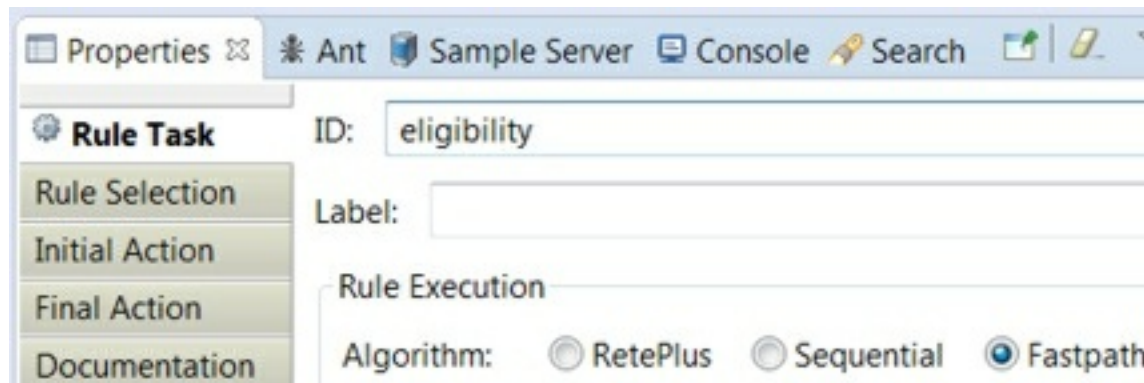
### Step 2: Modifying the ruleflow

You open the eligibility rule task in the `loanvalidation` ruleflow. The task uses an algorithm that prevents the approval rule from seeing changes to data in another rule in the task. You decide to reorganize the ruleflow by creating a task, and moving the approval rule to the new task. Data can then flow from the old task to the new task, where the approval rule can then see the data changes.

#### Procedure

1. Open the `loanvalidation` ruleflow in the Rule perspective.
2. Select the eligibility rule task. The rules in the **Agenda** tab are prefixed with `eligibility`, which is the name of the rule task.
3. Open **Properties > Rule Selection**, and expand `eligibility`. You see that the task contains both the approval action rule and the grade decision table.
4. Open the **Rule Task** tab. You see that the Fastpath algorithm is selected:






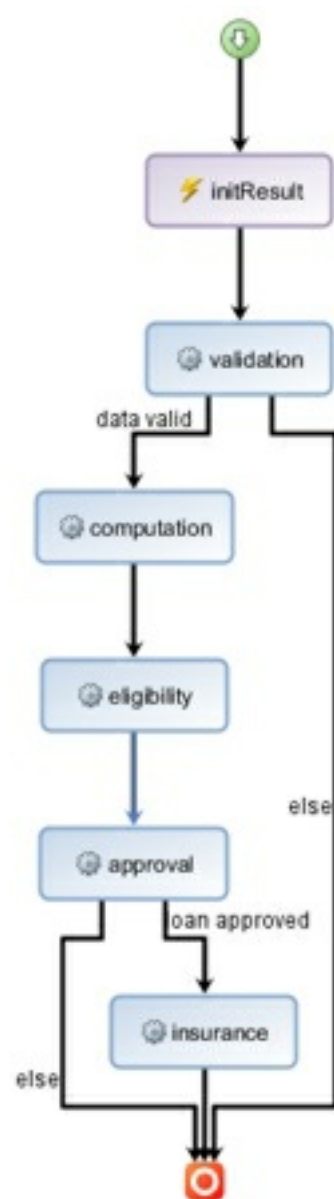
You know that the Fastpath algorithm does not support inference. The agenda is computed once in the rule task, but it is not changed if variables are modified by other rules in the task. The approval rule runs with an unset grade value, and the value is not changed by the grade decision table.

To fix the error, you decide to create a rule task, and move the approval action rule to the new task. Then, you organize the transitions so that data from the eligibility task flows to the new task. This arrangement allows the approval rule to see the data from the grade rule, which remains in the eligibility task.


5. Create a task node next to the insurance task in the ruleflow (see [Ruleflows](#)).

Enter the following parameters in the properties for the new task:

- Rule Task ID: approval
  - Rule Selection: eligibility.approval
6. Click the transition line from the eligibility task to the end node, and reset it to run from the approval task to the end node.
  7. Click the transition line from the eligibility task to the insurance task, and reset it to run from the approval task to the insurance task.
  8. Create a transition line between the eligibility task and the approval task.
  9. Click the **Layout All Nodes** button : The ruleflow now looks as follows:



10. Edit the eligibility task to remove the approval action rule from the **Rule Selection** tab. Remove the eligibility package, and then import all the eligibility rules except the approval rule:

 eligibility.checkCreditScore  
 eligibility.checkIncome  
 eligibility.grade

Now, the grade decision table runs in the eligibility task, and its grade value passes to the approval rule in the approval task.

11. Save your work.
12. Run the decision service by using the run loan validation configuration in the Run Configurations. The decision service runs correctly, and produces the expected report:

```
The zip code of the borrower is null
The borrower's age is valid.
The borrower's name is not empty.
The borrower's SSN area number belongs to an authorized area.
The borrower's SSN is well formatted.
monthly repayment calculated.
Borrower: First name Smith Last name John born Feb 1, 1950 SSN 123-12-1234
 - Yearly income 20,000
 - Credit Score 200
Loan: Amount 100,000 Starting Jul 27, 2017 Duration 48 month(s) Loan to value 120%
Report: Valid data true Approved true
 - Rate 0.055
 - Monthly repayment 570.83
 - Insurance required true Insurance rate 2%
 - Message Low risk loan
Congratulations! Your loan has been approved
```

13. Remove all the breakpoints as shown at the end of Task 3, and save your changes. Your decision service now works correctly. You can now make the new version of the decision service available to the business user for further review.

**Tip:** In this tutorial, you created a new rule task in a ruleflow, and changed the rule selection. This approach is not always possible. Alternatively, you can use inference by selecting the RetePlus algorithm in the Rule Task properties of the eligibility task (see [RetePlus algorithm](#)).

## Results

You have completed the tutorial. It showed you how to add breakpoints, and use a debugging session to follow the breakpoints through a decision service to find problems in rules.

[< Previous](#)



[< Previous](#) | [Next >](#)

## Getting started with decision modeling

This tutorial shows the basics of creating a decision model service. You do all the modeling and authoring of a decision service in the Decision Center Business console.

### Learning objectives

You do the following tasks:

- Create a decision model service.
- Author the decision logic.
- Create data types.
- Edit existing data and decision nodes.
- Test and deploy a decision model service.

### Time required

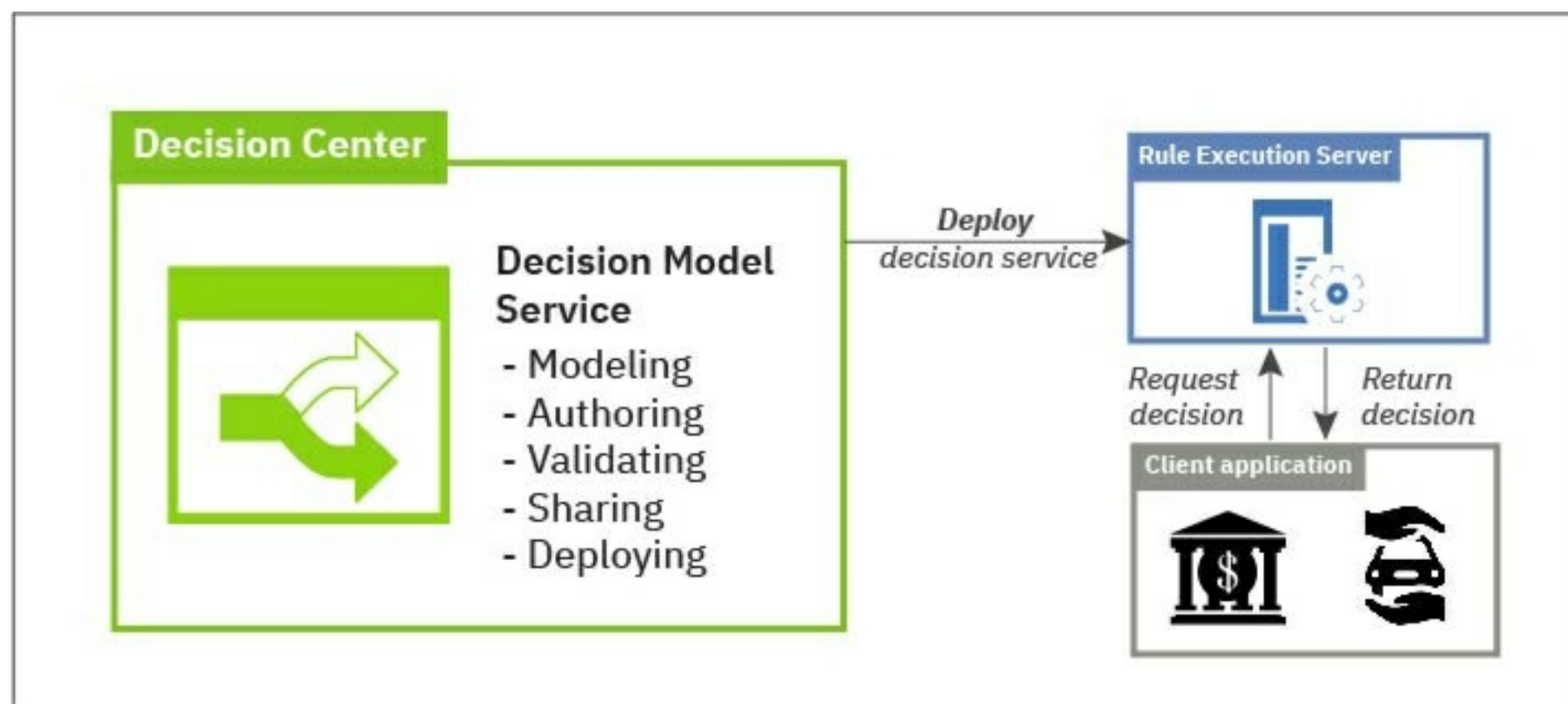
40 minutes

[< Previous](#) | [Next >](#)

## Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

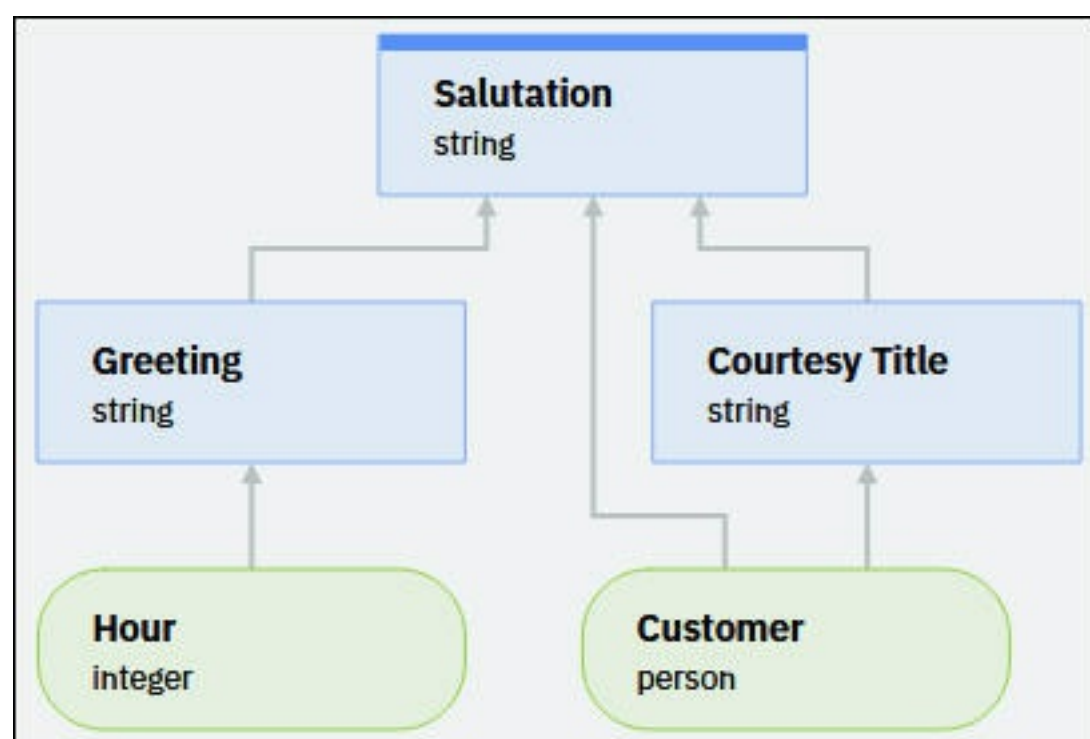
In the Decision Center Business console, you can create both the model and the logic of a decision service, and then deploy it to the execution environment, where it can be called by the client application.



The term *decision model service* identifies a decision service in which you create the model in the Business console. In a decision model service, you use a diagram to create data and decision nodes, and then author the decision logic of each decision node.

You create a *main* branch in this tutorial. The main branch serves as the foundation of a decision model service. After you complete a main branch, you should work in branches based on the main branch to avoid introducing errors to the original branch.

This tutorial guides you through creating, testing, and deploying a decision model service, and highlights differences with regular decision services. The decision model service provides a salutation to a customer. Initially, your salutation says Hello to the customer. Then, you add nodes to provide a salutation based on time of day, the gender of the customer, and any degree the customer might have. The final diagram looks as follows:



## Audience

This tutorial is intended for business users who want to create decision model services in the Business console.

## Prerequisites

The sample decision service and tutorial files are in English. You must use a supported browser (see [IBM® Operational Decision Manager on Cloud - Detailed System Requirements](#).)

## More information

For additional information on decision modeling in Decision Center, see the following sections:

- [Modeling decisions in the Business console](#)

- [Introducing the Business console](#)

## Lessons in this tutorial

### [\*\*Task 1: Creating the decision model service\*\*](#)

In this task, you create the project, the first nodes in the diagram, and the initial decision logic for your decision model service.

### [\*\*Task 2: Adding a decision node\*\*](#)

You add a decision node that affects the final decision.

### [\*\*Task 3: Creating a data type\*\*](#)

You create a new type of data, and a decision node that uses the new type.

### [\*\*Task 4: Modifying an existing decision\*\*](#)

You create an enumeration data type, add an attribute to the data type that you created in the previous task, and edit the model to take these changes into account.

### [\*\*Task 5: Testing and deploying\*\*](#)

You prepare and run a test suite, and then deploy a RuleApp from your decision model service to the execution environment.

[< Previous](#) | [Next >](#)

## Task 1: Creating the decision model service

In this task, you create the project, the first nodes in the diagram, and the initial decision logic for your decision model service.

### About this task

In a decision model service, you use a diagram to model data and decision nodes, and then author the decision logic of each decision node. Data nodes represent the data that is received from the client application, and decision nodes contain the logic in the form of business rules.


In this task, the data node is for the customer name, and the decision node returns a salutation with two possible replies:

- Says Hello to the customer and adds the customer's name.
- Replies with a default statement if no name is provided.

### Step 1: Creating the project

You first create the **Salutation2** project for your decision model service. The project contains all the information required to build and deploy a decision service. The project can also be exported as a .zip file.

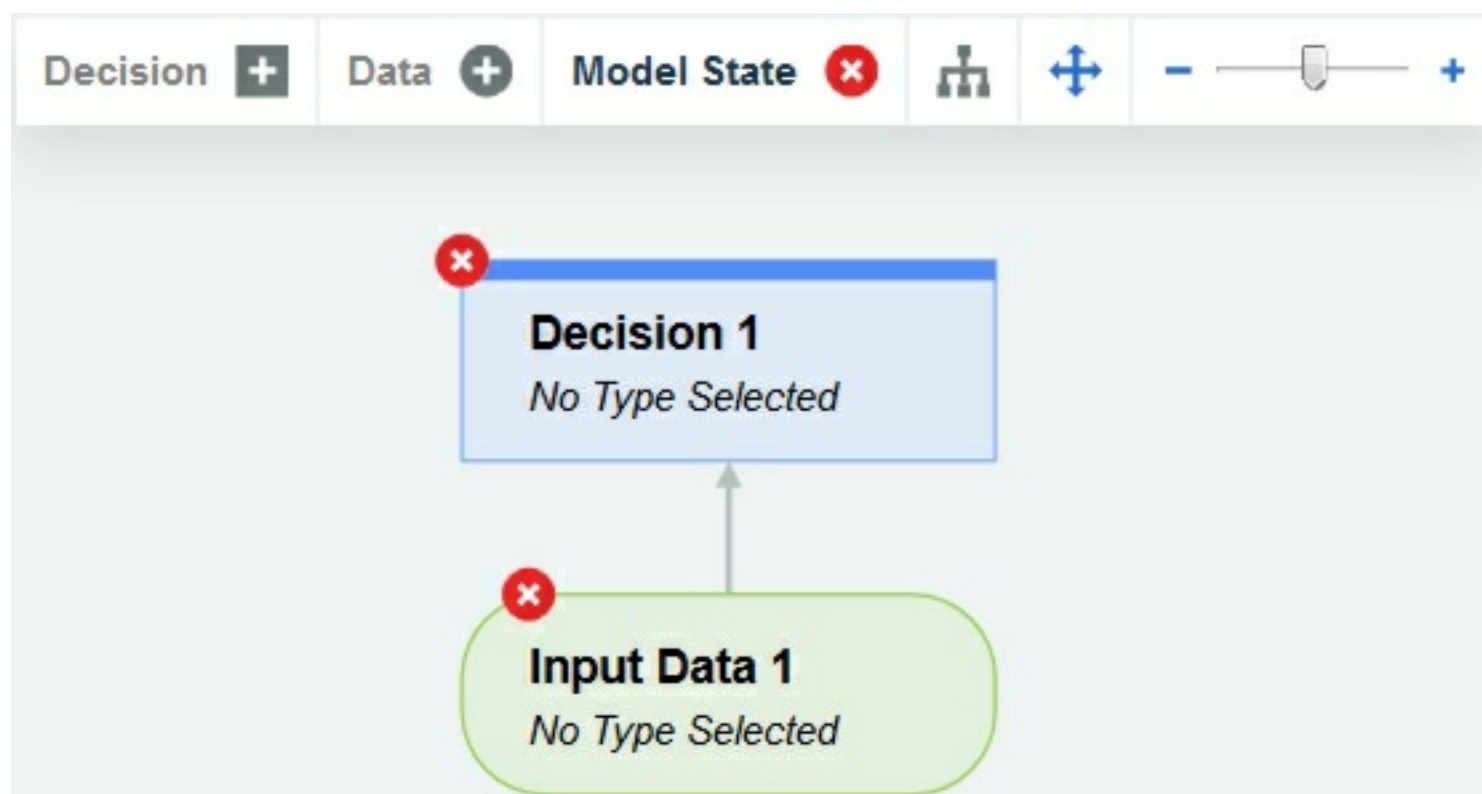
#### Procedure

1. Log in to your instance of Operational Decision Manager on Cloud, and open the Decision Center Business console.
2. In the **Library** tab, click the **Create New Decision Model Service** button .
3. Enter **Salutation2** as the project name for your decision model service. For the description, you can enter the following information:

Getting started with decision model tutorial. Decision model service composes greeting that includes the time, degree, and gender.

**Note:** This tutorial does not cover the decision governance framework, so you do not have to select it.

4. Click **Create**.
5. Make sure that the **Branches** tab is selected, and then click **main** to access the main branch. You can see the modeling diagram, which contains a decision node and a data node:




Decision Center assigns a new version number to the decision model every time you save it. The decision model works as one versionable element. Changes to individual rules or nodes can be consulted through the version of the decision model.

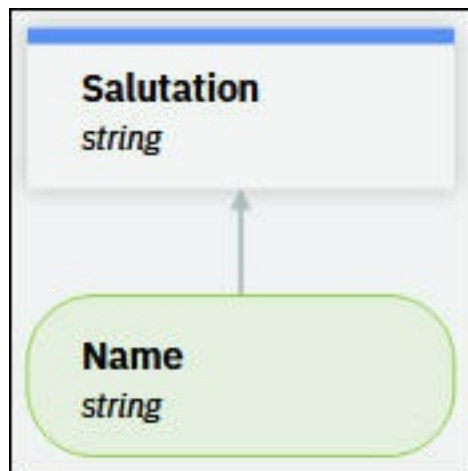
**Note:** The nodes are in an error state until you give them a type.

### Step 2: Creating the initial nodes

You create and define the relationship between the data that is provided by the client application, and the decision that is returned by your decision model service. Initially, you want your decision model to accept a customer name as input data, and provide a salutation to the customer.

#### Procedure

1. Click the **Edit** button  below the tab names, just above the diagram. Version 1.0 of the decision model diagram is now editable.
2. In the diagram, click the default data node **Input Data 1**.
3. In the **Model** tab, change the **Output variable name** of the node to **Name**. You can do this in **Output variable name** field or the name field just above the **Details** field.
4. Under **Output type**, click **Make a selection**, and then click **string** to select it as the type to handle the customer name as text. The type determines what the node can receive as input and send as output. Data modeling requires you to carefully assign the correct type to each node.
5. In the diagram, click once on the **Decision 1** decision node.
6. Name your new decision node **Salutation**, and choose **string** as the type. Your decision model now has a data node feeding a top-level decision node:



### Step 3: Creating the decision logic

You have the minimum node structure: one data node feeding one decision node. The decision node accepts input data from the nodes that point to it, and produces a decision that is based on the logic that you author.

#### About this task

Authoring the decision logic involves two aspects:

- Writing business rules that provide a decision that is usually based on conditions.
- Writing sufficient business rules to cover the possible cases that the client application encounter.

The nodes in the diagram are available in the business rules as variables of the form the 'node name'. For now, you have the **Name** data node available as 'the name', and the **Salutation** decision node as 'the salutation' in the rule and decision table editors.

#### Procedure

1. Select the **Salutation** node in the diagram to edit its content.
2. In the **Model** tab, under **Decision logic**, click **Add rule**. The wizard proposes a starting point for the rule that is based on conditions that you might want to apply to the data that it receives.
3. There are no conditions to consider for the name, so click **Create rule**.
4. In the middle panel, change the name from **rule-1** to **salutation rule**.
5. Click the `<a string>` placeholder, and select '**the name**' from the proposed options that appear. In this state, your decision model service simply returns the name provided as input. To add more value to the process, change the rule as follows:

```
set decision to "Hello " + 'the name' + "!" ;
```

**Note:** Add a space in "Hello " to insert a space before the name.

The keyword **decision** is available in each decision node to designate the output value of the node. Replacing with 'the salutation' is exactly equivalent.

6. Still under **Decision logic**, click **Set output default**, and replace the rule as follows:


```
set decision to "We don't have correct information to greet you!";
```

7. Click **Save and Continue** at the top of the window, and then **Create New Version** to save version 1.1 of your decision model service.

### Step 4: Validating the decision model

The decision modeling tool enables you to validate your model immediately. You run your decision model service on data that you provide.

#### Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  beside **Data sets**.
- 3. Enter Jones in the **Name** field for your data set, and click **Run**. The validation report shows the output from running your decision model, and information that is useful when troubleshooting:


▼ **Decision outputs**

| Salutation output | Salutation type |
|-------------------|-----------------|
| "Hello Jones!"    | string          |

▼ **Messages**

No output messages to display.

▼ **Run history**

| Node                                                                                                  | Type   | Rule interaction | Output         | Rules |
|-------------------------------------------------------------------------------------------------------|--------|------------------|----------------|-------|
| ▼  <b>Salutati</b> | string | sequence         | "Hello Jones!" | 2     |

- 4. Click the three dots next to **Name**, and click **Delete item** to clear the name from the data set.

▼ **Name**

Jones

Delete item

- 5. Click **Run** again. The report shows the default message.

"We don't have correct information to greet you!"

- 6. Click **Cancel**, and then **Yes** because the next tasks use different data.

**What to do next**

In the next task, you add to the salutation by making the final decision dependent on another decision.



## Task 2: Adding a decision node

You add a decision node that affects the final decision.

### About this task




You want your salutation to reflect the time of day. Instead of just saying Hello, for example, you want it to say Good morning. To do this, you create a new decision node and link it to the existing decision node.

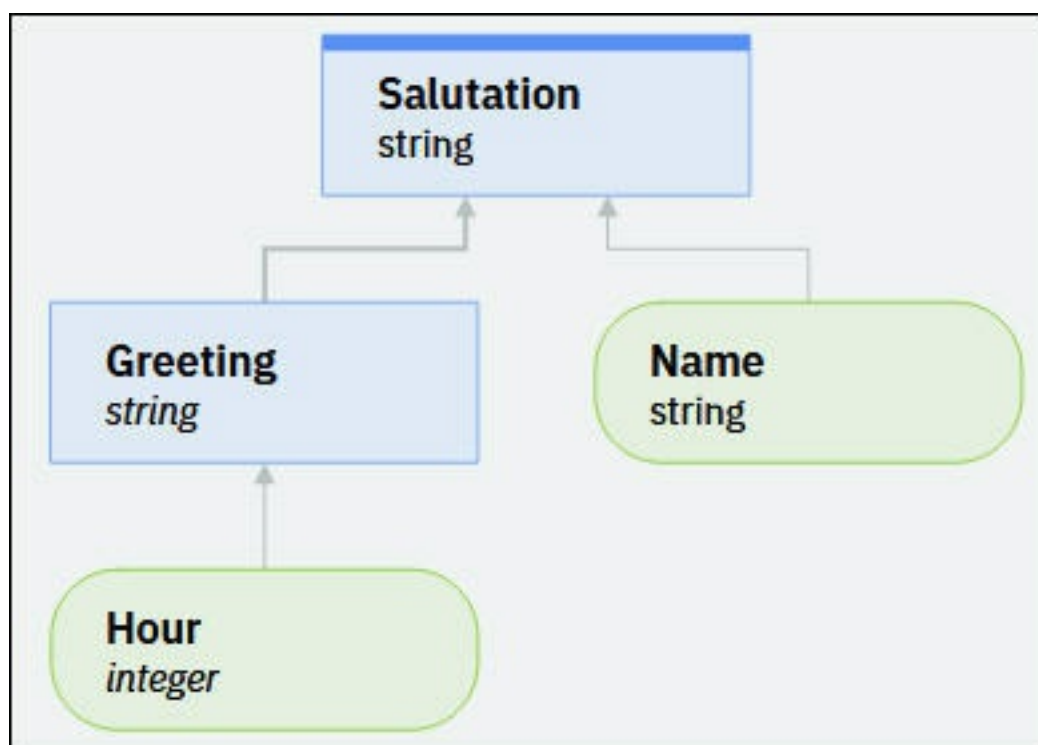
Each node in your diagram *feeds* the nodes to which it points. In other words, a node is *dependent* on the nodes that feed it. The new node feeds the existing decision node.

### Step 1: Adding nodes to the diagram

You create a decision node, called **Greeting**, that uses **Hour** as input. Then, you link this decision node so that it feeds data to the **Salutation** node.

#### Procedure

1. Click the **Edit** button  to continue editing the decision model.
2. In the floating toolbar, click **Data** to add a data node.
3. Name the new node **Hour**, and set its type to integer.
4. Hover over the **Hour** data node, click the **Add** button , and then click **Decision** to create another decision node.
5. Name the new node **Greeting**, and set its type to string.
6. Hover your mouse over the **Greeting** decision node, click and hold the **Add** button , and drag and drop the node onto the **Salutation** node. **Salutation** now depends on both **Greeting** and **Name**. The diagram looks as follows:



### Step 2: Adding the decision logic

You add the logic that decides which greeting is appropriate based on the provided time of day. Because there are many possibilities, you create a decision table, which is ideal for grouping business rules that have a common structure.

#### Procedure

1. In the diagram, click once on the **Greeting** node to select it.
2. In the **Model** tab, under **Decision logic**, click **Add table**.
3. In the displayed panel, select '**the hour**' as the data on which you place conditions, and click **Create table** in the lower right corner of the window. You now have an empty decision table that has one condition column and one decision column.
4. Rename your decision table **greeting table**, and then double-click the cells and enter the following values:

|   | Hour |     | Greeting       |
|---|------|-----|----------------|
|   | min  | max |                |
| 1 | 0    | 5   | Good night     |
| 2 | 5    | 12  | Good morning   |
| 3 | 12   | 18  | Good afternoon |
| 4 | 18   | 23  | Good evening   |
| 5 | 23   | 24  | Good night     |

- 5. Close the decision table panel.
- 6. Click the **Salutation** node in the diagram.
- 7. Under **Decision logic**, click **salutation rule** and change it as follows:

```
set decision to 'the greeting' + " " + 'the name' + "!" ;
```


Notice how the **Greeting** node is represented as 'the greeting', and the **Name** node as 'the name'.

- 8. Click **Save and Continue**, and then **Create New Version**.

Step 3: Validating the changes

After each modification, you can validate your decision model.

Procedure

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  beside **Data sets**.
- 3. Enter Jones in the **Name** field, and enter 20 in the **Hour** field.
- 4. Click **Run**. The validation report displays the Salutation output:

```
Good evening Jones!
```

- 5. Click **Cancel**, and then **Yes**.

What to do next

In the next task, you create a new type to make your decision model easier to read and maintain.



## Task 3: Creating a data type

You create a new type of data, and a decision node that uses the new type.

### About this task



Creating a custom type is important to creating a data model because it groups related data under one entity. By doing so, you produce a diagram that is easier to read and maintain. A second benefit is that you don't have to change the diagram when you add information to a custom type.

In this task, you modify your decision model so that it greets the customer with a courtesy title instead of just the customer's name. The courtesy title is either Mr . or Mrs . , depending on the gender of the customer.

### Step 1: Creating a new data type

Your decision model already uses the name of the customer. Now, you want to add the gender of the customer. You start by creating a **person** data type that groups these attributes together.

#### Procedure

1. Click the **Edit** button  to continue editing your decision model.
2. In the **Types** tab, click the **Add** button to create a new data type.
3. Click **Add type**, and name your new data type **person**.
4. Click the **Add** button  twice to add the following attributes to the **person** data type:

Name

person



Attributes

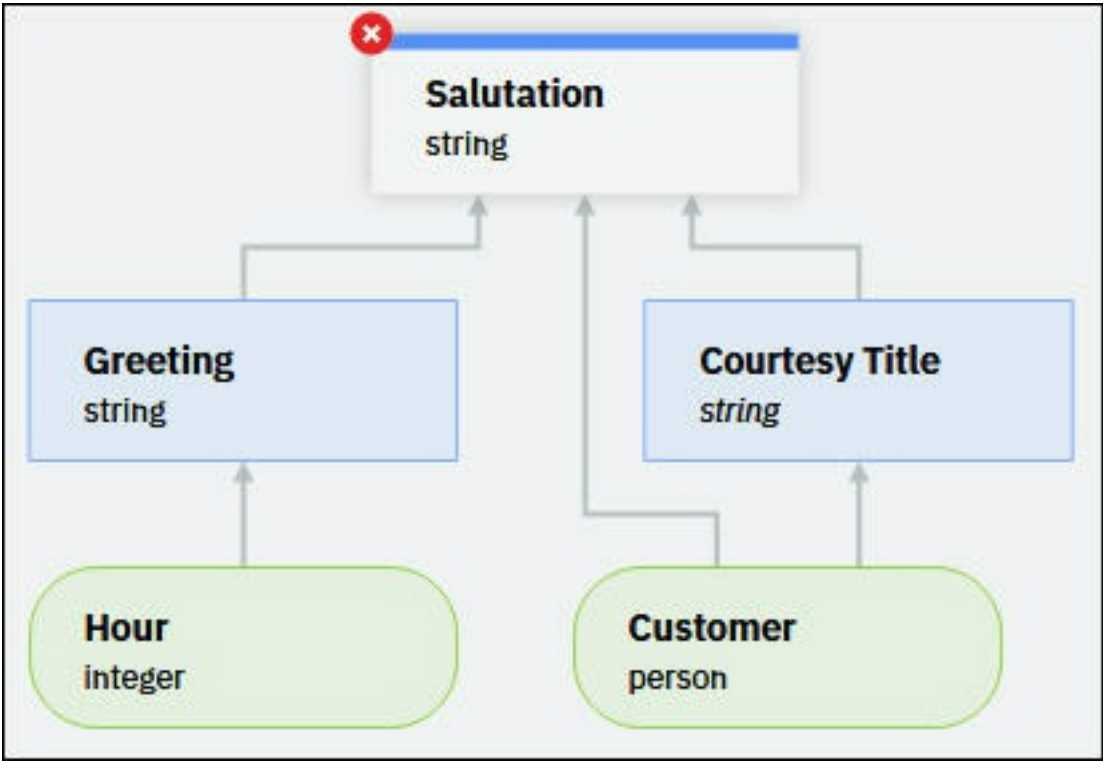
| Name   | Type   |
|--------|--------|
| name   | string |
| gender | string |

### Step 2: Creating a data node that uses the new type

You create a new decision node to greet the customer with a courtesy title instead of just the name. Your new decision node uses the new **person** data type.

#### Procedure

1. Click the **Model** tab.
2. Click the **Name** data node. Change its name to **Customer**, and its type to **person**. The **Salutation** node shows an error.
3. Hover over the **Customer** data node, click the **Add** button , and click **Decision** to create another decision node.
4. Name the new node **Courtesy Title**, and set its type to string.
5. Hover over the **Courtesy Title** decision node, click and hold the **Add** button , and drag and drop the new node onto the **Salutation** node. The node structure in the diagram is now in its final state, but it displays an error, which you fix in the next step. Along with the **Greeting** and **Courtesy Title** nodes, the **Salutation** node is now dependent on the **Customer** data node for the name of the customer. **Courtesy Title** is also dependent on the **Customer** data node for the gender of the customer:



### Step 3: Adding and adjusting the decision logic

You create the logic of your new decision node and adjust the top-level node. You also see how these nodes handle the custom data type.

**Procedure**

- 1. In the diagram, click once on the **Courtesy Title** node to select it.
- 2. In the **Model** tab, under **Decision logic**, click **Add table**.
- 3. In the displayed panel, select the `gender` of `'the customer'` as input to the node on which you place conditions, and then click **Create table**. You now have an empty decision table that contains a condition column and a decision column.
- 4. Change the name of your decision table to **courtesy table**, and enter the following values:

| gender | Courtesy Title |
|--------|----------------|
| Male   | Mr.            |
| Female | Mrs.           |

- 5. Close the decision table panel, and click the **Salutation** node in the diagram.
- 6. Click **salutation rule** and change it as follows:

```
set decision to 'the greeting' + " " + 'the courtesy title' + " " + the
name of 'the customer' + "!" ;
```


Notice how the name of the customer is usable in the rules as the `name of 'the customer'`.

- 7. Click **Save and Continue**, and then **Create New Version**.

### Step 4: Validating the changes

You add data and validate your changes.

**Procedure**

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  to create a new data set.
- 3. Enter the following data:
  - Name: Jones
  - Gender: Female
  - Hour: 20
- 4. Click **Run**. The validation report displays the Salutation output:

```
Good evening Mrs. Jones!
```

- 5. Click **Cancel**, and then **Yes**.

**What to do next**

In the next task, you add another type of data and explore some other features of a decision model service.



## Task 4: Modifying an existing decision

You create an enumeration data type, add an attribute to the data type that you created in the previous task, and edit the model to take these changes into account.

### About this task




An *enumeration* is a data type in which you establish all the possible values beforehand. You should use enumerations over text entries whenever possible. They facilitate data entry in the editors, the correctness of your rules is verified immediately, and gaps in decision tables are identified.

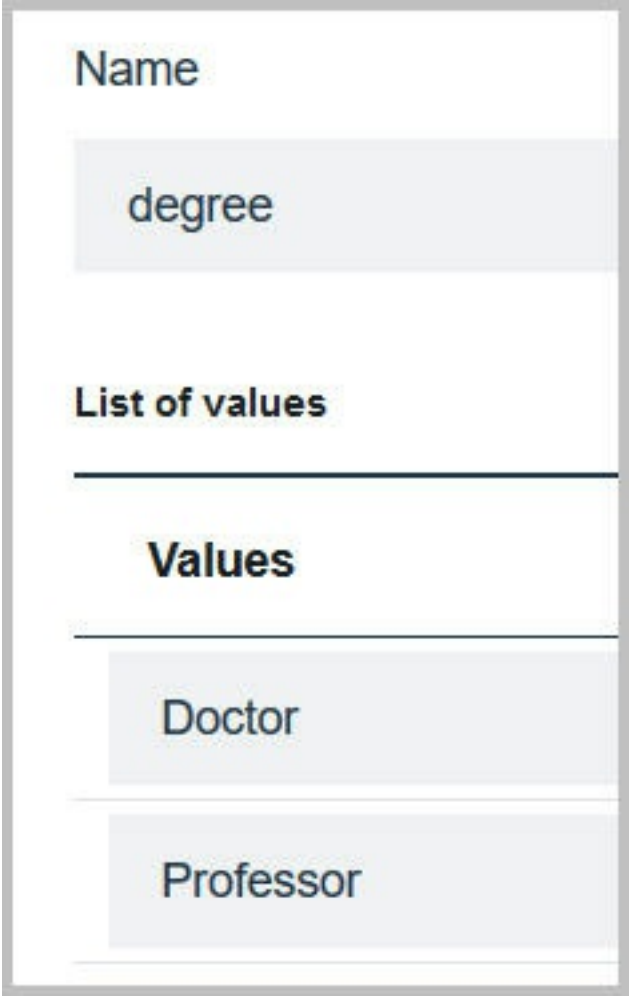
In this task, you create an enumeration that contains values that correspond to an academic degree, either Doctor or Professor. Then, you edit the decision table and rule to take this enumeration into account.

### Step 1: Adding an enumeration

You add an enumeration to the decision model.

#### Procedure

1. Click the **Edit** button  to continue editing of your decision model.
2. In the **Types** tab, click the **Add** button  to create a new data type.
3. Click **Add enumeration type**, and name your new data type **degree**.
4. Click the **Add** button  twice to add the following values to your enumeration:



The screenshot shows a dialog box for adding an enumeration type. At the top, the 'Name' field is filled with 'degree'. Below this, the 'List of values' section is expanded, revealing a 'Values' list. In this list, 'Doctor' and 'Professor' are entered as values, each in its own row.

5. Still in the **Types** tab, click the **person** data type. Add an attributed named **degree**, and give it the type **degree**.
6. Close the panel.

### Step 2: Adjusting the logic

You define the logic for the degree condition.

#### Procedure

1. In the diagram, click the **Courtesy Title** node, and open **courtesy table**.
2. Right-click the header of the **gender** column, and select **Insert column > Condition before**.
3. In the new column, change the heading to **Degree**.
4. Right-click the heading of the **Degree** column, and click **Define column**.
5. In the editor, add the following condition:

```
the degree of 'the customer' is <an object>
```

6. Click **Define**.

- 7. In the table, right-click the number 1 of the first row and click **Insert row > Above**. Repeat to create a second empty row.
- 8. Complete the table with the following values. Notice that you can select **Doctor** and **Professor** from the drop-down list:



|   | Degree ▼  | Gender ▼ | Courtesy Title |
|---|-----------|----------|----------------|
| 1 | Doctor    |          | Dr.            |
| 2 | Professor |          | Prof.          |
| 3 |           | Male     | Mr.            |
| 4 |           | Female   | Mrs.           |

- 9. Close the panel.
- 10. Click **Save and Continue**, and then **Create New Version**.

**Step 3: Validating the changes**

You validate your changes. An important aspect of decision modeling is to ensure that you cover situations where different rules apply. This is referred to as rule interaction. Validating as you go helps identify these situations.

**Procedure**

- 1. Click the **Validate** tab.
- 2. Click the **Add** button  to create a data set.
- 3. Enter the following data:
  - Name: Jones
  - Gender: Female
  - Degree: Professor
  - Hour: 20
- 4. Save your data set. Click the **Edit** button  next to **New input**, and replace it with **Prof Jones**.
- 5. Click **Run**. The validation report displays the **Salutation** output:

Good evening Mrs. Jones!

In this situation, customer Jones corresponds to both Mrs. and Prof. You decide to give priority to the degree.

- 6. Click the **Model** tab.
- 7. In the diagram, click the **Courtesy Title** node.
- 8. In the **Decision logic** section, click **Rules are applied in sequence** and change it to **First rule applies**. You might have to scroll down to reach the setting.
- 9. Return to the **Validate** tab and run the same data set. This time the validation report displays the following message:


Good evening Prof. Jones!

- 10. Click **Save and Close**, and then **Create New Version**.

**Task 4: Comparing versions**

Each time you save, Decision Center creates a new version of the decision model. You can compare versions to see the progress, and restore previous versions when needed.

**Procedure**

- 1. Click the **Open the decision model view** button  next to the edit button.
- 2. Click **Compare**. The comparison window opens with the current version already selected.
- 3. Select version 1.1, and then click **Compare**. All the changes made since the initial version of the decision model are shown.
- 4. Click **main** in the breadcrumbs to return to the **Decision Model** tab.

## What to do next

In the next step, you create a test suite, and then deploy your decision model service.

[< Previous](#) | [Next >](#)



## Task 5: Testing and deploying

You prepare and run a test suite, and then deploy a RuleApp from your decision model service to the execution environment.


### About this task

Now that your decision model service is complete, you create a test suite to do regression testing, and then deploy the service as a RuleApp to the execution environment.

### Step 1: Generating the scenario file

The first step to creating a test suite is to generate an empty *scenario file*. Scenarios represent real or fictitious use cases that you use to validate the behavior of your rules. Each scenario contains all the information that is needed to run your rules properly.

#### Procedure

1. Click the **Tests** tab.
2. Click the **Generate Scenario File** button .
3. Name your scenario file **salutation**.
4. In the tests to include, select **Salutation**.
5. Click **Download**, and save the file to a location on your computer.

### Step 2: Entering the scenarios and expected results

You create scenarios of representative cases for the client application. Each row in the first spreadsheet represents a scenario. Its columns indicate where to put the data for the scenarios. In the second spreadsheet, you enter the expected results of each scenario.

#### Procedure

1. Open the scenario file on your computer.
2. Enable editing if required, and enter the following scenarios in the **Scenarios** tab:

| Scenario ID | description | customer  |        |       | hour |
|-------------|-------------|-----------|--------|-------|------|
|             |             | degree    | gender | name  |      |
| Mr Jones    |             |           | Male   | Jones | 10   |
| Mrs Jones   |             |           | Female | Jones | 16   |
| Dr Jones    |             | Doctor    | Female | Jones | 21   |
| Prof Jones  |             | Professor | Male   | Jones | 3    |

3. At the bottom of the spreadsheet, click the **Expected Results** tab to open it.
4. Enter the following expected results for the scenarios:


| Scenario ID | Salutation equals          |
|-------------|----------------------------|
| Mr Jones    | Good morning Mr. Jones!    |
| Mrs Jones   | Good afternoon Mrs. Jones! |
| Dr Jones    | Good evening Dr. Jones!    |
| Prof Jones  | Good night Prof. Jones!    |

5. Save and close the file.

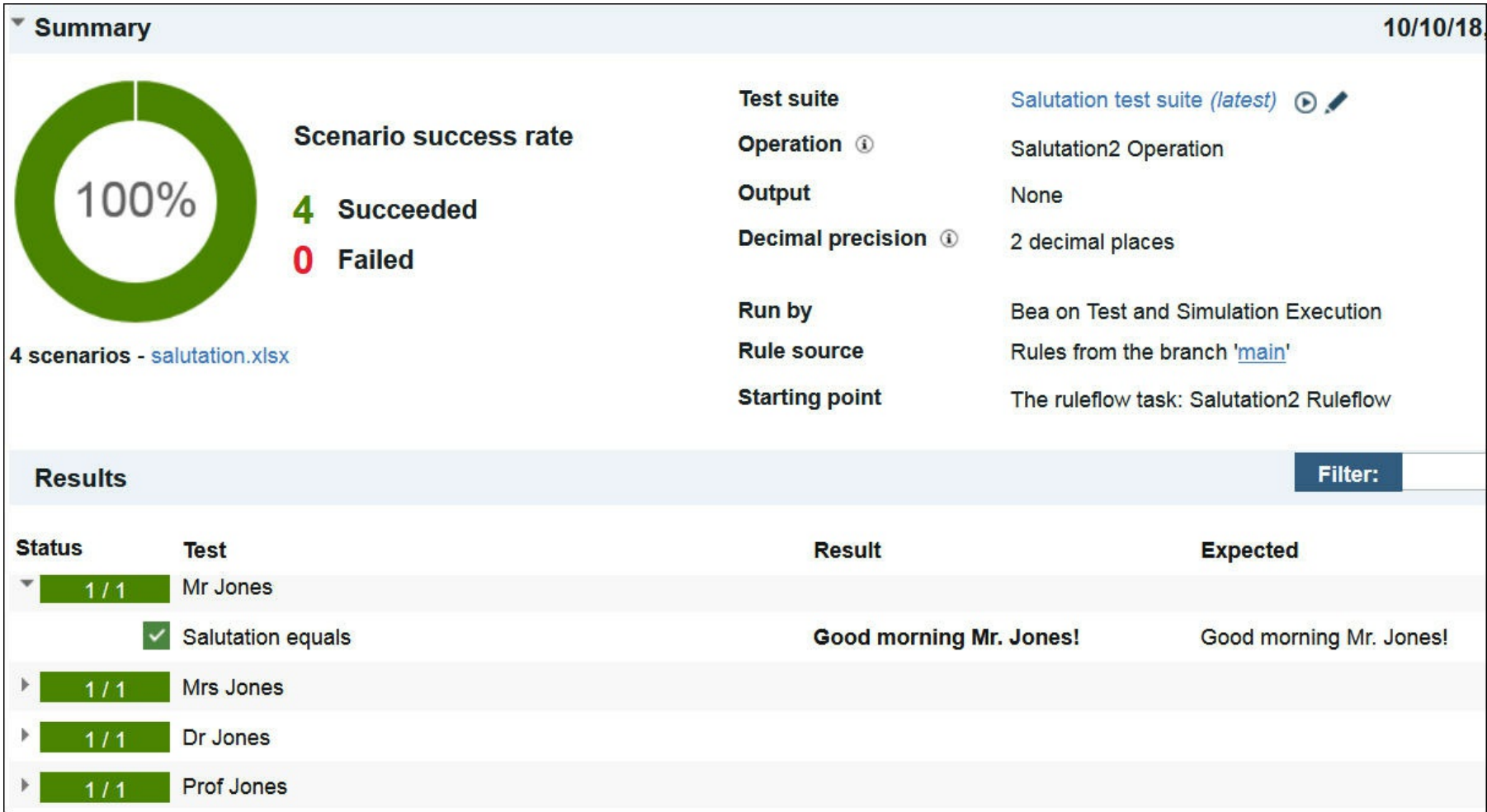
### Step 3: Creating and running the test suite

You run the test suite to compare your expected results with the actual results.

#### Procedure

1. In the Business console, click **main** in the breadcrumbs.
2. On the **Tests** tab, click the **Add** button  to create a new test suite.
3. Name your test suite **Salutation test suite**.

- 4. In the **File to use** section, click **Choose** and upload the **salutation** scenario file from your computer.
- 5. Click **Save and Run > Create New Version**, and then **OK** to the message on the progress report.
- 6. When the status of the test is complete, click the report name. You see that the scenarios ran successfully. The expected results that you entered are the same as the returned results.



- 7. Close the report.

(Optional) Step 4: Deploying the decision model service

About this task

A *deployment configuration* contains all the information needed to deploy a *RuleApp* from your decision model service, including the target server and the decision operation. The *decision operation* defines the rules for testing or deployment, and the input and output parameters. In a decision model service, the decision operation is created automatically and updated every time you save.

The RuleApp contains the *ruleset* developed in the decision model service. The execution environment runs the ruleset for the client application (see [Deployment configurations](#)).

**Restriction:** You must be a release manager to do this step. You cannot do this step if you are a business user.

Procedure

- 1. On the **Library** tab, click the **Salutation2** decision service, and then click **main** on the **Branches** tab.
- 2. Click the **Deployments** tab. The **Salutation2 Deployment** deployment configuration is automatically generated when you save the project.
- 3. Click **Salutation2 Deployment**. Look at the different subtabs to understand what is contained in the deployment configuration.
- 4. In the top banner, click **Deploy**.
- 5. Select **Server Development Environment** as the target server, and then click **Deploy**.
- 6. Click **OK** to close the status report message.
- 7. When the status shows a check mark, click the report link to show the results of the deployment. The report shows that the deployment was successful.
- 8. Click **Close**, and then log out.

Results

You have completed this tutorial, and learned how to create, test, and deploy a decision model service.

[< Previous](#)



[< Previous](#) | [Next >](#)

## Creating a ruleflow in the Business console

This tutorial shows you how to create a ruleflow in a decision service in the Decision Center Business console.

### Learning objectives

You do the following tasks:

- Explore the contents of a ruleflow.
- Create a ruleflow.
- Modify an existing ruleflow to call a subflow.
- Specify an order for running rules.
- Create a decision operation that uses the new ruleflow.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Best practices

This tutorial includes the following best practices for creating a ruleflow in the Business console:

- Create a subfolder in a project folder to group similar rules. [Example...](#)
- Create a ruleflow within a ruleflow to sequence certain rule together. [Example...](#)
- Select properties that optimize the running of the rules. [Example...](#)

### Time required

40 minutes

[< Previous](#) | [Next >](#)

## Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a ruleflow in implementing a new business policy. You work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans.

The decision service has two folders that contain business rules for different decisions:

| Folder      | Decision                                        |
|-------------|-------------------------------------------------|
| validation  | Determines whether the submitted data is valid. |
| eligibility | Determines whether the loan can be approved.    |

The decision service also contains a ruleflow that states the sequence for running the rules. If a loan request passes the rule in the validation folder, the ruleflow directs the loan data to the rules in the eligibility folder. Otherwise, the decision service rejects the loan.

The loan company wants you to add an eligibility policy that requires the borrower’s credit score to be at least 30 times bigger than the requested loan duration. To implement the policy, you create an action rule. Because of the growing number of eligibility rules, you decide to implement separate testing at this decision point. To do so, you create a separate ruleflow for eligibility, and a decision operation that uses the ruleflow.

### Audience

The tutorial introduces is for users who want to create ruleflows in the Decision Center Business console.

### Prerequisites

You work on a branch that you create in the decision service that is provided in the Miniloan Service sample project. The project is available in the Rule Designer section of the cloud portal.

To publish the decision service and create your branch, see [Preparing and removing the tutorial project](#).

**Important:**

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

### More information

If you are not familiar with the Decision Center Business console or the ruleflows, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Editing ruleflows](#)

### Lessons in this tutorial

**[Task 1: Exploring a ruleflow](#)**

You explore a ruleflow to determine how it works, and create an action rule for a business policy.

**[Task 2: Creating a ruleflow](#)**

You create a ruleflow, and modify another ruleflow.

**[Task 3: Setting rule task properties](#)**

You change the properties that define activities in a task.

**[Task 4: Applying a ruleflow in a decision operation](#)**

You create a decision operation that uses the new ruleflow, and test the ruleflow.

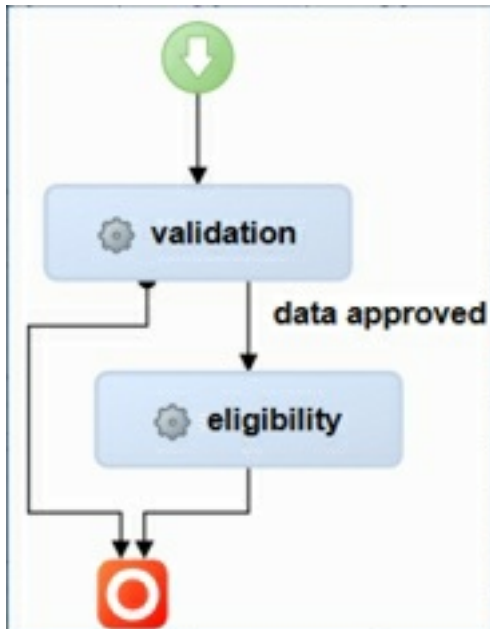
## Task 1: Exploring a ruleflow

You explore a ruleflow to determine how it works, and create an action rule for a business policy.

### About this task

A decision service keeps its decision points in projects and folders. The Miniloan Service decision service contains one project, and two folders for different decision points. The rule in the validation folder determines whether incoming loan data is valid, and the rules in the eligibility folder apply various conditions for eligibility.

The decision service also contains a ruleflow that states the sequence for applying the rules:



The ruleflow starts by running the validation rule. If the data from a loan request passes the validation rule, the ruleflow directs the data to the eligibility rules. However, if the validation decision point does not approve the data, the ruleflow skips the eligibility decision point, and the decision service stops processing the loan request.

You start by opening the branch that you created for this tutorial (see the software prerequisites in [Before you start](#)). Then, you create the action rule that applies the new policy.

#### Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Step 1: Opening your branch

You open the branch that you created for this project in the Miniloan Service decision service.

#### Procedure

1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Hover over the Miniloan Service box, and click anywhere except the name.
4. Click **My Tutorial** to open your branch. If you gave your branch a different name, click that name to open your branch.
5. Open the **Decision Artifacts** tab. It contains the following components:



**Tip:** To see all the components, click **All Types** at the top of the tab, and select **All Types**.

### Step 2: Taking a snapshot

Before you continue, you take a snapshot to capture the initial state of the main branch of the decision service. You cannot edit the contents of the snapshot, but you can compare it to other versions of the branch. You can also use the snapshot to restore the branch to its initial state.

## Procedure

1. Return to the list of artifacts, and click **Take Snapshot**.
2. Name your snapshot something like My snapshot. Do not reuse the name of an existing snapshot. For example, you can personalize your snapshot by using the initials of your name.
3. Click **Create**

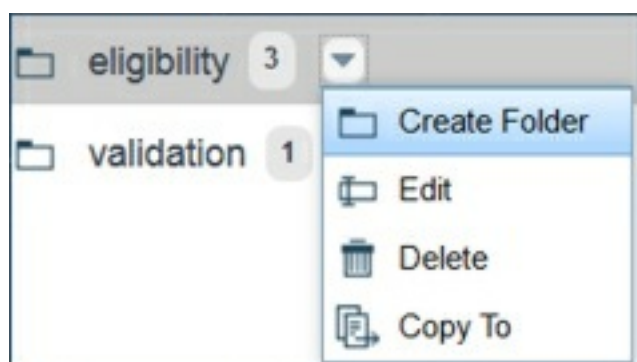
You can find the new snapshot in the **Snapshots** tab.


## Step 3: Adding an eligibility rule

The new policy is an eligibility constraint. To apply it, you create a folder and an action rule in the eligibility folder.

## Procedure

1. In the **Decision Artifacts** tab, hover over the eligibility folder, click the down arrow, and select **Create Folder** in the drop-down menu.



2. Enter duration as the name of the folder, and then click **Create**. The eligibility folder now contains the new folder.
3. Open the duration folder, and click the **Create** button  and select **New Rule**.
4. Enter max duration and score as the name of the rule, and then click **Create**.
5. Enter the following rule:

```
if
 the duration of 'the loan' / 12 is more than the credit score of 'the
 borrower' / 30
then
 reject 'the loan' ;
 add "Sorry, your credit score must be at least 30 times bigger than
 the loan duration" to the messages of 'the loan' ;
```

### Tip:

You can copy and paste the rule into the editor, or you can press Ctrl+Space to use the completion menu. (For information on the completion menu, see [Building rules using the Intellirule editor](#). The [Rule editor flash demo](#) on the IBM® Customer Support site also provides an English-only tour of the rule-editing features.)

6. Click **Save**, enter the following comment, and then click **Create New Version** to save the rule.

Created an action rule for the new eligibility policy.

You can find the new rule in the duration folder.

## What to do next

In the next task, you create a separate ruleflow for the eligibility rules.

[< Previous](#) | [Next >](#)

## Task 2: Creating a ruleflow

You create a ruleflow, and modify another ruleflow.

### About this task


A ruleflow links rules into a series of decisions. It states when, how, and why rules are run, and serves as a key component of a decision operation. When you deploy a decision service as a ruleset, the decision operation includes the ruleflow. The deployed ruleset contains all the rules in the decision service, but it only runs the rules that are referenced by the ruleflow.

The miniloan ruleflow in Miniloan Service uses all the rules in the decision service. You create a ruleflow that orders only the eligibility rules, and then you update the miniloan ruleflow to call the new ruleflow.

### Step 1: Creating a ruleflow

You create a ruleflow for the eligibility rules.

#### Procedure

1. Click **Miniloan Service** to display the decision artifacts.
2. Click the **Create** button , and select **New Ruleflow**.
3. Enter **eligibilityflow** as the name of the new ruleflow, and then click **Create**. The ruleflow editor opens. It displays instructions for creating tasks and transitions, and a ruleflow that contains a start task and an end task:





4. Click **Save**. Type the following comment, and then click **Create New Version**.

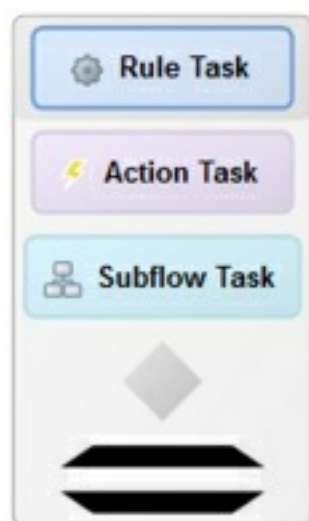
Created a ruleflow for the eligibility rules.

### Step 2: Adding and defining a rule task

You now add the tasks that call the rules, and the transitions between the tasks.

#### Procedure

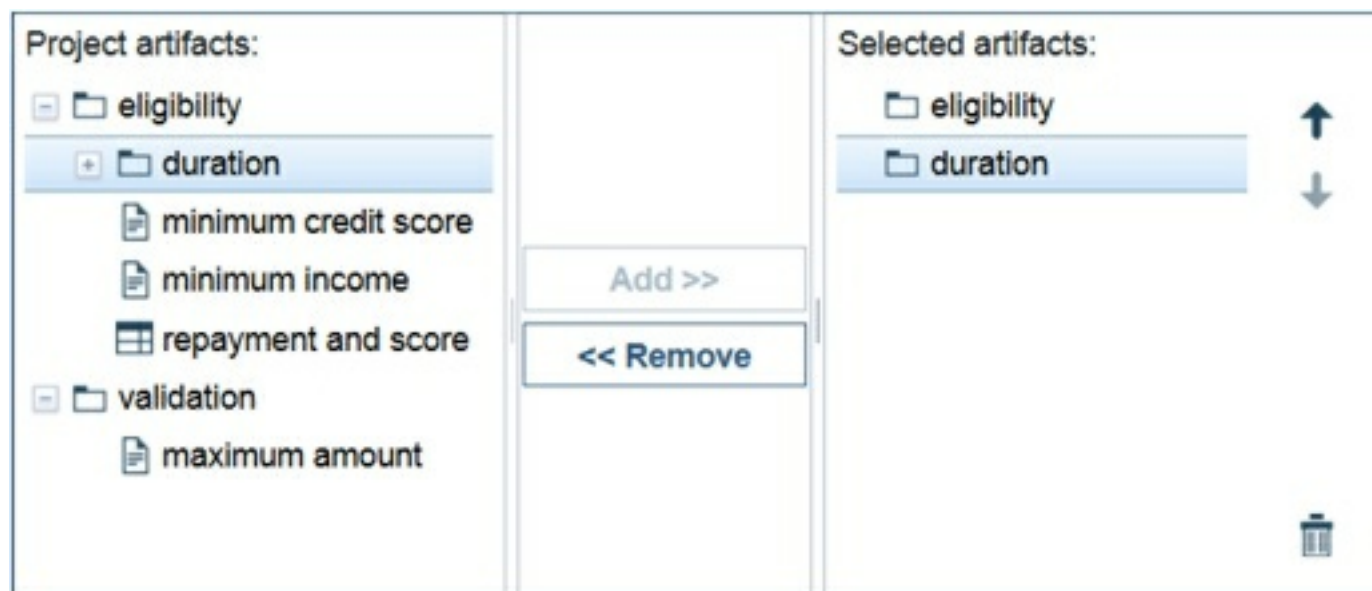
1. Hover over **eligibilityflow** in the list of decision artifacts, and click the **Edit** button  to open the file in the ruleflow editor.
2. Hover over the transition line between the start and end tasks.
3. Click the **Creation** button . The button opens a menu that contains different types of tasks:



Each task provides a different function:

- **Rule Task**: Contains one or more rules to run at a specific point in the ruleflow.
- **Action Task**: Contains one or more rule statements to run at a specific point in the ruleflow.
- **Subflow Task**: Calls another ruleflow.
- **Branch Node**: Organizes conditional transitions.
- **Fork Node**: Splits the run flow into parallel paths.
- **Join Node**: Combines parallel paths back into a run flow.

- Click **Rule Task** to add a task.
- Click the new task in the ruleflow. In the task editor, enter `eligibility task` as the label, and following description in the Documentation field:  
  
`Runs the eligibility rules.`
- Click **Edit** next to **Uses**.
- In the Rule Selection Editor, select the `eligibility` folder, and then click **Add** to add the folder to the rule task. The rule task runs all the rules in the folder. You can add more rules to the task by placing them in the folder.
- Add the `duration` folder to the list of selected artifacts.



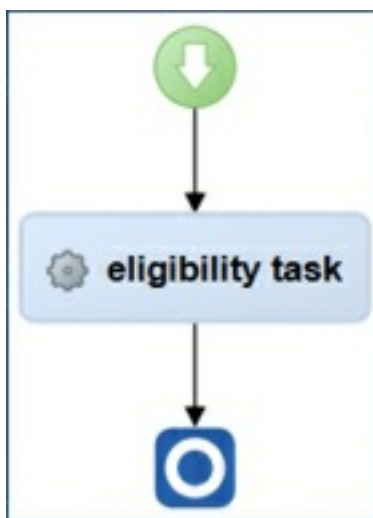
You must explicitly add the `duration` folder even though it is a subfolder in the `eligibility` folder. When you run the ruleflow, it applies the rules in the two folders.

- Click **OK**, and close the task editor.
- Click the End node, and then click the **Edit** button.
- Enter the following instructions, and then click **OK**:

```
print "Eligibility rules have been evaluated : result is :" + the
approval status of 'the loan' ;
```

The instructions give you some visibility into the running of the ruleflow. You can apply the same instructions in an action task after a rule task.

- Click the **Arrange** button  to optimize the layout. The ruleflow looks as follows:





- Click **Save**, and then click **Create New Version**.

### Step 3: Modifying the miniloan ruleflow

Now that you have a ruleflow that is dedicated to the eligibility rules, you must replace the eligibility task in the miniloan ruleflow with a call to the eligibilityflow ruleflow. You create a subflow task so that you do not have to update the miniloan ruleflow every time you change the eligibilityflow ruleflow.




#### Procedure

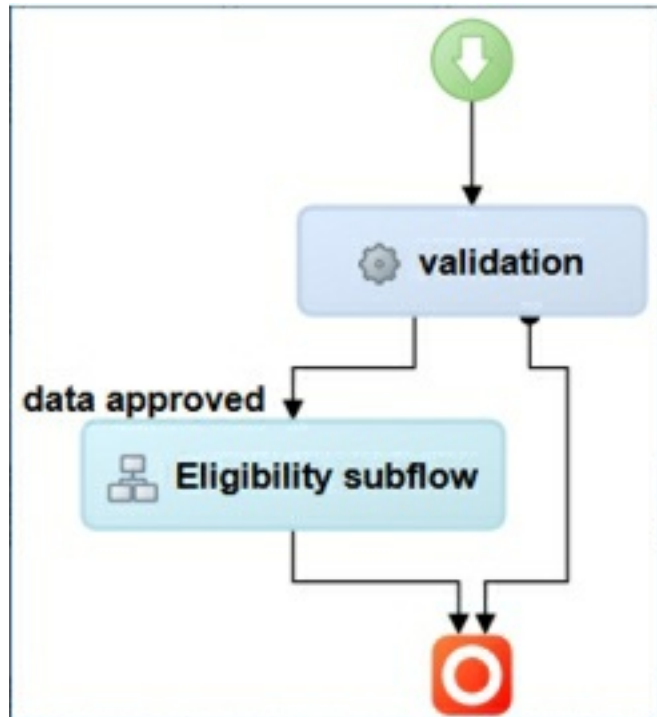
- Click the miniloan ruleflow, and click the **Edit** button  in the display area.
- Click the data approved transition link between the validation and eligibility tasks anywhere except the **Create** button . You see the following condition:

```
'the loan' is approved
```



When the condition is met, the ruleflow moves to the next task. Otherwise, ruleflow goes to the end task.

3. Close the condition editor.
4. Hover over the transition link, click the **Create** button , and click **Subflow Task**.
5. Hover over the subflow task, click the **Create** button  at the bottom, and drag it to the end task.
6. Click the eligibility task, and press Delete or Backspace to delete the task.
7. Click the **Arrange** button  to optimize the layout.
8. Click the subflow task and change the label to Eligibility subflow.
9. In the Subflow field, select eligibilityflow, and then close the task editor. The ruleflow looks as follows:



10. Click **Save**, and then click **Create New Version**.

### What to do next

In the next task, you look at setting the order for running rules in a rule task.

[< Previous](#) | [Next >](#)

## Task 3: Setting rule task properties

You change the properties that define activities in a task.

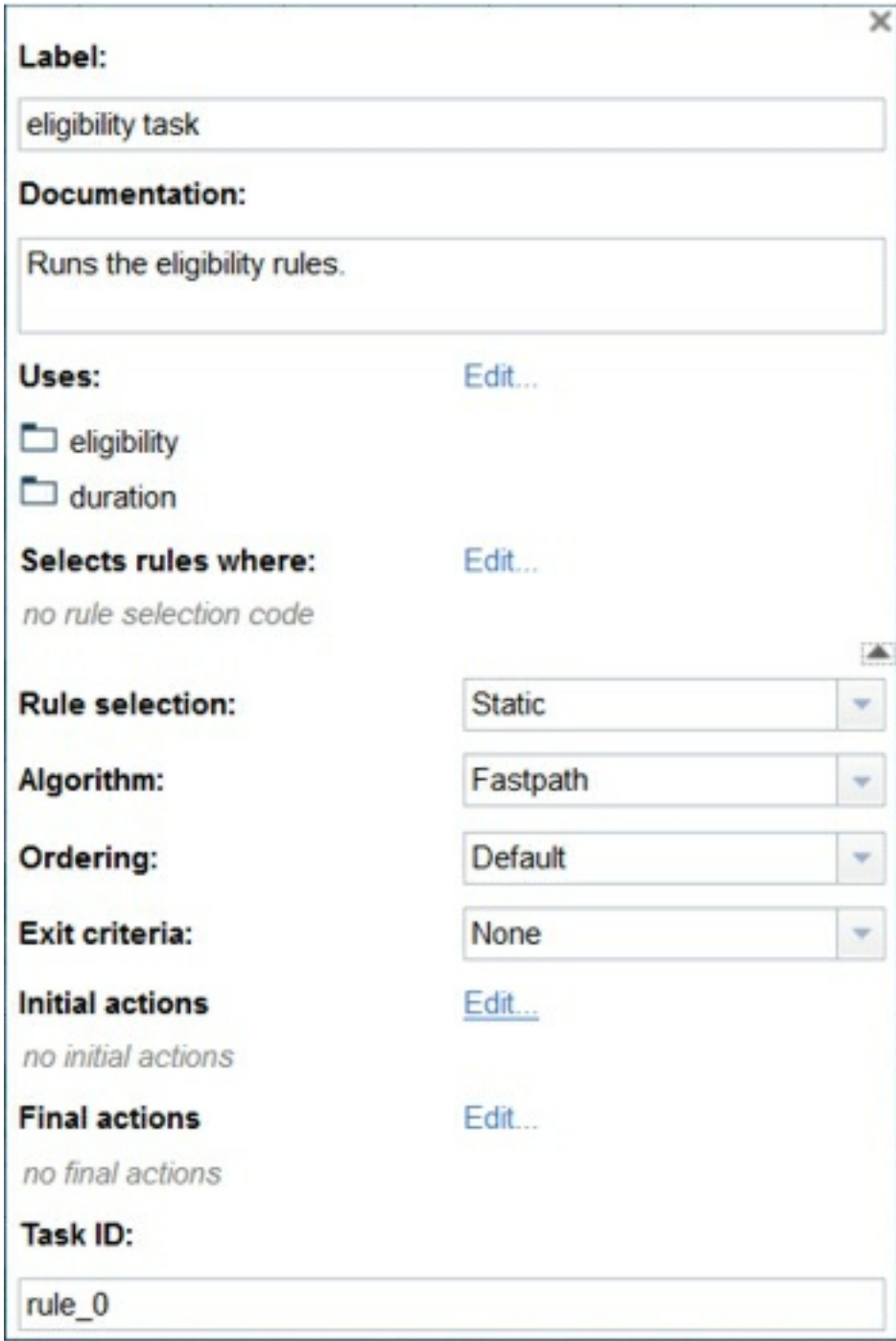
### About this task

The task treats the list of selected rules as pool of rules. You must set the ordering of the rules to literal to make the task run the rule artifacts in the order they are listed.

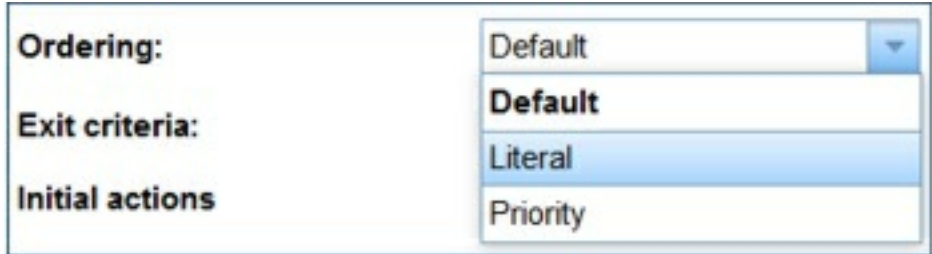
You set the ordering of the rule artifacts in the task properties. You can also set other properties for the task, including the engine algorithm and the number of rules the engine should run. For more information on these properties, see [Execution Properties for rule tasks](#).

### Procedure

1. Open eligibilityflow in the ruleflow editor.
2. Click the eligibility task to open the task editor.
3. Click the arrow at the bottom of the task editor to expand the window.

A screenshot of the 'eligibility task' editor window. The window has a title bar with a close button. It contains several sections: 'Label:' with a text field containing 'eligibility task'; 'Documentation:' with a text area containing 'Runs the eligibility rules.'; 'Uses:' with a tree view showing 'eligibility' and 'duration' folders, and an 'Edit...' link; 'Selects rules where:' with a text area containing 'no rule selection code' and an 'Edit...' link; 'Rule selection:' with a dropdown menu set to 'Static'; 'Algorithm:' with a dropdown menu set to 'Fastpath'; 'Ordering:' with a dropdown menu set to 'Default'; 'Exit criteria:' with a dropdown menu set to 'None'; 'Initial actions' with a text area containing 'no initial actions' and an 'Edit...' link; 'Final actions' with a text area containing 'no final actions' and an 'Edit...' link; and 'Task ID:' with a text field containing 'rule\_0'.

4. In the **Ordering** field, select **Literal** from the drop-down menu.

A screenshot of the 'Ordering' dropdown menu. The menu is open, showing four options: 'Default', 'Default', 'Literal', and 'Priority'. The 'Literal' option is highlighted with a blue background.

The ruleflow now runs the rules in the eligibility folder before it runs the rules in the duration folder.

5. Click **Save**, and then click **Create New Version**.

### What to do next

In the next task, you create a decision operation that uses the new ruleflow, and run a test to check the ruleflow.



## Task 4: Applying a ruleflow in a decision operation

You create a decision operation that uses the new ruleflow, and test the ruleflow.



### About this task

In this task, you create a decision operation that uses the new ruleflow. Then to check the results of ruleflow, you run a test that uses the decision service.

### Step 1: Creating a decision operation

You create a decision operation that uses the new ruleflow.

#### Procedure

1. In the **Decision Artifacts** tab, click the Operations folder.
2. Select Miniloan Service Operation, and click the **Copy** button .
3. Select the Operations folder as the location for the new decision operation, change the name to Miniloan Eligibility Operation, and then click **OK**.
4. Hover over Miniloan Eligibility Operation and click the **Edit** button. .
5. For **Main Ruleflow**, select the eligibilityflow ruleflow in the drop-down menu.
6. Click **Save**, and then click **Create New Version**.

### Step 2: Creating a test scenario file



You now run a test to check the results of the ruleflow.

#### Procedure

First, you must create a scenario file that contains loan data and expected results. When the test runs the data, it compares the actual results to the expected results, and creates a report. (see [Testing sets of rules in the Business console](#)).

The following steps show you how to make a scenario file for testing rules. A ready-made scenario file is provided in the sample project that is downloaded from the Operational Decision Manager on Cloud portal (see the software prerequisites in [Before you start](#)). If you downloaded the sample project, you can use the miniloan-test.xlsx scenario file at `<Install Dir>/Miniloan Service/scenarios/`.

**Attention:** To do this step, you need Open Office, or Excel 2007 or later.

1. Open the **Tests** tab.
2. Click the **Generate Scenario File** button .
3. Select the Miniloan Eligibility Operation decision operation, and click **OK**.
4. Enter eligibility as the file name.
5. In Tests, expand the loan.
6. Select **approved** and **messages**.
7. In the Operator menu for **messages**, select **contains**.
8. Click the **Download** button , and save the generated Excel file to a directory on your computer, for example, `<InstallDir>/Miniloan Service/scenarios`.
9. Open the scenario file in Excel or Open Office.
10. In the **Scenario** tab, enter the following values. Leave the description cells empty. You can copy and paste the values into your scenario file. After you enter the values for the first scenario, duplicate the row to create the second scenario.

**Tip:** Instead of completing the table, if you downloaded the sample project, you can use the included eligibility.xlsx scenario file to complete the tutorial.

| Scenario ID      | name | credit score | yearly income | amount | duration | yearly interest rate |
|------------------|------|--------------|---------------|--------|----------|----------------------|
| Bad credit score | Joe  | 250          | 80000         | 80000  | 240      | 0.05                 |
| Debt to income   | Joe  | 600          | 80000         | 25000  | 5        | 0.05                 |

Your **Scenario** table should look as follows:

| Scenario ID      | description | the borrower |              |               | the loan |          |                      |
|------------------|-------------|--------------|--------------|---------------|----------|----------|----------------------|
|                  |             | name         | credit score | yearly income | amount   | duration | yearly interest rate |
| Bad credit score |             | Joe          | 250          | 80000         | 80000    | 240      | 0.05                 |
| Debt to income   |             | Joe          | 600          | 80000         | 25000    | 5        | 0.05                 |

11. Open the **Expected Results** tab, and enter the following values.

| Scenario ID      | the loan is approved equals | the messages of the loan contains                                                |
|------------------|-----------------------------|----------------------------------------------------------------------------------|
| Bad credit score | FALSE                       | Sorry, your credit score must be at least 30 times bigger than the loan duration |
| Debt to income   | FALSE                       | Too big Debt-To-Income ratio                                                     |

Your **Expected Results** table should look as follows:




| Scenario ID      | the loan is approved equals | the messages of the loan contains                                                |
|------------------|-----------------------------|----------------------------------------------------------------------------------|
| Bad credit score | FALSE                       | Sorry, your credit score must be at least 30 times bigger than the loan duration |
| Debt to income   | FALSE                       | Too big Debt-To-Income ratio                                                     |

12. Save your changes, and close the file.

Step 3: Running a test suite

You now run a test to see the results that are produced by the new ruleflow. If the ruleflow works correctly, the test shows successful results for both test scenarios.

Procedure

1. Return to the **Tests** tab.
2. Click the **Create** button .
3. Select the Miniloan Eligibility Operation decision operation, and then click **OK**.
4. Enter Eligibility Test Suite as the name, and Eligibility Test Report as the report name.
5. Under **Scenarios**, in **File to use**, click **Choose** and navigate to <InstallDir>/Miniloan Service/. Select miniloan-simu.xlsx, and click **Open**.
6. Under **Report**, select the following run options:
  - The list of rules fired
  - The list of ruleflow tasks
7. Click **Save**, and then click **Create New Version**.
8. Hover over the **Eligibility Test Suite** row and click the **Run** button .
9. Click **OK** in the Run Test Suite window. The test run begins.
10. Wait until the status for **Eligibility Test Report** shows a check mark , and then click the name of the report to open the report. The report shows that 100% of the scenarios ran successfully.



11. Under Results, expand the Debt to income test, and its run details. The run details show that the test ran the rules and tasks in the eligibilityflow ruleflow.

|   |       |                                   |                                                                                                    |
|---|-------|-----------------------------------|----------------------------------------------------------------------------------------------------|
| ▼ | 2 / 2 | Debt to income                    | Joe                                                                                                |
| ▼ | i     | Execution Details                 |                                                                                                    |
|   |       | The list of rules fired           | Details...                                                                                         |
|   |       | The list of all tasks             | eligibilityFlow>rule_0<br>miniloan<br>miniloan>validation<br>miniloan>subflow_0<br>eligibilityFlow |
| ✓ |       | the loan is approved equals       | false false                                                                                        |
| ✓ |       | the messages of the loan contains | Details...                                                                                         |

12. Click the **Close** button  to close the report.

### Results

You have completed the tutorial. It showed you how to create a ruleflow in a decision service.

### Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the decision service, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

[< Previous](#) | [Next >](#)

## Creating a simulation in the Business console

This tutorial shows you how to create a simulation in a decision service in the Decision Center Business console.

### Learning objectives

You do the following tasks:

- Create metrics and KPIs.
- Create and upload a scenario data file.
- Create a template for a report format.
- Create and run a simulation configuration.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Best practice

This tutorial includes the following best practice for creating a simulation in the Business console:

- Use the completion editor to define metrics and KPIs. [Example...](#)

### Time required

40 minutes

[< Previous](#) | [Next >](#)

## Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you create a simulation to review the output of a decision service when you run the service on representative business data.

You work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans. When you run the decision service, it produces measurable information such as the amounts, durations, and credit scores of loans. You create a simulation that gathers this information as key performance indicators (KPIs), and displays the KPIs in a report.

### Audience

The tutorial introduces you to the simulation feature in the Decision Center Business console. It is for Business console users who want to run simulations in decision services.

### Prerequisites

You work on a branch that you create in the decision service that is provided in the Miniloan Service sample project. The project is available in the Rule Designer section of the cloud portal. You must have Rule Designer installed on your computer to publish the decision service to Decision Center.

To publish the decision service and create your branch for the tutorial, see [Preparing and removing the tutorial project](#).

#### Important:

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

### More information

If you are not familiar with the Decision Center Business console or the simulation feature, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Overview: Simulations in the Business console](#)

### Lessons in this tutorial

#### [Task 1: Creating metrics and KPIs](#)

You create metrics and use them to define KPIs.

#### [Task 2: Creating and uploading a scenario file](#)

You generate a scenario file, and add data to it offline. Then, you upload the file for use in simulations.

#### [Task 3: Formatting a report](#)

You create a template for the simulation report.

#### [Task 4: Running a simulation](#)

You configure and run a simulation in a decision service.

## Task 1: Creating metrics and KPIs

You create metrics and use them to define KPIs.

### About this task

A metric is a measurement of data. In the Miniloan Service decision services, the metrics include the amount, duration, and credit score of a loan. The decision service produces these metrics for each loan request. You gather these metrics for comparison in key performance indicators (KPIs). For example, a KPI can be the average amount of a group of loans, or it can be the amount of loans by credit score.

You start by opening the branch that you created in the decision service (see the software prerequisites in [Before you start](#)), and making a snapshot of the branch. Then, you create the metrics and KPIs that you use in the tutorial.

#### Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Step 1: Opening your branch

You open the branch that you created for this project, and create a snapshot. You can use the snapshot to compare versions of the branch, or to reinitialize the branch.

#### Procedure


1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Hover over the Miniloan Service box, and click anywhere except the name.
4. Click **My Tutorial** to open your branch. If you gave your branch a different name, click that name to open your branch.
5. Click **Take Snapshot**.
6. Name your snapshot something like My snapshot. Do not reuse the name of an existing snapshot. For example, you can personalize your snapshot by using the initials of your name.
7. Click **Create**

You can find the new snapshot in the **Snapshots** tab.

### Step 2: Creating metrics

The Business console uses the contents of the decision service to determine the types of metrics that you can create for the simulation.

#### Procedure

1. Open the **Simulations** tab, and then open the **Metrics** subtab.
2. Click the **Create** button  to make a metric.
3. Enter Amount of the loan as the name of the metric, and leave **Numeric** as the type. The type determines the expression that you can declare for the metric. You must set the type before you compose the expression.
4. In the metric expression field, enter the following text:

```
the amount of 'the loan'
```

#### Tip:

You can copy the expression into the editor, or you can press Ctrl+Space to use the completion menu. (For information on the completion menu, see [Building rules using the Intellirule editor](#).)

5. Click **Save**, and then **Create New Version**.
6. Repeat steps 2 - 5 to make the following metrics:

| Name         | Type    | Metric expression                  |
|--------------|---------|------------------------------------|
| Credit score | Numeric | the credit score of 'the borrower' |
|              |         |                                    |




|                      |                |                                     |
|----------------------|----------------|-------------------------------------|
| Approved loan        | <b>Boolean</b> | 'the loan' is approved              |
| Duration of the loan | <b>Numeric</b> | the duration of 'the loan'          |
| Yearly income        | <b>Numeric</b> | the yearly income of 'the borrower' |

Step 3: Creating KPIs

A KPI compiles similar metrics into a representative value. In a simulation report, you can present a KPI as a string or a comparative graph.

Procedure

- 1. Open the **KPIs** subtab.
- 2. Click the **Create** button  to make a KPI.
- 3. Enter Average credit score as the name of the KPI.
- 4. In the KPI expression field, enter the following text:

average 'Credit score'

**Tip:**  
You can copy the expression into the editor, or you can press Ctrl+Space to use the completion menu.

- 5. Click **Save**, and then **Create New Version**.
- 6. Repeat steps 2 - 5 to create the following KPIs:

| Name                     | KPI expression                                                             |
|--------------------------|----------------------------------------------------------------------------|
| Average income           | average 'Yearly income'                                                    |
| Average loan amount      | average 'Amount of the loan'                                               |
| Income and duration      | sum of 'Duration of the loan' grouped by 'Yearly income' with a 6 interval |
| Loan by credit score     | average 'Amount of the loan' grouped by 'Credit score' with a 10 interval  |
| Number of approved loans | number of 'Approved loan'                                                  |

What to do next

In the next task, you create a scenario data file, and import it into the Business console.

## Task 2: Creating and uploading a scenario file

You generate a scenario file, and add data to it offline. Then, you upload the file for use in simulations.

### About this task


You create a scenario file that holds loan data for the simulation that you run on Miniloan Service. You generate a file, download it to your computer, and add data in Excel or OpenOffice. When you complete the scenario file, you upload it back to the Business console. You can use the file in more than one simulation configuration, and in different instances of the Business console.

**Attention:** To work on the scenario file, you must have OpenOffice, or Excel 2007 or later.

### Step 1: Creating a scenario file

The following steps show you how to make the miniloan-simu scenario file. Alternatively, you can skip this step and use the miniloan-simu.xlsx scenario file in the sample project downloaded from your instance of Operational Decision Manager on Cloud.

#### Procedure

1. Open the **Data** subtab in the **Simulations** tab.
2. Click the **Generate Scenario File** button .
3. Provide the following information in the dialog box:
  - File name: miniloan-simu
  - Operation: Miniloan Service Operation
  - Scenario file format: Excel Workbook (.xlsx)
  - Locale: English (United States)
4. Save the file to a directory on your computer, for example, `<Install Dir>/Miniloan Service/`.
5. Open the miniloan-simu.xlsx file in Excel or OpenOffice. The expected values of the rules in the decision service determine the contents of the scenario file:

|             |             | the borrower |              |               | the loan |          |                      |
|-------------|-------------|--------------|--------------|---------------|----------|----------|----------------------|
| Scenario ID | description | name         | credit score | yearly income | amount   | duration | yearly interest rate |
| Scenario 1  |             |              |              |               |          |          |                      |

Each scenario row represents a loan application from a customer.

6. Enter the following values into the file. After you enter the values for the first scenario, duplicate the row to create the following scenarios.

**Tip:** Instead of completing the table, if you downloaded the sample project, you can use the included miniloan-simu.xlsx scenario file to complete the tutorial.

|             |             | the borrower |              |               | the loan |          |                      |
|-------------|-------------|--------------|--------------|---------------|----------|----------|----------------------|
| Scenario ID | description | name         | credit score | yearly income | amount   | duration | yearly interest rate |
| Scenario 1  |             | Joe          | 600          | 8000          | 140000   | 240      | 0.05                 |
| Scenario 2  |             | Joe          | 500          | 80000         | 240000   | 240      | 0.05                 |
| Scenario 3  |             | Joe          | 600          | 80000         | 130000   | 240      | 0.05                 |
| Scenario 4  |             | Joe          | 600          | 90000         | 300000   | 300      | 0.05                 |
| Scenario 5  |             | Joe          | 600          | 80000         | 560000   | 240      | 0.05                 |
| Scenario 6  |             | Joe          | 600          | 80000         | 240000   | 240      | 0.05                 |
| Scenario 7  |             | Joe          | 600          | 80000         | 130000   | 120      | 0.05                 |
| Scenario 8  |             | Joe          | 600          | 80000         | 700000   | 240      | 0.05                 |
| Scenario 9  |             | Joe          | 600          | 80000         | 140000   | 240      | 0.05                 |
| Scenario 10 |             | Joe          | 1000         | 80000         | 240000   | 120      | 0.05                 |
| Scenario 11 |             | Joe          | 600          | 80000         | 130000   | 240      | 0.05                 |
| Scenario 12 |             | Joe          | 600          | 90000         | 300000   | 300      | 0.05                 |
| Scenario 13 |             | Joe          | 600          | 80000         | 560000   | 240      | 0.05                 |
| Scenario 14 |             | Joe          | 600          | 80000         | 240000   | 240      | 0.05                 |
| Scenario 15 |             | Joe          | 600          | 80000         | 130000   | 60       | 0.05                 |
| Scenario 16 |             | Joe          | 400          | 100000        | 600000   | 340      | 0.05                 |


7. Save your changes and close the scenario file.

### Step 2: Uploading a scenario file

You upload the miniloan-simu.xlsx file to use it in your simulation.

#### Procedure



1. Return to the **Data** subtab, and click the **Create** button .
2. Enter Miniloandata as the name, and select Miniloan Service Operation as the operation.
3. Click **Choose**, navigate to the miniloan-simu.xlsx file, and then click **Open**.

**Tip:** If you have skipped step 1, you can find the premade scenario file at the following location on your computer: <Install Dir>/Miniloan Service/miniloan-simu.xlsx.

4. Click **Create** to add the data file to the **Data** tab.

### What to do next

In the next task, you format a report for your simulation.

[< Previous](#) | [Next >](#)

## Task 3: Formatting a report

You create a template for the simulation report.

### About this task

You add KPIs to a report template, and format their presentation. When you do so, you also tell the Business console which KPIs to process in the simulation.



You can present KPIs as strings or graphs. When a simulation produces a single value for a KPI, the value is presented as a line of text in the simulation report. However, if the KPI compares two metrics, you can present the information in a graph.

You format the presentation of the KPIs individually. You use a text editor to format strings, and a graph editor to format graphs. The graph editor gives you a choice of display types: bar, area, line, and pie. You can also add labels, and set the colors for different elements.

### Step 1: Creating a report template

You create a report template and add KPIs.

#### Procedure

1. Open the **Report Formats** subtab in the **Simulations** tab.
2. Click the **Create** button  to open the report editor.
3. Enter `Miniloan Simulation` as the name.
4. Click the **Create** button  twice to add two sections.
5. In the first section, click **New Section**, and type `Results` as the new name. This section holds the strings.
6. In the second section, click **New Section**, and type `Graphs` as the new name. This section holds the graphs.
7. In the column of Key Performance Indicators, drag the `Average credit score` KPI to the `Results` section.
8. Repeat step 7 for the following KPIs:
  - `Average income`
  - `Average loan amount`
  - `Number of approved loans`

**Note:** Your browser might affect the order in which you list the KPIs.



9. Drag the following KPIs to the `Graphs` section:
  - `Income and duration`
  - `Loan by credit score`

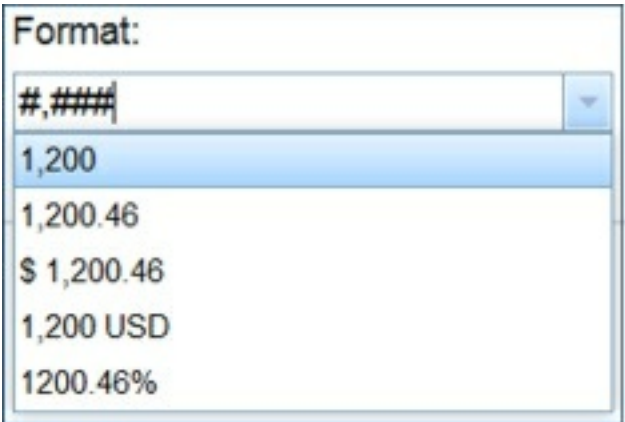
10. Click **Save**, and then click **Create New Version**.

### Step 2: Formatting the KPIs

You edit the appearance of the KPIs.

#### Procedure

1. Hover over `Miniloan Simulation`, and click the **Edit** button .
2. In the `Results` section, hover over the `Average credit score` KPI and click the **Configuration** button .
3. Open the **Format** menu, select **1,200**, and then click **OK**:



4. Format the other `Results` KPIs as follows:

- Average loan amount: \$1,200.46
- Average income: \$1,200.46
- Number of approved loans: 1,200

5. Format the KPIs in the Graphs section as follows:

**Income and duration KPI**

- Display Type: Bar
- Chart Title: Income to duration
- X-Axis Title: Income
- X-Label Format: 6
- Y-Axis Title: Duration
- Y-Label Format: 35
- Color: Blue (default)
- Outline Width: Default
- Style: Neon

**Loan by credit score KPI**

- Display Type: Pie
- Chart title: Average amount of loans by credit score
- Legend Format: 10
- Data Label: Value
- Outline Width: Default
- Style: Neon

6. Click **Save**, and then **Create New Version**.

**What to do next**

In the next task, you create and run a simulation configuration.

[< Previous](#) | [Next >](#)

[< Previous](#)

## Task 4: Running a simulation

You configure and run a simulation in a decision service.


### About this task

You create and run a simulation configuration. When you run the simulation, the configuration brings your report template and scenario file together with the rule artifacts in the decision service. The simulation produces a report that shows the results of the simulation as defined in your report template.

### Step 1: Creating a simulation configuration

You create a simulation configuration.

#### Procedure

1. Open the **Simulations** subtab in the **Simulation** tab.
2. Click the **Create** button .
3. Enter `Miniloan Service Simulation` as the name of the configuration, and enter `Miniloan Service Simulation Report` as the title of the simulation report.
4. Make sure that the following parameters are set as shown:
  - Server: `Testing and Simulation Runtime`
  - Report format: `Miniloan Simulation`
  - Input data: `Miniloandata`



The configuration editor automatically selects available parameters.

5. Check the sample report view to make sure that it shows your KPIs as you formatted them.
6. Click **Save**, and then click **Create New Version**.

### Step 2: Running a simulation

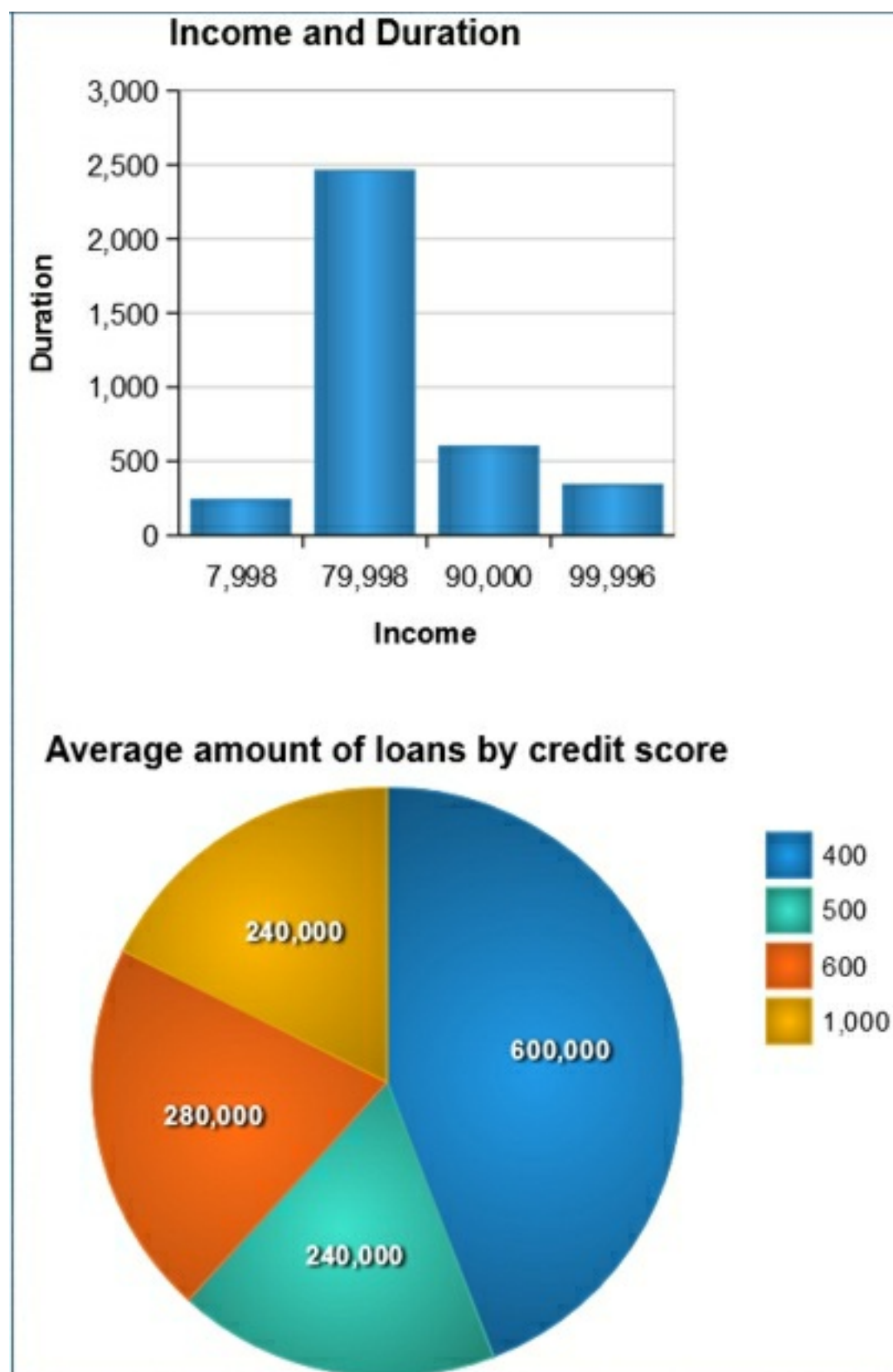
You run a simulation from your simulation configuration.

#### Procedure

1. Hover over the `Miniloan Service Simulation` configuration, and click the **Run** button .
2. Click **OK** to start the simulation. The **Reports** subtab opens to display the progress of your simulation.
3. When the status of the simulation changes to a check mark , click `Miniloan Service Simulation Report` to open the report. The report displays the results of the simulation:

| Results                            |
|------------------------------------|
| Average credit score: 606          |
| Average loan amount: \$ 298,750.00 |
| Average income: \$ 78,000.00       |
| Number of approved loans: 9        |

The report also contains graphs for comparative purposes:



The property window lists source, performance, and status information for the simulation, as well as links to the simulation configuration and the rules in the decision service:

|                                                        |                                                      |
|--------------------------------------------------------|------------------------------------------------------|
| Simulation                                             | <a href="#">Miniloan Service Simulation (latest)</a> |
| Operation                                              | Miniloan ServiceOperation                            |
| Input data                                             | Miniloandata                                         |
| Run date                                               | September 28, 2016 3:52 PM                           |
| Run duration                                           | 5 seconds                                            |
| Processed scenarios                                    | 16                                                   |
| Unsuccessful scenarios                                 | 0                                                    |
| Status                                                 |                                                      |
| <a href="#">View the rules used in this simulation</a> |                                                      |

4. Close the report.

## Results

You have completed the tutorial. You created the elements for a simulation, and ran the simulation. The simulation produced a report that showed your KPIs. In developing a decision service, you can make changes to the decision service, run a simulation repeated, and then compare the reports from the simulation runs. In doing so, you can determine how to improve the decision service.

## Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the Miniloan Service decision service in the Business console, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

[< Previous](#) | [Next >](#)

## Merging branches in the Business console

This tutorial shows you how to merge changes between branches in a decision service in the Decision Center Business console.

### Learning objectives

You do the following tasks:

- Compare branches before merging changes.
- Select changes and update options.
- Merge changes between branches.
- Verify your changes by using a snapshot.

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Best practices

This tutorial includes the following best practices for merging branches:

- Keep **Lock branches before merge** selected to prevent other users from making changes to the branches while you merge them. [Example...](#)
- Check the merge results by comparing snapshots. [Example...](#)

### Time required

30 minutes

[< Previous](#) | [Next >](#)

[< Previous](#) | [Next >](#)

## Before you start

Prepare for the tutorial by reading its description, and checking the audience and prerequisites.

In this tutorial, you work on Miniloan Service, a decision service that a fictitious loan company uses to determine whether borrowers are eligible for loans. The decision service processes data that includes loan durations, credit scores, and loan amounts.

You make your changes in the Decision Center Business console, which is a shared rule-editing environment. When you work on changes that might be used in a decision service, you can create a branch that isolates your work. The changes that you make in your branch do not affect the other branches in the decision service until you share them.

To add your changes to another branch, you can use the merge feature in the Business console. It lets you select the branches, changes, and update options, and it provides progress information on your updates.

In this tutorial, you create a branch in a decision service. You change three rules in the branch, and merge changes into another branch. Then, you check the second branch to ensure that it contains your changes.

## Audience

The tutorial is for users who want to merge branches in the Business console.

## Prerequisites

You work in the Miniloan Service decision service, the sample project that is available in the Rule Designer section of the cloud portal. You must have Rule Designer installed on your computer to publish the decision service to Decision Center.

To publish the decision service and create one of the branches for the tutorial, see [Preparing and removing the tutorial project](#).

### Important:

The sample project is in English. You must use the project in an English instance of Operational Decision Manager on Cloud.

## More information

If you are not familiar with the Decision Center Business console or the merge function, you can learn more in the following sections:

- [Introducing the Business console](#)
- [Merging branches](#)

## Lessons in this tutorial

### [Task 1: Creating a branch and modifying rules](#)

You duplicate a branch in the decision service, and then modify three rules in the new branch.

### [Task 2: Merging the branches](#)

You merge two branches to synchronize their rules, and then you check your changes by using a snapshot.

[< Previous](#) | [Next >](#)



## Task 1: Creating a branch and modifying rules

You duplicate a branch in the decision service, and then modify three rules in the new branch.

### About this task

You open the Business console and locate the branches in the Miniloan Service decision service (see [Before you start](#)). You make a branch, and then update three rules in the branch.


#### Attention:

The tutorial does not cover collaborative development in the decision governance framework. You work in an ungoverned branch of the decision service, and not in a release.

### Step 1: Creating a branch

You open the Miniloan Service decision service in the Business console, and create a branch.

#### Procedure

1. Log in to the Operational Decision Manager on Cloud portal and launch the Decision Center Business console.
2. Open the **Library** tab, which displays the available decision services.
3. Click **Miniloan Service** to open the decision service.
4. Open the **Branches** tab, and expand main. You see the My Tutorial branch that you created when you published the decision service.
5. Click the **Add** button .
6. Enter Increase Minimum as the name, and select **main** as the parent branch.

**Note:** To distinguish your branch from other branches in the decision service, add a personal identifier to the name of the branch, for example, the initials of your name or your job title.

7. Enter the following goal, and then click **Create**:


Test to increase the minimum score.

The Business console creates the Increase Minimum branch, which opens in the console. It contains the same contents as the main branch.

### Step 2: Modifying rules

You modify two action rules and a decision table in the Increase Minimum branch.

#### Procedure



1. Open the **Decision Artifacts** tab in the Increase Minimum branch.
2. Click **All Projects**, select **Rules and Decision Tables**, and click **Apply** to see the folders that contain business rules.
3. Click the eligibility folder to see its contents in the preview window.
4. Hover over the minimum credit score action rule in the preview window, and click the **Edit** button  to open the rule in the rule editor.
5. Change 200 to 300 in the condition and the action statements:

```
if
 the credit score of 'the borrower' is less than 300
then
 add "Credit score below 300" to the messages of 'the loan' ;
 reject 'the loan' ;
```

6. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the minimum credit score to 300.


The Decision Artifacts tab opens, and shows the contents of the eligibility folder.

7. Hover over the repayment and score decision table, and click the **Edit** button . Click **OK** in the dialog box to keep the default settings. The table opens in the decision table editor.
8. Change 200 to 300 in rows 1 and 2, and then click the **Optimize** button .

9. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the minimum credit score to 300.

The Decision Artifacts tab opens, and shows the contents of the eligibility folder.

10. Click the validation folder to see its contents in the preview window.
11. Hover over the maximum amount action rule in the preview window, and click the **Edit** button  to open the rule in the rule editor.
12. Change 1,000,000 to 100,000 in the condition and the action statements.
13. Click **Save**, enter the following comment, and then click **Create New Version**:

Changed the maximum amount to 100,000.

### What to do next

In the next task, you merge your changes into the My Tutorial branch.

[< Previous](#) | [Next >](#)

[< Previous](#)

## Task 2: Merging the branches

You merge two branches to synchronize their rules, and then you check your changes by using a snapshot.

### About this task

You use the merge feature to add changes from the Increase Minimum branch to the My Tutorial branch.

### Step 1: Selecting the branches

You select the branches to be merged.

#### Procedure

1. In the **Increase Minimum** branch, click the **Merge Branches** button , and then expand main in the dialog box to see the following list:



2. Click **My Tutorial** to select the branch. Leave **Lock branches before merge** selected to prevent other users from making changes to the branches while you merge them.
3. Click **Merge**. The merge page opens. It shows the selected branches in a table, and options for merging the branches:







### Step 2: Merging the rules

You compare the rules, select the options for updating the branches, and merge the branches.

#### Procedure

1. Click **Expand all** to see all the artifacts in the table:

| Name                                                                                                                 | Increase Minimum | My Tutorial | Action                                |
|----------------------------------------------------------------------------------------------------------------------|------------------|-------------|---------------------------------------|
| ▼  Miniloan Service               |                  |             |                                       |
| ▼  eligibility                    |                  |             |                                       |
|  minimum credit score    modified |                  | -           | <a href="#">update in My Tutorial</a> |
|  repayment and score    modified  |                  | -           | <a href="#">update in My Tutorial</a> |
| ▼  validation                     |                  |             |                                       |
|  maximum amount    modified       |                  | -           | <a href="#">update in My Tutorial</a> |

2. Hover over the row for the minimum credit score rule, and click the **Compare** button . The comparison page shows the rule in the two branches, and highlights the differences:

200 was changed to 300
200' was changed to 300'

|                                                                                                                                                                      |                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>version 1.0 (current) in My Tutorial</b><br>Created by paulreleasemanager@gmail.com on Feb 22, 2017                                                               | <b>version 4.0 (current) in Increase Minimum</b><br>Created by paulreleasemanager@gmail.com on Feb 22, 2017                                                       |
| <i>if</i><br>the credit score of 'the borrower' is less than 200<br><i>then</i><br>add "Credit score below 200" to the messages of 'the loan'<br>reject 'the loan' ; | <i>if</i><br>the credit score of 'the borrower' is less than 300<br><i>then</i><br>add "Credit score below 300" to the messages of 'the I'<br>reject 'the loan' ; |

- Close the comparison page, and do the same comparison operation for repayment and score and maximum amount to see the differences between the branches.
- After you compare the rules, return to the minimum credit score row, and click **update in My Tutorial**.


A menu opens with three update options:

update in My Tutorial

do not modify

update in My Tutorial

update in Increase Minimum

- Set the merge options for the rules as follows:
  - minimum credit score: **update in My Tutorial** (The changes in the Increase Minimum rule are made in the same rule in My Tutorial.)
  - repayment and score: **update in Increase Minimum** (The changes in the Increase Minimum table are replaced with variables in the same table in My Tutorial.)
  - maximum amount: **do not modify** (No changes are made between the branches.)
- Click the **Apply Merge** button  at the top of the table.
- Enter the following comment, and select **Create snapshots of the branches before merging**:

Merged changes between branches.

- Click **Apply Merge**. When the operation finishes, the merge table shows the following message:

Merge completed successfully

- Click **Expand All**. The table shows the following information:


| Name                 | Increase Minimum | My Tutorial |
|----------------------|------------------|-------------|
| Miniloan Service     |                  |             |
| eligibility          |                  |             |
| minimum credit score | -                | updated     |
| repayment and score  | updated          | -           |

The minimum credit score rule was updated in the My Tutorial branch, and the repayment and score decision table was updated in the Increase Minimum branch. The table does not show the maximum amount rule because no update was made to the rule in either branch.

### Step 3: Checking the merge results

You compare the My Tutorial branch to its snapshot to see what was changed by the merge operation.

#### Procedure

- Click **Miniloan Service** in the breadcrumbs to return to the **Branches** tab.
- Expand main, and click My Tutorial to open the branch.
- Open the **Snapshots** tab, and click Before merge from 'Increase Minimum' to open the snapshot.
- Click the **Compare** button . The following list opens:





Current State of the Project

✓



Before merge from 'Increase Minimum'


Created by paulreleasemanager@gmail.com on Feb 22, 2017 at 4:35 PM


State of this branch before the merge with the branch 'Increase Minimum'.

✓

5. Click **Compare**. A page opens with the results of the comparison. It contains a table that shows the modified rule:

1 artifact has been updated in *current state* since *Before merge from 'Increase Minimum'* snapshot

|                                                                                                                                   | Before merge from 'Increase Minimum'                            | current state                                                        |
|-----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|----------------------------------------------------------------------|
|                                                                                                                                   | Created by paulreleasemanager@gmail.com<br>Feb 22, 2017         | Created by paulreleasemanager@gmail.com<br>Feb 22, 2017              |
|  <b>minimum credit score</b><br>In eligibility | v1.0<br>Created by paulreleasemanager@gmail.com<br>Feb 22, 2017 | v4.0<br>Last updated by paulreleasemanager@gmail.com<br>Feb 22, 2017 |
|                                                                                                                                   |                                                                 |                                                                      |

6. Hover over the minimum credit score row, and click the **Compare** button . The comparison window opens, and shows the same results that are in step 2.2:

★ 200 was changed to 300

★ 200' was changed to 300'

| version 1.0                                                                                                                                                                                                | version 4.0 (current)                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Created by paulreleasemanager@gmail.com on Feb 22, 2017                                                                                                                                                    | Created by paulreleasemanager@gmail.com on Feb 22, 2017                                                                                                                                                    |
| <div><div>if</div><div>the credit score of 'the borrower' is less than 200</div><div>then</div><div>add "Credit score below 200" to the messages of 'the loan' ;</div><div>reject 'the loan' ;</div></div> | <div><div>if</div><div>the credit score of 'the borrower' is less than 300</div><div>then</div><div>add "Credit score below 300" to the messages of 'the loan' ;</div><div>reject 'the loan' ;</div></div> |

Results

You have completed the tutorial. You created a branch, updated the new branch, and merged your changes into another branch. Then, you compared the second branch to a snapshot to check the merged changes.

Deleting the decision service from Decision Center

If you plan to do another Business console tutorial, you can return the decision service to its original state by using the Initial State snapshot. However, if you no longer need the Miniloan Service decision service in the Business console, remove it from Decision Center. Removing your project prevents it from conflicting with other tutorial projects. To delete the decision service, see [Preparing and removing the tutorial project](#).

[< Previous](#)

## Preparing and removing the tutorial project

The Miniloan Service decision service is used in most of the tutorials. This topic provides information for setting up the project for a tutorial and removing it when it is no longer needed.

### About this task

The sections in this topic cover different activities for preparing and removing the Miniloan Service project. You do not have to follow all the instructions in this topic with every tutorial. Follow the instructions pertinent to your tutorial.

## Downloading the decision service

The files for the project are kept in a GitHub repository. You download a compressed file, and extract its files to your computer.

### Procedure

1. Sign in to the Operational Decision Manager on Cloud portal.
2. Click **Download** in the Decision Server Rule Designer section of the development environment.
3. Click **Miniloan Service project and Miniloan Server web application**. A download dialog opens.
4. Select **Save File**, and click **OK**.
5. Select a directory for the download file, and click **Save**. A compressed file is downloaded to the directory: `<InstallDir>\odm-cloud-getting-started-master.zip`. InstallDir is your directory for the GitHub files.
6. Click **Close** in the Rule Designer download dialog.
7. Open the compressed file, `<InstallDir>\odm-cloud-getting-started-master.zip`.
8. Extract the contents of the compressed file to the same directory. If you keep the original name, the new folder is `odm-cloud-getting-started-master`.

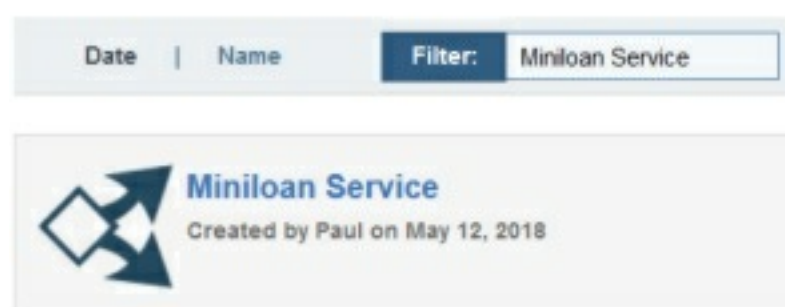
## Checking the availability of the decision service

You determine whether the Miniloan Service decision service is already in the cloud portal.

### Procedure

1. Sign in to the cloud portal.
2. Launch the Decision Center Business console in the development environment. The console opens to its home page.
3. Click **LIBRARY** to open the list of decision services that are currently in Decision Center.
4. Enter Miniloan Service in the filter to look for the decision service.

#### Decision Services




If the Miniloan Service decision service is in the library, skip step 3, and do step 4.

## Importing the decision service into Decision Center

You import the decision service into the Business console.

### Procedure

1. Open the folder that you created in the previous step, `odm-cloud-getting-started-master`.
2. Compress the Miniloan Service and miniloan-xom files into a .zip file named, for example, `miniloan.zip`.
3. Open the **LIBRARY** tab in the Business console, and click the **Import Decision Service** button .
4. Click **Choose**, and navigate to the compressed file that you created, `miniloan.zip`.
5. Select your compressed file, and click **Open**.
6. Click **Import**. The decision service is added to the library.


## Creating a branch

You create a branch to isolate your changes to the decision service.

### About this task

The main branch in the decision service contains the rule artifacts in their original form. To update the decision service, you create a branch based on the main branch, and work in the new branch.

### Procedure

1. Click **Miniloan Service** to open the decision service.
2. Open the **Branches** tab, and expand the main branch. Look at the names of the existing branches. When you name your branch, do not reuse the name of an existing branch.
3. Click the **New Branch** button .
4. Enter a name for your branch, for example, **My Tutorial**. Remember not to reuse the name of an existing branch. For instance, you can personalize your branch by using the initials of your name.
5. Select **main** as the parent branch, and then click **Create**. The Business console duplicates the artifacts of the main branch in a new branch that you can modify.

### Results

You can now work on the decision service in the tutorials.

## Deleting your branch from Decision Center

### About this task

When you no longer need your branch in the Miniloan Service decision service, you can delete it from Decision Center.

### Procedure

1. Open **LIBRARY** in the Business console
2. Open the Miniloan Service decision service, and open the Branches tab.
3. Expand the main branch to see your branch.
4. Hover over the name of your branch, click the down arrow to open the command menu, and click **Delete**:



5. Click **Yes** in the Delete Branch dialog.

### Results

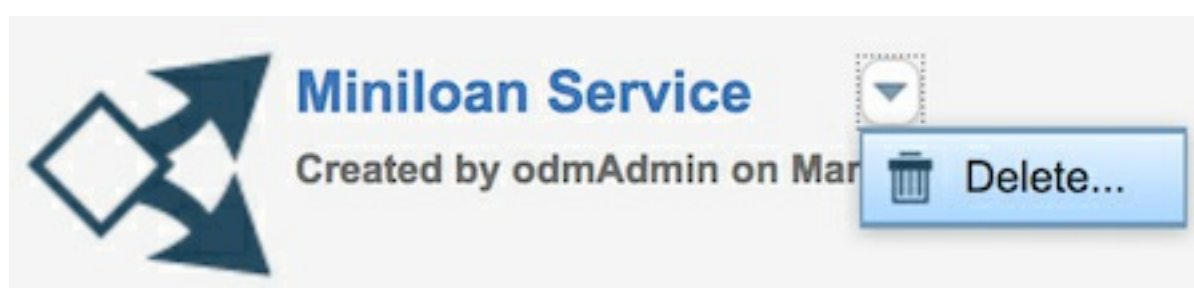
Your branch is deleted from Decision Center, and is no longer displayed in the Business console.

## Deleting the decision service from Decision Center

You remove the Miniloan Service decision service from the Decision Center database.

### Procedure

1. Open the **Library** tab in the Business console.
2. Hover over the Miniloan Service box, open the drop-down menu and click **Delete**:



A warning message opens. It shows the decision service that you selected.

3. Click **Delete**. The decision service is removed from Decision Center.

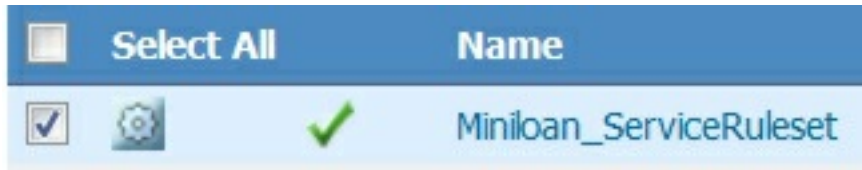
## Removing a ruleset from Rule Execution Server

### About this task

You can remove a ruleset that you deployed to Rule Execution Server.

**Procedure**

- 1. Open the Operational Decision Manager on Cloud portal.
- 2. Launch the Rule Execution Server console.
- 3. Click **Explorer** to open the tab.
- 4. Open the RuleApp for your decision service, for example, RuleApps/Miniloan/1.0.
- 5. Select the ruleset for your decision service in the RuleApp view, for example, Miniloan\_ServiceRuleset:



- 6. Click **Remove** to delete the ruleset. A warning message opens. It shows the ruleset that you selected.
- 7. Click **Confirm** in the warning message. The console deletes your ruleset. The RuleApp view opens, and it no longer shows your ruleset.

**Results**

You removed your ruleset from the Rule Execution Server console.



## Creating decision services

Your rule applications start in Rule Designer, where you create decision services for collaborative development.

### Developing rulesets in Rule Designer

A ruleset is a set of business rules that serve as an executable decision unit. You develop the contents of rulesets in decision services and rule projects, and call the rulesets from client applications.

### Designing projects for rule authoring

You work in projects to define a data model for rules and a vocabulary for business users. You also author different types of business rules.

### Building and running rules

You can execute business rules on different platforms, optimize their execution, and automate certain tasks.

### Publishing decision services to Decision Center

To enable developers and business users to collaborate on projects, you publish decision services from Rule Designer to Decision Center.

## Developing rulesets in Rule Designer

The set of business rules that are put together as one executable decision unit is called a ruleset. You develop decision services and rule projects to define the contents of the ruleset that is called by the client application.

### Setting up rule projects

The first step in developing a ruleset is to create a decision service with its rule projects, either from scratch or by using a template. You can then set up the rule project structure and add rule packages, to make your application more modular.

### Orchestrating ruleset execution

You combine rules and rule artifacts into a ruleset for subsequent execution. You can pass data and share code in a ruleset, and manage the flow of rule execution.

### Defining the ruleset content and signature

You define the content of a ruleset and its signature.

## Setting up rule projects

The first step in developing a ruleset is to create a decision service with its rule projects, either from scratch or by using a template. You can then set up the rule project structure and add rule packages, to make your application more modular.

### [Setting up a project hierarchy](#)

You can organize your content as a hierarchy of rule projects.

### [Defining rule project references](#)

You use project references to make your business rule application more modular.

### [Defining a folder structure for rule project items](#)

When you create a rule project, a folder is automatically created for each type of rule project item.

### [Rule project item naming conventions](#)

When you create a new rule project item, follow the naming conventions.

### [Improving performance on large rule projects](#)

In large rule projects, you can improve the performance of Rule Designer.

**Parent topic:** [Developing rulesets in Rule Designer](#)

## Setting up a project hierarchy

You can organize your content as a hierarchy of rule projects.

You create a hierarchy by establishing dependencies between projects. Use rule project hierarchies to separate and share the rules and the BOM among different projects, especially as your application becomes more complex. Using different rule projects allows for easier maintainability, lets you test different parts of the logic independently, and organize for various decision points. You can then set up the projects to refer to one another.

The project organization that you set up in Rule Designer also impacts how you manage these projects in Decision Center.

### Table of contents

- [Organizing the rule project structure](#)
- [Organizing the project structure for multiple rulesets](#)

### Organizing the rule project structure

Decision services are designed to facilitate the creation of the project structure. When you create a decision service, you create a main rule project that serves as the top-level project of the rule project hierarchy. If the project contains many rules, you create other types of projects to group these rules by domain and to store the BOM. Then, you set dependencies between the projects. All projects that are directly or indirectly referenced by the main project become part of the decision service. For example, a main decision project might reference two standard rule projects, which in turn both reference a project that contains the BOM. All four projects form the decision service.

You can create the different types of projects in the **Decision Service Rule Project** section of the New Rule Project wizard as follows:

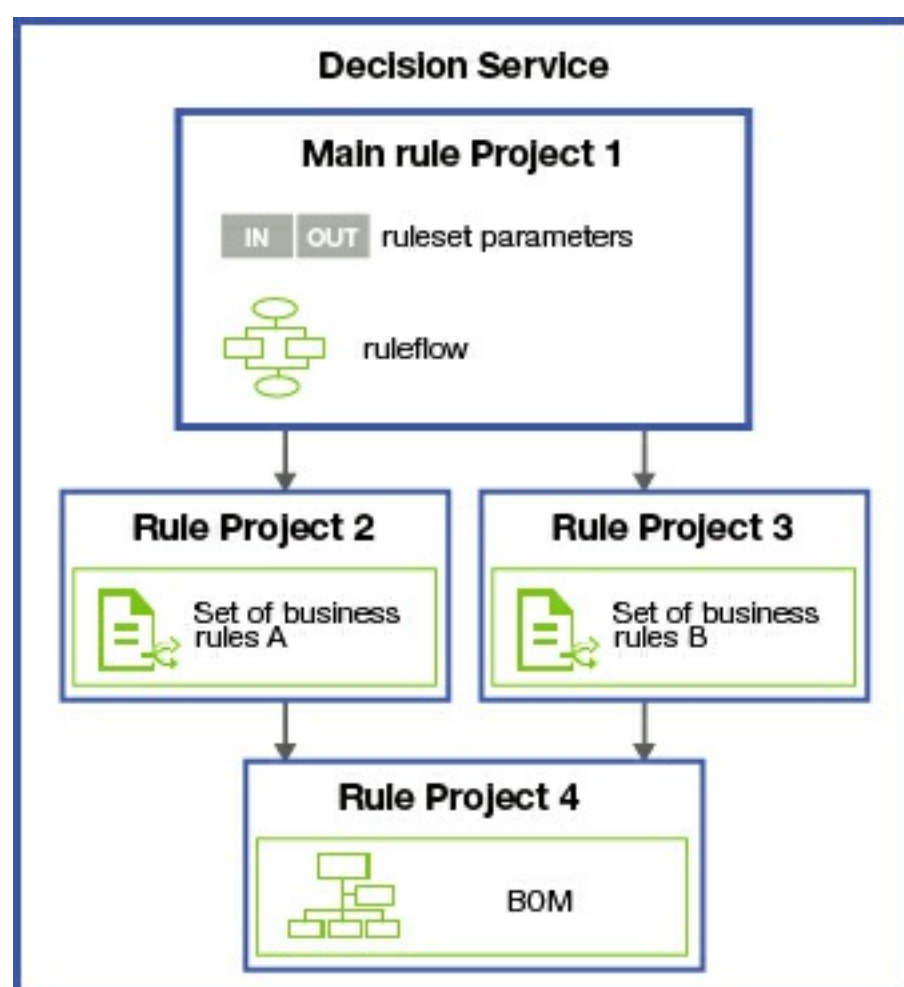
- Select **Main Rule Project** to create a project that is the entry point of the decision service and that might reference other projects.
- Select **Standard Rule Project** to create one or more projects to store your rules.
- Select **Rule Project with a BOM** to create a project to store the BOM.

#### Note:

It is possible to change the type of any project to transform it into a main project or vice versa, in **Properties > Decision Service**. You can change a rule project to a main rule project when the rule project is not referenced by another project. Changing a project can require changes to the way projects reference each other, and can affect the deployment and synchronization of the project.

In Rule Designer, many operations take place at the decision service level, such as build, queries, refactoring, and ruleset extraction. As a consequence, grouping rule projects in separate decision services limits the scope of these operations.

The following diagram shows a possible rule project organization in a decision service. The main project contains a ruleflow that uses rules from projects in the hierarchy, and the projects share a BOM that is kept in one of the projects. All four projects are part of the same decision service because rule projects 2, 3, and 4 are directly or indirectly referenced by the main rule project.

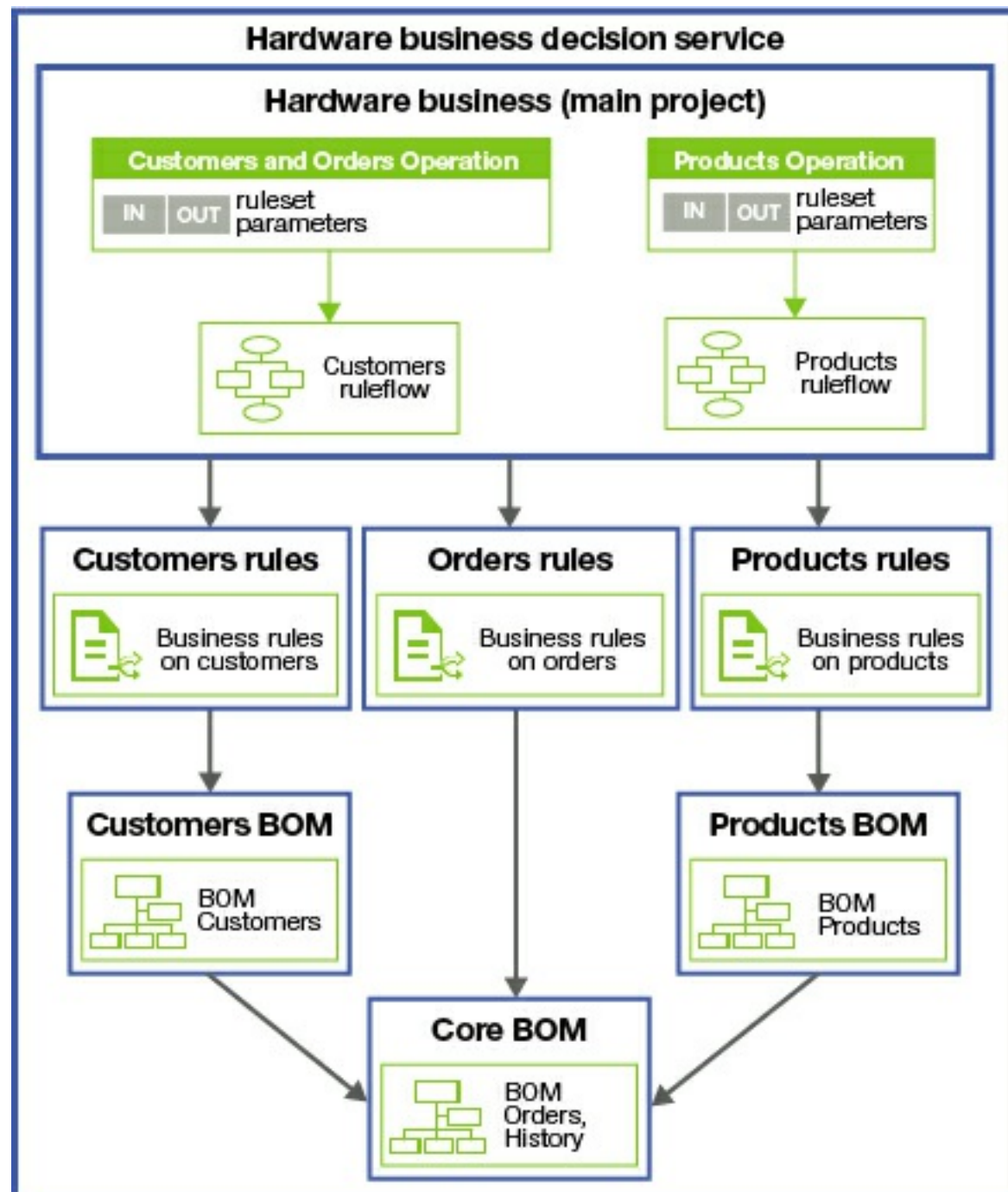


## Organizing the project structure for multiple rulesets

If the client application calls different decision points, you should organize your project structure for multiple rulesets.

If an operation and its rules require only part of a BOM, you can keep those rules and that part of the BOM in a separate rule project hierarchy that specifies the vocabulary that is available to those rules. The size of the BOM can result in usability and performance issues. To reduce the number of completion entries in the rule editors, and to make them more relevant to your rules, use categories to organize your vocabulary into subsets.

You can package all decision operations, ruleflows, and rules in the same decision service. The following diagram shows a decision service that is designed to distribute the decision points in separate rule projects. It also shows a strategy for splitting the BOM for efficiency.



**Parent topic:** [Setting up rule projects](#)

### Related concepts:

[Defining the ruleset content and signature](#)  
[Executing rulesets in the decision engine](#)

### Related tasks:

[Running and debugging decision operations](#)

### Related information:

[Categories](#)  
[Improving performance on large rule projects](#)  
[Business object model \(BOM\)](#)

# Defining rule project references

You use project references to make your business rule application more modular.

## About this task

A rule project can reference other rule projects. All items in the referenced rule project are then available to you for use in your current rule project.

Rule project references are a way to make your business rule application modular, by reusing rule project items such as business object models or templates in several rule projects. The decision service approach to creating your rule project hierarchy is perfectly suited to handle the dependencies of the rule projects it contains.

You can either define rule project references when you create a decision service, or add them later using the rule project Properties dialog.

## Procedure

To add project references:

1. In the Rule Explorer view, select the rule project, and on the **Project** menu click **Properties**.
2. Click **Project References** to display a list of rule projects that you can reference.
3. Select each rule project that you want to reference and then click **OK**.

## Results

Your rule project references are now defined.

**Parent topic:** [Setting up rule projects](#)

## Defining a folder structure for rule project items

When you create a rule project, a folder is automatically created for each type of rule project item.

### About this task

When you create a rule project, you automatically create folders to store the different types of rule project items:

- BOM entries: bom
- Queries: queries
- Rule artifacts: rules
- Resources: resources
- Templates: templates
- Deployment: deployment

### Procedure

1. In the Rule Explorer view, select the rule project and then on the Project menu click **Properties**.
2. In the pane of the Rule Project Properties dialog, click **Rule Project Folders** to display the list of rule project folders.
3. Optional: If you want to modify the Resource folder:
  - a. In the Rule project folders area, select the Resource Folder path, and then click **Edit**.
  - b. In the Select Folder dialog, either select an existing folder, or create a new one. To create a new folder, select the rule project, and click **Create New Folder**, specify the new folder name and click **OK**.
  - c. Click **OK** to close the Select Folder dialog.
4. Optional: If you want to modify the Output folder:

The Properties dialog displays the new Output folder path.

- a. Click **Edit** next to the Output folder field.
  - b. Either select the folder you want to use as the output folder, or create a new one.
5. Click **OK** to close the Properties dialog.

### Results

Your rule project folders are now redefined.

**Parent topic:** [Setting up rule projects](#)

## Rule project item naming conventions

When you create a new rule project item, follow the naming conventions.

Make sure that you use the Java™ naming conventions when you create a name for your rule project items. You can use spaces because they are processed into valid characters when you execute, but do not use spaces at the beginning of a name.

The name of a rule project item must not:

- Be empty
- Be longer than 255 characters
- Start or end with white space characters
- Contain the characters ", ., :, \*, /, <, >, ?, \, or |
- Contain a character that has a ASCII or Unicode code below 0x1f, included

|                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Restriction:</b> Be careful with the use of Japanese, Korean, and Chinese (simplified and GB18030 encoding) characters in your rule project names. |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|

**Parent topic:** [Setting up rule projects](#)

**Related information:**

[Overview. Ways to express business rules](#)



## Improving performance on large rule projects

In large rule projects, you can improve the performance of Rule Designer.

When your rule projects grow larger, you can usually improve Rule Designer performance by using some Rule Designer build configurations that are better suited to large projects, limiting the BOM footprint, and reducing the size of business rule artifacts.

### Disabling the automatic build

By default in Eclipse, a build process is started as soon as a resource changes in the workspace. As a consequence, as soon as you save a file in Rule Designer, a build runs. The build is usually incremental and fast, so there is no particular issue with the fact that it runs regularly. However, if you make changes that affect the rule project globally, such as modifications on the business object model (BOM), the build might have to recheck many elements. The build can be slow if your project is large.

Deselect **Project > Build Automatically** to disable the automatic build.

#### Note:

If you disable the Eclipse automatic build, rule refactoring might not always be activated. In this case, you must explicitly ask for a build to refresh the errors in the Problems view.

### Setting up preferences for a faster build

When you have a large rule project, particularly one with many decision tables and ruleflows, the build can become slower. In Rule Designer you can define preferences for the rule project build.

1. To access the build preferences page, select **Windows > Preferences**. (On Mac, click **Eclipse > Preferences**.)
2. Select **Rule Designer > Build**

When you save a rule project item, Rule Designer automatically generates IRL code and runs some checks on this code, including rule analysis. You can choose to disable the rule analysis checks only, or to disable all checks by clearing the **Perform IRL checks during build** option.

#### Note:

If you disable these options, you do not receive warning and error messages when saving your rules, but the rule project build is faster. However, more time is required when you extract a ruleset archive from the rule project, because the IRL code is generated and checked at that point. You can also disable IRL code generation to make the build even faster.

### Limiting the number of business rules in rule packages

Rule Designer is more efficient when you work with several small or medium-sized packages, as opposed to a few large packages, for the following reasons:

- Rule Designer loads only the required packages. If all your rule artifacts are stored in a single package, Rule Designer must load all the rule artifacts when you start it. If your rules are organized into packages, Rule Designer loads only the rule artifacts from the required packages.
- To resolve the fully qualified name of a rule, Rule Designer navigates through packages to find the rule. If the tree of packages is well distributed, Rule Designer accesses the rule faster.

#### Note:

The fully qualified name of a rule is the short name of the rule prefixed with its package name.

However, make sure that you do not have too many small packages, because looking for rule model elements slows down Rule Designer.

### Reducing the business object model footprint

When your rule project contains a large business object model (BOM), the features related to business rule language are slower. Parsing, checking, Content Assist, and navigation in business rules and technical rules is slower with large business object models.

To reduce the BOM footprint, you can apply the following techniques:

#### BOM size

- Reduce the BOM size by removing unused verbalizations, BOM members and classes: When you define a BOM entry from a XOM, the BOM entry might include numerous technical classes and methods that are not actually used in your rules. Because the size of the BOM affects Rule Designer performance, it is better to remove all unused business elements from the BOM.
- Split the BOM into several smaller BOMs that can be spread among your rule projects. See [Setting up a project hierarchy](#).

## BOM tree

Collapse the BOM tree in the Rule Explorer View before saving any changes to the BOM and starting a build. A collapsed BOM tree significantly reduces the time taken to build the project.

## Vocabulary parsing

A large BOM with many verbalizations might cause noticeable pauses in Rule Designer. To improve performance, you can reduce the time taken by BRL parser generation by disabling this option: **Window > Preferences > Rule Designer > Business Rule Editing > Enable Quick Fix**. As a consequence, the user is no longer prompted with Quick Fix suggestions when editing rules with the IntelliRule editor. The guided editor behavior is not affected because it does not support the Quick Fix feature.

## Limiting the size of decision tables and decision trees

Decision tables that contain more than 3,000 rows, given a reasonable number of columns, significantly increase editing and build time. Tables with many columns should have fewer rows, because the main scalability factor is the number of cells.

In Rule Designer, you can significantly improve the usability and performance of large decision tables and trees:

- If possible, break up your large decision tables into smaller tables to improve their response time and make it easier to maintain their contents.
- Disable overlap and gap checking.
- Set up the build to disable IRL generation and checking. See [Setting up preferences for a faster build](#).

## Limiting the size of RuleApps

When a RuleApp archive is generated, the number and size of its rule artifacts account for most of the resource consumption.

To reduce the resource usage cost of a business rule application, you can apply the following techniques:

- Transform each large rule project into several smaller ones.
- Keep each rule project as it is, and generate multiple smaller rulesets out of it. You can use ruleset extractors, one per ruleset, to extract part of the rule project. In this case, you might have to change the application that executes the rulesets.

## Upgrading to 64-bit Java SDK

If your system configuration supports it, run Rule Designer with 64-bit Java™ SDK to help building large projects if memory runs scarce.

**Parent topic:** [Setting up rule projects](#)

## Orchestrating ruleset execution

You combine rules and rule artifacts into a ruleset for subsequent execution. You can pass data and share code in a ruleset, and manage the flow of rule execution.

Rules apply decisions, but they do not have a defined sequence or succession. A ruleflow organizes rules into a sequence of decisions by assembling the rules into a group of rule tasks that uses a set execution pattern.

Each rule task is evaluated to produce a result, or decision. All these results and decisions are combined to produce a single business decision, which is represented by a ruleflow. Ruleflows also specify the transitions between rule tasks. The transitions determine how, when, and under what conditions to use each rule task.

The following diagram shows the levels of refinement for selecting the rules.



The evaluation during ruleset execution selects rules in the following manner:

### 1. Ruleflow scope selection

Each rule task in a ruleflow has a scope that is defined by a list of rule packages and individual rules. At run time, ruleflow scope selection generates a list of the rule packages and rules that you defined for each task. When a task is run, the rule engine considers only the rules within this scope.

### 2. Runtime rule selection

The runtime rule selection process further determines which rules the rule engine must consider. Filtering is typically based on the value of rule properties and execution parameters.

### 3. Rule overriding

After the other selection mechanisms are executed, certain rules that you defined beforehand override other rules. The overridden rules are filtered out of the selection.

The rule engine evaluates the conditions of the selected rules, and runs their rule actions on the matching objects.

### [Working with ruleflows](#)

In Rule Designer, you create ruleflows and add different types of tasks to control the execution of rules.

### [Configuring rule execution](#)

You can notify the rule engine of changes in object states. You can also specify how rules in a ruleset are executed at run time by setting up the ordering, priorities, or overriding of rules.

**Parent topic:** [Developing rulesets in Rule Designer](#)

# Working with ruleflows

In Rule Designer, you create ruleflows and add different types of tasks to control the execution of rules.

## **Overview: Ruleflows**

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

## **Creating a ruleflow**

You can create a ruleflow in Rule Designer, and add elements to your ruleflow in the Ruleflow Editor. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

## **Configuring rule task execution**

You define how rules are selected at run time and specify the general order in which the rule tasks associated with these rules are executed.

## **Configuring ruleflow properties**

You configure the properties of ruleflow elements in the **Properties** view of each element.

**Parent topic:** [Orchestrating ruleset execution](#)

## Overview: Ruleflows

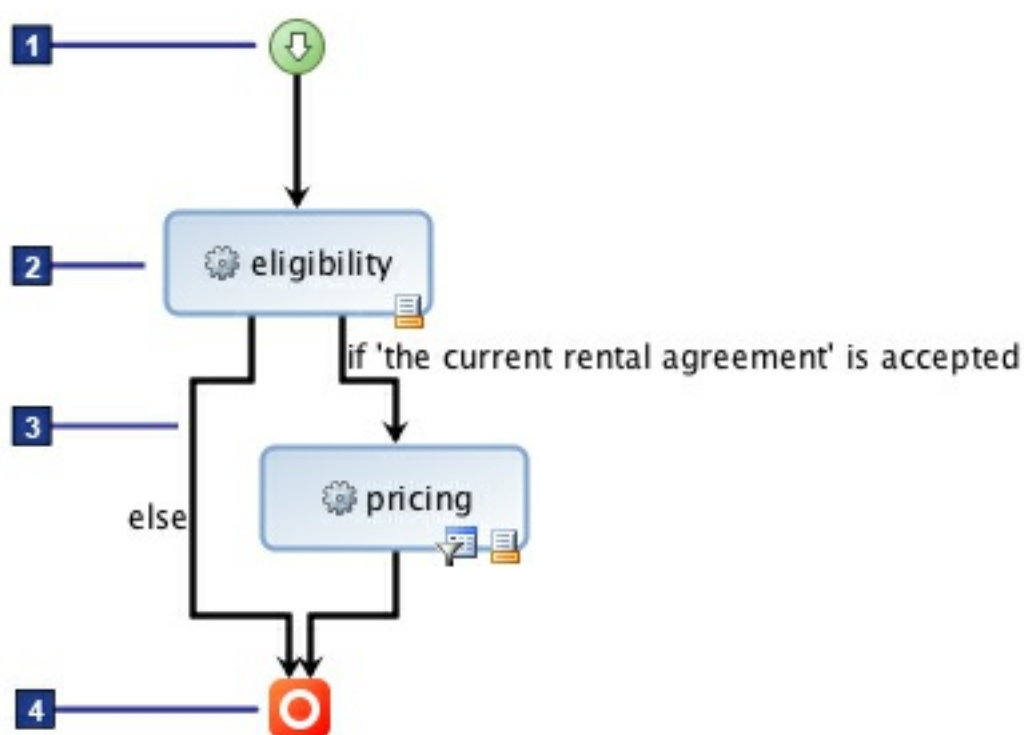
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



**Note:** The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

### Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

### Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

#### Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter


statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

## Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

## Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

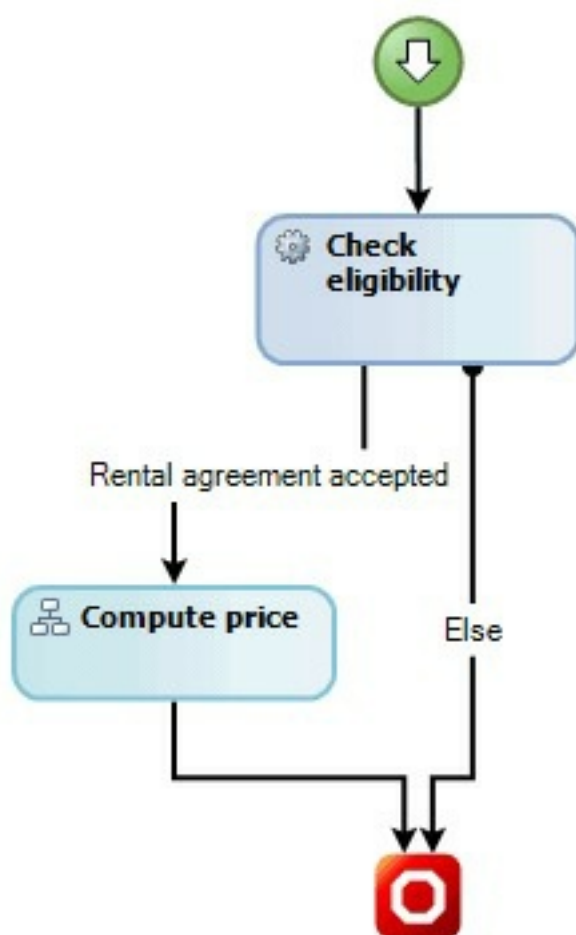
You can specify initial and final actions on subflow tasks.

## Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.



**Note:** In transition conditions, variables cannot reference objects in the working memory. They can only reference ruleset parameters.


A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.


## Branches



A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch. Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

## Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

## Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

**Note:** Like action tasks, initial and final actions cannot call methods on objects in the working memory. They can only call the methods of ruleset parameters.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

**Parent topic:** [Working with ruleflows](#)

# Creating a ruleflow

You can create a ruleflow in Rule Designer, and add elements to your ruleflow in the Ruleflow Editor. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

## About this task

You can add a ruleflow to a project, or any of its packages. You can have multiple ruleflows in a project, but if you have more than one ruleflow in a project, you must define one of them as the main ruleflow. After adding a ruleflow to a rule project or package, you define the structure by adding the elements that you need in the ruleflow diagram. To understand the use of each element, see [Ruleflows](#). Then, you connect these elements by adding transitions between them.

## Procedure

1. In the Rule Explorer, select the name of the package or project you want to add a ruleflow to, and click



**New Ruleflow** on the New Rule Project Item toolbar.

In the **New Ruleflow** dialog that opens, enter the relevant information for your ruleflow. If you are adding the ruleflow to a package, make sure the name of this package is added in the **Package** field. No entry is needed for a ruleflow created under the top folder.

2. Use the buttons in the Ruleflow Editor palette to add elements in the ruleflow diagram as required. You must create one start node for your ruleflow, and at least one end node.
3. Create rule tasks, and add rules to be executed at this point in the ruleflow.
  - a. Add a rule task element in the diagram.
  - b. Set the properties that you need in the **Properties** view. For more information about rule tasks properties, see [Configuring ruleflow properties](#).
  - c. In the **Rule Selection** tab of the **Properties** view, click **Edit** to open the **Select Rules** dialog, and add rules to a rule task. You can use the **Up** and **Down** buttons to order the rules and packages. Depending on the rule execution properties of the task, this order might impact the output of ruleflow execution.

**Tip:** You can also drag an existing rule package or rule from the Rule Explorer into the ruleflow diagram. The rule task has the name of the element and already contains the rules and packages.

4. Optional: If you need to execute rule action statements, add action tasks in your ruleflow, and set the action statement, initial action, and final action in the **Properties** view.
5. Add transitions between the tasks to define the flow of your ruleflow.
  - a. To specify a condition for the transition, in the **Properties** view for the transition, click **Condition**.
  - b. Provide a name for the condition in the **Label** field.
  - c. Select **Use BAL for transition condition** and type a condition statement. For example:

**'the current rental agreement' is accepted**

You can also write the condition using IRL. In this case, make sure the text fields contain a valid Boolean expression. If you do not use BAL, in the ruleflow diagram the transition arrow displays the label and the expression.

In transition conditions, the variable scope is restricted to ruleset parameters and variables. It does not include access to the working memory.

6. Optional: You can create multiple, parallel paths in your ruleflow, if you need to execute rules simultaneously. For example, if you are checking the eligibility of a customer for a loan, you might want to check whether the customer meets the criteria for the loan, and also if the amount requested is valid. To do so, you use forks and joins in your ruleflow.
  - a. Add a fork node where you want your ruleflow to execute several rules in parallel. You can then add your rule tasks in the ruleflow.
  - b. Add a join where you want to combine the transitions created from the fork. Make sure to create all transitions between the different elements.

The transitions from a fork node to a join node must not have conditions, because the ruleflow follows all paths in parallel between the fork and the join.


7. Optional: You can add branches to the ruleflow to organize conditional transitions, in the same way that you could start several conditional transitions from a task.

### Important:

When multiple transitions originating from a branch or a task define overlapping conditions, the path taken to execute the ruleflow is unpredictable. Make sure the conditions you define for multiple



transitions do not overlap.

- a. Add a branch node where you want your ruleflow to organize the different conditions.
  - b. To name the branch, in the ruleflow diagram click the branch icon and in the **Properties** view, click **Branch Node** and enter the name in the field provided.
  - c. Add transitions to and from the branch.
  - d. Add transition conditions for each transition from the branch. One of the transitions must be an Else transition.
8. Optional: If you want to execute another ruleflow at some point in your main ruleflow, add a subflow task in the diagram:
- a. In the **Properties** view for the subflow task, click **Subflow Task**.
  - b. Click **Select** to select a ruleflow to include in the subflow task.
9. To align the ruleflow automatically, click  **Layout All Nodes** in the Ruleflow Editor toolbar. You can also align items manually by selecting them, right-click the ruleflow, and select **Align**. Select the alignment options in the pop-up menu that opens.
10. Save the ruleflow (Ctrl+S).

**Parent topic:** [Working with ruleflows](#)

**Related concepts:**

[Overview: Ruleflows](#)

**Related information:**

[Execution properties for rule tasks](#)

## Configuring rule task execution

You define how rules are selected at run time and specify the general order in which the rule tasks associated with these rules are executed.

### Runtime rule selection

You can define a selection filter on a rule task to specify dynamically what rules of the rule task must execute.

### Execution properties for rule tasks

You can define rule task execution properties to select the rule engine algorithm that is used to execute the rules, and refine the number and order of rules that the rule engine executes.

**Parent topic:** [Working with ruleflows](#)

## Runtime rule selection

You can define a selection filter on a rule task to specify dynamically what rules of the rule task must execute.

You specify the rules selected at run time in the **Rule Selection** tab of the **Properties** view. You can select individual rules, or packages containing several rules, that are considered for execution in this rule task, and filter out some of them. You can specify this filter with dynamic or static BAL constructs, or IRL. The filter is applied at run time, and the rule engine evaluates only the rules that pass through the filter.

### Note:

An empty list of rules and packages means that all the rules from the project are selected for execution at run time.

### Selecting rules with BAL

Typically, the filtering process is based on the values of rule properties and execution parameters. The filter tests each rule in the ruleflow task to determine whether the rule should be selected for execution. For example, the following code filters on the name of the rule and is called for every rule in the task.

```
the name of 'the rule' contains "Age"
```

There is no difference between dynamic BAL and static BAL. The filter is evaluated each time the rule task is invoked. Candidate rules can be selected based on the ruleset parameter state. For example, you can specify that the expiry date of the rule is after the date of the loan.

### Selecting rules with IRL

#### Static body

The list of rules is specified simply by including the names inside the rule task body.

```
body = { R1, R2 }
```

#### Static body using the select keyword

Here is an example of selecting rules using the select keyword:

```
body = select(?rule) { <boolean returning code> }
```

This first example shows that a body of the rule task can be an IRL predicate that selects the relevant rules among all the rules of the ruleset.

In this second example, the specification selects all the rules available in the ruleset:

```
body = select(?rule) { return true; }
```

### Important:

There is no difference between `dynamicselect` and `select`. `select` behaves the same way as `dynamicselect` where the statement is called each time the task is executed.

#### Dynamic body

The list of rules is selected by a function-like expression that takes an [IlrRule](#) object as its argument and evaluates to a Boolean expression. The pseudo-variable `?rule` represents the `IlrRule` argument of the expression. This filter applies to each rule in the ruleset each time the rule task is activated. Hence, this expression might well depend on environment variables such as ruleset parameters or fields whose values can change between two executions of the rule task.

```
body = dynamicselect(?rule) {
 return myRuleSelection(?rule);
}
```

#### Dynamic body in a domain

The domain is an expression that evaluates to a Java™ array or collection of [IlrRule](#) objects. The dynamic body is also an expression. You can use expressions to select the subset of the rules of the domain for the next rule task execution. The dynamic body is evaluated each time the rule task is activated.

It accepts two different signatures.

- **Dynamic body that returns a collection of rules:**

The dynamic body is an expression that evaluates to a Java array or collection of an `IlrRule` object.

```
body = dynamicselect() {
 return myRuleDomainSelection();
}
in myInitialRuleDomain();
```

**Attention:**

There is no check at run time that the collection of rules returned by the dynamic body is really a subset of the initial rule domain. You must enforce such verification.

- **Dynamic body that returns a Boolean value:**

The dynamic body is a function-like expression that takes an [IlrRule](#) object as its argument and evaluates to a Boolean expression. The pseudo-variable `?rule` represents the single `IlrRule` argument of the expression. The engine passes to this rule filter only the rules that belong to the initial rule domain.

```
body = dynamicselect(?rule) {
 return myRuleSelection(?rule);
}
in myInitialRuleDomain();
```

See [Memory consumption reduction](#) for more information.

## Scope

The scope defines in a symbolic way what rules compose the body of a rule task. This provides a simpler way of describing the selection than using IRL expressions such as an `in` expression. The scope makes use of the package organization.

**Note:**

Because the IRL `in` expression and the scope provide alternate selection methods, you cannot use them together.

**Parent topic:** [Configuring rule task execution](#)

**Related information:**

[Working with ruleflows](#)

[Setting simple priorities among rules](#)

[Ordering rules in a ruleset](#)

# Execution properties for rule tasks

You can define rule task execution properties to select the rule engine algorithm that is used to execute the rules, and refine the number and order of rules that the rule engine executes.

You specify the rule task execution properties in the **Rule Task** tab of the **Properties** view to control rule execution in different execution modes. You can choose a rule engine algorithm to execute the rules in a particular rule task, and use execution control properties to further refine the number and order of rules the rule engine should execute.

In general, the order of the rule artifacts in the list of selected rules is not guaranteed, and the list of selected rules is considered as a pool of rules to be executed. However, with **Ordering** set to **Literal**, the order of the rule artifacts selected becomes important.

Table 1. Control settings for rule tasks

| Action                  | Location                                                                                                                            | Description                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Rule selection          | <b>Rule Selection</b> tab in <b>Properties</b> view for rule task.                                                                  | Click <b>Edit</b> to add and change the order of rules and rule packages by using the <b>Select Rules</b> dialog.                                                                                                                                                                                                                                                                                                             |
| Runtime rule selection  | <b>Dynamic BAL</b> , <b>Static BAL</b> and <b>IRL</b> in the <b>Rule Selection</b> tab of the <b>Properties</b> view for rule task. | Control which rules are evaluated in a rule task.                                                                                                                                                                                                                                                                                                                                                                             |
| Execution mode          | <b>Rule Task</b> tab in <b>Properties</b> view for rule task.                                                                       | Specifies an execution mode for the rule task. See <a href="#">Changing the execution mode</a> for details.                                                                                                                                                                                                                                                                                                                   |
| Rule task ordering      | <b>Rule Selection</b> tab in <b>Properties</b> view for rule task.                                                                  | <p>The default ordering of a rule task:</p> <ul style="list-style-type: none"><li>• <b>Default</b> the ordering of rules depends on the execution mode.</li><li>• <b>Literal</b> follows the rule task ordering.</li><li>• <b>Priority</b> sorts rules according to the execution mode in operation.</li></ul> <p>See <a href="#">Deciding on an execution mode</a> and <a href="#">Rule execution order</a> for details.</p> |
| Rule task exit criteria | <b>Rule Task</b> tab in <b>Properties</b> view for rule task.                                                                       | <b>Choose exit criteria</b> specifies how rules are executed before the task terminates. See <a href="#">Exit criteria property</a> for details.                                                                                                                                                                                                                                                                              |

## Control properties for rule tasks in execution modes

You can set specific properties to order rule tasks. These settings operate differently in each execution mode: RetePlus, Sequential, and Fastpath. See [Engine execution modes](#) for a description of each execution mode.

The following table outlines how ruleflow control properties operate in RetePlus execution mode.

Table 2. Task ordering properties for RetePlus

| Ordering property                                                                       | Exit Criteria property                                                                                                                                                       | Advanced Properties only                                                                                           |
|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <p><b>Default</b></p> <p>A RetePlus network with full agenda management is created.</p> | <p><b>None</b></p> <p>All the rules in the task body are activated. Rule instances are executed until the agenda is empty.</p> <p>Equivalent to:</p> <p>firing=allrules,</p> | <p>firinglimit &gt; 0: instances are executed until either the given number is reached or the agenda is empty.</p> |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                               |                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                         | firinglimit=0                                                                                                                                                                                                                                                                 |                                                                                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>Rule</b><br><br>Only the highest priority rule of the task body is activated. Rule instances are executed until the agenda is empty.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=0                                                                               |                                                                                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>RuleInstance</b><br><br>Only the highest priority rule of the task body is activated. Just the first instance of the rule is executed, or none if the agenda is empty.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=1                                             |                                                                                                               |
| <b>Literal</b><br><br>A RetePlus network is created but there is no agenda. Rules are activated one by one according to the order provided by the body, and the instances are executed without reevaluation.<br><br>Or:<br><br><b>Priority</b><br><br>A RetePlus network is created but there is no agenda. Rules are first sorted in decreasing order of priority. They are then activated one by one and the instances executed without reevaluation. | <b>None</b><br><br>A loop is made on each of the rules provided in the task body, their instances computed and executed. Instances are executed until the end of the loop.<br><br>Equivalent to:<br><br>firing=allrules, firinglimit=0                                        | firinglimit > 0: rule instances are executed until either the given number or the end of the loop is reached. |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>Rule</b><br><br>A loop is made on each rule provided in the task body. As soon as a rule can be instantiated (that is, at least one rule instance is created), all those instances are executed and the loop ends.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=0 |                                                                                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>RuleInstance</b><br><br>A loop is made on each rule provided in the task body. As soon as a rule can be instantiated (that is, at least one rule instance is created) that instance is executed and the loop ends.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=1 |                                                                                                               |

The following table outlines how ruleflow properties operate in sequential execution mode.

Table 3. Task ordering properties for Sequential

| Ordering property                                                                                                            | Exit Criteria property | Advanced Properties only |
|------------------------------------------------------------------------------------------------------------------------------|------------------------|--------------------------|
| <b>Default</b><br><br>In the sequential mode, the <b>Default</b> property acts the same way as the <b>Priority</b> property. |                        |                          |

|                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                  |                                                                                                                                                       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Literal</b></p> <p>Rules are compiled according to the order provided by the task body. Rules are evaluated and executed sequentially against the incoming tuple objects.</p> <p>Or:</p> <p><b>Priority</b></p> <p>Rules are sorted according to their priorities. They are then compiled according to the sorted order. The rules are evaluated and executed sequentially against the incoming tuple objects.</p> | <p><b>None</b></p> <p>All the rules provided in the task body are evaluated. All the rules evaluated as true are executed until there are no more rules to select. The order is the one determined by the ordering property.</p> <p>Equivalent to:</p> <p>firing=allrules,<br/>firinglimit=0</p> | <p>firinglimit &gt; 0:<br/>rules evaluated as true are executed until the given number of rules is reached, or there are no more rules to select.</p> |
|                                                                                                                                                                                                                                                                                                                                                                                                                          | <p><b>Rule</b></p> <p>The rules provided in the task body are evaluated sequentially. As soon as one rule evaluates to true, it is executed and the loop ends.</p> <p>Equivalent to:</p> <p>firing=rule</p> <p>This property forces the firinglimit to 1.</p>                                    | <p>firinglimit</p> <p>Not applicable.</p>                                                                                                             |
|                                                                                                                                                                                                                                                                                                                                                                                                                          | <p><b>RuleInstance</b></p> <p>The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), that instance is executed and the loop ends.</p> <p>Equivalent to:</p> <p>firing=rule, firinglimit=1</p>                       |                                                                                                                                                       |

The following table outlines how ruleflow properties operate in Fastpath execution mode.

Table 4. Task ordering properties for Fastpath

| Ordering property                                                                                                                                                                                                                                                                                                                                                                                                    | Exit Criteria property                                                                                                                                                                                                                                                                   | Advanced Properties only                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Default</b></p> <p>In the Fastpath mode, the <b>Default</b> property acts the same way as the <b>Priority</b> property.</p>                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                          |                                                                                                                                                       |
| <p><b>Literal</b></p> <p>Rules are compiled according to the order provided by the task body. Rules are evaluated and executed sequentially against the incoming tuple objects.</p> <p>Or:</p> <p><b>Priority</b></p> <p>Rules are sorted according to their priority. They are then compiled according to the sort order. The rules are evaluated and executed sequentially against the incoming tuple objects.</p> | <p><b>None</b></p> <p>All the rules provided in the task body are evaluated. All the rules evaluated as true are executed until there are no more rules to select. The order is determined by the ordering property.</p> <p>Equivalent to:</p> <p>firing=allrules,<br/>firinglimit=0</p> | <p>firinglimit &gt; 0:<br/>rules evaluated as true are executed until the given number of rules is reached, or there are no more rules to select.</p> |
|                                                                                                                                                                                                                                                                                                                                                                                                                      | <p><b>Rule</b></p> <p>The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), those instances are</p>                                                                                                        |                                                                                                                                                       |

|  |                                                                                                                                                                                                                                                                     |  |
|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
|  | executed and the loop ends.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=0                                                                                                                                                                                 |  |
|  | <b>RuleInstance</b><br><br>The rules provided in the task body are evaluated. As soon as one rule can be instantiated (that is, one rule instance is created), that instance is executed and the loop ends.<br><br>Equivalent to:<br><br>firing=rule, firinglimit=1 |  |

**Rule execution order**

When the rule artifacts for a rule task are selected at compile time or run time, the packages in the list of selected rules are expanded. The expansion starts at the top of the list. If a business rule is listed separately above its package, it retains its execution position in the list. However, if it is listed separately under its package, the business rule executes from the package. Under the package, the business rule is redundant because it is already in the package.

For example, consider a package that contains rules R1, R2 and R3:

- If you add R2 and then the package, the expanded list is R2, R1 and R3.
- If you add the package and then R2, the expanded list is R1, R2 and R3.
- To force R2 to be last in the list, you must list the rules separately: R1, R3, and then R2.

**Exit criteria property**

The Exit Criteria property specifies whether all the rules execute or just the first instance of the first rule executes. You can set the property to one of the following values:

- None: All the instances of all the applicable rules execute for a tuple before moving to the next tuple.
- Rule: All the rule instances of the first applicable rule execute for a tuple before moving to the next tuple.
- Rule Instance: Only the first rule instance of the first applicable rule executes for a tuple before moving to the next tuple.

**Parent topic:** [Configuring rule task execution](#)

**Related reference:**

[firing](#)  
[firinglimit](#)

**Related information:**

[Choosing an execution mode](#)  
[Working with ruleflows](#)  
[Setting simple priorities among rules](#)



# Configuring ruleflow properties

You configure the properties of ruleflow elements in the **Properties** view of each element.

To display the **Properties** view below the editor, you must select a ruleflow element in the diagram. The **Properties** view opens, with a list of tabs where you set the properties for your ruleflow elements. When you add a node or a task to your ruleflow, you must set their properties.

To set the properties for the whole ruleflow, select an empty area in the diagram to open the **Properties** view.

Some properties, or tabs, are common to all ruleflow elements, and are described in the section [Common properties for ruleflow elements](#). Specific properties for each ruleflow element are detailed in the following sections:

- [Ruleflow properties](#)
- [Rule tasks properties](#)
- [Action tasks properties](#)
- [Subflow tasks properties](#)
- [Transitions properties](#)

## Common properties for ruleflow elements

Some properties, or tabs, can be found in the **Properties** view for ruleflows, start nodes, end nodes, rule tasks, action tasks, subflows, branches, fork nodes, join nodes, and transitions.

**Attention:** A start node does not contain the **Final action** tab, and an end node does not contain the **Initial action** tab.

Table 1. Common properties in the tab corresponding to the ruleflow element name

| Property | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ID       | <p>An ID is the technical name for a ruleflow element. All ruleflow elements have an ID that identifies them in an XML schema.</p> <p>In Rule Designer Java™ code, the ID of a ruleflow element generates the IRL code that executes at run time. To view this code, click the <b>IRL</b> tab under the Ruleflow Editor.</p> <p>If you duplicate a ruleflow element in a ruleflow, it has a unique ID.</p> <p>IDs are not locale-dependent. When you specify an ID, it is good practice to use only characters (a-z, A-Z) and numbers (0-9).</p>                             |
| Label    | <p>You can specify a label for the ruleflow element that is shown in the Diagram page. A label has no impact on the IRL that is generated.</p> <p>Labels are locale-dependent. You can specify different labels in a new locale and recover the labels in the original locale for elements that do not change. For example, if you change the ID of a ruleflow element, you lose the labels defined for this element in all other locales except the original.</p> <p>When you specify a label, it is good practice to use only characters (a-z, A-Z) and numbers (0-9).</p> |

Table 2. Initial action tab

| Property       | Purpose                                                                                                                                         |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Initial action | <p>The action that launches before the task starts.</p> <p>Select <b>Use BAL for action</b> to define the action in BAL, or enter IRL code.</p> |

Table 3. Final action tab

| Property     | Purpose                                                                                                                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Final action | <p>The action that launches after the task ends.</p> <p>Select <b>Use BAL for action</b> to define the action in BAL, or enter IRL code.</p> |

Table 4. Documentation tab

| Property          | Purpose                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Documentat<br>ion | <p>This property displays optional information about the start node. Use this property to provide notes, comments, and other useful information that open as a tooltip.</p> |

|  |  |
|--|--|
|  |  |
|--|--|

### Ruleflow properties

**Note:** To show the properties, ensure that you click in an empty area of the Diagram page and do not select a ruleflow node.

Table 5. Properties tab


| Property              | Purpose                                                                                                                                                                                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>locale</b>         | This property indicates the language setting of the ruleflow. The default setting is en_US. You cannot edit this field.                                                                                                                                                                                                               |
| <b>main flow task</b> | Use this property to indicate if the ruleflow is the main task. Double-click the value to set it to true or false.                                                                                                                                                                                                                    |
| <b>name</b>           | The name of the ruleflow.                                                                                                                                                                                                                                                                                                             |
| <b>tags</b>           | <div> Use this property to view and edit the values used for the import tags. Click a field and  to open the Values for Tags dialog. </div> <div> <b>Important:</b><br/> Tags are supported for compatibility purposes. Leave the tags empty. </div> |

Table 6. Category Filter tab

| Property               | Purpose                                                                                                                                    |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Category Filter</b> | Use this property to edit the rule category filter for the selected ruleflow.<br><br>Click <b>Edit</b> to open the Category Filter dialog. |

Table 7. Imports tab

| Property                                     | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>View the imports used for BOM classes</b> | <div> This property displays the imports for BOM classes. You edit this list to manually add or delete imports. </div> <div> The imported BOM or Java classes required by the IRL code in a ruleflow can be viewed and edited in the ruleflow Properties view. The list is populated automatically when you type in IRL code with Content Assist. Edit the list to remove unused imports, that is, the imports that were not removed automatically, or manually add classes if you have entered IRL code without Content Assist. </div> <div> If a ruleflow in a rule package contains IRL code that references variables of the default package, you must import these variables with the use <code>&lt;variable&gt;</code> declaration. </div> |

### Rule task properties

Table 8. Rule Task tab

| Property             | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Algorithm</b>     | Use this section to specify the processing algorithm for the rule task: <ul style="list-style-type: none"> <li><b>RetePlus</b></li> <li><b>Sequential</b></li> <li><b>Fastpath</b></li> </ul> See <a href="#">Engine execution modes</a> for more information.                                                                                                                                                                              |
| <b>Exit Criteria</b> | Use this section to specify if all rules, one rule, or one rule instance execute: <ul style="list-style-type: none"> <li><b>None:</b> The default setting that all rules are executed until conditions terminate execution. The rules are executed in a particular order determined by the selected ordering.</li> <li><b>Rule:</b> Execution terminates after the chosen rule is executed, according to the selected algorithm.</li> </ul> |

|                            | <ul style="list-style-type: none"><li>• <b>RuleInstance:</b> A single instance of one rule is executed. This rule is determined by the selected ordering.</li></ul>                                                                                                                                                                                                                                                                                                                                                                |                   |             |                  |                                                                                                                                                                                                                                                     |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ordering</b>            | Use this section to specify the order of rule execution in a rule task as follows: <ul style="list-style-type: none"><li>• <b>Default:</b> The algorithm determines the execution order.</li><li>• <b>Literal:</b> The order of the rules in the rule task determines the order of rule execution</li><li>• <b>Priority:</b> the rules are executed following a static priority in decreasing order.</li></ul>                                                                                                                     |                   |             |                  |                                                                                                                                                                                                                                                     |
| <b>Advanced properties</b> | <p>The following table summarizes some features you can set using advanced properties.</p> <p><i>Table 8. Some features set by advanced properties</i></p> <table><tr><th>Advanced property</th><th>Description</th></tr><tr><td>firinglimit = n;</td><td>This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to <b>none</b> and in Advanced Properties add firing = allrules rule and firinglimit = n, n&gt;0. For n = 0 use the <b>Exit Criteria</b> options.</td></tr></table> | Advanced property | Description | firinglimit = n; | This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to <b>none</b> and in Advanced Properties add firing = allrules rule and firinglimit = n, n>0. For n = 0 use the <b>Exit Criteria</b> options. |
| Advanced property          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                   |             |                  |                                                                                                                                                                                                                                                     |
| firinglimit = n;           | This property sets a specific number of rule executions. To use an execution limit, set exitCriteria to <b>none</b> and in Advanced Properties add firing = allrules rule and firinglimit = n, n>0. For n = 0 use the <b>Exit Criteria</b> options.                                                                                                                                                                                                                                                                                |                   |             |                  |                                                                                                                                                                                                                                                     |

Table 10. Rule Selection tab

| Property                                                 | Purpose                                                                                                                                                                                |
|----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>List the rules and rule packages in the rule task</b> | Use this property to list the rules and rule packages that are considered when the rule task is executed. To edit this list and reorder it, use the <b>Up</b> and <b>Down</b> buttons. |
| <b>Dynamic BAL</b>                                       | Use this property to specify runtime rule selection using a dynamic statement BAL. The property runs each time you call the task.                                                      |
| <b>Static BAL</b>                                        | Use this property to specify runtime rule selection using static statement in BAL. The property runs the first time that you call the task.                                            |
| <b>IRL</b>                                               | Use this property to specify runtime rule selection using IRL.<br><br>Define a rule filter in IRL with "body = . . . ."                                                                |

Action task properties

Table 11. Action Task tab

| Property              | Purpose                                                                              |
|-----------------------|--------------------------------------------------------------------------------------|
| <b>Rule Execution</b> | You can select the algorithm, exit criteria, and ordering for running the rule task. |

Table 12. Rule Selection tab

| Property                             | Purpose                                                                        |
|--------------------------------------|--------------------------------------------------------------------------------|
| <b>Edit</b>                          | You list the rules and rule package that are used in the rule task.            |
| <b>Choose runtime rule selection</b> | You select the runtime language for the rule: Dynamic BAL, Static BAL, or IRL. |

Subflow properties

Table 13. Subflow Task tab

|  |  |
|--|--|
|  |  |
|--|--|

| Property          | Purpose                                                 |
|-------------------|---------------------------------------------------------|
| Select a ruleflow | Click <b>Edit</b> to open the Select a ruleflow dialog. |

Transition properties

Table 14. Condition tab

| Property   | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Conditions | <p>In the <b>Conditions</b> tab, you must first select the rule language you use to write the transition: <b>BAL</b> or <b>IRL</b>. In the rule language you chose, you write an expression whose return value must be true or false, for example:</p> <ul style="list-style-type: none"><li>• <b>BAL:</b></li></ul> <div>'the loan' is approved</div> <ul style="list-style-type: none"><li>• <b>IRL:</b></li></ul> <div>(bicycle.speed &gt; 10) &amp;&amp; (bicycle.speed &lt;= 30)</div> <p>If you have several transitions originating from a task, there must be one (and only one) transition with no conditions, which is considered as the default transition, also called <code>else</code> transition. By default, if you do not specify a label, it is displayed as "else".</p> <p>With several transitions, you must make sure that transitions do not overlap, which means that no more than one transition condition must return "true". Typically, you might want to use two transitions, to choose between one ruleflow path with a specific condition and one default <code>else</code> transition for the remaining cases.</p> <p>If there is only one transition that links two tasks, you must not specify any conditions.</p> |

Parent topic: [Working with ruleflows](#)

## Configuring rule execution

You can notify the rule engine of changes in object states. You can also specify how rules in a ruleset are executed at run time by setting up the ordering, priorities, or overriding of rules.

### Updating object states in the rule engine

When rules are executed with the RetePlus execution mode, you can control whether the rule engine is notified of changes in the state of objects in the working memory that result from rule execution. Notifying the rule engine of an object state change causes the rules to be matched against the new state, and can result in new rule instances being added to the agenda.

### Ordering rules in a ruleset

The order in which rules are presented in the ruleset affect the order in which they are placed in the ruleset archive during the final packaging of the ruleset for execution. You can sort rules alphabetically, or manually.

### Setting simple priorities among rules

Rules have priorities, which can be static or dynamic. Use a constant to define a static priority, or an expression containing a ruleset variable to define a dynamic priority.

### Rule overriding

Rule overriding is the last selection mechanism after ordering and priorities. Overridden rules are not selected for execution. You can combine rule overriding with the hierarchical property.

**Parent topic:** [Orchestrating ruleset execution](#)

## Updating object states in the rule engine

When rules are executed with the RetePlus execution mode, you can control whether the rule engine is notified of changes in the state of objects in the working memory that result from rule execution. Notifying the rule engine of an object state change causes the rules to be matched against the new state, and can result in new rule instances being added to the agenda.

### Object state update

In RetePlus execution mode, objects are stored in working memory. You set the "object state update" option to notify the rule engine of a change, and match the rules against the object new state.

### Notifying the rule engine of object updates

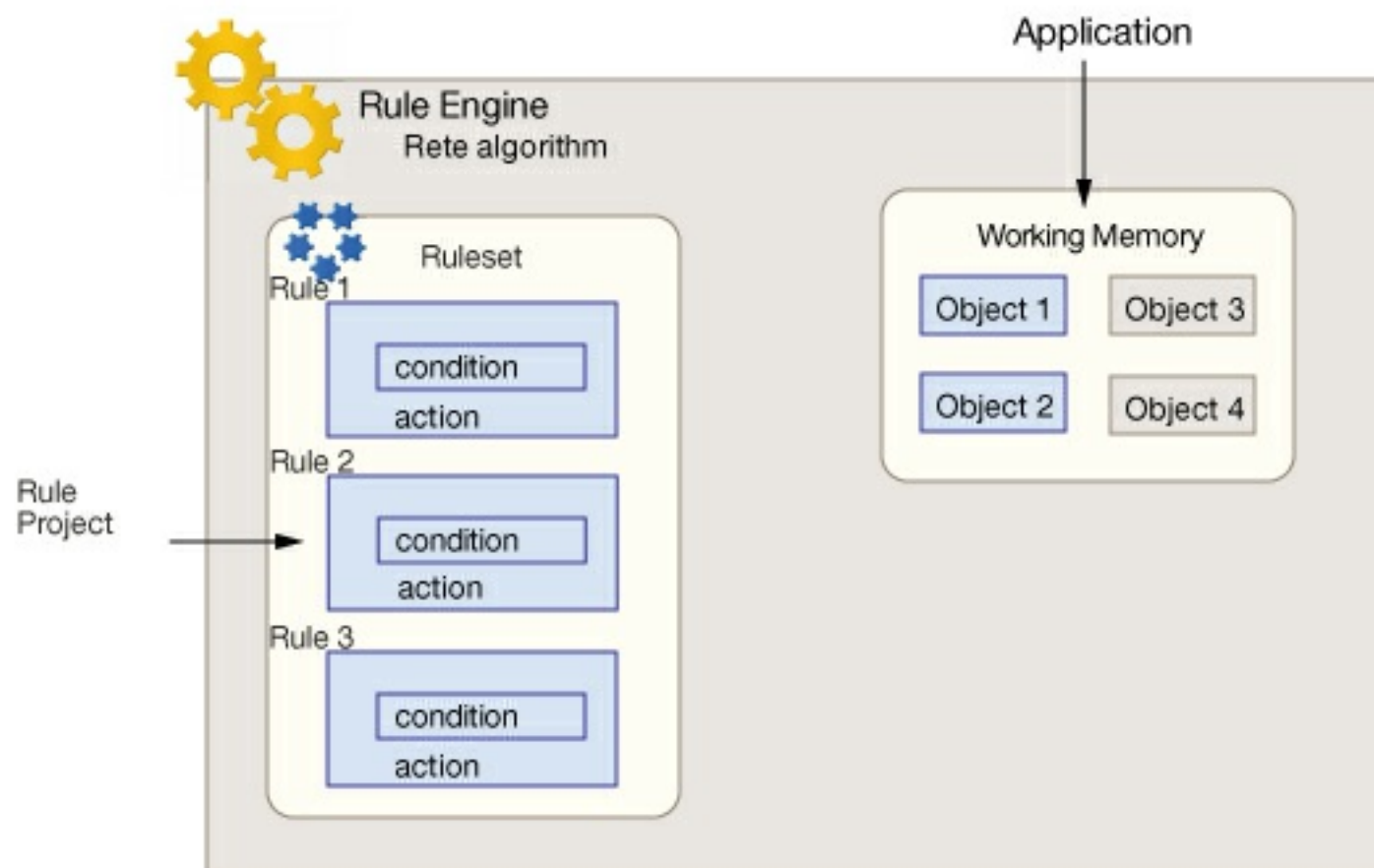
When operating in RetePlus execution mode, you can specify which methods to use to notify the rule engine of object updates.

**Parent topic:** [Configuring rule execution](#)

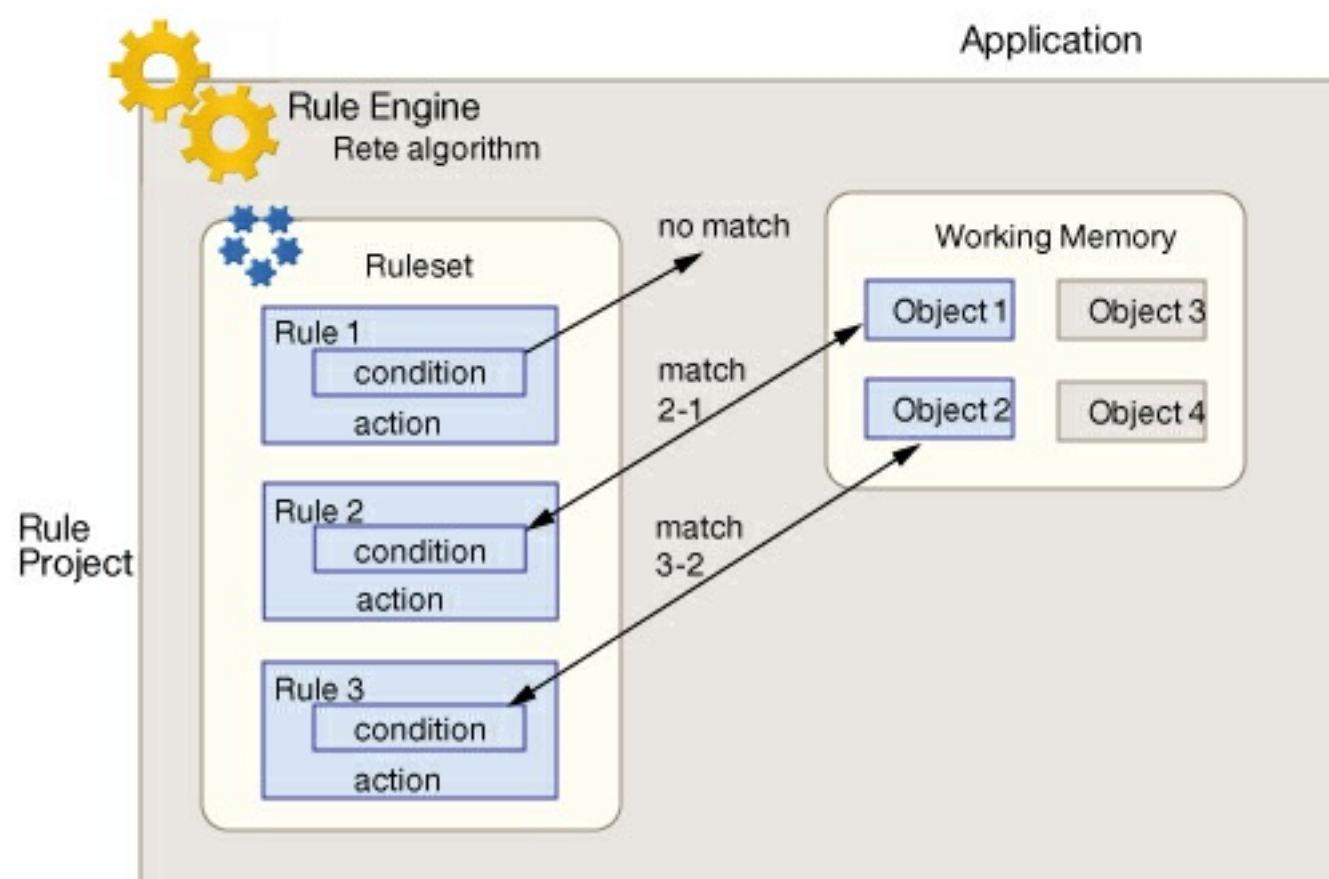
## Object state update

In RetePlus execution mode, objects are stored in working memory. You set the "object state update" option to notify the rule engine of a change, and match the rules against the object new state.

When you execute rules using the RetePlus execution mode, the rules are compiled into a ruleset and sent to the rule engine. In the meantime, application objects are loaded into the rule engine in what is called the working memory.

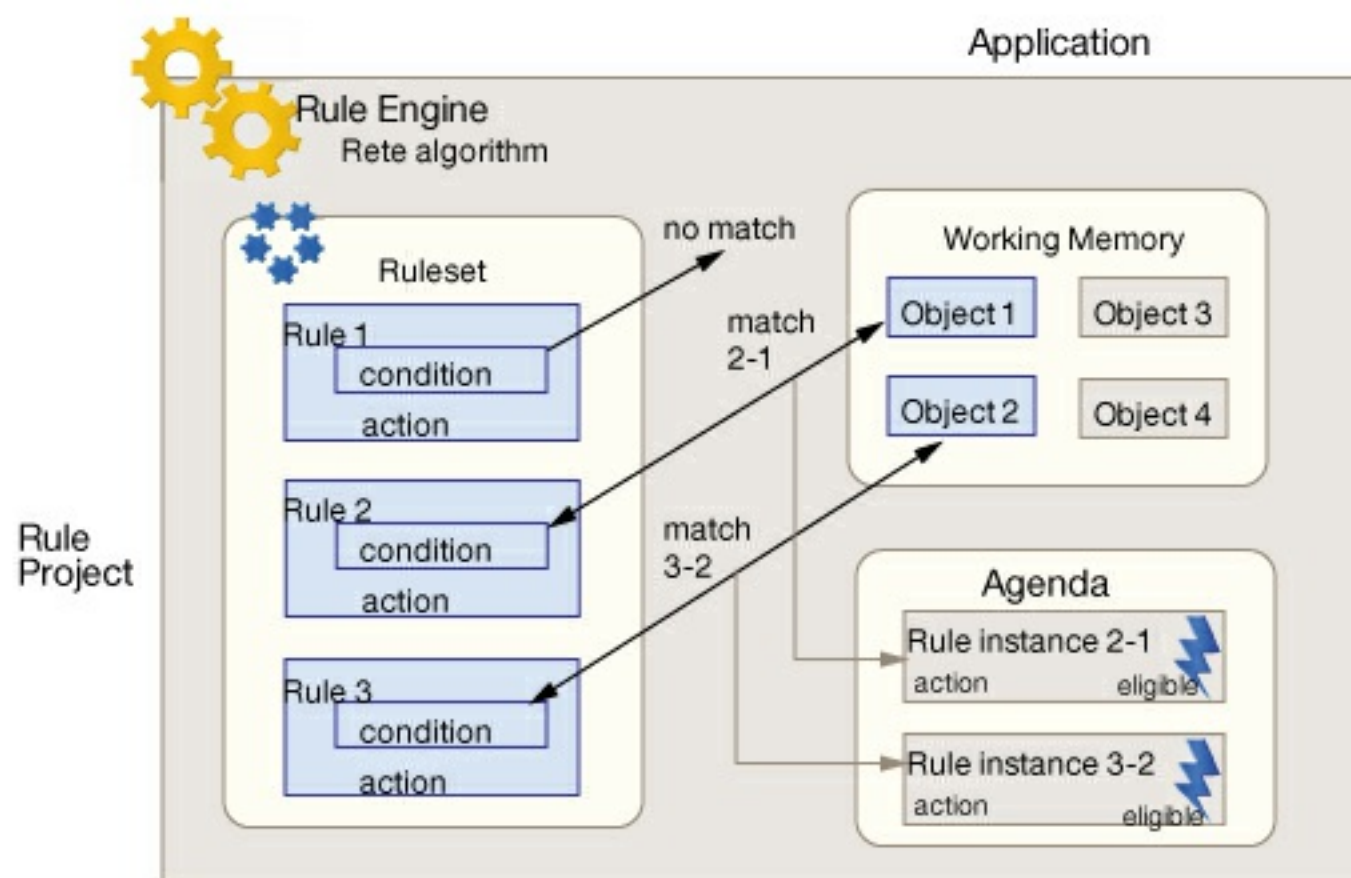


The condition part of the rules is then matched against the objects in working memory.

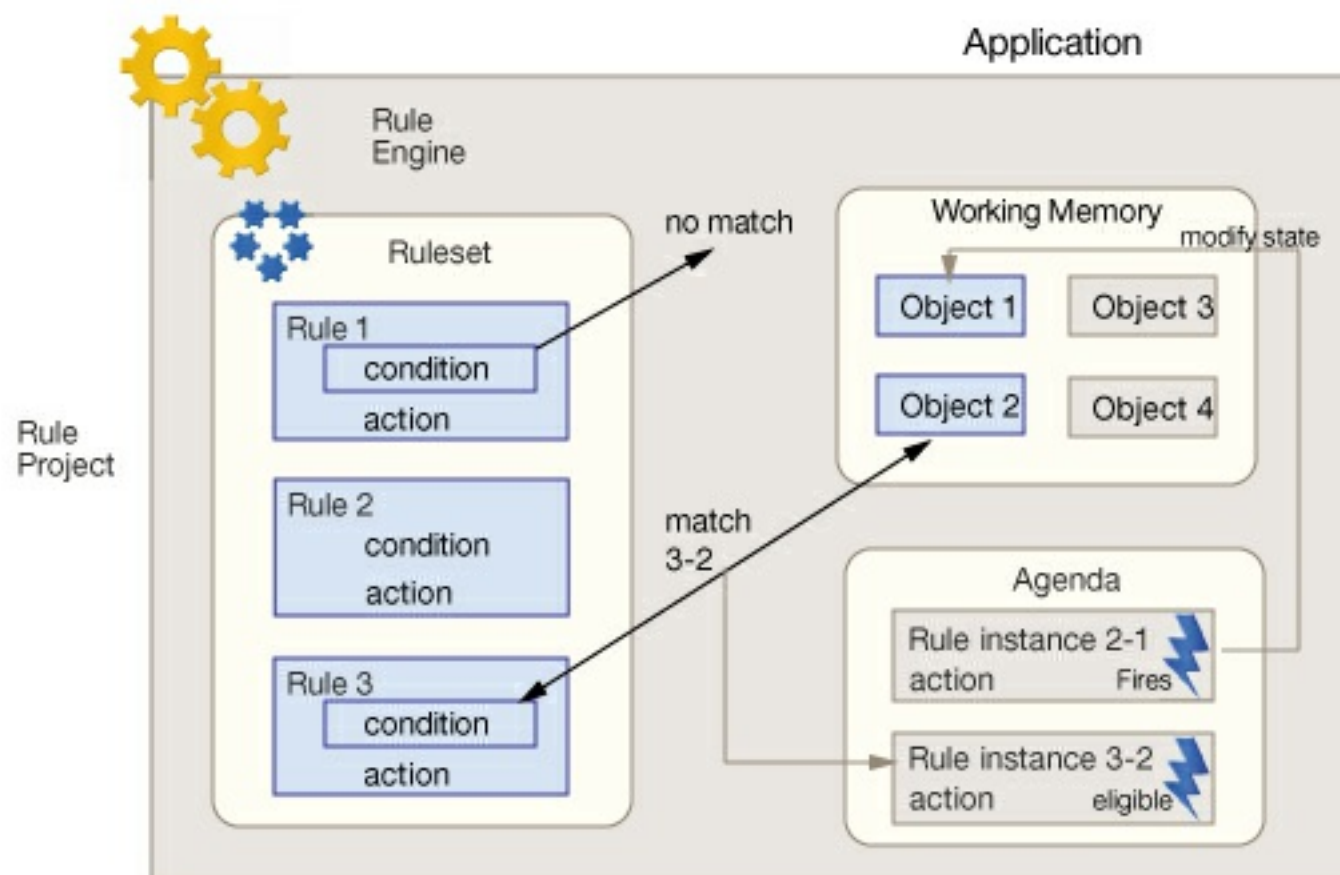


For each match between the conditions of the rules and an object, a rule instance is put into the rule execution agenda.



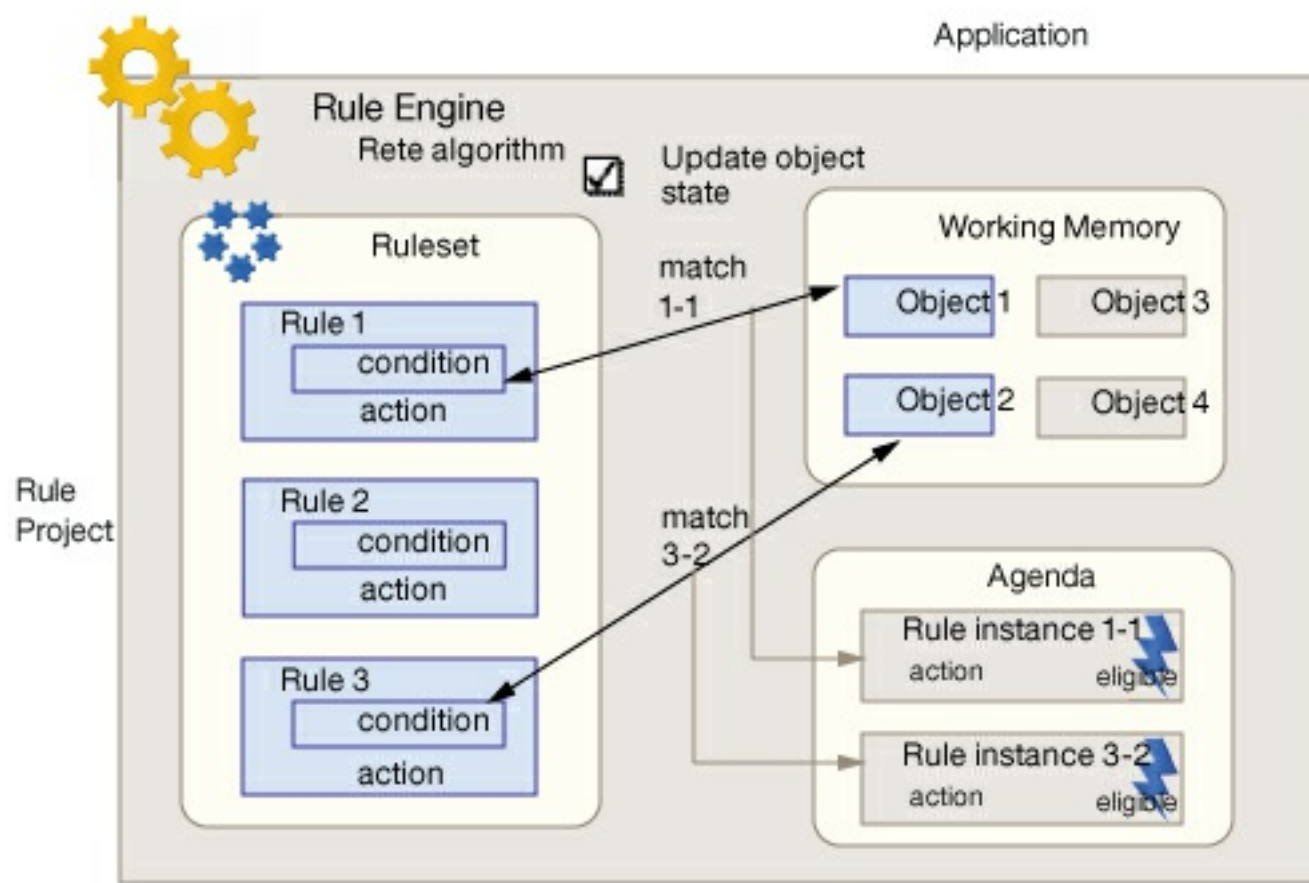


The first rule in the agenda is then executed.



The action of this rule sometimes modifies the state of an object in working memory. When the **Update object state** box is not selected in the BOM Editor for the method that is executed in the rule actions, the rule engine is not notified of this change in object state. If you select the **Update object state** box, the rule engine is notified of the change and rules are matched against the new state of the object, which might result in new rule instances being added to the agenda.





**Parent topic:** [Updating object states in the rule engine](#)

## Notifying the rule engine of object updates

When operating in RetePlus execution mode, you can specify which methods to use to notify the rule engine of object updates.

### About this task

If you want the rule engine to be notified when a method changes the state of an object, ruleset parameter, or ruleset variable when rules are executed with the RetePlus execution mode, you can specify which methods to use to activate a notification.

### Procedure

To specify the methods that can activate updates:

1. In the Outline view, click the method you want to specify.
2. On the Member page of the BOM Editor, in the General Information section, select the **Update object state** check box.
3. Save the BOM entry.

Now when you execute rules with the RetePlus execution mode, the rule engine is notified of changes made by this method on ruleset parameters, ruleset variables, or objects.

**Parent topic:** [Updating object states in the rule engine](#)


### Related information:

[Object state update](#)

# Ordering rules in a ruleset

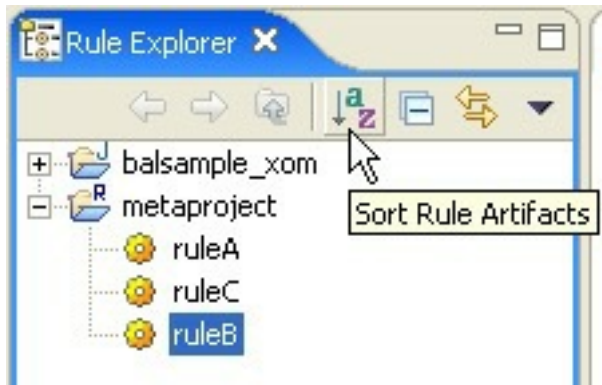
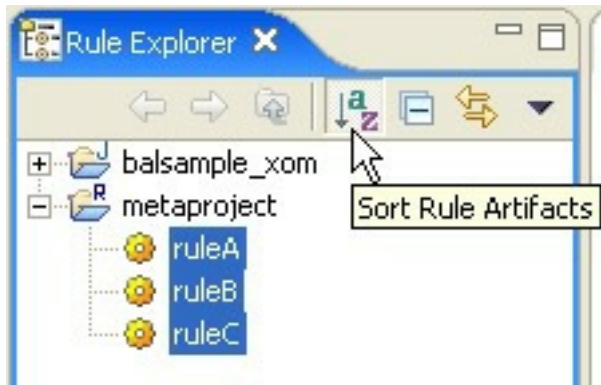
The order in which rules are presented in the ruleset affect the order in which they are placed in the ruleset archive during the final packaging of the ruleset for execution. You can sort rules alphabetically, or manually.

## Order rules alphabetically

In the rule project, you can sort rule artifacts alphabetically. Select your rule project in the Rule Explorer, and click  **Sort Rule Artifacts**.

You can display the rules in a project alphabetically without changing the order in which the rules run.

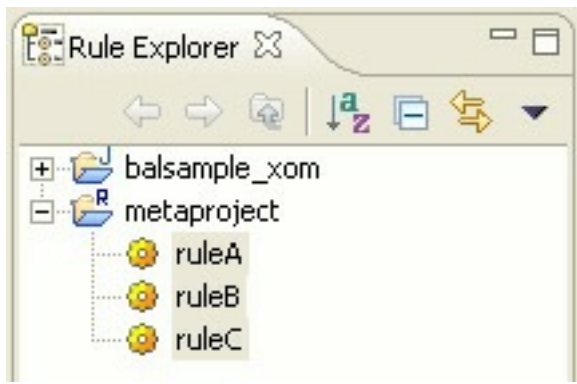
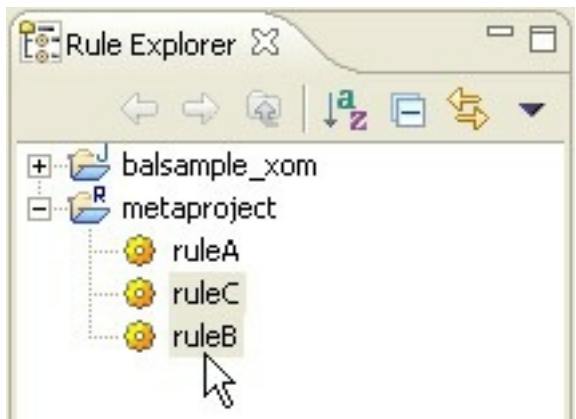
**Note:** The button remains active until you click it again.

| BEFORE: Rules in random order                                                      | AFTER: Rules in alphabetic order                                                    |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |

## Order rules manually

One of the ways of determining the initial rule order is to order them manually in the Rule Explorer. Drag each rule you want to move, and drop them over the new location.

**Note:** The **Sort Rule Artifacts** icon must be deactivated before you can use the drag-and-drop method. If it is activated, click it again to deactivate it.


| BEFORE: Rules not in required order                                                 | AFTER: ruleB moved to required position                                              |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
|  |  |

Parent topic: [Configuring rule execution](#)

## Setting simple priorities among rules

Rules have priorities, which can be static or dynamic. Use a constant to define a static priority, or an expression containing a ruleset variable to define a dynamic priority.

To define the order in which rules are executed, you set the priority of a rule in the **Properties** view of the relevant rule artifact, with the **priority** property.

**Note:** To be able to edit the property, make sure that the “hand” icon  is visible next to the **Value** column header in the **Properties** view.

### Static priority

Use a static priority to change the sequence of rule execution among rules. Static priorities are integers, the relative values of which determine the priorities among the rules. You can also use a static priority to change the order of execution between several instances of the same rule when they are eligible for execution at the same time.

You define static priorities using a constant. In the **Value** field of the **priority** property, you type a number that represents the priority. The number can be any Java™ integer value between  $-10^9$  and  $+10^9$ . The larger the number, the higher the execution priority of the rule.

### Dynamic priority

A dynamic priority is an expression whose value depends on ruleset variables bound in the condition part of a rule.

**Note:** An error message is displayed when a ruleset archive includes a ruletask in sequential mode that selects rules with a dynamic rule priority.

In the **Value** field of the **priority** property, you enter an expression that uses a ruleset variable. The expression can use any variables defined in the condition part of the rule provided that its scope is for the entire ruleset. If the expression returns a number that is not an integer, it is converted to an integer following the Java language specification.

Here we have two examples:

- `prior1 + ?p`
- `-?a`

**Parent topic:** [Configuring rule execution](#)

# Rule overriding

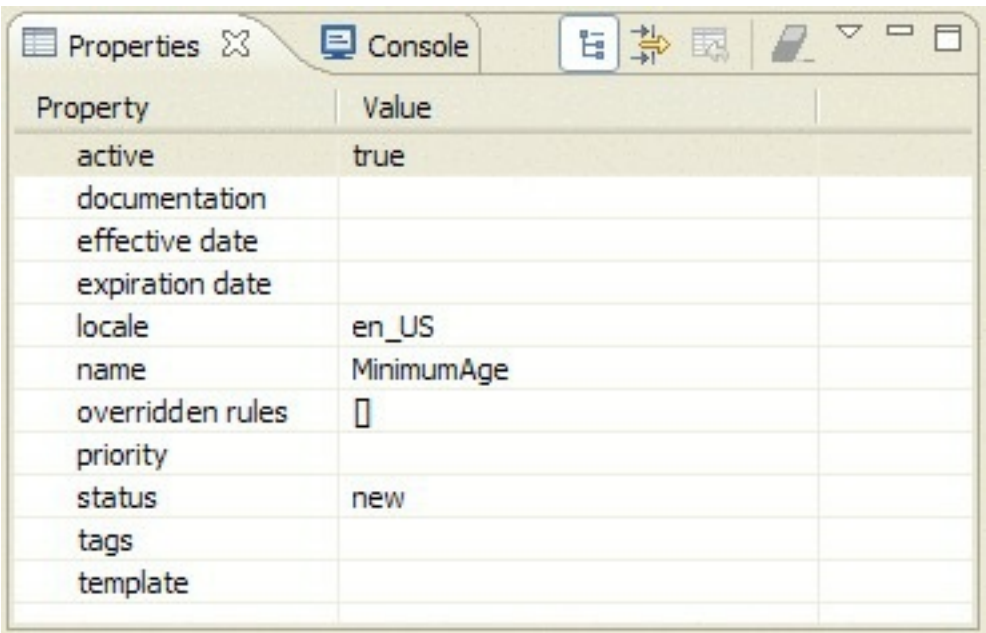
Rule overriding is the last selection mechanism after ordering and priorities. Overridden rules are not selected for execution. You can combine rule overriding with the hierarchical property.


Rule overriding is the final stage of rule selection. After all other selection mechanisms, you can specify that certain rules override other rules. Overridden rules are filtered out and not selected for execution.

A rule can override one or more other rules if these rules are selected in the same rule task at run time. Rule overriding occurs at the individual rule, decision table, or decision tree level. You can set an entire business decision table to override another decision table or tree, for example.

You create a rule override in the **Properties** view of the relevant artifact, with the **overridden rules** property. Click the . . . button next to the property name, and click **Add** to add rules to be overridden by the current artifact. In the **Values for overridden rules** dialog, enter one of the following attributes:

- The name of the rule that you want this artifact to override
- The name of each rule that you want this artifact to override, separated by commas; or
- Select from a list by clicking the . . . button.

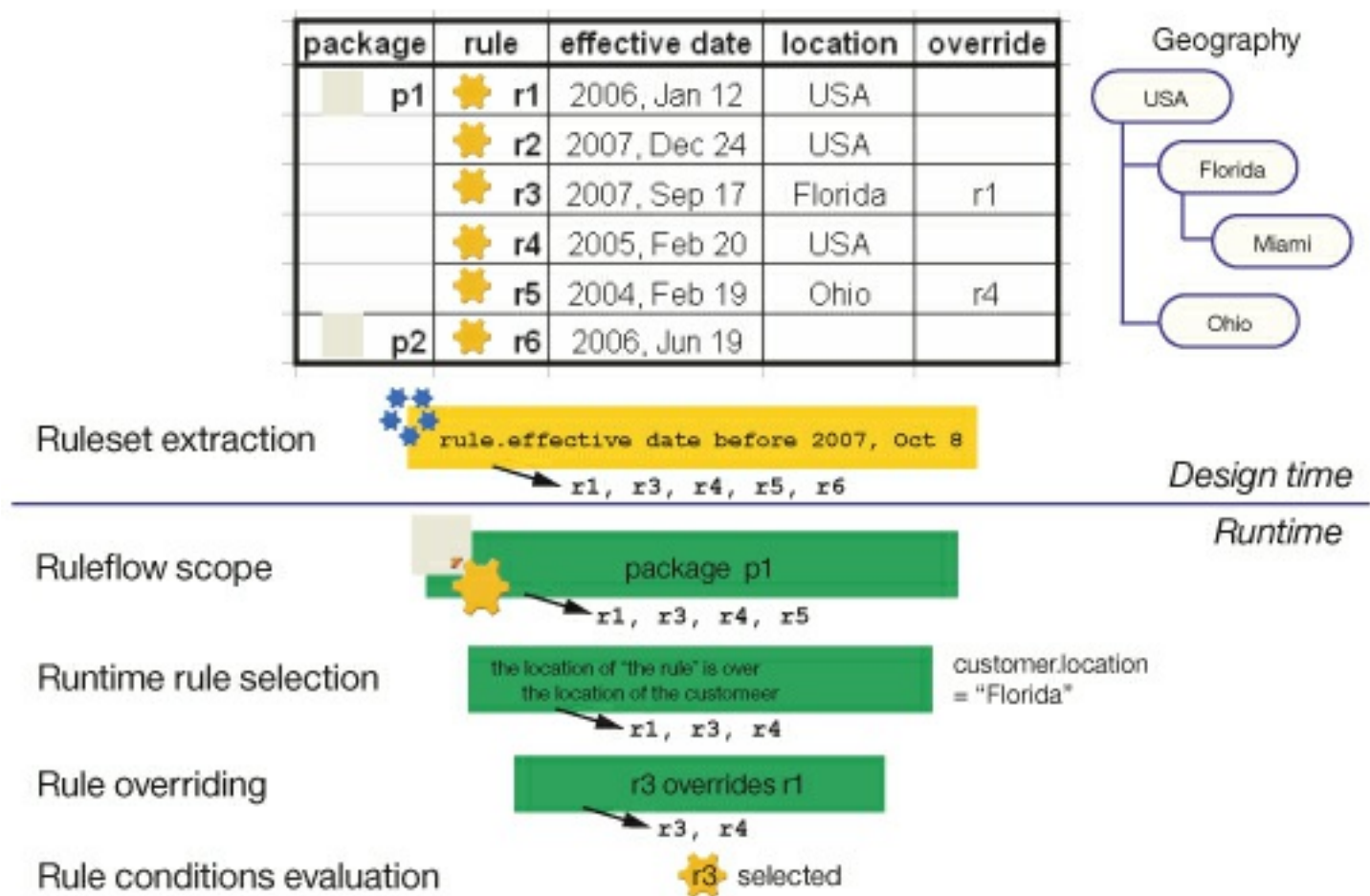


**Note:** To be able to edit the property, make sure that the “hand” icon  is visible next to the **Value** column header in the Properties view.

## Combine rule overriding with the hierarchical property

You can also combine rule overriding with the hierarchical property, enabling specific rules to override more general rules. You specify overriding in Rule Designer by setting the overriddenRules rule property. Using the overriddenRules property, you can select which rule artifacts are to be overridden and specify the overriding relationship between a rule and other rules.

The following diagram shows how rule overriding works in a hierarchical rule structure.



Selection operates as follows:

1. The first selection is based on the effective date property. This drops rule r2 from the selection, leaving r1, r3, r4, r5, and r6.
2. At run time, only the first package is selected (for example, p1 has been added to the rule task, but p2 has not). As a result, r6 is filtered out of the selection.
3. Runtime rule selection is set to the location of the rule **is over** 'Florida', so only USA and Florida rules are considered for execution (see the Geography hierarchy tree in the diagram). As a result, r5 is filtered out of the selection.
4. Finally, r3 overrides r1. Therefore, only the conditions of r3 and r4 are left to be evaluated. The rules whose actions match the required conditions are executed.

**Parent topic:** [Configuring rule execution](#)

**Related reference:**

[firing](#)  
[firinglimit](#)

**Related information:**

[Working with ruleflows](#)  
[Setting simple priorities among rules](#)  
[Ordering rules in a ruleset](#)



## Defining the ruleset content and signature

You define the content of a ruleset and its signature.

A ruleset is an executable package that includes rule artifacts and other elements. It contains a set of rules that can be executed by the rule engine. You must define the ruleset content and the parameters that allow the client application to exchange information with the ruleset.

In a decision service, the decision operation includes all the settings needed to define the contents of the ruleset and its parameters.

### Ruleset content

The ruleset content is defined by specifying the following information:

- The source rule project. All the rules and variables contained in this project and any of its dependent projects become eligible to be included in the ruleset.
- You can specify any ruleflow to include in the ruleset.
- Any queries or validators to filter the rules, so that only a subset of the rules of the source rule project and its dependent projects are included in the ruleset. Typically, this is used to include rules from specific packages of a project or according to a rule property such as status.

### Ruleset signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation for a decision service. The ruleset signature is stored in a ruleset archive as an XML file.

The parameters of a ruleset can have three directions:

- IN: The parameter value is provided as input to the ruleset on execution.
- OUT: The parameter value is set by the execution of the ruleset and provided as output from the ruleset at execution completion.
- IN\_OUT: The parameter value is provided as input to the ruleset on execution and its value can be modified by the ruleset and provided as output at execution completion.

In a decision service, you create the parameters from any of the ruleset variables that are available in the rule projects that are part of the scope of the decision operation. Ruleset variables are internal to a ruleset and provide a way to exchange data between rules, functions, and tasks.

**Note:**

Only ruleset variables that are defined at the top-level package are eligible to be used as ruleset parameters.

**Parent topic:** [Developing rulesets in Rule Designer](#)

**Related concepts:**

[Setting up a project hierarchy](#)

[Executing rulesets in the decision engine](#)

**Related tasks:**

[Running and debugging decision operations](#)

# Designing projects for rule authoring

You use projects to design the data model for rules, and define a vocabulary fitted for business users. You can also author different types of business rules.

## [Designing business object models](#)

The way you design a business object model (BOM) depends on the nature of the data you use as its basis. You can design a BOM for a Java™ model or for an XML model.

## [Configuring the BOM for rule authoring](#)

After designing a BOM for an underlying object model, you configure it to maximize the efficiency of rule authoring activities.

## [Authoring business rules](#)

You use Rule Designer to create and edit different types of rules. You can use automatic variables, ruleset variables, templates, and categories to simplify the rule creation process.

## [Reviewing a rule project](#)

Manage rule project items and run queries and reports using the features provided in Rule Designer.



# Designing business object models

The way you design a business object model (BOM) depends on the nature of the data you use as its basis. You can design a BOM for a Java™ model or for an XML model.

## Overview: BOM and execution object model (XOM)

The XOM is the model used to execute the rules.

## Creating BOM entries

BOM entries are files that describe a part of a Business Object Model (BOM).

## Designing a BOM for a Java model

You design a BOM for a Java model by building a XOM from compiled Java classes or from an XML schema, mapping BOM elements to XOM elements, and configuring BOM updates.

## Designing a BOM for an XML model

You can use data other than Java as the basis for your XOM.

## Updating the BOM when the XOM changes

How you manage XOM changes in the BOM depends on whether your rule project references a Java project as a XOM.

**Parent topic:** [Designing projects for rule authoring](#)

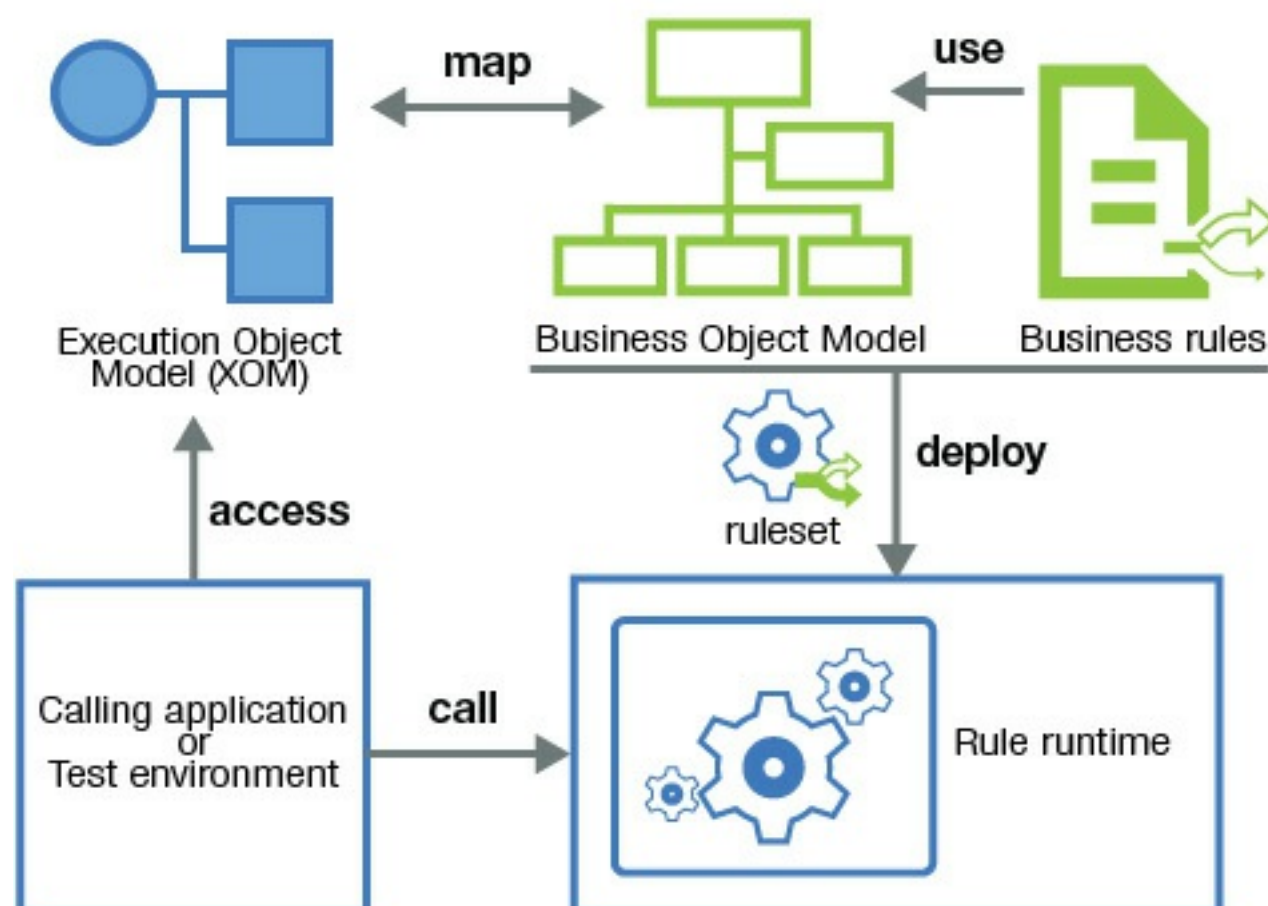
## Overview: BOM and execution object model (XOM)

The XOM is the model used to execute the rules.

You can build the XOM from different data sources.

The execution object model (XOM) is the model against which you run rules. It references the application objects and data, and is the base implementation of the business object model (BOM). Rule projects reference the XOM.

Through the XOM, the rule engine can access application objects and methods, which can be Java™ objects, XML data, or data from other sources. At run time, rules that were written against the BOM are run against the XOM.



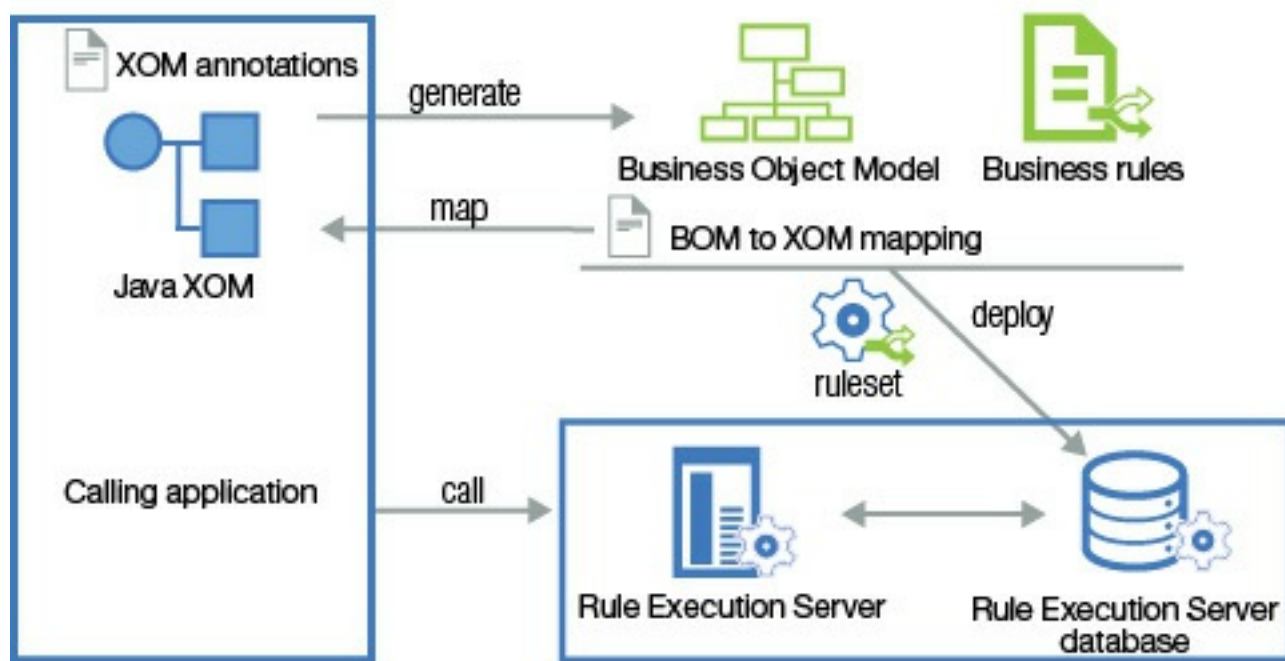
Every BOM element (business element) must have a corresponding XOM element (execution element). The correspondence between execution elements and business elements does not need to be one-to-one. If a business element originates from an execution element, you do not need to specify an explicit mapping. If a business element does not originate from an execution element, you must specify a BOM-to-XOM mapping.

Some internal rules exist for the implicit BOM-to-XOM mapping when you generate a BOM from a XOM. For example, a BOM class is implicitly mapped to a XOM class of the same name.

### Designing a BOM for a Java model

When your data model is in Java, you can generate a BOM directly from this Java Execution Object Model (XOM), see [Mapping of a BOM created from a Java XOM](#). You can use XOM annotations to configure BOM generation, see [XOM annotations](#). If you want to extend the BOM with business classes and methods, you can then map these business elements to XOM elements using BOM-to-XOM mapping in the BOM Editor. You have two means of specifying BOM-to-XOM mapping: extender mapping or IRL mapping, see [Rule language and extender mapping](#). Business users can then author business rules without having to worry about the underlying data types and platform.

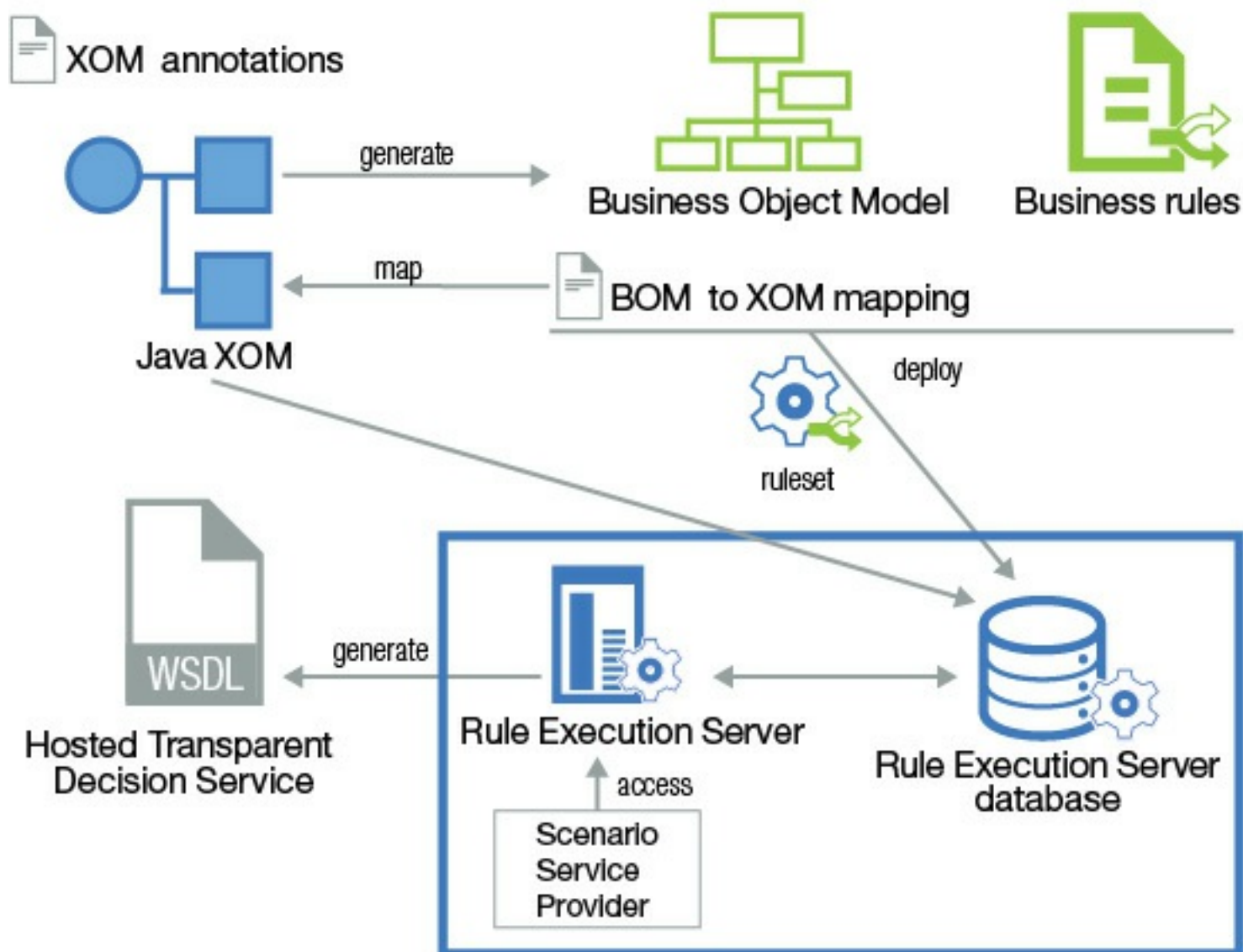
At deployment time, if your application is a Java or web application, then the Java XOM is packaged into the application and there are no specific deployment steps you need to take for the XOM. You package the rules and BOM-to-XOM mapping definition into a ruleset archive, which is then available to your calling application through the rule runtime, Rule Execution Server.



If you do not have a Java or web calling application, either because you want to deploy to a testing server for testing and simulation, or because you want to expose your rules as a Hosted Transparent Decision Service (HTDS), then you must deploy the Java XOM to the Rule Execution Server database.

When you test and simulate rule execution, the Scenario Service Provider (SSP) accesses Rule Execution Server for execution, and Rule Execution Server retrieves the XOM from the database.

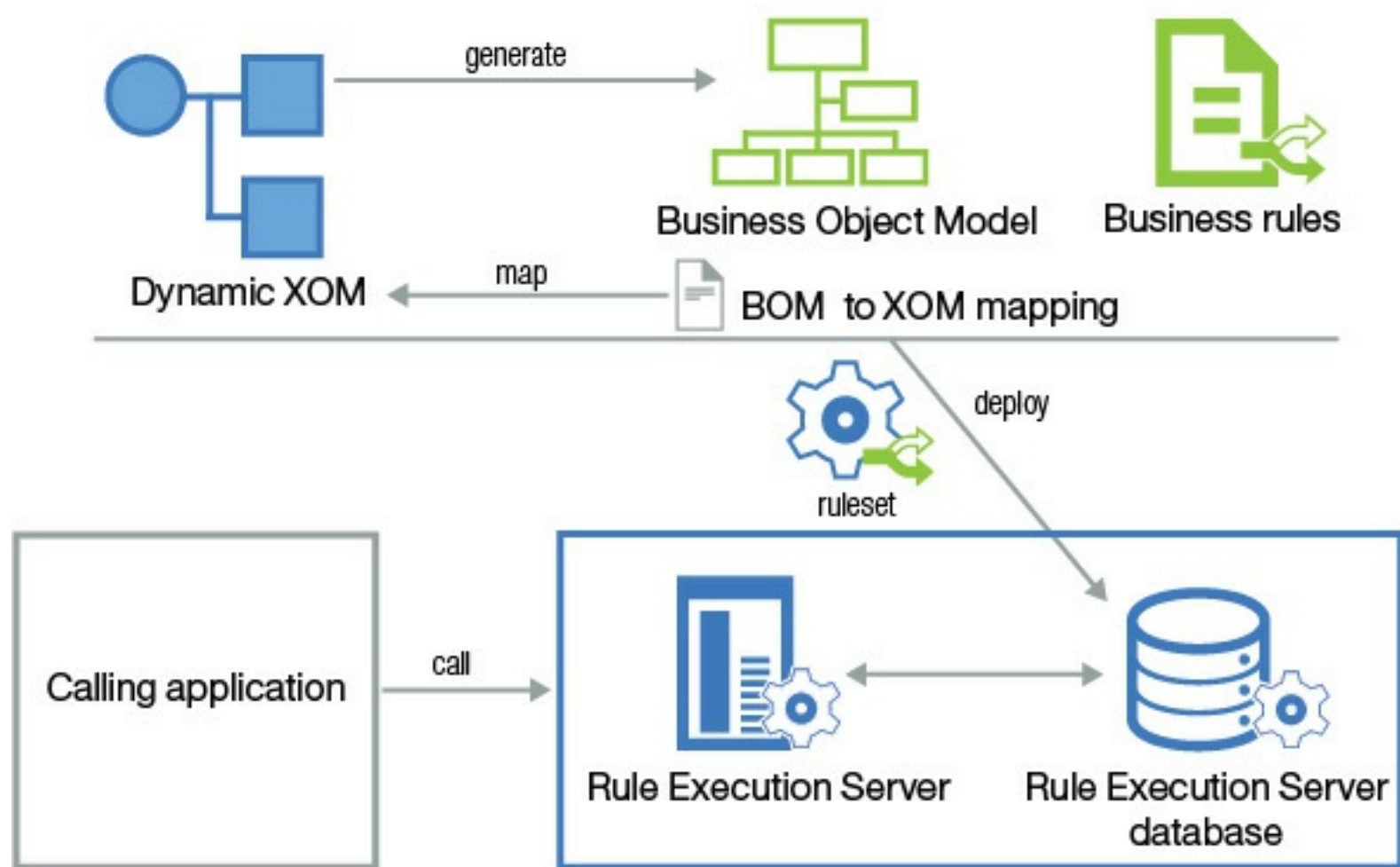
When you generate a HTDS, Rule Execution Server retrieves the XOM from the database in the same way (see [Downloading a HTDS WSDL file](#)).



## Designing a BOM for an XML model

When your data model is in XML, you can generate a BOM directly from this Dynamic XOM, see [Overview: Dynamic execution object model \(XOM\)](#) and [Built-in simple types](#). If you want to extend the BOM with business classes and methods, you can then map these business elements to XOM elements using BOM-to-XOM mapping in the BOM Editor. You specify this mapping with IRL mapping, see [Rule language and extender mapping](#). Business users can then author business rules without having to worry about the underlying data types and platform.

At deployment time, the Dynamic XOM and BOM-to-XOM mapping files are packaged with the rules in the ruleset archive, which is then available to your calling application through the rule runtime, Rule Execution Server.



**Parent topic:** [Designing business object models](#)

**Related information:**

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

[Defining BOM-to-XOM mapping](#)

## Creating BOM entries

BOM entries are files that describe a part of a Business Object Model (BOM).

### [Creating a BOM without a XOM](#)

You can work with BOM entries to create business elements and map them to execution elements in the XOM, or reference BOM entry files in other rule projects.

### [Creating a BOM entry from a XOM](#)

If you create a BOM entry from a XOM, you can subsequently extend the BOM without modifying the XOM.

### [Specifying the order of BOM entries](#)

If you have an element of the same name in more than one BOM entry, you can control which element to use.

**Parent topic:** [Designing business object models](#)

## Creating a BOM without a XOM

You can work with BOM entries to create business elements and map them to execution elements in the XOM, or reference BOM entry files in other rule projects.

### [Creating an empty BOM entry](#)

If you want to create business elements without having to consider how the execution elements are implemented, you can create an empty BOM entry that you then map to the XOM.

### [Disabling BOM to XOM mapping checks](#)

To stop the display of build warnings when defining initial BOM entries, you disable BOM to XOM mapping checks.

### [Associating an empty BOM entry with a XOM](#)

You associate an empty BOM entry with a XOM by setting a reference from the BOM entry to the XOM.

**Parent topic:** [Creating BOM entries](#)

# Creating an empty BOM entry

If you want to create business elements without having to consider how the execution elements are implemented, you can create an empty BOM entry that you then map to the XOM.

## About this task

You can create business elements without considering how the execution elements are implemented. In this case, you create an empty BOM entry that you later map to the Execution Object Model (XOM). You can also extend the BOM without modifying the original XOM.

## Procedure

To create an empty BOM entry:

1. In the Rule Explorer view, select the bom folder and then, on the **File** menu, click **New > BOM Entry**.

The New BOM Entry wizard opens.

2. In the **Name** field, type a name for the BOM entry and then select the option **Create an empty BOM entry**.
3. Click **Finish**.

In the Explorer view, the bom folder displays a new BOM entry.

## Results

You can now add business elements to this BOM entry. Later, you can map them to an execution object model (XOM).

**Parent topic:** [Creating a BOM without a XOM](#)

### Related tasks:

[Creating a BOM entry from a XOM](#)

[Disabling BOM to XOM mapping checks](#)

[Associating an empty BOM entry with a XOM](#)

[Specifying the order of BOM entries](#)

### Related information:

[Business object model \(BOM\)](#)



## Disabling BOM to XOM mapping checks

To stop the display of build warnings when defining initial BOM entries, you disable BOM to XOM mapping checks.

### About this task

When you work on an empty BOM entry, the business elements you create are not yet mapped to the XOM and so warnings are generated when you build. To suppress these warnings, you can disable checks on the BOM to XOM mapping.

### Procedure

To disable BOM to XOM mapping checks:

1. On the **Window** menu, click **Preferences**. (On Mac, click **Preferences** on the Eclipse menu.)
2. In the Preferences dialog, expand **Rule Designer**, and then click **Build**.
3. On the Build page, clear the check box **Perform BOM to XOM checks during build**.
4. Click **Apply**, and then click **Apply and Close**.

BOM to XOM warnings are now disabled at build time.

**Parent topic:** [Creating a BOM without a XOM](#)



## Associating an empty BOM entry with a XOM

You associate an empty BOM entry with a XOM by setting a reference from the BOM entry to the XOM.

### About this task

If you define an empty BOM entry and then want to associate it with a XOM, you set the origin of the BOM entry as the path to the XOM.

The BOM origin is a reference from a BOM entry to a XOM. Rule Designer uses this reference to update the BOM entry to:

- Import some additional classes from the XOM into the BOM entry
- Update some BOM classes from a modified version of the XOM

The BOM origin can refer to a JAR, a Java™ project, an XSD, a class folder, or a rule project. If it refers to a rule project, it refers to the XOM of the current rule project.

### Procedure

To associate an empty BOM entry with a XOM:

1. Right-click your rule project and then click **Properties**.
2. In the Properties dialog, click **Business Object Model**.
3. In the **Required BOM** entries field, expand the BOM entry and then double-click the item **Origin:none**.
4. In the BOM Entry Origin dialog, in the **Origin URL** field, type the path to the XOM.

Use this format: `xom:/myRuleProject/myXOM`

5. Click **OK**.

The empty BOM entry is now associated with a XOM.

**Parent topic:** [Creating a BOM without a XOM](#)

# Creating a BOM entry from a XOM

If you create a BOM entry from a XOM, you can subsequently extend the BOM without modifying the XOM.

## About this task

You can create BOM entries from an existing XOM. You can later extend the BOM without modifying the original XOM.

## Procedure

To create a BOM entry from a XOM:

1. In the Rule Explorer view, select the bom folder and then, on the **File** menu, click **New > BOM Entry**.
2. In the New BOM Entry wizard, in the Name field, type a name for the BOM entry, and select the option **Create a BOM entry from a XOM**.
3. Click **Next**.

In the BOM Entry step, you select the XOM entries that you have defined in your rule project to be used as a basis for the BOM entry. These XOM entries can be in your current rule project or in a referenced rule project.

### Important:

You can also select the Java Runtime Environment (JRE) or Java Development Kit (JDK) system libraries as a XOM. However, it can make the BOM entry creation fail because some classes cannot be loaded directly. You can click **Browse XOM** again to reload the XOM types. The second time, the failing classes are ignored.

4. In the **Choose a XOM entry** field, type a path with the format `xom:/myRuleProject/myXOM`, or click **Browse XOM** to select a XOM entry.

The classes in the XOM entry you selected are now shown in the **Select classes** field.

5. In the **Select classes** field, select the classes that you want to have in the BOM entry.

Any classes that you do not select are not available from any of the rule project items. When you select a package or a class, all its members are automatically included in the selection. You can redefine the list of selected classes later on.

**Important:** BOM types that are derived from XML simple types are filtered out because they do not have a binding at run time. To prevent runtime errors that are caused by a BOM created in a previous release, do any of the following:

- Manually delete BOM types that are derived from XML simple types.
- Regenerate your BOM to ensure that the XML simple types are no longer generated into the BOM.

6. Click **Next**.

In the BOM Verbalization step, you define which BOM elements you want to be verbalized. BOM elements that you do not verbalize are not available in business rule artifacts, but you can still use them in technical rule artifacts. You can redefine the elements to be verbalized later on.

7. Click **Finish**.

In the Rule Explorer view, the bom folder displays a new BOM entry.

## Results

You can now edit the verbalization of business elements in the BOM Editor, and use them in your rule artifacts.

**Parent topic:** [Creating BOM entries](#)

### Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

### Related tasks:

[Disabling BOM to XOM mapping checks](#)

[Associating an empty BOM entry with a XOM](#)

[Specifying the order of BOM entries](#)

[Creating an empty BOM entry](#)

### Related information:

[Defining BOM-to-XOM mapping](#)

[Business object model \(BOM\)](#)

## Specifying the order of BOM entries

If you have an element of the same name in more than one BOM entry, you can control which element to use.

### About this task

The business object model (BOM) defines a path to a set of BOM entries. These BOM entries can be in the current rule project or in a referenced rule project.

If you have several BOM entries, an element in the first BOM entry in the path overrides any other element with the same name in the other BOM entries. You can modify the order of the BOM entries in the path to control which element is selected at run time.

### Procedure

To modify the order of BOM entries:

1. In the Rule Explorer view, select the rule project and, on the **Project** menu, click **Properties**.
2. In the side pane of the Rule Project Properties dialog, click **Business Object Model** to display the list of BOM entries in the business object model.
3. Click the **Order** tab.
4. On the Order page, select a BOM entry and then use the **Up** and **Down** buttons to change the order in which they are listed.
5. Click **Apply and Close**.

The order in which the BOM entries are found in the business object model path is now changed.

**Parent topic:** [Creating BOM entries](#)

## Designing a BOM for a Java model

You design a BOM for a Java™ model by building a XOM from compiled Java classes or from an XML schema, mapping BOM elements to XOM elements, and configuring BOM updates.

### Defining a Java XOM for a rule project

You can define a Java XOM for a rule project, add resources and references, and order Java XOM entries. You can also deploy a Java XOM using the Java Build Path.

### Defining BOM-to-XOM mapping

You can map different business elements to the XOM. These elements include business classes, attributes, method calls, and constructor calls. You can also implement synthetic objects, and access ruleset variables, parameters, and functions from rule language mapping code.

### Mapping of a BOM created from a Java XOM

When you create a BOM from the XOM, Rule Designer processes the Java compiled classes into business elements.

### Adding annotations to the XOM

You can add annotations to the XOM to customize the way that the BOM is created from Java classes.

**Parent topic:** [Designing business object models](#)

## Defining a Java XOM for a rule project

You can define a Java™ XOM for a rule project, add resources and references, and order Java XOM entries. You can also deploy a Java XOM using the Java Build Path.

### Defining a Java XOM

You can specify the Java XOM when creating the rule project, or later using the rule project Properties dialog.

### Adding a JAR file or class folder to a Java XOM

You can add a JAR file or a class folder to a Java XOM.

### Setting the source location of a library

You must specify the location of the source files for a library.

### Defining the order of Java XOM entries

You can specify the order in which Java XOM entries are listed.

**Parent topic:** [Designing a BOM for a Java model](#)

## Defining a Java XOM

You can specify the Java™ XOM when creating the rule project, or later using the rule project Properties dialog.

### Procedure

To define a Java XOM using the Properties dialog:

1. In the Rule Explorer view, select the rule project and then on the **Project** menu, click **Properties**.  
You can also right-click the project name and select **Properties** from the contextual menu.
2. Click **Java Execution Object Model** in the list of properties.
3. From the list **Required Java projects**, select each Java project that you want for the XOM by clicking the check box next to it.  
Click the **Select All** button if all the Java projects are necessary.
4. Click **Apply and Close**.

### Results

The Java XOM is now defined. Using the other tabs of the same dialog, you can also add a JAR file or class folder, and define the order of Java XOM entries.

**Parent topic:** [Defining a Java XOM for a rule project](#)

### Related tasks:

[Adding a JAR file or class folder to a Java XOM](#)

[Setting the source location of a library](#)

[Defining the order of Java XOM entries](#)

# Adding a JAR file or class folder to a Java XOM

You can add a JAR file or a class folder to a Java™ XOM.

## Procedure

To add a JAR file or a class folder:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.

You can also right-click the project name and select **Properties** from the contextual menu.

2. Click **Java Execution Object Model** in the list of properties.
3. Click the **Libraries** tab to display the Libraries page so that you can select the required JAR files and class folders.
  - To add JAR files that have been imported in the Eclipse workspace, click **Add JARs**. A dialog opens for you to browse JAR files in the workspace.
  - To add JAR files that are not in the workspace, click **Add External JARs**. A dialog opens for you to browse the JAR files in your file system.
  - To add class folders, click **Add Class Folder**. A dialog opens for you to browse directories in the workspace. In this same dialog, you can add external class folders by creating a folder in the workspace that links to an external folder in the file system.

### Note:

By default, the Java Runtime Environment (JRE) library is already in the list of libraries. This library is the same as the one used to run Eclipse. You cannot remove it. You can browse the JRE library to see the JAR files used by this library.

4. Click **Apply and Close**.

## Results

You must now specify the location of the source to be used to reference a library.

**Parent topic:** [Defining a Java XOM for a rule project](#)

### Related tasks:

[Defining a Java XOM](#)

[Setting the source location of a library](#)

[Defining the order of Java XOM entries](#)

## Setting the source location of a library

You must specify the location of the source files for a library.

### Procedure

To set a source location for a library:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.  
You can also right-click the project name and select **Properties** from the contextual menu.
2. Click **Java Execution Object Model** in the list of properties.
3. On the Libraries tab, expand the item that you want to edit and then select the **Source** item.
4. Click **Edit**.

A dialog opens for you to select a .zip file or a source folder from the workspace or the file system.

**Parent topic:** [Defining a Java XOM for a rule project](#)

### Related tasks:

[Defining a Java XOM](#)

[Adding a JAR file or class folder to a Java XOM](#)

[Defining the order of Java XOM entries](#)



## Defining the order of Java XOM entries

You can specify the order in which Java™ XOM entries are listed.

### Procedure

To order Java XOM entries:

1. In the Rule Explorer view, select the rule project, and then on the **Project** menu, click **Properties**.

You can also right-click the project name and select **Properties** from the contextual menu.

2. Click **Java Execution Object Model** in the list of properties.
3. To display a list of all selected Java XOM entries, click the **Order and Export** tab.

Using this tab, you can specify what libraries you want to export when the XOM is deployed to Rule Execution Server.

#### Note:

All items are checked by default except the JDK.

4. Select one or more entries and then use the **Up** and **Down** buttons to move them relative to each other.

The classes are searched for in the order in which they are shown in this list,

5. Click **Apply and Close**.

### Results

If you want to deploy a Java XOM that has Java class dependencies, use the export function of the Java project used to build the BOM.

**Parent topic:** [Defining a Java XOM for a rule project](#)

#### Related tasks:

[Defining a Java XOM](#)

[Adding a JAR file or class folder to a Java XOM](#)

## Defining BOM-to-XOM mapping

You can map different business elements to the XOM. These elements include business classes, attributes, method calls, and constructor calls. You can also implement synthetic objects, and access ruleset variables, parameters, and functions from rule language mapping code.

### [Rule language and extender mapping](#)

Rule artifacts that are created from the BOM must be mapped to the XOM to run at run time. You can use the default mapping, or mappings based on rule language functions or Java extender classes.

### [Execution class mapping](#)

You can map a business class to an execution class. By default, the elements of the business class are mapped to the elements of the execution class. Conversely, the BOM-to-XOM mapping also helps to deduce the business class from a particular class instance, such as for testing and simulation.

### [Rule language mapping](#)

You can use the rule language mapping code to define the BOM-to-XOM mapping.

### [Extender mapping](#)

Extender mapping is designed to use Java rather than rule language. In a Java extender class, you create static elements that have the same name as business class elements.

**Parent topic:** [Designing a BOM for a Java model](#)

## Rule language and extender mapping

Rule artifacts that are created from the BOM must be mapped to the XOM to run at run time. You can use the default mapping, or mappings based on rule language functions or Java™ extender classes.

The BOM-to-XOM mapping mechanism can translate BOM-based rule artifacts into XOM-based rule artifacts at run time. The mechanism provides a default mapping, and supports two kinds of explicit mapping:

- Rule language mapping: You associate a business element with rule language code that is based on the XOM. You can call functions, ruleset parameters and variables, and rule instances.
- Extender mapping: In a Java extender class, you create static elements that have the same name as business class members. You can use extender mapping only on a Java XOM. If you have a dynamic XOM, use rule language mapping.

|                                                                                                         |
|---------------------------------------------------------------------------------------------------------|
| <b>Important:</b> For rule language mapping, write the mapping in ARL, which is stored in a .b2xa file. |
|---------------------------------------------------------------------------------------------------------|

The BOM-to-XOM mapping is defined in a BOM-to-XOM mapping XML schema.

At run time, the rule engine uses the following order to find the right mapping:

1. Explicit mapping in rule language.
2. Explicit mapping with extender class.
3. Implicit mapping, by default when nothing is specified.

If the rule engine does not find a mapping by default, it gives you an error.

### Unexpected results for overriding methods

The BAL object operator `is one of` does not use BOM-to-XOM redefined equals. If you apply BOM-to-XOM mapping to a method that overrides another method, you might receive an unexpected result. If there is a direct call to the overriding method, the BOM-to-XOM mapping is applied. However, in an indirect call, such as a call to the method in a superclass or interface, the BOM-to-XOM mapping is not applied to the overriding method.

For example, if you override the method `Object.equals(Object)` in a BOM class and provide a BOM-to-XOM implementation, the BOM-to-XOM implementation is called when an instance of such a class is inside a collection, such as when `HashSet.add(Object)` or `ArrayList.contains(Object)` is called.

**Parent topic:** [Defining BOM-to-XOM mapping](#)

#### Related concepts:

[Consistency checking](#)

[Overview: BOM and execution object model \(XOM\)](#)

#### Related information:

[Business object model \(BOM\)](#)

# Execution class mapping

You can map a business class to an execution class. By default, the elements of the business class are mapped to the elements of the execution class. Conversely, the BOM-to-XOM mapping also helps to deduce the business class from a particular class instance, such as for testing and simulation.

To map a business class to an execution class, you must specify the name of the execution class in the BOM Editor.

You define the execution class in the Execution name field of the **BOM to XOM Mapping** section of the BOM Editor. For complex mapping, you can use a tester to test the instances of the execution class by using rule language mapping.

When you run the rules, the execution class is used instead of the business class whenever needed.

For best results, apply the following rules to the mapping:

- When the business class is a utility class, map it to void.
- When the business class is an enumerated class, do not provide a tester.
- When you do provide a tester, write it carefully so that it filters out all non-matching class instances, by returning false.

For example, you can map the business class RichCustomer to an execution class Customer: In the BOM Editor, in the BOM to XOM Mapping section, type a rule language statement that returns a Boolean value in the Tester field. Use this to represent the current object. For example:

```
return this.money > 100000;
```

If members of the business class are not explicitly mapped, the BOM-to-XOM mechanism assumes that they are the same as the members of the execution class. The following table describes how the BOM-to-XOM mechanism uses these members:

Table 1. Implicit mapping to members of an execution class

| Member of business class<br>BusinessClass                   | Case                                                                 | Mapping to execution<br>member of class<br>ExecutionClass |
|-------------------------------------------------------------|----------------------------------------------------------------------|-----------------------------------------------------------|
| Constructor<br>BusinessClass(MyBizClass B, MyBizClassC)     | Call by new                                                          | Constructor<br>ExecutionClass(MyClass B, MyClassC)        |
| Attribute MyBizClassA attr                                  | Assignment                                                           | Attribute attr (not read-only)                            |
|                                                             | Access                                                               | Attribute attr (not write-only)                           |
| Method MyBizClassA<br>myBizMethod(MyBizClassB, MyBizClassC) | Invocation                                                           | Method MyClassA<br>myMethod(MyClassB, MyClassC)           |
| BusinessClass is used                                       | Use of operator InstanceOf, or cast, or classification in conditions | ExecutionClass                                            |

Parent topic: [Defining BOM-to-XOM mapping](#)

# Rule language mapping

You can use the rule language mapping code to define the BOM-to-XOM mapping.

You define the rule language mapping by selecting an item in the Outline view and entering the mapping in the **BOM to XOM Mapping** section of the BOM Editor.

This table shows members of the business class. The table assumes that an execution class name ExecutionClass is used for the business class BusinessClass.

Table 1. IRL and ARL mapping possibilities

| Member of business class BusinessClass                      | Case                                                                    | S<br>c<br>o<br>p<br>e                | A function body in IRL or ARL where...                        |
|-------------------------------------------------------------|-------------------------------------------------------------------------|--------------------------------------|---------------------------------------------------------------|
| Constructor<br>BusinessClass(MyBizClassB, MyBizClassC)      | Call by new                                                             | -                                    | this cannot be used, returning an instance of ExecutionClass. |
| Attribute MyBizClassA<br>attr                               | Assignment                                                              | s<br>t<br>a<br>t<br>i<br>c           | value can be used and this cannot be used.                    |
|                                                             |                                                                         | i<br>n<br>s<br>t<br>a<br>n<br>c<br>e | value and this can both be used.                              |
|                                                             | Access                                                                  | s<br>t<br>a<br>t<br>i<br>c           | this cannot be used, returning an instance of MyClassA.       |
|                                                             |                                                                         | i<br>n<br>s<br>t<br>a<br>n<br>c<br>e | this can be used, returning an instance of MyClassA.          |
| Method MyBizClassA<br>myBizMethod(MyBizClassB, MyBizClassC) | Invocation                                                              | s<br>t<br>a<br>t<br>i<br>c           | this cannot be used.                                          |
|                                                             |                                                                         | i<br>n<br>s<br>t<br>a<br>n<br>c<br>e | this can be used.                                             |
| Tester                                                      | Use of operator<br>InstanceOf, or cast, or classification in conditions | -                                    | this can be used, returning a Boolean.                        |

## Class tester mapping

You can use a tester to test the instances of the execution class.

For example, you have a business class that is named RichCustomer in your BOM, and it corresponds to an execution class Customer, which has the attribute money greater than 100000:

- 1. Set the execution name to Customer.
- 2. Enter the following code in the Tester field:

```
return this.money > 100000;
```

## Attribute mapping

You can map a BOM attribute to an expression to return an attribute value at run time.

For example, you have a class that is named ShoppingCart in the XOM, and the class contains two methods that return the number of items in the shopping cart and the total value of the items:

```
// Execution class
public class ShoppingCart {
 public double getValue() ...
 public int getNumberOfItems() ...
}
```

The BOM contains an attribute that represents the average item price, which is computed by using the following expression:

```
// BOM class
public class ShoppingCart {
 public double averageItemPrice;
 ...
}
```

In the Getter field of the BOM Editor, enter the expression that computes the value of the attribute:

```
if (this.getNumberOfItems() == 0)
 return 0;
return this.getValue()/this.getNumberOfItems();
```

## Method call mapping

You can map a method call to any expression.

For example, you have a class Customer in the XOM, and the class contains a method that returns the age of the customer:

```
// Execution class
public class Customer {
 public int getAge() ...
}
```

In the BOM, you have a predicate that checks whether the customer is older than a certain age, and the age is passed to the method as a parameter:

```
public class Customer {
 public boolean isOlderThan(int age);
}
```

In the Body field of the BOM Editor, you express the BOM-to-XOM mapping as follows:

```
return this.getAge() > age;
```

## Constructor call mapping

You can map a constructor call to a specified expression.

For example, you have a class Customer in the XOM, and the class contains an age attribute and a constructor that takes the name of the customer parameter:

```
// Execution class
public class Customer {
 public Customer(String name);
 public int age;
}
```

The BOM has a different constructor that takes the name and the age of the customer:

```
public class Customer {
 public Customer(String name, int age);
}
```

In the Body field of the BOM Editor, you express the BOM-to-XOM mapping as follows:

```
Customer result = new Customer(name);
result.age = age;
return result;
```

## Synthetic object mapping

Instead of regular Java objects, you can use synthetic objects as application data. You can then use BOM-to-XOM mapping to implement a complete business object model by using only the synthetic objects.

In the following example, you use the `java.util.Map` interface to implement synthetic objects, where attributes are stored as pairs of key values. Therefore, only the regular `java.util.Map` interface exists in the XOM.

In the BOM, you have a business class `Customer` defined as follows:

```
Class Customer {
 Customer(String name);
 readonly String name;
 int money;
}
```

The execution class for `Customer` is `java.util.Map`. To facilitate the writing of the member mappings, add the following import:

```
import java.util.*;
import java.lang.*;
```

For the name attribute, the getter part of the BOM-to-XOM mapping is expressed as follows:

```
return (String)this.get("name");
```

For the money attribute, by using an entry `money` that contains an `Integer` in the `Map`, the getter part of the BOM-to-XOM mapping is expressed as follows:

```
return ((Integer)this.get("money"));
```

The setter part is expressed as follows:

```
this.put("money",value);
```

Because you can use the same execution class, which is `java.util.Map` in this example, for more than one business class, you must distinguish which instances belong to which business class. This information must be stored in the `Map` when objects are constructed, and is used by the tester. You express the BOM-to-XOM mapping body for the constructor as follows:

```
Map result = new HashMap(3);
result.put("name", name);
result.put("money",0);
// as assumed, it always contains an Integer
result.put("className", "Customer");
return result;
```

The tester part of the BOM-to-XOM mapping for the class `Customer` uses the `className` entry to differentiate instances of `java.util.Map`:

```
return "Customer".equals(this.get("className"));
```

**Parent topic:** [Defining BOM-to-XOM mapping](#)

## Extender mapping

Extender mapping is designed to use Java™ rather than rule language. In a Java extender class, you create static elements that have the same name as business class elements.

You define an extender class name for a class in the **Extender name** field of the **BOM to XOM Mapping** section of the BOM Editor. The BOM-to-XOM mapping mechanism then looks up extender elements that have the same name as your business elements in the extender class. Therefore, when you create the extender class, make sure that you create elements that can be found from the business element name.

The following table shows extender mapping options. It assumes that an extender class name ExtenderClass is used for a business class named BusinessClass.

Table 1. Extender mapping options

| Member of business class<br>BusinessClass                      | Case                                                | S<br>c<br>o<br>p<br>e        | Extender member of class<br>ExtenderClass                                |
|----------------------------------------------------------------|-----------------------------------------------------|------------------------------|--------------------------------------------------------------------------|
| Constructor<br>BusinessClass(MyBizClass<br>B, MyBizClassC)     | Call by new                                         | -                            | static ExtenderClass<br>create(MyClassB, MyClassC)                       |
| Attribute MyBizClassA attr                                     | Assignment                                          | st<br>at<br>ic               | static MyClassA attr<br><br>—OR—<br><br>static void<br>setAttr(MyClassA) |
|                                                                |                                                     | in<br>st<br>a<br>n<br>c<br>e | static void<br>setAttr(ExtenderClass,<br>MyClassA)                       |
|                                                                | Access                                              | st<br>at<br>ic               | static MyClassA<br><br>—OR—<br><br>static MyClassA getAttr()             |
|                                                                |                                                     | in<br>st<br>a<br>n<br>c<br>e | static MyClassA<br>getAttr(ExtenderClass)                                |
| Method MyBizClassA<br>myBizMethod(MyBizClassB,<br>MyBizClassC) | Invocation                                          | st<br>at<br>ic               | static MyClassA<br>myMethod(MyClassB,<br>MyClassC)                       |
|                                                                |                                                     | in<br>st<br>a<br>n<br>c<br>e | static MyClassA<br>myMethod(ExtenderClass,<br>MyClassB, MyClassC)        |
| InstanceOf operator (Tester)                                   | Use of operator<br>classification in<br>conditions. | -                            | static boolean<br>tester(ExtenderClass)                                  |

### Example

The following code shows an example of an extender class for a BOM class that is named EventDispatcher.

```
public class DispatcherExtender {
 /**
 * This extender method implements
 * EventDispatcher.exception(FinancialEvent, String)
 */
 public static void exception(GenericObject dispatcher,
 GenericObject event,
```



```

 String message) {
 System.out.println("###> Processing Exception : event "
 + event + message + "<###");
}

/**
 * This extender method implements
 * EventDispatcher.send(FinancialEvent, String)
 */
public static void send(GenericObject dispatcher,
 GenericObject event,
 String target) {
 System.out.println("===> Sending " + event + " to " + target + " <===");
}
}

```

## Class tester mapping

The mapping from a business class to an execution class can become complex. For example, you might have several business classes that are mapped to one execution class. To make mapping easier, you can specify a tester to test the instances of the execution class.

For example, you have a RichCustomer business class that corresponds to an execution class Customer:

- Set **Execution name** to Customer.
- Set **Extender name** to RichCustomerExt. This class has no naming conventions.
- Enter the following code in the Tester field:

```

public static boolean tester(Customer customer) {
 return customer.money > 100000;
}

```

The code specifies that the tester method is a Java static method of the extender class that is named tester. It returns a boolean and takes the execution class Customer as its parameter.

## Attribute mapping to an expression to return a value

You can use extender mapping to map a BOM attribute to an expression and return an attribute value at run time.

In this example, the method that is used when the attribute averageItemPrice is accessed is provided as a Java static method that is named getAverageItemPrice:

```

public class CartExtender {
 public static double getAverageItemPrice(ShoppingCart cart) {
 if (cart.getNumberOfItems() == 0)
 return 0;
 return cart.getValue()/ cart.getNumberOfItems();
 }
}

```

## Attribute mapping to an expression to set a value

You can map a BOM attribute to an expression that sets the value of the attribute at run time.

In this example, the method that is used when the attribute averageItemPrice is accessed is provided as a Java static method that is named setAverageItemPrice:

```

public class CartExtender {
 public static void setAverageItemPrice(ShoppingCart cart, double v) {
 cart.setValue(cart.getNumberOfItems()*v);
 }
}

```

## Method call mapping

You can map a method call to a specified expression.

In this example, the method that is used when the method `isOlderThan` is called is provided as a Java static method with the same name in the extender class:

```
public class ExtCustomer {
 public static boolean isOlderThan(Customer customer, int age) {
 return customer.getAge() > age;
 }
}
```

## Constructor call mapping

You can map a constructor call to any expression.

In this example, the method that is used when the constructor is called is provided as a Java static method that is called `create`:

```
public class ExtCustomer {
 public static Customer create(String name, int age) {
 Customer c = new Customer(name);
 c.setAge(age);
 return c;
 }
}
```

## Synthetic object mapping

You can use a synthetic object to simulate the state of real objects. Like a real object, a synthetic object can store attributes. Service Data Objects (SDO) is an example of a synthetic object framework. You can use synthetic objects instead of regular Java objects as application data. You can then use BOM-to-XOM mapping to implement a complete business object model that uses only synthetic objects.

Specify the name of your extender class in the **Extender name** field, and specify the name of the execution class in the **Execution name** field.

In this example, the extender class defines all the necessary mappings to the interface of the execution class `java.util.Map`:

```
import java.util.*;
public class CustomerExtender {
 public static String getName(Map customer) {
 return (String) customer.get("name");
 }

 public static int getMoney(Map customer) {
 return ((Integer) customer.get("money")).intValue();
 }

 public static void setMoney(Map customer, int value) {
 customer.put("money", new Integer(value));
 }

 public static Map create(String name) {
 Map result = new HashMap(3);
 result.put("name", name);
 result.put("money", new Integer(0));
 // as assumed it always contains an Integer
 result.put("className", "Customer");
 return result;
 }

 public static boolean tester(Map customer) {
 return "Customer".equals(customer.get("className"));
 }
}
```

**Parent topic:** [Defining BOM-to-XOM mapping](#)

# Mapping of a BOM created from a Java XOM

When you create a BOM from the XOM, Rule Designer processes the Java™ compiled classes into business elements.

The following table describes the default mapping for business elements originating from a Java XOM when you keep the option **Load getters and setters as attributes** selected in the New BOM Entry wizard.

Table 1. Business elements originating from a Java XOM

| Java XOM element                                                                                                                                                                                                                                                                                                                    | Becomes element in the BOM...                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nongeneric public class.                                                                                                                                                                                                                                                                                                            | Class with the same name.                                                                                                                                                                                                                                                                             |
| <div>Class that implements the java.util.Collection interface.</div> <div>For example:</div> <div>MyCollection implements java.util.Collection</div>                                                                                                                                                                                | <div>Class with a collection domain to be recognized as a collection when verbalized.</div> <div>For example:</div> <div>public class MyCollection implements java.util.Collection { domain 0, * ; }</div>                                                                                            |
| Public constructor.                                                                                                                                                                                                                                                                                                                 | Constructor with the same parameters.                                                                                                                                                                                                                                                                 |
| Public attribute.                                                                                                                                                                                                                                                                                                                   | Attribute with the same name and return type.                                                                                                                                                                                                                                                         |
| Final attribute.                                                                                                                                                                                                                                                                                                                    | Read-only attribute with the same name and return type.                                                                                                                                                                                                                                               |
| <div>Public static final attributes whose types are the current class.</div> <div>For example:</div> <div>public class Color { private Color(String name) {...} public static final Color red = new Color("red"); public static final Color blue = new Color("blue"); public static final Color green = new Color("green"); }</div> | <div>Enumerated domain of static references of the class.</div> <div>For example:</div> <div>public class Color { domain { static red, static blue, static green } public static final readonly Color red; public static final readonly Color blue; public static final readonly Color green; }</div> |
| <div>Public method that does not follow the JavaBeans convention for property accessors (void setFoo(PropertyType value) and PropertyType getFoo()).</div> <div>For example:</div> <div>public interface Customer { public boolean register(java.sql.Connection db); }</div>                                                        | <div>Method with equivalent parameters.</div> <div>For example:</div> <div>public interface Customer { public abstract boolean register(java.sql.Connection arg); }</div>                                                                                                                             |
| Public method that follows the JavaBeans convention, with a get method and no set method.                                                                                                                                                                                                                                           | Read-only attribute.                                                                                                                                                                                                                                                                                  |

|                                                                                                                                                                                                                              |                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <p>For example:</p> <pre>public interface Customer { public int getAge(); }</pre>                                                                                                                                            | <p>For example:</p> <pre>public interface Customer { public readonly int age; }</pre>                                                |
| <p>Public method that follows the JavaBeans convention, with only a set method.</p> <p>For example:</p> <pre>public interface Customer { public void setBirthDate(Date date); }</pre>                                        | <p>Write-only attribute.</p> <p>For example:</p> <pre>public interface Customer { public writeonly java.util.Date birthDate; }</pre> |
| <p>Public method that follows the JavaBeans convention, with both a get method and a set method.</p> <p>For example:</p> <pre>public interface Customer { public String getName(); public void setName(String name); }</pre> | <p>Attribute.</p> <p>For example:</p> <pre>public interface Customer { public java.lang.String name; }</pre>                         |

Business elements that originate from a dynamic XOM are processed differently. For information about elements that originate from a dynamic XOM, refer to [Designing a BOM for an XML model](#).

**Parent topic:** [Designing a BOM for a Java model](#)

**Related concepts:**  
[Overview: BOM and execution object model \(XOM\)](#)

**Related tasks:**  
[Creating a BOM entry from a XOM](#)

**Related information:**  
[Business object model \(BOM\)](#)  
[Updating the BOM when the XOM changes](#)

# Adding annotations to the XOM

You can add annotations to the XOM to customize the way that the BOM is created from Java™ classes.

## XOM annotations

You can use annotations in your Java source code. Annotations are a type of metadata that you can add to classes, members and parameters.

## @NotBusiness

You use the @NotBusiness annotation to filter out classes and members that you do not want to import.

## @BusinessName

You use the @BusinessName annotation to provide a name for a class, a member, or a parameter in the BOM.

## @BusinessType

You use the @BusinessType annotation to change the type of a member or a parameter in the BOM.

## @BoundedIntDomain

You use the @BoundedIntDomain to specify an interval between two bounding values.

## @CollectionDomain

You use the @CollectionDomain to specify the cardinality and the type of collection elements.

## @PatternDomain

You use the @PatternDomain annotation to define a pattern domain on a member or parameter.

## @CustomProperty

You use the @CustomProperty annotation to set a property to a class, a member, or a parameter.

## @CustomProperties

You use the @CustomProperties annotation to set several properties to a class, a member, or a parameter.

## @java.beans.ConstructorProperties

You can use the @java.beans.ConstructorProperties annotation to provide a name for a constructor.

**Parent topic:** [Designing a BOM for a Java model](#)

## XOM annotations

You can use annotations in your Java™ source code. Annotations are a type of metadata that you can add to classes, members and parameters.

You can add annotations to the execution object model (XOM) to customize the way that the business object model (BOM) is created from Java classes. You use the annotations to apply changes to classes, members, or parameters in the BOM.

For example, you can add an annotation to a class to define a business name for the class. In the BOM, the class takes the name defined by the annotation.

When you create the BOM from the XOM, the BOM Verbalization page of the New BOM Entry wizard displays the business name of the class.

### Important:

To use the annotations from the [ilog.rules.bom.annotations](#) package, you must add `<InstallDir>/studio/lib/jrules-engine.jar` to the classpath of your XOM. However, if you just want to use the `@java.beans.ConstructorProperties` annotation, you do not need to add the `jrules-engine.jar` to the classpath.

**Parent topic:** [Adding annotations to the XOM](#)

### Related concepts:

[Mapping of a BOM created from a Java XOM](#)

## @NotBusiness

You use the @NotBusiness annotation to filter out classes and members that you do not want to import.

### Purpose

Filters out classes and members that you do not want to import in the BOM.

### Syntax

```
@NotBusiness
```

### Description

The classes and members with the @NotBusiness annotation are not imported in the BOM. For example, if you encounter problems when loading a class, you can filter out the member that uses this class by setting the @NotBusiness annotation to this member.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 public Customer(String name) {...}
 public String getName() {...}

 @NotBusiness
 public Object readResolve() throws ObjectStreamException {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(string arg);
 readonly string name;
}
```

**Parent topic:** [Adding annotations to the XOM](#)

## @BusinessName

You use the @BusinessName annotation to provide a name for a class, a member, or a parameter in the BOM.

### Purpose

Provides a name for a class, a member, or a parameter in the BOM.

### Syntax

```
@BusinessName(<a string>)
```

### Description

In Java™, the parameter names are not stored in the class. You can set the @BusinessName annotation to give a business name to a parameter. The parameter names are stored in the BOM, and they are used in the BOM to XOM mapping and in the testing and simulation constructor for testing.

A discrepancy between a BOM name and a XOM name can cause an error. To resolve the error, you must edit the BOM to XOM mapping.

For information on annotation for changing the name of a parameter in the BOM, see [Annotation Type BusinessName](#).

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 public Customer(@BusinessName("name") String name) {...}
 public String getName() {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(string name);
 readonly string name;
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related reference:**

[@java.beans.ConstructorProperties](#)



## @BusinessType

You use the @BusinessType annotation to change the type of a member or a parameter in the BOM.

### Purpose

Changes the type of a member or a parameter in the BOM.

### Syntax

```
@BusinessType(<a string>)
```

### Description

The @BusinessType annotation changes the type of a member or a parameter in the BOM. For example, if you want to change a type int to an enumeration in the BOM to verbalize it. An enumeration is a class with a domain set as an enumeration of static references.

When you apply the @BusinessType annotation to a method, the return type of the method is modified.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 public Customer(@BusinessType("Category") int category) {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(Category category);
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**

[Domains](#)

## @BoundedIntDomain

You use the @BoundedIntDomain to specify an interval between two bounding values.

### Purpose

Specifies an interval between two bounding values on a member or a parameter of type int.

### Syntax

```
@BoundedIntDomain(min = <an int>, max = <an int>)
```

### Description

The @BoundedIntDomain provides a bounded domain to specify an interval between two bounding values on a member or a parameter of type int.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 @BoundedIntDomain(min = 0, max = 120)
 public int age;
}
```

The result in the BOM is the following code:

```
class Customer {
 int age domain [0,120];
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**

[Domains](#)

## @CollectionDomain

You use the @CollectionDomain to specify the cardinality and the type of collection elements.

### Purpose

Specifies the cardinality and the type of collection elements on a member or a parameter where a collection type is used.

### Syntax

```
@CollectionDomain(<see the example>)
```

### Description

The @CollectionDomain provides a collection domain to specify the type of collection elements and the cardinality.

The collection domain and the element type are used by the Business Action Language (BAL).

### Example

In the XOM, the annotation can be used as follows:

```
public class Cart {
 @CollectionDomain(elementType = "Item")
 public List items;

 @CollectionDomain(min = 1, max = 12)
 public Person[] passengers;
}
```

The result in the BOM is the following code:

```
class Cart{
 List items domain 0,* class Item;
 Person[] passengers domain 1,12 class Person;
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**

[Domains](#)

## @PatternDomain

You use the @PatternDomain annotation to define a pattern domain on a member or parameter.

### Purpose

Defines a pattern domain for a member or a parameter.

### Syntax

```
@PatternDomain(<a string>)
```

### Description

The @PatternDomain annotation enables you to specify a pattern domain for a member or a parameter.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 @PatternDomain("[A-Za-z]")
 public String getName() {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 readonly string name domain "[A-Za-z]";
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**  
[Domains](#)

## @CustomProperty

You use the @CustomProperty annotation to set a property to a class, a member, or a parameter.

### Purpose

Sets a property on a class, a member, or a parameter.

### Syntax

```
@CustomProperty(name=<a string>, value=<a string>)
```

### Description

The @CustomProperty annotation sets a property on a class, a member, or a parameter.

You can use any BOM property.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {

 @CustomProperty(name = "dataio.default", value = "true")
 public Customer(@BusinessName("name") String name);
 public String getName() {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(string name)
 property dataio.default "true";
 readonly string name;
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**

[BOM properties](#)

## @CustomProperties

You use the @CustomProperties annotation to set several properties to a class, a member, or a parameter.

### Purpose

Sets several properties on a class, a member, or a parameter.

### Syntax

```
@CustomProperties(names={<strings>},values={<strings>})
```

### Description

The @CustomProperties annotation sets several properties on a class, a member, or a parameter.

You can use any BOM property.

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 @CustomProperties(names = {"java_length", "other" }, values = { "9",
 "true"})
 public String getName() {...}
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(string name)
 property java_length "9"
 property other "true";
 readonly string name;
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related concepts:**

[BOM properties](#)

## @java.beans.ConstructorProperties

You can use the `@java.beans.ConstructorProperties` annotation to provide a name for a constructor.

### Purpose

Provides a name for a constructor.

### Syntax

```
@java.beans.ConstructorProperties({<strings>})
```

### Description

You can use the `@java.beans.ConstructorProperties` annotation to provide a name for a constructor. You can also use the `@BusinessName` on the parameters to give a name to each argument in the BOM.

For more information about the `@java.beans.ConstructorProperties` annotation, see [ConstructorProperties](#).

### Example

In the XOM, the annotation can be used as follows:

```
public class Customer {
 @ConstructorProperties("name", "age")
 public Customer(String name, int age);
}
```

The result in the BOM is the following code:

```
class Customer {
 Customer(string name, int age);
}
```

**Parent topic:** [Adding annotations to the XOM](#)

**Related reference:**

[@BusinessName](#)

## Designing a BOM for an XML model

You can use data other than Java™ as the basis for your XOM.

### Overview: Dynamic execution object model (XOM)

The dynamic execution object model (XOM) can be generated from native Java classes or from XML data. The rule engine accesses XML directly.

### Defining a dynamic XOM for a rule project

Define the dynamic XOM when creating a rule project, or later in the properties of the rule project.

### Mapping between XML schema and dynamic classes

XML binding maps XML schema types to XOM dynamic classes and objects. It supports the map function, schema types defined from other types, and XML declarations, subject to limitations.

**Parent topic:** [Designing business object models](#)



## Overview: Dynamic execution object model (XOM)

The dynamic execution object model (XOM) can be generated from native Java™ classes or from XML data. The rule engine accesses XML directly.

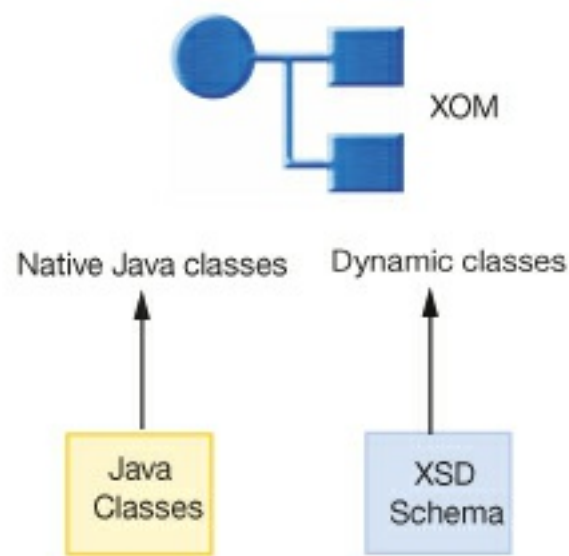
The execution object model (XOM) is an object model against which rules are executed. Business rules are written against the Business Object Model (BOM), and then translated into the ILOG® Rule Language (IRL) and run against the XOM. The rule engine evaluates the rules against the application objects and executes them when appropriate.

In Decision Server, a XOM can be generated from native Java classes or from dynamic classes, or derived from XML schema definitions (XSDs). The word *dynamic* means that no native Java object is constructed or generated. Instead, dynamic XML objects are created to represent the instances of the dynamic classes.

- A XOM generated from native Java classes is known as a **Java XOM**.
- A XOM generated from dynamic classes is known as a **Dynamic XOM**.

When the rule engine looks for XOM classes, it looks first in the dynamic XOM, then in the Java XOM.

The following diagram shows the mapping between a XOM and Java classes and dynamic classes.



**Parent topic:** [Designing a BOM for an XML model](#)

**Related concepts:**

[Overview: BOM and execution object model \(XOM\)](#)

**Related information:**

[Mapping between XML schema and dynamic classes](#)

## Defining a dynamic XOM for a rule project

Define the dynamic XOM when creating a rule project, or later in the properties of the rule project.

### About this task

You can define a dynamic XOM either in the New Rule Project wizard when you create the project, or later, using the rule project Properties dialog. The procedure is the same in each case.

### Procedure

To define a dynamic XOM using the Properties dialog:

1. In the Rule Explorer view, select the rule project and then on the **Project** menu click **Properties**.
2. In the pane of the Properties dialog, click **Dynamic Execution Object Model**.
3. In the Dynamic Execution Object Model wizard, on the Dynamic Bindings tab, select **Add XSD** or **Add External XSD**:
  - To add XSDs that have been imported into the Eclipse workspace, click **Add XSD**. Select each of the required XSD files from the **XSD Files** dialog and then click **OK**.
  - To add XSDs that are held in your file system, click **Add External XSD**. Select each of the required XSD files from the **External XSD Files** dialog and then click **Open**.

For each schema namespace, a package name is defined in which the XOM classes are stored. You can expand each list entry to see the XML namespaces and the package names.

At the bottom of the Dynamic Bindings tab, a hidden section lists the XSD files that are included in the ruleset archive. To see this list, click the black arrow next to **Schemas to be included in Ruleset archive for enabling the Transparent Decision Service**.

This list of files might be important when you use the hosted transparent decision service.

4. The default package name is derived from the namespace. If you want to rename a package:
  - a. Select the package and then click **Edit**.
  - b. Type the new package name in the **Package Name Configuration** dialog and then click **OK**.
5. If you want to change the order in which the selected dynamic binding files are listed, click the **Order** tab and then use the up or down arrows to move the required files.

You might want to change the order of the files if the same element exists in more than one file and you want to dictate which schema or file to use first to identify it.

6. When you have selected all your dynamic binding files, click **Apply and Close** to close the **Dynamic Execution Object Model** dialog.

The dynamic XOM is now defined. The new packages are shown in the BOM Entry and behave like any normal package.

7. To remove dynamic binding files from the XOM, select each of the files you want to remove and then click **Remove**.

**Parent topic:** [Designing a BOM for an XML model](#)

**Related concepts:**  
[Transparent decision services](#)

# Mapping between XML schema and dynamic classes

XML binding maps XML schema types to XOM dynamic classes and objects. It supports the map function, schema types defined from other types, and XML declarations, subject to limitations.

## Map function

The map function is used to design a dynamic type when its related schema type is known.

## Schema types

You use schemas to define types and refine your definition by deriving previously defined types.

## XML declarations

XML elements, attributes, groups, and appinfo structures map to dynamic classes.

## Schema-related markup in XML documents

Schema-related markup in XML documents includes `xsi:nil`, `xsi:type`, `xsi:schemaLocation`, and `xsi:noNamespaceSchemaLocation`.

## Schema mapping limitations

Some features or keywords are not mapped by the current XML binding and the value types of some schema types are lost.

**Parent topic:** [Designing a BOM for an XML model](#)

## Map function

The map function is used to design a dynamic type when its related schema type is known.

The map function is used to design a dynamic type when its related schema type is known.

- If  $T$  is a *complex* type,  $map(T)$  represents the dynamic class related to  $T$ .
- If  $T$  is an *atomic* simple type,  $map(T)$  represents its mapped dynamic type by following these rules:
  1. If  $T$  is a built-in type, the dynamic type is given by XXX.
  2.  $map(T) = map(base(T))$  where  $base(T)$  is the direct base simple type of  $T$ .

### Example of a map function

With a simple type MyT defined as:

```
<simpleType name="MyT">
 <restriction base="string">
</simpleType>
```

You have,  $map(MyT) = java.lang.String$ .

The following expression:

```
class C
{
 [map(MyT)] myField;
}
```

Stands for:

```
class C
{
 java.lang.String myField;
}
```

**Parent topic:** [Mapping between XML schema and dynamic classes](#)

**Related information:**

[Schema types](#)

# Schema types

You use schemas to define types and refine your definition by deriving previously defined types.

## **Built-in simple types**

Built-in schema types map to XOM dynamic class types.

## **Simple type mapping**

Simple types map schemas to dynamic models.

## **Simple types derived from other simple types**

You can derive simple types from other simple types and incorporate restrictions to the more general simple type using facets.

## **List types and union types**

You can use list and union types in a schema and XML document.

## **Local simple types mapped to inner classes**

Embedded local simple types map to inner classes.

## **Complex types**

You can use mixed content in a schema and an XML document.

## **Extension of simple type content in complex types**

You can extend simple type content in a schema and an XML document.

## **Complex type restriction**

The XOM takes complex type restrictions into account in the XOM.

## **Complex type extension**

The XOM takes complex type extensions into account.

## **Local complex types mapped to inner classes**

Local complex types map to inner classes.

## **Default constructor dynamic methods**

To map default constructors, a method is generated for each dynamic class.

## **Type identifier mapping**

Simple type names map to XOM class names according to a number of rules.

**Parent topic:** [Mapping between XML schema and dynamic classes](#)

# Built-in simple types

Built-in schema types map to XOM dynamic class types.

The following table summarizes how built-in schema types are mapped to dynamic class types.

Table 1. Built-in type mapping between XML schemas and XOM dynamic classes

XML schema type	XOM dynamic class type
boolean	boolean
decimal	java.math.BigDecimal
double	double
integer	java.math.BigInteger
int	int
ID	java.lang.String
nonPositiveInteger	java.math.BigInteger
negativeInteger	java.math.BigInteger
nonNegativeInteger	java.math.BigInteger
positiveInteger	java.math.BigInteger
unsignedLong	java.math.BigInteger
unsignedInt	long
unsignedShort	int
unsignedByte	short
normalizedString	java.lang.String
token	java.lang.String
language	java.lang.String
name	java.lang.String
ENTITY	java.lang.String
IDREFS	java.util.Vector
ENTITIES	java.util.Vector
float	float
duration	ilog.rules.xml.types.IlrDuration
dateTime	ilog.rules.xml.types.IlrDateTime
time	ilog.rules.xml.types.IlrTime
gYearMonth	ilog.rules.xml.types.IlrGYearMonth
gMonthDay	ilog.rules.xml.types.IlrGMonthDay
gDay	ilog.rules.xml.types.IlrGDay
gMonth	ilog.rules.xml.types.IlrGMonth
date	ilog.rules.xml.types.IlrDate
gYear	ilog.rules.xml.types.IlrGYear
NMTOKENS	java.util.Vector
anyURI	java.lang.String
QNAME	java.lang.String
NOTATION	java.lang.String
hexBinary	byte[]
base64Binary	byte[]
IDREF	java.lang.String
string	java.lang.String
NCNAME	java.lang.String
NMTOKEN	java.lang.String
long	long
short	short

**Note:**

IDRef is mapped to a String. Management of reference and ID is currently not available.

Parent topic: [Schema types](#)

**Related information:**

- [Complex types](#)
- [Local simple types mapped to inner classes](#)
- [Simple type mapping](#)



# Simple type mapping

Simple types map schemas to dynamic models.

The following table shows some simple type mappings between a schema and a dynamic model.

Table 1. Standard simple type mapping

Case	Schema	Dynamic model
Global simple type derived by restriction	<pre>&lt;simpleType name="st1" &gt;   &lt;restriction base="st0"&gt;     ...   &lt;/restriction&gt; &lt;/simpleType&gt; &lt;element name="e1" type="st1"&gt;</pre>	<pre>type s1 extends st0 {   // [map(st0)]   javatype;   [map(st0)]   getMinInclusive();   [map(st0)]   getMaxInclusive();   [map(st0)]   getMinExclusive();   [map(st0)]   getMaxExclusive();   String   getPattern();   int getLength();   int   getMaxLength();   int   getMinLength();   [map(st0)][]   getEnumerations();   int   getTotalDigits();   int   getFractionDigits() } [map(st0)] e1;</pre>
Local simple type derived by restriction	<pre>&lt;complexType name="ct1"&gt;   &lt;sequence&gt;     &lt;element name="e1" &gt;       &lt;simpleType name="st1" &gt;         &lt;restriction base="st0"&gt;           ...         &lt;/restriction&gt;       &lt;/simpleType&gt;     &lt;/element&gt;   &lt;/sequence&gt; &lt;/complexType&gt;</pre>	<pre>class Ct1 {   type s1 extends st0   {     [map(st0)]   }   javatype;   [map(st0)] e1; }</pre>
Simple type derived by list	<pre>&lt;simpleType name="st1" &gt;   &lt;list itemType="st0"/&gt; &lt;/simpleType&gt; &lt;element name="e1" type="st1"&gt;</pre>	<pre>type s1 {   java.util.Vector javaType; } java.util.Vector e1;</pre>



Simple type derived by union	<pre>&lt;simpleType name="stUnion" &gt;   &lt;union memberTypes="st0 st1"/&gt; &lt;/simpleType&gt; &lt;element name="e1" type="st1"&gt;</pre>	<pre>type stUnion {   java.lang.Object   javaType; } java.util.Object e1;</pre>

Parent topic: [Schema types](#)

Related information:

[Built-in simple types](#)

[Simple types derived from other simple types](#)

[Map function](#)

[Complex types](#)

## Simple types derived from other simple types

You can derive simple types from other simple types and incorporate restrictions to the more general simple type using facets.

The decision to derive a simple type from another simple type implies incorporating restrictions. To incorporate such restrictions, you use facets. Facets are sets of predefined constraints that help define new simple types by restricted built-in types.

### Example of a facet: a zip code

An example of a facet is a pattern where a zip code is defined as containing only five digits.

The following schema shows a simple type definition:

```
<simpleType name="myType">
 <restriction base="string">
 <pattern value="*A"/>
 </restriction>
</simpleType>
<attribute name = "typeValue" type = "myType"/>
```

In this example, the simple type `myType` is defined as a subtype of `string`. The `typeValue` attribute is mapped to a dynamic class field of type `java.lang.String`.

When simple type definitions are mapped to dynamic class types, the type hierarchy tree is searched. Starting from the most restrictive type, the hierarchy tree is searched upwards until a built-in type is found that maintains the mapping type as the simple type definition mapping.

The following zip code example shows how user schema simple types are represented in the XOM and which methods are generated to manipulate them. A set of static methods are declared. They can be called from a rule. Two rules named `checkFloatInInterval` and `findZipCodePattern` are used to test the `ZipCode` objects.

Here is the schema:

```
<simpleType name="zip-code">
 <restriction base="string">
 <pattern value="A[0-9]{5}"/>
 </restriction>
</simpleType>
```

```
<simpleType name="Interval">
 <restriction base="float">
 <minInclusive value="1.1"/>
 <maxInclusive value="3.1"/>
 </restriction>
</simpleType>
```

Here is the XOM representation:

```
class ZipCode extends IlrXmlObject
{
 static String getPattern();
 static int getMinLength();
 static int getMaxLength();
 static getLength();
 static String[] getEnumerations();
}
```

```
class Interval
{
 static String getPattern();
 static int getMinLength();
 static int getMaxLength();
 static getLength();
 static float[] getEnumerations();
 static float getMinInclusive ();
}
```

```
static float getMaxInclusive ();
static float getMinExclusive ();
static float getMaxExclusive ();
static int getTotalDigits();
static int getFractionDigits();
}
```

And here are the rules:

```
rule checkFloatInInterval
{
 when {
 f: Float (i: floatValue(); Interval.getMinInclusive() <= i;
 i <= Internal.getMaxInclusive());
 }
 then {
 out.println (f + " in Interval");
 }
}
```

```
rule findZipCodePattern
{
 when {
 s: String (equals ZipCode.getPattern());
 }
 then {
 out.println (s + " equals zip code pattern");
 }
}
```

The simple type dynamic method provides information on type facets values.

The fact that the method returns the last facet definition in the type hierarchy has the following consequences:

- If a facet is defined twice in the type hierarchy, the nearest definition is returned.
- If a facet is not defined, a special value (usually null) is returned depending on the facet type.

Facets can be of the following types:

- Range facets: `getMinInclusive`, `getMaxInclusive`, `getMinExclusive`, `getMaxExclusive`
- Lexical space facets: `getPattern`, `getMinLength`, `getMaxLength`, `getLength`
- Enumeration facets: `getEnumerations`
- Numeric facets: `getTotalDigits`, `getFractionDigits`

The following table shows the returned values when the facet is not defined.

Table 1. Returned values with undefined facet

Types	getMinInclusive getMinExclusive	GetMaxInclusive getMaxExclusive	getMinLength getMaxLength  getLength getTotalDigits getFractionDigits	getEnumerations
float	- Float.MAX_VALUE	Float.MAX_VALUE	IlrXmlObject.UNKNOWN_POSITIVE_VALUE	float[0]
double	- Double.MAX_VALUE	Double.MAX_VALUE		double[0]
byte	Byte.MIN_VALUE	Byte.MAX_VALUE		byte[0]
int	Integer.MIN	Integer.MAX		int[0]

	VALUE	VALUE	
long	Long.MIN_VALUE	Long.MAX_VALUE	long[0]
Object	null	null	Object[0]

**Parent topic:** [Schema types](#)

**Related information:**

[Built-in simple types](#)

[Simple type mapping](#)

## List types and union types

You can use list and union types in a schema and XML document.

This example demonstrates the use of list types and union types in a schema and an XML document. The rules `displayExtendedIntMeasure` and `displayStandardInt` are used to test the `MeasureSet` objects.

Here is the schema:

```
<simpleType ExtendedInt>
 <union memberTypes="int string"/>
</simpleType>
```

```
<simpleType IntMeasures>
 <list itemType="ExtendedInt"/>
</simpleType>
```

```
<element name="measureSet" >
 <complexType>
 <sequence>
 <element name="intMeasures" type="IntMeasures" />
 <element name="extendedInt" type="ExtendedInt" />
 </sequence>
 </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<measureSet>
 <intMeasures>1 2 3 4 ten 5 unknown</intMeasures>
 <extendedInt>3</extendedInt>
</measureSet>
```

Here is the XOM representation:

```
class MeasureSet extends IlrXmlObject
{
 Vector intMeasures; // list of (Integer| String) instances
 Object extendedInt; // either Integer or String instances
 ...
}
```

And here are the rules:

```
rule displayExtendedIntMeasure
{
 when {
 ms: MeasureSet ();
 m: String () in ms.intMeasures;
 }
 then {
 out.println (m + " is an extended int.");
 }
}
```

```
rule displayStandardInt
{
 when {
 ms: MeasureSet ();
 m: Integer () from ms.extendedInt;
 }
 then {
 out.println (m + " is a standard int.");
 }
}
```

```
}
```

**Parent topic:** [Schema types](#)

**Related information:**

[Built-in simple types](#)

[Complex types](#)

## Local simple types mapped to inner classes

Embedded local simple types map to inner classes.

**Note:**

The inner class policy is activated by default.

### Mapping embedded local simple types to inner classes

This example shows how to map embedded local simple types to inner classes.

The example includes an element named `ident` that contains a simple type restriction. The simple type is represented in the XOM by the dynamic class `BorrowedBook.Ident`.

Here is the schema:

```
<complexType name="borrowed-book">
 <sequence>
 <element name="ident"/>
 <simpleType>
 <restriction base="string">
 <pattern value="A[0-9]*"/>
 </restriction>
 </simpleType>
 </sequence>
</complexType>
```

Here is the excerpt from an XML document:

```
<borrowed-book>
 <ident>12345</ident>
</borrowed-book>
```

Here is the XOM representation:

```
class BorrowedBook extends IlrXmlObject
{
 class Ident
 {
 ...
 };
}
```

And here is the rule:

```
rule identifyBorrowedBook
{
 when {
 b: BorrowedBook ();
 BorrowedBook.Ident equals "12345";
 }
 then {
 ...
 }
}
```

**Parent topic:** [Schema types](#)

**Related information:**

[Built-in simple types](#)

[Simple type mapping](#)

[Map function](#)

# Complex types

You can use mixed content in a schema and an XML document.

## Mixed content in a schema and XML document

This example demonstrates the use of mixed content in a schema and an XML document. The schema contains an element of type `int` and an element of type `string`.

Here is the schema:

```
<element name="observation">
 <complexType mixed="true">
 <sequence>
 <element name="temperature" type="int">
 <element name="city" type="string">
 </sequence>
 ...
 </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<observation>
It is <temperature>10</temperature>degrees in <city>London</city>.
</observation>
```

And here is the XOM representation:

```
class Observation extends IlrXmlObject
{
 int temperature;
 String city;
 String content;
}
```

The content of the observation element is translated to a single `String` that concatenates all the strings and put in a special attribute named `content`.

## Complex type mapping

The following table shows some complex type mappings between a schema and dynamic model.

Table 1. Standard complex type mapping

Case	Schema	Dynamic model
Element with local complex type of unary component	<pre>&lt;element name="e1"&gt;   &lt;complexType&gt;     &lt;sequence&gt;       &lt;element name="e2" type="t2"/&gt;       &lt;element name="e3" type="t3"/&gt;     &lt;/sequence&gt;     &lt;attribute name="a1" type="t4"/&gt;   &lt;/complexType&gt; &lt;/element&gt;</pre>	<pre>class E1 extends IlrXmlObject {   [map(t2)] e2;   [map(t3)] e3;   [map(t4)] a1; } E1 e1;</pre>
Element with local complex type of unary component	<pre>&lt;element name="e1"&gt;   &lt;complexType&gt;     &lt;sequence&gt;</pre>	<pre>class E1 extends IlrXmlObject {</pre>



	<pre>&lt;element name="e2" type="t2" maxOccurs="unbounded"/&gt;   &lt;element name="e3" type="t3" minOccurs="2" /&gt; &lt;/sequence&gt; &lt;/complexType&gt; &lt;/element&gt;</pre>	<pre>// collection of [map(t2)] Vector e2List; // collection of [map(t3)] Vector e3List; } E1 e1;</pre>
Typed element	<pre>&lt;complexType name="t1"&gt;   &lt;sequence&gt;     &lt;element name="e2" type="t2"/&gt;     &lt;element name="e3" type="t3"/&gt;   &lt;/sequence&gt; &lt;/complexType&gt; &lt;element name="e1" type="t1" /&gt;</pre>	<pre>class T1 extends IlrXmlObject {   [map(t2)] e2;   [map(t3)] e3; } T1 e1;</pre>
Typed element with collection group	<pre>&lt;complexType name="t1"&gt;   &lt;sequence maxOccurs="1001"&gt;     &lt;element name="e2" type="t2"/&gt;     &lt;element name="e3" type="t3"/&gt;   &lt;/sequence&gt; &lt;/complexType&gt; &lt;element name="e1" type="t1" /&gt;</pre>	<pre>class T1 extends IlrXmlObject {   // collection of [map(t2)] Vector e2List;   // collection of [map(t3)] Vector e3List; } T1 e1;</pre>
Local complex type: when inner classes are permitted	<pre>&lt;complexType name="t1"&gt;   &lt;sequence&gt;     &lt;element name="e1"&gt;       &lt;complexType&gt;         ...       &lt;/complexType&gt;     &lt;/complexType&gt;   &lt;/complexType&gt; &lt;/element&gt;</pre>	<pre>class T1 extends IlrXmlObject {   class E1 extends IlrXmlObject   {     ...   }   E1 e1; }</pre>
Local complex type: when inner classes are not permitted	<pre>&lt;complexType name="t1"&gt;   &lt;sequence&gt;      &lt;element</pre>	<pre>Class E1 extends IlrXmlObject {  }</pre>

	<pre> name="e1"&gt;     &lt;complexType&gt;         ...     &lt;/complexType&gt; &lt;/complexType&gt; &lt;/element&gt; </pre>	<pre> class T1 extends IlrXmlObject {     E1 e1; } </pre>
--	-------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------

### Simple content mapping

The following table shows some simple content mappings between a schema and dynamic model.

Table 2. Simple content mapping

Case	Schema	Dynamic model
Complex type with simple content	<pre> &lt;simpleType name="st0" &gt; ... &lt;/simpleType&gt; &lt;complexType name="ct0"&gt;     &lt;simpleContent&gt;         &lt;extension base="st0"&gt;     &lt;attribute name="a1" type="st1" /&gt;     &lt;/extension&gt;     &lt;simpleContent&gt; &lt;/complexType&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {     [map(st0)] content;     [map(st1)] a1; } </pre>
Complex type inheriting by extension from complex type with simple content	<pre> &lt;complexType name="ct0" &gt;     &lt;simpleContent&gt;         ...     &lt;simpleContent&gt; &lt;/complexType&gt; &lt;complexType name="ct1"&gt;     &lt;simpleContent&gt;         &lt;extension base="ct0"&gt;     &lt;attribute name="a1" type="st1" /&gt;     &lt;/extension&gt;     &lt;/simpleContent&gt; &lt;/complexType&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {     [map(st0)] content;     ... } class Ct1 extends Ct0 {     [map(st0)] content;     [map(st1)] a1;     ... } </pre>
Complex type inheriting by restriction from complex type with simple content	<pre> &lt;complexType name="ct0" &gt;     &lt;simpleContent&gt;         ...     &lt;simpleContent&gt; &lt;/complexType&gt; &lt;complexType name="ct1"&gt;     &lt;simpleContent&gt;         &lt;restriction base="ct0"&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {     [map(st0)] content;     ... } class Ct1 extends Ct0 { </pre>

	<pre>         &lt;attribute name="a1" type="st1" /&gt;       &lt;/restriction&gt;       &lt;simpleContent&gt; &lt;/complexType&gt; </pre>	
--	-------------------------------------------------------------------------------------------------------------------------------------------	--

### Complex content mapping

The following table shows some complex content mappings between a schema and dynamic model.

Table 3. Complex content mapping

Case	Schema	Dynamic model
Complex type inheriting by extension	<pre> &lt;complexType name="ct0"&gt;   &lt;attribute name="a1" type="st1" /&gt; &lt;/complexType&gt; &lt;complexType name="ct0"&gt;   &lt;complexContent&gt;     &lt;extension base="ct0"&gt;       &lt;attribute name="a2" type="st2" /&gt;     &lt;/extension&gt;     &lt;complexContent&gt; &lt;/complexType&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {   [map(st1)] a1; } class Ct1 extends Ct0 {   [map(st2)] a2; } </pre>
Complex type inheriting by restriction	<pre> &lt;complexType name="ct0"&gt;   &lt;attribute name="a1" type="st1" /&gt; &lt;/complexType&gt; &lt;complexType name="ct0"&gt;   &lt;complexContent&gt;     &lt;restriction base="ct0"&gt;       &lt;attribute name="a1" type="st2" /&gt;     &lt;/restriction&gt;     &lt;complexContent&gt; &lt;/complexType&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {   [map(st1)] a1; } class Ct1 extends Ct0 { } </pre>
Complex type with mixed content, inheriting by extension	<pre> &lt;complexType name="ct0" &gt;   &lt;simpleContent&gt;     ...   &lt;/simpleContent&gt; &lt;/complexType&gt; &lt;complexType name="ct0"&gt;   &lt;complexContent mixed="true"&gt; </pre>	<pre> class Ct0 extends IlrXmlObject {   [map(st0)] content; } class Ct1 extends Ct0 {   [map(st1)] a1; } </pre>

	<pre>&lt;extension base="st0"&gt;   &lt;attribute name="a1" type="st1" /&gt; &lt;/extension&gt; &lt;simpleContent&gt; &lt;/complexType&gt;</pre>	<pre>}</pre>
--	--------------------------------------------------------------------------------------------------------------------------------------------------	--------------

**Parent topic:** [Schema types](#)

**Related information:**  
[Complex type extension](#)  
[Complex type restriction](#)

## Extension of simple type content in complex types

You can extend simple type content in a schema and an XML document.

The example extends temperature of base type `int` to include a `unit` attribute of type `string`.

Here is the schema:

```
<complexType temperature>
 <simpleContent>
 <extension base="int">
 <attribute name="unit" type="string"/>
 </extension>
 </simpleContent>
</complexType>
```

Here is the excerpt from an XML document:

```
<temperature unit= "celsius">15</temperature>
```

Here is the XOM representation:

```
Class Temperature extends IlrXmlObject
{
 int content ;
 String unit ;
 ...
}
```

**Parent topic:** [Schema types](#)

**Related information:**

[Complex types](#)

[Built-in simple types](#)

# Complex type restriction

The XOM takes complex type restrictions into account in the XOM.

In the following example, the complex type `book` represents a standard book and the complex type `old-book` is a restriction because a lexical pattern constraint (`pattern value = "[0-9]{2} - [A-Z]{5} - [0-9]"`) has been added. The example includes a rule named `processOldBook` that matches “Hamlet” with the title of an `OldBook` object.

**Note:**  
  
The `OldBook` dynamic class does not redefine nor restrict the `Book` class attributes, even if the `ident` field is more restricted.

Here is the schema:

```
<complexType name= "book">
 <sequence>
 <element name="ident" type="string"/>
 <element name="title" type="string" />
 </sequence>
</complexType>

<complexType name="old-book">
 <complexContent>
 <restriction base="book">
 <sequence>
 <element name="ident">
 <simpleType>
 <restriction base="string">
 <pattern value="[0-9]{2} - [A-Z]{5} - [0-9]">
 </restriction>
 </simpleType>
 </element>
 </sequence>
 </restriction>
 </complexContent>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
 ...
 String ident;
 String title;
 ...
}
```

```
class OldBook extends Book
{
 ...
}
```

And here is the rule:

```
rule processOldBook
{
 when {
 OldBook (title equals "Hamlet");
 }
 then {
 ...
 }
}
```

**Parent topic:** [Schema types](#)

**Related information:**  
[Complex type extension](#)  
[Complex types](#)

## Complex type extension

The XOM takes complex type extensions into account.

### Extending a complex type by adding an attribute

In this example, the complex type book has been extended to computer-book by adding a new attribute named language. The example includes a rule named processComputerBook that matches “Java” with the title of a ComputerBook object.

Here is the schema:

```
<complexType name= "book" >
 <sequence>
 <element name="ident" type="string"/>
 <element name="title" type="string" />
 </sequence>
</complexType>
```

```
<complexType name="computer-book">
 <complexContent>
 <extension base="book">
 <sequence>
 <element name="language" type="string"/>
 </sequence>
 </extension>
 </complexContent>
</complexType>
```

Here is an excerpt from an XML document:

```
<computer-book>
 <ident>12345</ident>
 <title>Decision Server Reference Manual</title>
 <language>Java IRL</language>
</computer-book>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
 ...
 String ident;
 String title;
 ...
}
```

```
class ComputerBook extends Book
{
 String language;
 ...
}
```

And here is the rule:

```
rule processComputerBook
{
 when {
 ComputerBook (language equals "Java");
 }
 then {
 ...
 }
}
```



**Parent topic:** [Schema types](#)

**Related information:**

[Complex type restriction](#)

[Complex types](#)

[Extension of simple type content in complex types](#)

## Local complex types mapped to inner classes

Local complex types map to inner classes.

**Note:**

The inner class policy is activated by default.

This example shows how to map embedded local complex types onto inner classes.

The example includes a rule named `processBorrowedBook` that matches “Smith” with the attribute `surname` of an instance of the inner class `borrower`.

Here is the schema:

```
<complexType name="borrowed-book">
 <sequence>
 <element name="ident" />
 <element name="borrower">
 <complexType>
 <sequence>
 <element name="name">
 <element name="surname">
 </sequence>
 </complexType>
 </element>
 </sequence>
 </complexType>
 </element>
 </sequence>
</complexType>
```

Here is the excerpt from an XML document:

```
<borrowed-book>
 <ident>12345</ident>
 <borrower>
 <name>John</name>
 <surname>Smith</surname>
 </borrower>
</borrowed-book>
```

Here is the XOM representation:

```
class BorrowedBook extends IlrXmlObject
{
 class Borrower extends IlrXmlObject
 {
 String name;
 String surname;
 };
 String ident;
 Borrower borrower;
}
```

And here is the rule:

```
rule processBorrowedBook
{
 when {
 b: BorrowedBook ();
 BorrowedBook.Borrower (surname equals "Smith") from b.borrower;
 }
 then {
 ...
 }
}
```

**Parent topic:** [Schema types](#)

**Related information:**  
[Complex types](#)  
[Complex type extension](#)  
[Map function](#)

## Default constructor dynamic methods

To map default constructors, a method is generated for each dynamic class.

### Mapping of default constructors

This example shows how to map a default constructor. A method is generated for each dynamic class. This method sets the attributes with their default values, if they exist. The following example includes a function that adds an instance of a dynamic class Book, with the default attribute title equal to None.

Here is the schema:

```
<complexType name="book">
 <sequence>
 <element name="ident"/>
 <element name="title" minOccurs="0" default="None" />
 </sequence>
</complexType>
```

Here is the XOM representation:

```
Class Book extends IlrXmlObject
{
 String ident ;
 String title ;
 Book (); // default constructor
}
```

And here is a function in IRL:

```
function void insertBook() {

 // Add a new book, default value: (ident = null, title="None")
 insert (new Book());
}
```

**Parent topic:** [Schema types](#)

**Related concepts:**  
[Functions](#)

**Related tasks:**  
[Creating functions](#)

**Related reference:**  
[function](#)

## Type identifier mapping

Simple type names map to XOM class names according to a number of rules.

Simple type name map to XOM class names according to the following rules:

- The first type letter is translated into lowercase.
- All characters that could be contained in a Java™ identifier are translated without change.
- The other characters are discarded and the next letter is translated to uppercase.
- If the resulting class name is already used as a class or simple type identifier, a numeric suffix is appended to the name. For example, name becomes name\_0.

Following these rules, zip-code becomes zipCode.

**Parent topic:** [Schema types](#)

**Related information:**

[Simple type mapping](#)

## XML declarations

XML elements, attributes, groups, and appinfo structures map to dynamic classes.

### Groups

Groups can be choice groups, composite groups, or collection groups. They are mapped using complex types and specific dynamic methods.

### appinfo

appinfo structures are mapped to XOM properties.

### Translation of schema member names to dynamic class field names

Schema member names translate to dynamic class field names according to a number of rules.

**Parent topic:** [Mapping between XML schema and dynamic classes](#)

## Groups

Groups can be choice groups, composite groups, or collection groups. They are mapped using complex types and specific dynamic methods.

Group mapping is presented in three subsections:

- [Choice groups](#)
- [Composite groups](#)
- [Collection groups](#)

In each sections, an example is used to demonstrate the various types of mapping. The examples include a schema, an XML document, and the XOM representation. The examples of choice group mapping and composite group mapping also include the rules to help clarify the explanation.

### Choice groups

A choice group is an XSD group of elements. Only one of them is shown in the XML document related to the group schema.

Additional dynamic methods are generated to help handle the choice group. For example, the dynamic method `getChoice` returns the dynamic attribute value selected in a group, and the dynamic method `getChosenAttribute` returns the dynamic attribute name selected in a group. The `String` parameter helps to locate the group when there are multiple groups in a complex type. The selected group is identified by the XOM name of a field of one of its children. When the method is used with no argument, the first choice is selected by deep search.

#### Example of choice group mapping

The following example uses the complex type `observation` which contains three elements: `temperature`, `pressure`, and `velocity`.

Here is the schema:

```
<complexType name="observation">
 <choice>
 <element name="temperature" type="double"/>
 <element name="pressure" type="double"/>
 <element name="velocity" type="double"/>
 </choice>
</complexType>
```

Here is the excerpt from an XML document:

```
<observation>
 <pressure>10.0</pressure>
</observation>
```

Here is the XOM representation:

```
class Observation extends IlrXmlObject
{
 double temperature ;
 double pressure ;
 double velocity ;
 ...
 // choice dynamic methods
 Object getChoice (String attrName) ;
 String getChosenAttribute (String attrName);
 Object getChoice();
 String getChosenAttribute();
}
```

And here are the rules:

```
rule findTemperatureObservation
{
 when {
```

```

 Observation (temperature isknown);
 }
 then {
 ...
 }
}

```

```

rule processObservationValue
{
 when {
 obs: Observation ();
 v: Double () from obs.getChoice ();
 attr: String () from obs.getChosenAttribute ();
 }
 then {
 ...
 }
}

```

## Composite groups

A composite group is a combination of elements and groups.

### Note:

The W3C specification does not define the concept of a composite complex type. This concept has been introduced here to describe a new mapping state and differentiate it from the sequence or choice groups.

### Example of composite group mapping

The following example reuses the same complex type observation already used in [Choice groups](#). For the purpose of mapping a composite group, one more choice group has been added to the observation type. This additional choice group has two options: error and comment.

Here is the schema:

```

<complexType name="observation">
 <sequence>
 <choice>
 <element name="temperature" type="double"/>
 <element name="pressure" type="double"/>
 <element name="velocity" type="double"/>
 </choice>
 <choice>
 <element name="error" type="double"/>
 <element name="comment" type="string" />
 </choice>
 </sequence>
</complexType>

```

Here is the excerpt from an XML document:

```

<observation>
 <pressure>10</pressure>
 <error>0.2</error>
</observation>

```

Here is the XOM representation:

```

class Observation extends IlrXmlObject
{
 double temperature ;
 double pressure ;
 double velocity ;
 double error;
 String comment;
}

```



```

...
// choice dynamic methods
Object getChoice (String attrName) ;
String getChosenAttribute (String attrName);
}

```

And here are the rules:

```

rule findTemperatureWithErrorObservation
{
 when {
 Observation (temperature isknown; error isknown);
 }
 then {
 ...
 }
}

```

```

rule processObservationValue
{
 when {
 obs: Observation ();
 v: Double () from obs.getChoice ("velocity");
 attr: String () from obs.getChosenAttribute ("velocity");
 }
 then {
 ...
 }
}
rule displayErrorOrComment
{
 when {
 obs: Observation ();
 c: Object () from obs.getChoice ("error");
 attr: String () from obs.getChosenAttribute ("error");
 }
 then {
 out.println (attr + '=' + c.toString());
 }
}

```

## Collection groups

The following example demonstrates how to map a collection of choice groups. This example reuses again the complex type observation already used in the [Choice groups](#) and [Composite groups](#) sections. The collection of the choice group is defined in the complex type with maxOccurs="unbounded".

Here is the schema:

```

<complexType name="observations">
 <choice maxOccurs="unbounded">
 <element name="temperature" type="double"/>
 <element name="pressure" type="double"/>
 <element name="velocity" type="double"/>
 </choice>
</complexType>

```

Here is the excerpt from an XML document:

```

<observations>
 <velocity>12.3</velocity>
 <temperature>11.5</temperature>
 <velocity>9.2</velocity>
</observations>

```

And here is the XOM representation:

```
class Observations extends IlrXmlObject
{
 Vector temperatureList;
 Vector pressureList;
 Vector velocityList;
 ...
}
```

**Parent topic:** [XML declarations](#)

## appinfo

appinfo structures are mapped to XOM properties.

XML schema appinfo structures are automatically mapped to XOM properties when the name is equal to “http://www.ilog.com/rules/xml”.

### Example of appinfo mapping

The following example shows how a simple appinfo structure is translated in the XOM.

Here is the schema:

```
<appinfo xmlns:irl="http://www.ilog.com/rules/xml">
 <irl:property name="property1">property1Value</irl:property>
</appinfo>
```

Here is the XOM representation:

```
property property1 "property1Value"
```

### appinfo as a class property

The following example demonstrates how to create class properties using appinfo structures. This example maps the complex type book, which has the bookRenderer and position properties, to a class named Book.

#### Example of mapping appinfo as a class property

Here is the schema:

```
<complexType name="book">
 <annotation>
 <appinfo xmlns:irl="http://www.ilog.com/rules/xml">
 <irl:property name="bookRenderer">Renderer1</irl:property>
 <irl:property name="position">
 <irl:property name="x">10</irl:property>
 <irl:property name="y">20</irl:property>
 </irl:property>
 </appinfo>
 </annotation>
 <sequence>
 <element name="ident"/>
 <element name="title"/>
 </sequence>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
 property bookRenderer "Renderer1"
 property position {
 x "10",
 y "20"
 }
 String ident;
 String title;
}
```

### appinfo as a field property

The following example demonstrates how to create field properties using appinfo structures. This example maps the complex type book, which has the bookRenderer and position properties, to a field named ident.

Here is the schema:

```
<complexType name="book">
 <sequence>
```

```
<element name="ident">
 <annotation>
 <appinfo xmlns:irl="http://www.ilog.com/rules/xml">
 <irl:property name="bookRenderer">Renderer1</irl:property>
 <irl:property name="position">
 <irl:property name="x">10</irl:property>
 <irl:property name="y">20</irl:property>
 </irl:property>
 </appinfo>
 </annotation>
</element>
<element name="title"/>
</sequence>
</complexType>
```

Here is the XOM representation:

```
class Book extends IlrXmlObject
{
 String ident
 property bookRenderer "Renderer1"
 property position {
 x "10",
 y "20"
 };
 String title;
}
```

**Parent topic:** [XML declarations](#)

## Translation of schema member names to dynamic class field names

Schema member names translate to dynamic class field names according to a number of rules.

A schema member name is translated into a dynamic class field name according to the following rules:

- The first type letter is translated into lowercase.
- All characters that could be contained in a Java™ identifier are translated without change.
- The other characters are discarded and the next letter is translated into uppercase.
- If the resulting member name is already used as a member identifier, a numeric suffix is appended to that name. For example, field becomes field\_0.

Following these rules, my-address becomes myAddress.

**Parent topic:** [XML declarations](#)

## Schema-related markup in XML documents

Schema-related markup in XML documents includes `xsi:nil`, `xsi:type`, `xsi:schemaLocation`, and `xsi:noNamespaceSchemaLocation`.

The schema-related markup in XML documents includes the following elements:

- [xsi:nil](#)
- [xsi:type](#)
- [xsi:schemaLocation and xsi:noNamespaceSchemaLocation](#)

### **xsi:nil**

When an XML element is set to `xsi:nil` in the XML document instance, its related dynamic class field is set to null. The `isunknown` operator applied on this field returns `true`. In other words, when an XML object is serialized via the `driver.writeObject` method, the following information is added to this object:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

### **Example of mapping xsi:nil in an XML document**

The following example demonstrates the mapping of `xsi:nil` in an XML document. The rule `detectPersonWithMissingAddress` uses the `isunknown` keyword to find a person whose address is not known.

In this example:

- `xsi` is the namespace name
- `http://www.w3.org/2001/XMLSchema` instance is the URL
- `nil` is one element of the namespace

Here is the schema:

```
<element name="person">
 <complexType>
 <sequence>
 <element name="surname"/>
 <element name="address" nillable="true"/>
 </sequence>
 </complexType>
</element>
```

Here is the excerpt from an XML document:

```
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <surname>Smith</surname>
 <address xsi:nil="true"/>
</person>
```

And here is the rule:

```
rule detectPersonWithMissingAddress
{
 when {
 Person (address isunknown);
 }
 then {
 ...
 }
}
```

XML namespaces are designed to provide universally unique names for elements and attributes. Developers can use them to carry out the following actions:

- Combine fragments from different documents without any naming conflicts.

- Write reusable code modules that can be called for specific elements and attributes. Universally unique names guarantee that such modules are called only for the correct elements and attributes.
- Define elements and attributes that can be reused in other schemas without any risk of name conflicts.

## **xsi:type**

To refine the type of an element in the XML document, you can use xsi polymorphism. This feature substitutes a subtype for the standard type of the element.

### **Example of mapping xsi:type in an XML document**

The following example demonstrates how to use xsi polymorphism. The example uses a schema and an excerpt from an XML document.

Here is the schema:

```
<element name="person" type="person"/>
<complexType name=" person " >
 <sequence>
 <element name="name">
 <element name="surname">
 </sequence>
</complexType>
```

```
<complexType name=" inhabitant ">
 <complexContent>
 <extension base="person">
 <sequence>
 <element name="address"/>
 </sequence>
 </extension>
 </complexContent>
</complexType>
```

And here is an excerpt from an XML document:

```
<person xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:type="inhabitant">
 <name>John</name>
 <surname>Smith</surname>
 <address>123 Downing Street, London</address>
</person>
```

## **xsi:schemaLocation and xsi:noNamespaceSchemaLocation**

These functions are supported by the parser. However, the resulting information is not used in XML binding.

**Parent topic:** [Mapping between XML schema and dynamic classes](#)

## Schema mapping limitations

Some features or keywords are not mapped by the current XML binding and the value types of some schema types are lost.

### Unmapped schema features and keywords

Some schema features or keywords are not mapped by the current XML binding. These keywords are correctly parsed but they are not processed and not translated into XOM dynamic classes or XML objects:

- `key`, `keyref`, `unique`
- `abstract`, `block`
- `blockDefault`, `finalDefault`
- `any`, `anyAttribute`
- `skip`, `lax`, `strict`
- `processContents`

### Lost value types

The following schema types are mapped without errors to `java.lang.Object` by the current XML binding but the original value type is lost at runtime:

- `anySimpleType`
- `anyType`

If you want to process such elements with their original value types (for example, by assignation), you need to cast them to your required value.

**Parent topic:** [Mapping between XML schema and dynamic classes](#)



## Updating the BOM when the XOM changes

How you manage XOM changes in the BOM depends on whether your rule project references a Java™ project as a XOM.

### Managing changes to the XOM from a Java project

If your rule project references a Java project as a XOM, you can update the BOM from the Java project.

### Managing changes to the XOM from a rule project

If your rule project does not reference a Java Project as execution object model (XOM), you can update the BOM from the rule project.

### Configuring BOM updates to ignore differences

You can configure the BOM update feature to selectively ignore certain differences between the BOM and the XOM.

**Parent topic:** [Designing business object models](#)

## Managing changes to the XOM from a Java project

If your rule project references a Java™ project as a XOM, you can update the BOM from the Java project.

### About this task

Update the BOM directly from the Java project. The rule project must reference a Java project as Execution Object Model (XOM).

When you rename an element in a Java XOM, the Rename Compilation Unit dialog displays the impact on the related BOM.

### Procedure

Accept the changes in the Rename Compilation Unit dialog. The BOM and its associated vocabulary are automatically updated. Any reference to a XOM class in the BOM to XOM mapping section of a business element is also updated.

**Parent topic:** [Updating the BOM when the XOM changes](#)

## Managing changes to the XOM from a rule project

If your rule project does not reference a Java™ Project as execution object model (XOM), you can update the BOM from the rule project.

### About this task

Update the BOM from the rule project.

### Procedure

To update the BOM:

1. In the Rule Explorer, in the bom folder, right-click the BOM entry you want to update and select **BOM Update**.
2. In the **BOM Update** view, do the suggested actions required to update your BOM.

### Results

Changes to the XOM are taken into account in the BOM. Any elements you added to the BOM when extending it remain as they are. If they have a BOM to XOM mapping, the mapping is updated. Members that have been modified in the XOM, through a change of name or a change to their arguments, are added as new business elements in the BOM.

**Parent topic:** [Updating the BOM when the XOM changes](#)

### Related information:

[Configuring BOM updates to ignore differences](#)

## Configuring BOM updates to ignore differences

You can configure the BOM update feature to selectively ignore certain differences between the BOM and the XOM.

If you add a `.cfg` file with new properties for the BOM file being updated, you can filter the differences that are detected when running a BOM update. That is, the BOM update feature filters out the differences that you do not want to display. However, ignoring the differences between the BOM and the XOM does not necessarily prevent them from being modified. The BOM methods and attributes can still be updated in action. For example, if the attribute that you ignore is the only difference between the BOM and the XOM class, then the BOM update feature does not propose an action. However, if you change another attribute in the same BOM class, the BOM update action updates the entire class, including the attributes that you choose to ignore.

The `.cfg` file should be placed in the same directory as the `.bom` file to which it applies. The name of the file must be identical to the `.bom` file, but with a `.cfg` file extension.

Values are comma-separated globs. Each glob must be preceded by either a plus "+" sign (inclusion) or a minus "-" sign (exclusion). The order in which the globs are shown is important. Globs are converted to REGEX by replacing "\*" and "?" with JRE REGEX equivalents, and "." and "\" are escaped. All other characters are not converted, and are shown in the REGEX as written. The REGEX is prefixed with a "^" and is appended with a "\$".

The following example provides an annotated `.cfg` file:

```
/** This property ignores the following missing BOM elements:
 * + the 'ignore' package and all sub-packages
 * - except for the ignore.exception package and all sub-packages
 * + the XOM class 'bank.Loan'
 * + all classes in the 'branch' package as well as all sub-packages
 * + the method 'bank.Customer.doSomething' taking a java.lang.String
 * + all classes with 'test' in their package name
 * + the attribute bank.Customer.age
 * + classes starting with the name 'Test' followed by any 3 characters
 */
update.ignoreMissingBomElement = +ignore.,\
-ignore.exception.*,\\
+bank.Loan,\\
+branch.*,\\
+bank.Customer.doSomething([Ljava.lang.String[]),\\
+*.test.*,\\
+bank.Customer.age,\\
+*.Test???
```

```
/** This property ignores the following missing XOM elements:
 * + the BOM class 'new.BomClass'
 */
update.ignoreMissingXomElement = +new.BomClass
```

```
/** This property ignores the following differences between BOM and XOM
classes:
 * + all attributes on 'bank.Customer'
 * - except for the bank.Customer.name attribute
 */
update.ignoreDifferences = +bank.Customer.,-bank.Customer.name
```

**Parent topic:** [Updating the BOM when the XOM changes](#)

**Related tasks:**

[Managing changes to the XOM from a rule project](#)

# Configuring the BOM for rule authoring

After designing a BOM for an underlying object model, you configure it to maximize the efficiency of rule authoring activities.

## Overview: A data model for rule authoring

You use a vocabulary to write rules. You define the vocabulary in the business object model (BOM).

## Business object model (BOM)

The BOM is the object model for business rules. You edit BOM members in the BOM editor.

## Creating BOM elements

You can extend the business object model (BOM) with new business elements, and manage values in the editor with specific classes.

## Defining a vocabulary

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model.

## Working with categories

A category is an identifier that you can assign to business rules and certain business elements to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. The rule editor drop-down list shows only the business elements that a rule can see.

## Working with domains

You can use domains to extend the business object model (BOM).

## Translating an existing business vocabulary

To translate an existing vocabulary that is defined on the BOM, you can copy the vocabulary file or edit the verbalizations.

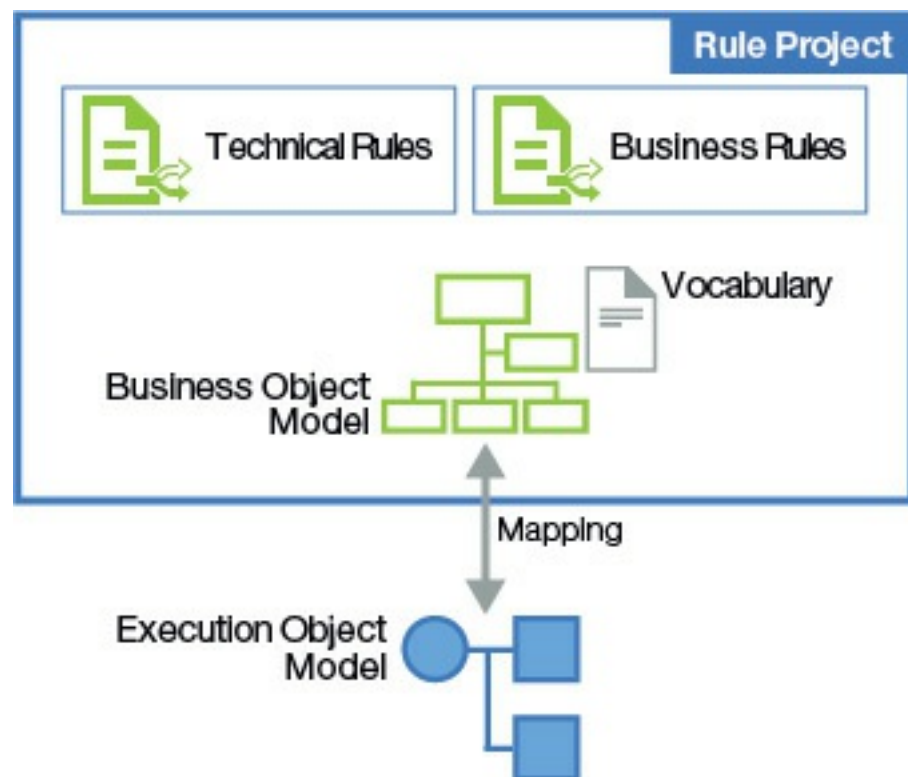
**Parent topic:** [Designing projects for rule authoring](#)

## Overview: A data model for rule authoring

You use a vocabulary to write rules. You define the vocabulary in the business object model (BOM).

The vocabulary defines what policy managers can write business rules about. The elements of the vocabulary are defined in a Business Object Model (BOM).

You can define the BOM from scratch or create it from an Execution Object Model (XOM) that references compiled Java™ classes and other data sources. You can extend the BOM without modifying the XOM. The XOM describes quite extensively the elements that you need for rule editing



You can define a BOM in one of two ways:

- **Bottom-up:** In this case, you create a BOM entry from the XOM. If you find that you require additional elements in the BOM to facilitate rule editing for policy managers, you can extend the BOM by creating business elements and mapping them to the XOM.
- **Top-down:** In this case you create business elements without necessarily considering the way that the execution elements are going to be implemented. At a later date, when you implement the execution elements, you can specify a mapping for all your business elements.

The rule project contains:

- Business rules, which express a business policy. Business rules have a list of conditions to meet before doing a list of actions. You write business rules using the Business Action Language (BAL).
- Technical rules, which comprise a *condition* part and an *action* part. The condition part binds variables to objects and attribute values and specifies tests on attribute values. The action part specifies the actions to be carried out if the rule is executed. You write technical rules in Rule Designer.

**Parent topic:** [Configuring the BOM for rule authoring](#)

### Related concepts:

[Overview: BOM and execution object model \(XOM\)](#)

[Vocabulary](#)

[Action rules](#)

### Related information:

[Business object model \(BOM\)](#)

[Technical rules](#)

## Business object model (BOM)

The BOM is the object model for business rules. You edit BOM members in the BOM editor.

### Introducing the business object model (BOM)

You use the BOM to make business rule editing user-friendly by providing tools to set up a natural language vocabulary. With this vocabulary, policy managers can describe their business logic in a business rule language.

### Collections

In the BOM, arrays and collections represent a set of objects. In the BAL, you must set a collection domain on a collection of type `java.util.Collection`.

**Parent topic:** [Configuring the BOM for rule authoring](#)

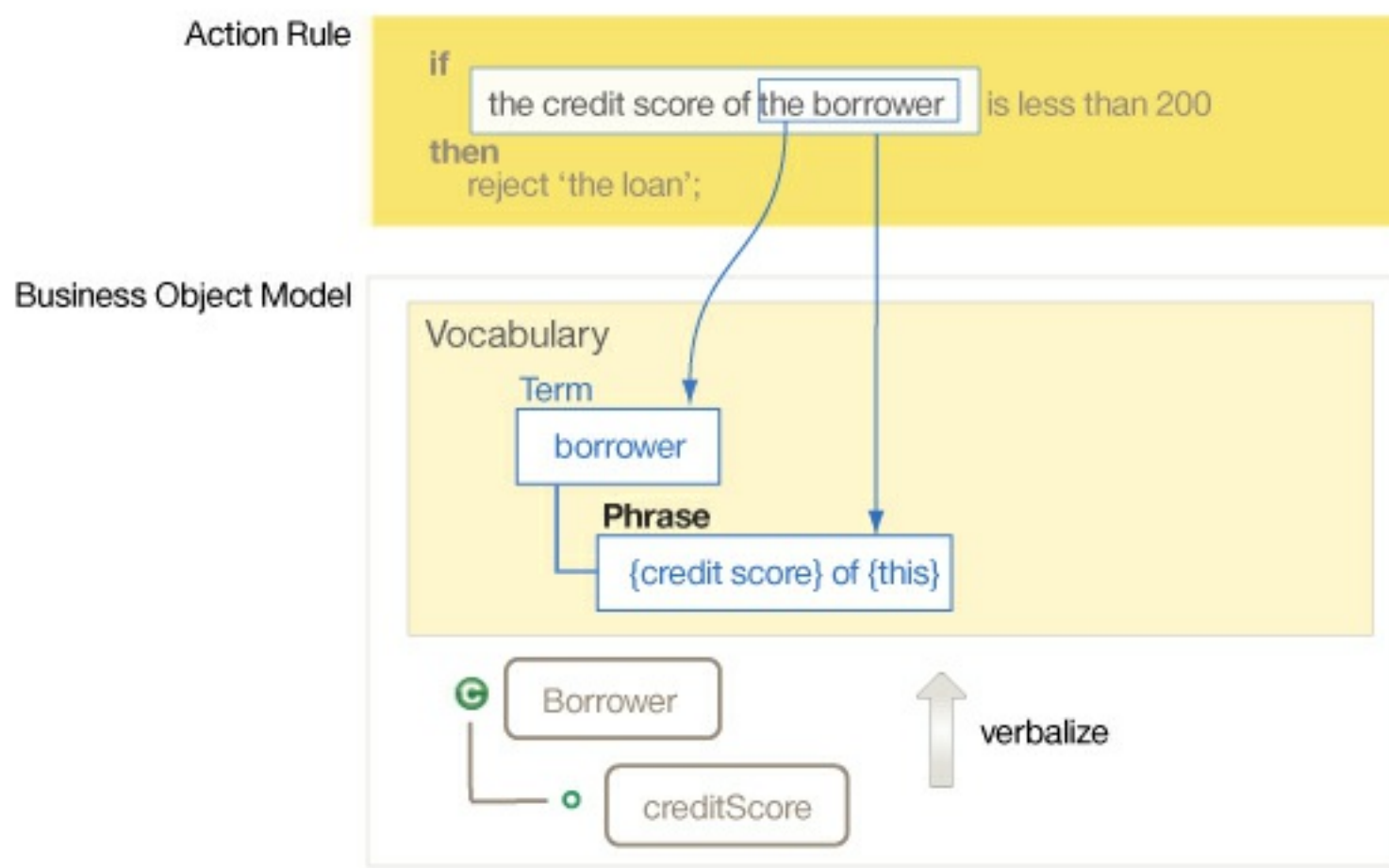
## Introducing the business object model (BOM)

You use the BOM to make business rule editing user-friendly by providing tools to set up a natural language vocabulary. With this vocabulary, policy managers can describe their business logic in a business rule language.

The BOM is the basis for the vocabulary used in business rules. It is an object model similar to a Java™ object model, and contains elements that map to those of the XOM.

A BOM contains the classes and methods that rule artifacts act on. As an object model, the BOM is very similar to a Java object model. It consists of classes grouped into packages. Each class has a set of attributes, methods and, possibly, other nested classes.

BOM-to-XOM mapping defines the correspondence between the BOM and the execution object model (XOM) used at runtime.



### System BOM

By default, the BOM always includes classes that map to specific JDK classes, and basic date and time-related classes. This set of classes is called the System BOM. For example, to compare the parts of a date, the System BOM contains the following classes, which map to the parts of a `java.util.Date` and have associated value editors:

- `ilog.rules.brl.SimpleDate`
- `ilog.rules.brl.Time`
- `ilog.rules.brl.DayOfWeek`
- `ilog.rules.brl.Month`
- `ilog.rules.brl.Year`

If you have a BOM member of type `java.util.Date`, you can change that type to one of the System BOM date types. The mapping is carried out automatically.

### BOM entries

A business object model comprises one or more BOM entries. A BOM entry defines a set of business elements in the business object model.

You can order BOM entries so that if you have two business elements with the same name in two BOM entries, the one in the first BOM entry in the path overrides the other.

A BOM entry comprises:

- A BOM file, which describes the structure of the BOM
- A VOC file, which is locale-specific and describes the vocabulary associated to the BOM
- A B2X file, which describes the mapping between the BOM and the XOM



**Parent topic:** [Business object model \(BOM\)](#)

**Related concepts:**

[Overview: BOM and execution object model \(XOM\)](#)

[BOM properties](#)

[Collections](#)

[Vocabulary](#)

**Related tasks:**

[Defining and assigning categories](#)

[Setting up automatic variables](#)

**Related information:**

[Overview: A data model for rule authoring](#)

[Object state update](#)

[Creating BOM entries](#)

[Creating BOM elements](#)

[Defining a vocabulary](#)

## Collections

In the BOM, arrays and collections represent a set of objects. In the BAL, you must set a collection domain on a collection of type `java.util.Collection`.

In the BOM, collections are members that are semantically linked to a BOM class with a one-to-many relationship.

Collections are shown in business rules with BAL operators and constructs such as:

- `<an object>` is one of `<objects>`
- the number of `<objects>` in `<objects>`

In the following example, the action rule uses collections:

```
definitions
set 'company1' to a company;
set 'customer1' to a customer in the employees of 'company1';
if
 the number of books in the items of the shopping cart of 'customer1' is more
 than 5
...
```

You can represent a set of objects in the BOM in two ways:

- Array (`java.lang.Object[]`)
- Collection (`java.util.Collection`)

Both are mapped to the concept of "collection of objects" in the BAL. From a BAL perspective, a collection is an element with multiple-cardinality. From a BOM perspective, it is an array, or a BOM member with a collection domain.

A collection is treated as a "collection of objects" in the BAL only if it has a collection domain. A collection domain specifies the cardinality and the type of collection elements, for example, `0,* class Customer`.

### Note:

Ruleset parameters or variables of type or subtype `java.util.Collection` are treated as "collection of objects" in the BAL.

When you edit a business rule using the Content Assist box in designer, you can access the constructs relative to collections if the member is of type:

- array
- `java.util.Collection` and has a collection domain.

### Collections of array type

Members (attributes, methods, constructors) of array type are automatically treated as collections.

For example, in the following BOM class, the method `getEmployees` is automatically treated in business rules as a collection of `Employee` objects.

```
public class Company {
 Employee[] getEmployees();
}
```

### Collections of `java.util.Collection` type

In the BAL, a member of type `java.util.Collection` is not treated as a collection until you set a collection domain.

For example, in the following BOM class, a collection domain is set and defined.

```
public class Company {
 java.util.Collection getEmployees() domain 0, * class Employee;
}
```

Use the BOM editor to set a domain on a member of type `java.util.Collection`. For information about

defining domains, refer to [Defining domains](#).

**Parent topic:** [Business object model \(BOM\)](#)

**Related concepts:**  
[Domains](#)

**Related tasks:**  
[Creating packages and classes](#)

## Creating BOM elements

You can extend the business object model (BOM) with new business elements, and manage values in the editor with specific classes.

### [Creating packages and classes](#)

Extend the BOM with new business elements or properties.

### [Adding nested classes](#)

You can enclose a class in an existing class.

### [Adding members to a class](#)

In the BOM editor, you can add members to a class.

**Parent topic:** [Configuring the BOM for rule authoring](#)

# Creating packages and classes

Extend the BOM with new business elements or properties.

## About this task

You can extend the business object model (BOM) with new business elements or new properties on BOM elements. You can also manage values in the BOM editor by developing specific classes and referencing them in BOM custom properties.

You start by adding a package to a BOM entry, and then adding a class to the BOM package.

## Procedure

To add a package to a BOM entry and add a class to the package:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in your BOM entry.

2. Click **New Package**.
3. In the New BOM Package wizard, in the **Name** field, specify the fully qualified name of the package you want to create.
4. Click **Finish**.

The BOM entry now contains the new package. You must now add a class to the BOM package.

5. To add a class, in the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in the BOM entry.

6. Click **New Class**.
7. In the New BOM Class wizard, in the **Package** field, specify the fully qualified name of the package in which you want to create the class.
8. In the **Name** field, specify the name of the class you want to create.
9. Click **Finish**.

## Results

You now have a new BOM package, with a new class added to it.

### Note:

When using the Intellirule editor to write rules, you can create business elements on the fly. If you use a new element in a rule, the Intellirule editor detects that it must be added to the BOM and displays a Quick Fix message so that you can add it automatically. See [Correcting errors by using Quick Fix](#).

**Parent topic:** [Creating BOM elements](#)

### Related tasks:

[Adding nested classes](#)

[Adding members to a class](#)

### Related information:

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

[Creating a BOM without a XOM](#)

## Adding nested classes

You can enclose a class in an existing class.

### About this task

You can add a new class in an existing class to create a nested class.

### Procedure

To add a nested class:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.

The Package tab of the BOM Editor displays the list of packages and classes in your BOM entry.

2. Click **New Class**.
3. In the New BOM Class wizard, select the **Enclosing class** box and then click **Browse** to select the enclosing class.
4. In the **Name** field, specify the name of the class you want to create.
5. Click **Finish**.

### Results

The new class is added as a nested class.

**Parent topic:** [Creating BOM elements](#)

### Related tasks:

[Creating packages and classes](#)

[Adding members to a class](#)

# Adding members to a class

In the BOM editor, you can add members to a class.

## About this task

Add members such as attributes, constructors, or methods to a class.

## Procedure

To add a member to a class:

1. In the Outline view, expand the BOM entry and click the class to which you want to add members.
2. In the Class tab of the BOM Editor, in the Members section, click **New**.
3. In the New Member wizard, select whether the new member is an attribute, a constructor, or a method.
4. Specify its name, type, and arguments if any.
5. Click **Finish**.

## Results

The Members section displays the new member.

**Parent topic:** [Creating BOM elements](#)

### Related tasks:

[Creating packages and classes](#)

[Adding nested classes](#)

### Related information:

[Business object model \(BOM\)](#)

[Creating BOM entries](#)

## Defining a vocabulary

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model.

### Vocabulary

The vocabulary comprises terms and phrases that are attached to the elements of the BOM, and terms that are defined for ruleset variables and ruleset parameters. You use the vocabulary to create business rules.

### Defining the vocabulary for a BOM entry

You can define a vocabulary for a complete BOM entry.

### Editing terms

You edit terms to change, for example, the singular or plural form of a term.

### Documenting terms

You document business terms to provide guidance to policy managers.

### Editing and creating phrases

You can create a new phrase, edit a phrase, including the subject of a phrase, and use a reduced verbalization for simpler rules.

### Editing the verbalization of constants

Constants correspond to static references in the BOM. You can change the verbalization of constants.

### Editing the verbalization of ruleset variables

The default verbalization of a ruleset variable is its name. You can edit this verbalization.

### Editing the verbalization of ruleset parameters

The default verbalization of a ruleset parameter is its name. You can edit this verbalization.

**Parent topic:** [Configuring the BOM for rule authoring](#)



# Vocabulary

The vocabulary comprises terms and phrases that are attached to the elements of the BOM, and terms that are defined for ruleset variables and ruleset parameters. You use the vocabulary to create business rules.

Rule Designer verbalizes BOM elements to make them visible in business rules. Rule Designer creates a default vocabulary that can be translated.

When you create a BOM entry, you can choose to verbalize the business elements to create a vocabulary. A default verbalization is then applied to all attributes, getters, setters, and static references in the BOM entry. You can create a code-like verbalization of all other methods by selecting the option **All Methods** in the New BOM Entry or Verbalize BOM wizards.

In addition to vocabulary elements created from the BOM, you create vocabulary terms by defining a verbalization for ruleset parameters and ruleset variables.

Only business rules use the vocabulary. Technical rules and functions do not use the vocabulary. If you do not verbalize a business element, that element is not part of the vocabulary, and therefore not visible from business rules.

You can translate the vocabulary, and use several vocabularies in different languages on top of the same business object model.

## **Vocabulary elements**

A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.

## **Phrase templates**

A phrase template is a pattern for the verbalization of phrases. The pattern changes for navigation, predicate, or action phrases.

## **Default verbalization**

Rule Designer applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization in the BOM Editor.

## **Vocabulary errors and warnings**

If you have vocabulary ambiguities, missing placeholders, or problems on business elements, Rule Designer raises errors and warnings on terms and phrases at editing or build time.

**Parent topic:** [Defining a vocabulary](#)

### **Related concepts:**

[Categories](#)

[Vocabulary elements](#)

[Phrase templates](#)

### **Related tasks:**

[Defining and assigning categories](#)

### **Related information:**

[Vocabulary errors and warnings](#)

[Overview: Ways to express business rules](#)

[Business object model \(BOM\)](#)

[Defining a vocabulary](#)

# Vocabulary elements

A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.

Whether a vocabulary element is a business term, a phrase, or a constant depends on the nature of its corresponding business element.

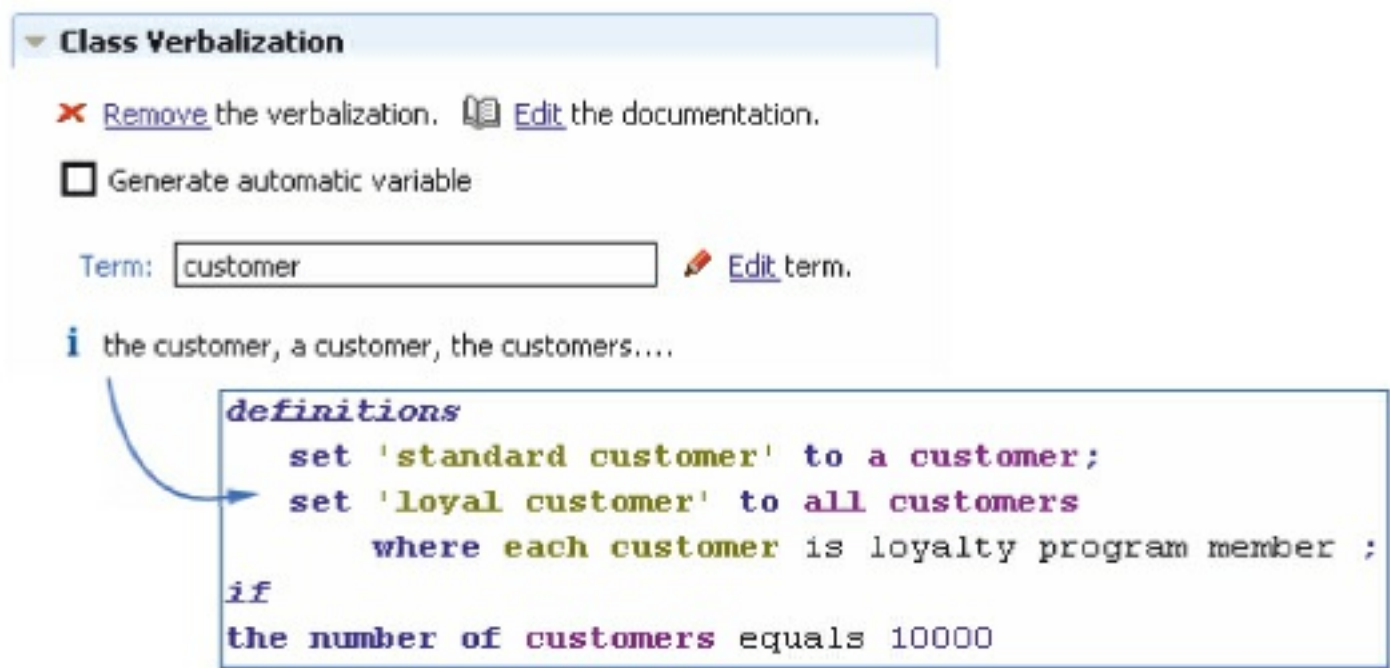
The drop-down lists of business rule editors show vocabulary elements.

**Important:**  
  
If you change the default verbalization for a vocabulary element, avoid using parentheses ( ). These characters are considered as delimiters and can cause false difference, overlap, or gap warnings in decision tables.

## Business term

A business term is a key concept handled in business rules. It is the verbalization of a class or nested class in the BOM.

For example, customer is a business term.



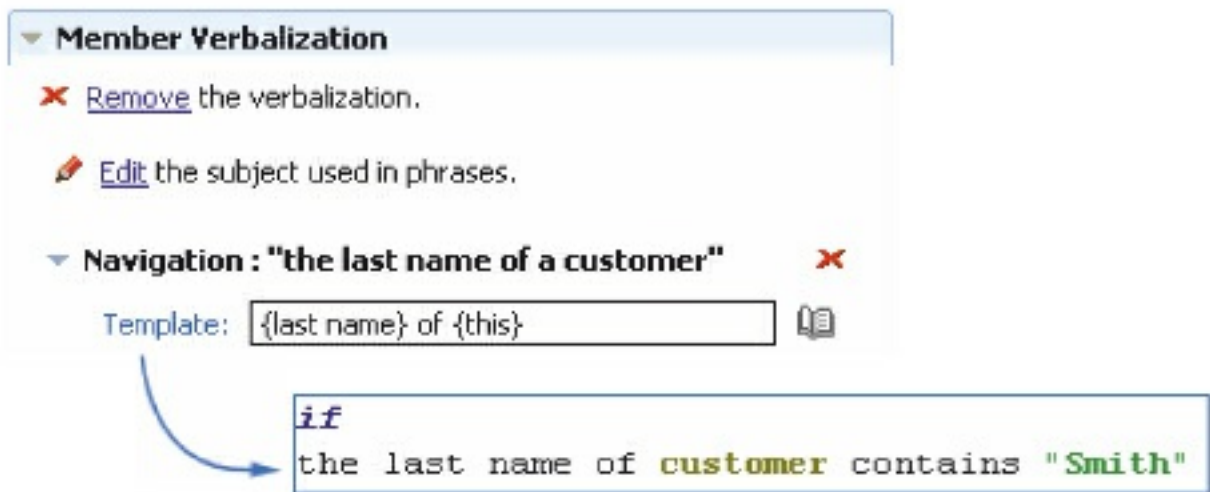
**Attention:**  
  
Do not use the characters “ and > in term labels. They cause warnings and ambiguity problems when used in rules.

## Navigation phrase

A navigation phrase is a phrase that associates two business elements. It can be the verbalization of:

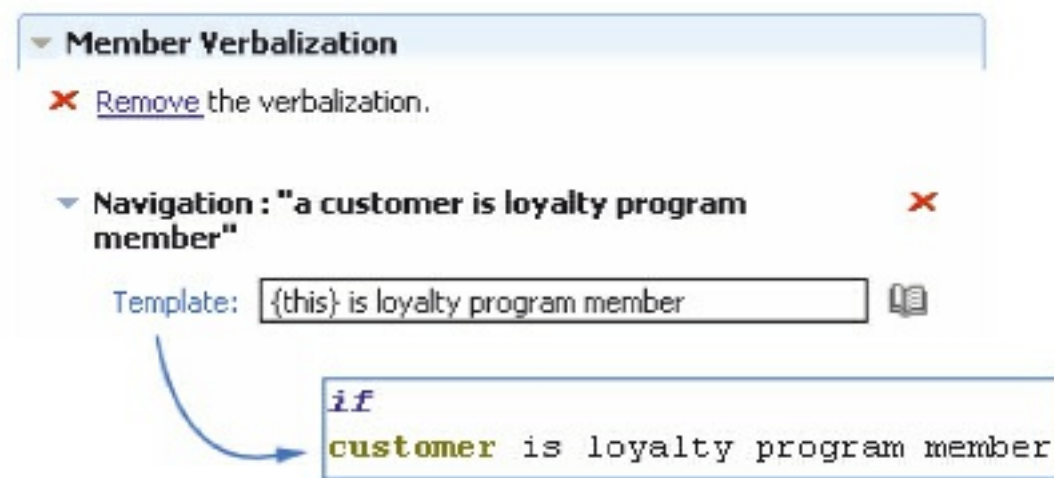
- A method that has a non void return type
- The getter part of an attribute

For example, {last name} of {this} is a navigation phrase for the attribute Customer.lastName.



You can also use predicate phrases in rules. A predicate phrase is a specific type of navigation phrase that verbalizes a non void method that returns a boolean or java.lang.Boolean. A predicate phrase has an implicit subject. For example, {this} is loyalty program member is a predicate phrase for the attribute

Customer.loyaltyProgramMember.



## Action phrase

An action phrase applies an action to an object. It can be the verbalization of:

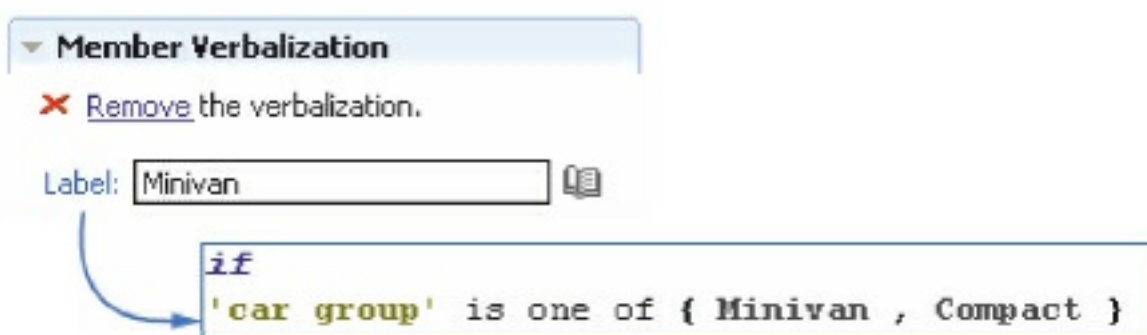
- A method that has a void return type
- The setter part of an attribute

For example, in {this}, display the message {0} is an action phrase for the method Session.sendMessage(String).



## Constant

A constant is the verbalization of the public static final attribute of a class with the same type as this class. For example, Minivan is a constant for the public static final attribute CarGroup.Minivan.



**Parent topic:** [Vocabulary](#)

**Related concepts:**

[Phrase templates](#)

[Vocabulary](#)

# Phrase templates

A phrase template is a pattern for the verbalization of phrases. The pattern changes for navigation, predicate, or action phrases.

Phrase templates combine placeholders and text. Text binds placeholders to compose a phrase. For example, in the phrase template {last name} of {this}, {last name} and {this} are placeholders, and of is text.

**Attention:**

Do not use the characters “ and { and } in phrase templates. They cause warnings and ambiguity problems when used in rules.

**Placeholders**

Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type.

**Simplified phrase templates**

To avoid wordiness and make phrases shorter, you can remove the {this} placeholder if there is no ambiguity between BOM members.

**Parent topic:** [Vocabulary](#)

**Related concepts:**

[Vocabulary elements](#)

[Placeholders](#)

[Vocabulary](#)

**Related tasks:**

[Creating a BOM entry from a XOM](#)

**Related information:**

[Default verbalization](#)

[Simplified phrase templates](#)

# Placeholders

Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type.

Vocabulary phrase templates contain placeholders. Placeholders represent gaps in phrases that can be filled in either automatically or manually when editing rules. Placeholders are identified by curly brackets {}.

There are three types of placeholder:

- Subject placeholders, which are completed automatically
- {this} placeholders, which you complete manually when editing rules
- Argument placeholders, which represent the arguments of a method, and which you complete manually when editing rules

To set a different text for an argument placeholder, use the syntax {x, "my text"} in the phrase template.

## Subject placeholder

Subject placeholders are shown in navigation phrases. The subject is a business term that you can edit in the BOM Editor. The subject corresponds to the return type of a member.

The subject placeholder is optional. When used in rules, the subject placeholder automatically takes the appropriate number and article for the context. In the placeholder text, the subject needs to be specified using the singular form, without article.

### {this} placeholder

The {this} placeholder represents the declaring class of a member. You cannot use this placeholder in the verbalization of static members.

For example, for an attribute `lastName` of type `String` in class `Customer`, you can specify the following phrase template:

```
{last name} of {this}
```

If there is no ambiguity, you can write simplified phrase templates by not using the {this} placeholder. See [Simplified phrase templates](#).

## Argument placeholder

Argument placeholders represent the arguments of methods. They are represented by the index of the argument.

For example, the action phrase in {this}, display the message {0} is the verbalization of the method `Session.sendMessage(String)`. {0} is the argument placeholder for a `String`.

▼ Member Verbalization

✖ Remove the verbalization.

▼ Action : "in a session, display the message : a string" ✖

Template:  

then

in 'the current session', display the message :

**Parent topic:** [Phrase templates](#)

**Related concepts:**  
[Phrase templates](#)

## Simplified phrase templates

To avoid wordiness and make phrases shorter, you can remove the {this} placeholder if there is no ambiguity between BOM members.

If there is no ambiguity between members of the BOM in different classes, you can remove the {this} placeholder for the declaring class from the verbalization. This produces shorter phrases and makes your business rules less verbose.

Removing the {this} placeholder causes a vocabulary warning by default, but as long as you know that the phrase is not ambiguous, you can set a preference to ignore this warning.

For example, if you change the verbalization of `Customer.name` from {name} of {this} to {name}, everywhere in your business rules you can use the phrase the name instead of the name of the customer. However, if you have `Company.name` verbalized as {name} in the same BOM, you must modify one of the phrase templates to avoid ambiguity. For example, you could verbalize `Customer.name` to {customer name}.

When using phrases that contain no {this} placeholder in their verbalizations, a binding on the working memory is generated in the IRL translation of the rule. In order to execute this kind of rule, an object of the right type must be inserted into the working memory.

**Parent topic:** [Phrase templates](#)

**Related concepts:**

[Placeholders](#)

[Phrase templates](#)

**Related tasks:**

[Using a reduced verbalization for simpler rules](#)

**Related information:**

[Vocabulary errors and warnings](#)



# Default verbalization

Rule Designer applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization in the BOM Editor.

When you verbalize a BOM, a default verbalization is applied to all the members you select in the New BOM Entry or Verbalize BOM wizards. You might need to edit the default verbalization afterwards to add new phrases, or to make phrases more readable.

Table 1. Default verbalization

Business element	Default verbalization
Class  For example:  Customer	Class name in lowercase  For example:  customer
Constructor	None
Getters (non write-only attributes and methods of type getXXX without arguments)  For example:  getAge	{XXX} of {this}  For example:  {age} of {this}
Boolean getters  For example:  isRich	{this} is XXX  For example:  {this} is rich
All other methods  For example:  displayMessage(String)	Code-like verbalization, that is, the method signature, without subjects.  For example:  {this}.displayMessage({0})
Setters (non read-only attributes and void methods of type setXXX with one argument)  For example:  setDiscount	set the XXX of {this} to {0}  For example:  set the discount of {this} to {discount}
Boolean setters  For example:  loyaltyProgramMember	make it {0} that {this} is XXX  For example:  make it {loyalty program member} that {this} is loyalty program member
Static references (public static final attribute of a class with same type as this class)  For example:  Minivan	Name of the static reference  For example:  Minivan

Parent topic: [Vocabulary](#)



**Related concepts:**

[Phrase templates](#)

**Related information:**

[Vocabulary errors and warnings](#)

[Defining a vocabulary](#)

# Vocabulary errors and warnings

If you have vocabulary ambiguities, missing placeholders, or problems on business elements, Rule Designer raises errors and warnings on terms and phrases at editing or build time.

Vocabulary elements, such as terms, phrases, articles and constants, might be made up of one or more words, called lexical units. These lexical units must comply with specific lexical rules. Vocabulary elements must be non ambiguous. You cannot verbalize two classes with the same term, and you cannot verbalize two members with the same phrase.

## Terms

Terms must not conflict with the elements of the business rule language and the System BOM. For example, if a word in a term can be interpreted as a number, you get an error.

Ambiguity errors are reported locally when editing a vocabulary in the same vocabulary file, and reported globally at build time if the ambiguity comes from duplicate terms in two separate files.

## Phrases

In phrases, you can use each type of placeholder only once.

You get a warning in the following circumstances:

- If the subject placeholder is not used in a navigation phrase
- If the {this} placeholder is missing

You can set a preference to ignore these warnings.

Errors in phrases can also come from their related business element:

- An action phrase for a read-only or final attribute
- A navigation phrase for a write-only attribute

Table 1. Examples of vocabulary errors

Message	Description
Character > is forbidden.	A vocabulary element uses a forbidden character. This error is reported when editing the vocabulary.
Character > in term for class balsample.Book is forbidden.	A vocabulary element uses a forbidden character. This error is reported at build time.
Term book is duplicated.	The term is already used for another class in the current vocabulary file.
Lexical conflict in production l-target(balsample.Book,SINGLE,INDEFINITE_ARTICLE,_,_,_) -> 'a' "book" "12" expecting construct "12" and having construct "<ilog.rules.brl.Number>"	Lexical conflict: the lexical unit 12 in the term associated to the class Book is interpreted as a number.
Placeholder "author" is missing in "balsample.Book: author"	The subject placeholder (author) is missing in the phrase associated to the member balsample.Book.author.
Duplicate verbalization: [balsample.Book: author] and [balsample.Book: title]	The members balsample.Book.author and balsample.Book.title have the same verbalization.
Action phrase not allowed for "balsample.Book: author" (member is read-only)	The attribute is read-only and must not be verbalized as an action phrase.
Navigation phrase not allowed for "balsample.Book: author" (member is write-only)	The attribute is write-only and must not be verbalized as a navigation phrase.

## Delimiters in numbers

Commas and spaces are used to separate parts of rules and expressions. In certain locales, they also serve as delimiters in numbers:

- A comma can be used to group three digits, as in 10,000.
- A space can also be used to group three digits, as in 10 000.
- A comma can be used as a decimal mark, as in 3,14159265.

Follow these instructions to avoid problems when expressing numbers:

- When possible, do not use digits in verbalizations.
- When using digits, avoid space and comma separators between digits, especially if those characters are valid group delimiters or decimal markers in the locale of the vocabulary.
- If a verbalization must contain digits separated with spaces or commas, surround the numbers with double quotation marks, for example, "3,14159265".

**Parent topic:** [Vocabulary](#)

**Related concepts:**

[Vocabulary](#)

[Phrase templates](#)

## Defining the vocabulary for a BOM entry

You can define a vocabulary for a complete BOM entry.

### About this task

The vocabulary is the set of terms and phrases defined in all the BOM entries of a business object model. If an element is not verbalized, it is not available in business rules. You can define a vocabulary for a complete BOM entry using the Verbalize BOM wizard.

### Procedure

To define the vocabulary for a BOM entry:

1. In the Rule Explorer view, double-click the BOM entry in the bom folder.
2. In the BOM Editor Package page, in the Tasks section, click **Verbalize the elements of this BOM entry**.
3. In the Verbalize BOM wizard, choose whether you want to verbalize only the Getters part of methods and attributes, only the Setters part, all methods, or only static references. Select or clear the relevant check boxes to include or remove vocabulary elements from the vocabulary.
4. Click **Finish**.

### Results

The BOM entry is now verbalized. You can edit the default verbalization in the BOM Editor to align the terms and phrases with your business model.

#### Attention:

If you change the default verbalization for terms, phrases, or constants, avoid using parentheses ( ). These characters are considered as delimiters and can cause false difference, overlap, or gap warnings in decision tables.

**Parent topic:** [Defining a vocabulary](#)

## Editing terms

You edit terms to change, for example, the singular or plural form of a term.

### About this task

You can edit terms in the BOM Editor to change, for example, the singular or plural form of a term.

### Procedure

To edit a term:

1. In the Outline view, click the class corresponding to the term you want to edit.
2. In the Class Verbalization section of the BOM Editor, click **Edit term**.

The Edit Term dialog opens.

3. Change the term, as required:
  - Change the singular or plural form of the term
  - Change its definite and indefinite articles

To edit fields in this dialog, clear the check box next to the field.

4. When you have finished, click **OK** to close the Edit Term dialog.

**Parent topic:** [Defining a vocabulary](#)

# Documenting terms

You document business terms to provide guidance to policy managers.

## About this task

To make business terms reusable and provide guidance to policy managers, it is important to document them. The documentation you add displays in the information box next to the Content Assist box when you edit business rules.

## Procedure

To document a term:

1. In the Outline view, click the class corresponding to the term you want to document.
2. In the Class Verbalization section of the BOM Editor, click **Edit the documentation**.
3. In the Business Term Documentation dialog, enter the required description or comment.
4. Click **OK**.

## Results

The documentation text is shown at the bottom of the Class Verbalization section.

**Parent topic:** [Defining a vocabulary](#)

### Related tasks:

[Documenting phrases](#)

## Editing and creating phrases

You can create a new phrase, edit a phrase, including the subject of a phrase, and use a reduced verbalization for simpler rules.

### Editing a phrase

Phrases correspond to methods in the BOM. You can edit the navigation and action phrases using the BOM Editor.

### Editing the subject of a phrase

You can edit the subject of a phrase using the Edit Term dialog.

### Creating new phrases

You can define several navigation and action phrases for a BOM element.

### Using a reduced verbalization for simpler rules

You can reduce the verbalization to simplify your business rules.

### Documenting phrases

You document phrases to provide guidance to policy managers.

**Parent topic:** [Defining a vocabulary](#)

## Editing a phrase

Phrases correspond to methods in the BOM. You can edit the navigation and action phrases using the BOM Editor.

### About this task

Phrases correspond to methods in the BOM. You can edit the verbalization of phrases.

### Procedure

To edit a phrase:

1. In the Outline view, click the element corresponding to the phrase you want to edit.
2. In the Member Verbalization section of the BOM Editor, edit the navigation and action phrases in the **Template** field, using text and Content Assist.

#### Note:

To edit the subject of a phrase, click **Edit the subject used in phrases** and change the term using the Edit Term dialog.

3. Save the BOM.

The new phrase verbalization is now visible in business rules.

**Parent topic:** [Editing and creating phrases](#)



## Editing the subject of a phrase

You can edit the subject of a phrase using the Edit Term dialog.

### About this task

In the Edit Term dialog, you can edit a subject phrase through the subject placeholder.

### Procedure

To edit the subject of a phrase:

1. In the Member Verbalization section of the BOM Editor, click **Edit the subject used in phrases**.

The Edit Term dialog opens.

2. Change the phrases, as required. You can make the following changes:

- Change the singular or plural form of the term
- Change its definite and indefinite articles

3. When you have finished, click **OK** to close the Edit Term dialog.

**Parent topic:** [Editing and creating phrases](#)

## Creating new phrases

You can define several navigation and action phrases for a BOM element.

### About this task

When you define several navigation or action phrases for a BOM element, rule authors can use different wordings while using the same method or attribute.

### Procedure

To add a phrase to a method:

1. In the Member Verbalization section of the BOM Editor, click either **Create a navigation phrase** or **Create an action phrase**.

The phrase is shown in the list of phrases. An error might be display because the new phrase is defined with the default verbalization. If you already have the default verbalization for an existing phrase, there is a conflict because you should not define the same verbalization twice in the vocabulary.

2. Click the phrase to expand it, and then in the **Template** field edit the verbalization.
3. Save the BOM.

### Results

You can now use the phrase in your business rules.

**Parent topic:** [Editing and creating phrases](#)

## Using a reduced verbalization for simpler rules

You can reduce the verbalization to simplify your business rules.

### About this task

If there is no ambiguity between members of the BOM in different classes, you can remove the {this} placeholder for the declaring class from the verbalization. This produces shorter phrases and makes your business rules less verbose.

### Procedure

To simplify the verbalization:

1. In the Outline view, click the member for which you want a simpler verbalization.
2. In the Member Verbalization section of the BOM Editor, expand the phrase to show the Template field.
3. Remove the of {this} part of the verbalization.

You can ignore the following warning for the time being:

Placeholder “this” is missing

4. On the **Window** menu, click **Preferences**. (On Mac, **Preferences** is in the **Eclipse** menu.)
5. In the side pane of the Preferences dialog, click **Rule Designer** > **Vocabulary**.
6. On the Vocabulary page, clear the box **Check {this} placeholders in phrases**.
7. Click **Apply**, and then **Apply and Close**.
8. Save the BOM. Click **Yes** if a message opens.

The warning is no longer shown and you can start using the simplified verbalization in your business rules.

**Parent topic:** [Editing and creating phrases](#)

# Documenting phrases


You document phrases to provide guidance to policy managers.

## About this task

You can provide guidance to policy managers by adding documentation to phrases. Then, when you edit business rules, the description or comment you added is displayed in the information box next to the Content Assist box.

## Procedure

To document a phrase:

1. In the Outline view, click the member you want to document.
2. In the Member Verbalization section of the BOM Editor, next to the **Template** field, click the  **Edit Phrase Documentation** button.
3. In the Phrase Documentation dialog, enter the required description or comment.
4. Click **OK**.

To view the documentation text again, click the  **Edit Phrase Documentation** button.

**Parent topic:** [Editing and creating phrases](#)

## Related tasks:

[Documenting terms](#)

## Editing the verbalization of constants

Constants correspond to static references in the BOM. You can change the verbalization of constants.

### About this task

Constants correspond to static references in the BOM. A static reference can be a static final attribute of a type, for example, Borrower. To change the verbalization of a constant, you edit the corresponding static reference.

### Procedure

To edit a constant:

1. In the Outline view, click the static reference you want to edit.
2. In the Member Verbalization section of the BOM Editor, edit the verbalization in the **Label** field.

You cannot use placeholders and special characters such as % or \$.

3. Save the BOM.

The new constant verbalization is now visible in the business rules.

**Parent topic:** [Defining a vocabulary](#)

## Editing the verbalization of ruleset variables

The default verbalization of a ruleset variable is its name. You can edit this verbalization.

### About this task

By default, a ruleset variable is verbalized as its name. You can change the verbalization using the Variable Set Editor.

### Procedure

To edit the verbalization of a ruleset variable:

1. In the Rule Explorer view, double-click the variable set that contains the ruleset variable you want to verbalize.

The Variable Set Editor opens.

2. In the Variable Set Editor, in the Verbalization column, click the verbalization you want to change and then click **Refactor**.
3. In the Refactor Variable dialog, type a new verbalization for the ruleset variable.

You cannot use placeholders and special characters such as % or \$.

4. Click **Preview** to assess the impact of this change on the business rules.
5. Click **OK**.
6. Save the variable set.

The verbalization of the ruleset variable is now changed.

**Parent topic:** [Defining a vocabulary](#)

## Editing the verbalization of ruleset parameters

The default verbalization of a ruleset parameter is its name. You can edit this verbalization.

### About this task

By default, a ruleset parameter is verbalized as its name. You can change the verbalization using the Rule Project Properties dialog.

**Note:** You use ruleset parameters with classic rule projects only. For decision services, you use ruleset variables to define parameters for the decision operation.

### Procedure

To edit the verbalization of a ruleset parameter:

1. In the Rule Explorer view, right-click the rule project and then click **Properties**.
2. In the Rule Project Properties dialog, click **Ruleset Parameters** in the side pane.

The Ruleset Parameters page opens.

3. In the Verbalization column, click the verbalization you want to change and then click **Refactor**.
4. In the Refactor Parameter dialog, type a new verbalization for the ruleset parameter.
5. Click **Preview** to assess the impact of this change on the business rules, and then click **OK**.
6. In the Rule Project Properties dialog, click **OK**.

The ruleset parameter verbalization is now changed.

**Parent topic:** [Defining a vocabulary](#)

## Working with categories

A category is an identifier that you can assign to business rules and certain business elements to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. The rule editor drop-down list shows only the business elements that a rule can see.

### **Categories**

A category is a filter that is applied to business rules and to business elements such as classes and members.

### **Defining and assigning categories**

You define a category at the rule project level and then assign it to a business element.

**Parent topic:** [Configuring the BOM for rule authoring](#)



# Categories

A category is a filter that is applied to business rules and to business elements such as classes and members.

A category is an identifier that you can assign to business rules and certain business elements (classes and members) in order to filter which business rules can use which business elements. A rule can use a business element if the rule category filter specifies at least one of the categories assigned to the business element. Only business elements that a rule can see are shown in the rule editor drop-down list. You can assign one or more categories to business object model (BOM) classes and members, or to a business rule category filter.

By default, all business rules, classes, and members belong to the predefined category any. All rules can see the any category, whatever the category filter in use. Therefore, the default is that all classes and members can be used in all business rules.

**Note:**  
System BOM elements have no category, so are hidden from any business rule regardless of which rule category filter the rule uses.

Before you can use a category, you must define it at the rule project level. If you define a category in a project that is referenced by another project, the category is usable in both projects.

There is no inheritance between classes and members, but you can assign the same category to all members of the same class.

For more information about using categories, see:

- [Defining and assigning categories](#)
- [Applying a category filter](#)

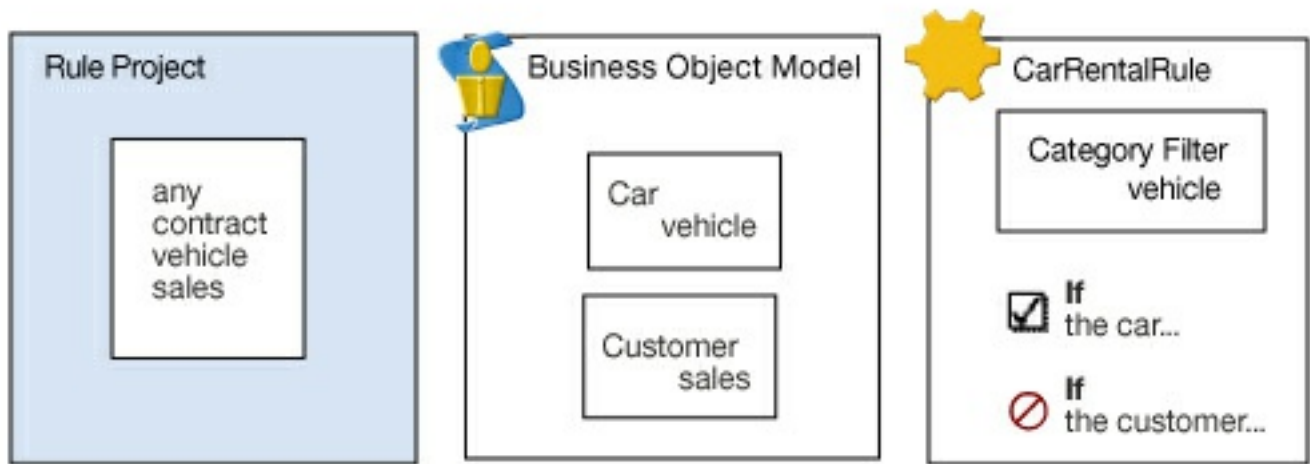
Table 1. Category usage summary

If a business element has the category...	And the business rule category filter is...	In that business rule, the business element is...
any	any	visible
any	another category	visible
another category	any	visible
category1 and category2	category1	visible
category1 and category2	category2	visible
category1 and category2	category1 and category2	visible
category1 and category2	category3 and category4	hidden

## Example

In your business object model, you have a Car class with a category vehicle. You also have a rule project that contains a business rule CarRentalRule, which has vehicle in its category filter. From CarRentalRule, you can see all the classes that have the category vehicle, as well as all classes that have the any category. CarRentalRule can therefore see the Car class.

If you then define a category sales and assign it to a class Customer, you do not see the Customer class in the rule editor drop-down list for the rule CarRentalRule. If you still decide to use the Customer class in the business rule, it is reported as a warning.



Similarly, if the Car class has a member with a category other than vehicle, you cannot use this member in the conditions and actions for CarRentalRule, even though you can define a variable of the Car type in the definitions part of the rule.

**Parent topic:** [Working with categories](#)

## Defining and assigning categories

You define a category at the rule project level and then assign it to a business element.

### About this task

You define categories at the rule project level. When you have defined a new category, you can assign it to business elements and use it in business rule category filters.

### Procedure

To define a new category and assign it to a business element:

1. In the Rule Explorer view, right-click the rule project and then click **Properties**.
2. In the side pane of the Properties dialog, click **Categories**.
3. On the Categories page, click **Add**.
4. In the New Category dialog, specify a name for the category and then click **OK**.

The Categories page displays the new category in the category list.

5. Click **Apply and Close** to close the Properties dialog.

The rule project now has a new category, which you can assign to a business element.

6. To assign a category to a business element, go to the Outline view and then click the business element to which you want to assign the category.
7. In the Categories section of the BOM Editor, click **Edit the categories**.
8. In the Categories dialog, select a category in the **All categories** field and then click > > to move it across to the **Selected categories** field.

You can also double-click the category to move it from one field to the other.

9. Repeat the previous step for each category you want to assign.
10. Click **OK** to close the Categories dialog.

### Results

The rule project now has a new category and the Categories section of the BOM Editor lists the categories that you defined for the business element.

#### Note:

When you delete a category from the rule project, it is not deleted from the business elements and rule category filters that use it. To delete it fully, you must also delete it using the BOM Editor. You do not get a warning message to prompt you to delete using the BOM Editor.

**Parent topic:** [Working with categories](#)

# Working with domains

You can use domains to extend the business object model (BOM).

## **Domains**

Make the BOM more specific by setting domains on members. Domains can be static, dynamic, enumerated, or complex.

## **Defining domains**

Use domains to extend the business object model (BOM).

## **Excel domain provider**

You can create enumerated domains based on data stored in an Excel file. The Excel domain provider handles the link between the data stored in Excel and your BOM.

## **Creating dynamic domains from Excel**

Create a dynamic domain and populate it with data from an Excel file.

## **Updating a single dynamic domain**

If Rule Designer is customized to use an Excel file or other external data sources for BOM domain values, you can update these values from the BOM Editor.

## **Updating all dynamic domains**

After populating or modifying the values of your enumerated domain, you can update all the BOM classes.

**Parent topic:** [Configuring the BOM for rule authoring](#)

# Domains

Make the BOM more specific by setting domains on members. Domains can be static, dynamic, enumerated, or complex.

A domain places a restriction on type elements in the BOM. You can set a domain on classes, attribute types, method return types, and arguments.

The main domains include:

- Static domains, comprising [Literals](#), [Static references](#), [Bounded](#), [Collection](#), and [Other](#) domain types.
- [Dynamic](#) domains, with values stored in an Excel file.
- Enumerated domains, comprising [Literals](#), [Static references](#), and [Dynamic](#) domain types.
- Complex domains, comprising [Bounded](#), [Collection](#), and [Other](#) domain types.

**Note:**

Not all kinds of BOM domain are enforced in BAL rules or other business rules. Business rules use only enumerated domains (literal, static reference, or dynamic). A semantic check is done to check that the business rule does not use a value outside its domain, and the Intellirule editor suggests values from the enumerated domains. However, the semantic check done at the business rule level is primitive and does not detect complex patterns of incorrect usage that involves operators other than `is` or `is not`. Other kinds of domain, such as bounded domains, are not enforced at business rule level.

You can use domains when working with business rules:

- Editing business rules: Code completion in the Intellirule Editor uses enumerated domains to propose valid settings.
- Checking business rules: You use enumerated domains to report errors and warnings that help you to validate the values you specify.
- Analyzing rules: You use all domain types in the BOM to check the consistency of business rule semantics.

If the XOM has a set of `public`, `static`, and `final` attributes typed to the declaring class, they are automatically considered as an enumeration of static references of the class in the BOM.

If the XOM has members of array type, they are automatically considered as a collection of the class in the BOM.

## Bounded

A bounded domain specifies an interval between two bounding values, such as `[0, 120]`.

In the designer BOM editor, when defining a bounded domain on an integer attribute, you can specify the `*` (asterisk) character as the lower or upper bound of the domain. When you specify `*`, the bound value is replaced in the corresponding `.bom` file by `-2147483648` for the lower bound, or `2147483647` for the upper bound. These values correspond to `Integer.MIN_VALUE` and `Integer.MAX_VALUE`, respectively.

**Note:**

Bounded domains are not enforced at business rule level. You can create a bounded domain on a number primitive type.

## Collection

A collection domain specifies the cardinality and the type of collection elements, for example, `0,* class Customer`.

If you have members of type `java.util.Collection`, set a collection domain on these members for them to be automatically considered as a collection in business rules. You can also create, add, and remove methods for the items in the collection domain by using the BOM Editor.

For more information about collections, see [Collections](#).

**Note:**

You can create a collection domain on a collection or an array.

## Dynamic

You can populate a domain in the BOM dynamically from a data source, and then synchronize the data source and the domain.

A dynamic domain is an enumerated domain with values from an Excel file.

**Note:**  
You can create a dynamic domain only on a class in the BOM class editor, but not on a Collection subclass.

**Literals**

A domain set as an enumeration of literals specifies a list of values, for example, {1, 2, 3}.

**Note:**  
You can create a literal domain on a primitive type or a string.

**Static references**

A domain set as an enumeration of static references specifies a list of references to constants, for example, {static GroupA, static GroupB, static GroupC}.

You can define attribute types and method return types and arguments as follows:

- If you have an attribute of type A, you can define a domain of static references on it using the static attributes of the class A (classic Java™ enumeration pattern).
- If you have an attribute of a primitive type, you can define a literal domain on it.

**Note:**  
You can create a static reference domain on a class, but not on a Collection subclass.

**Other**

The ‘other’ domain types were introduced to support most domains that come from the XML binding. In an XML schema, you can define pattern domains (even for numbers) and simultaneously an enumeration. You can also define pattern domains or intersection of domains in the business object model.

You can define other types of domains by using the syntax of regular expressions. For example, you can define a pattern for Strings as follows:

```
"a*n"
```

You can define an intersection of domains as follows:

```
{1, 3, 5, 7, 9}, [0,6])
```

**Note:**  
Other domains are not enforced at business rule level. You can set this type of domain on an array by using the following syntax: '{ (String)"a", (String)"b"}'.

**Parent topic:** [Working with domains](#)

**Related concepts:**  
[Collections](#)

**Related tasks:**  
[Defining domains](#)

# Defining domains

Use domains to extend the business object model (BOM).

## About this task

You can create domains of different types to extend the business object model (BOM).

### Note:

You can create dynamic domains only on a BOM class.

## Procedure

To define a domain:

1. In the Outline view, click the element to which you want to add the domain.
2. In the BOM Editor, in the Domain section, click **Create a domain**.

The Domains wizard displays the options that are available for the type of artifact that you are creating. The options can include items from the following list:

- **Bounded:**

To create a bounded domain, double-click **Bounded**, and then specify the bound values and whether they are included in the domain.

- **Collection:**

To create a collection, double-click **Collection**, and then click **Browse** to select the type of the collection. Note that a member of type `java.util.Collection` is not automatically treated as a collection in business rules until you set a collection domain on it.

- **Literals:**

To create an enumeration of literals, double-click **Literals**, and then click **Add** to add new values to the enumeration of literals.

- **Static References:**

To create an enumeration of static references, double-click **Static References**, and then click **Add** to add new static references to the list.

- **Other:**

To create any other type of domain, double-click **Other**, and then enter the domain definition in the field provided.

- **Excel:**

To create a dynamic domain from an Excel file, double-click **Excel**. For more information, see [Creating dynamic domains from Excel](#).

3. Click **Finish**.

The domain is added to the business element. You can view its definition in the Domain section of the BOM Editor.

You can now use the values of your domain when editing your rules.

**Parent topic:** [Working with domains](#)

### Related concepts:

[Domains](#)

### Related tasks:

[Creating dynamic domains from Excel](#)

[Updating a single dynamic domain](#)

[Updating all dynamic domains](#)



# Excel domain provider

You can create enumerated domains based on data stored in an Excel file. The Excel domain provider handles the link between the data stored in Excel and your BOM.

**Note:** You can use the Excel domain provider only on a BOM class.

Some properties must be defined on the BOM class to retrieve the information from the Excel file. To ensure that the properties are mapped correctly, you must create an Excel file with the required structure. The Excel file must contain a column for the values of the domain provider, a column for the label, and a column for the BOM to XOM mapping.

You must create one row for each value of the domain provider. You cannot merge cells.

The Excel file must have the following structure:

## Value column

You must create a value column to enter the values of the domain provider.

## Label column

You must create a label column to enter the verbalization for the domain value. This label is the name displayed for the value when editing a rule.

## BOM to XOM column

You must create a column to add the BOM to XOM mapping for each domain value.

## Optional columns

The documentation column is optional. You can create a column to enter documentation on a value.

You can add additional label and documentation columns for other locales. The default locale is the locale of your Eclipse application. The values for the other locales are used when changing the locale of your Eclipse application.

You can also add columns if you want to use custom properties for the values in your dynamic domain. The custom properties that you define through the Domains wizard apply to all the values in your dynamic domain.

## Sheets

You can have several domain providers in the same Excel file. Each worksheet corresponds to one domain provider.

For example, you could have a sheet for a domain called Currency, and another sheet for a domain called Status.

The following table is an example of the columns and values in a configured Excel file. In this example, the domain defines the status of the loan:

Values	BOM to XOM	English label	Documentation (En)	Nom français	Documentation (Fr)
BLOCKED	return "Blocked";	Blocked	The loan is blocked	Bloqué	Le prêt est bloqué
ACCEPTED	return "Accepted";	Accepted	The loan is accepted	Accepté	Le prêt est accepté
REJECTED	return "Rejected";	Rejected	The loan is rejected	Rejeté	Le prêt est rejeté
PENDING	return "Pending";	Pending	The loan is pending	En attente	Le prêt est en attente

After creating your Excel file with the required columns and data, you must add the file to the resources folder of your rule project.

You can then map the domain properties to the columns in your Excel file, see [Creating dynamic domains from Excel](#).

**Parent topic:** [Working with domains](#)

**Related concepts:**  
[Domains](#)

**Related tasks:**  
[Creating dynamic domains from Excel](#)

**Related information:**





# Creating dynamic domains from Excel

Create a dynamic domain and populate it with data from an Excel file.

## Before you begin

You can create a dynamic domain with values extracted from an Excel file.

Make sure that the Excel file containing the values to populate the enumerated domain has the correct structure, see [Excel domain provider](#).

### Important:

Make sure that the Excel file is in the resources folder of your rule project.

## Procedure

To create a dynamic domain from an Excel file:

1. Open the BOM class in the BOM editor.
2. In the Domain section, click **Create a domain**.
3. Select **Dynamic Domains > Excel**, and then click **Next**.
4. In the **Excel File** field, select the Excel file that you added to the resources folder of your rule project.
5. In the **Sheet** field, select the sheet for the domain provider.
6. Select the **Table with header** check box if you have created a header in your Excel file.

Selecting this check box displays the name of the columns in the drop-down lists instead of the default column letters.

7. In the **Value column**, **BOM to XOM column**, and **Label column** fields select the corresponding columns in your Excel file.

In the Label column, you must select at least the label column for the default locale.

8. Click **Finish**, or click **Next** to add custom properties to the values.
9. Optional: Click **Add** to add a custom property.
  - a. Enter the name of your custom property.
  - b. Select the corresponding column in your Excel file.

## Results

The values of the dynamic domain are displayed in the BOM editor.

The values are also available in the completion menu of the rule editor. You can now use them when editing your rules.

If you have defined labels or documentation for other locales, you must update the BOM for each locale. To do so, you must restart Rule Designer in the locale to update, and synchronize the BOM with the values in the Excel file. For more information, see [Updating a single dynamic domain](#) and [Updating all dynamic domains](#).

**Parent topic:** [Working with domains](#)

### Related concepts:

[Excel domain provider](#)

[BOM properties](#)

### Related tasks:

[Updating a single dynamic domain](#)

[Updating all dynamic domains](#)

## Updating a single dynamic domain

If Rule Designer is customized to use an Excel file or other external data sources for BOM domain values, you can update these values from the BOM Editor.

### About this task

You can populate and update the values of an enumerated domain on the BOM classes that have the required properties for the dynamic domain provider.

If you are working on a single BOM class, you can update the values from the BOM editor using the **Synchronize** link.

If you want to update several classes at the same time, see [Updating all dynamic domains](#).

### Procedure

To update a single BOM class:

1. Open the BOM Editor.
2. In the BOM Editor, open the class that defines the dynamic domain.
3. Click **Synchronize** to populate the enumerated domain.

If the value of the domain provider property is invalid, the Domain section displays following message:  
The value provider is invalid. Check the custom properties.

4. Save the changes in the BOM.

### Results

If a value is no longer used, the value is set to deprecated, and a warning is displayed in the Problems view.

**Parent topic:** [Working with domains](#)

### Related tasks:

[Updating all dynamic domains](#)

[Defining domains](#)

[Creating dynamic domains from Excel](#)

# Updating all dynamic domains

After populating or modifying the values of your enumerated domain, you can update all the BOM classes.

## About this task

You can populate and update the values of an enumerated domain on all BOM classes that have the required properties for the dynamic domain provider.

You can update the values for all the dynamic domains at the same time.

If you do not want to update all the dynamic domains at once, see [Updating a single dynamic domain](#).

## Procedure

To update all dynamic domains in your BOM at once:

1. Open the BOM Editor.
2. In the BOM Editor, in the Tasks section of the Package page, click **Update the dynamic domains of this BOM entry**.

The Dynamic Domains dialog opens. This dialog shows all the BOM classes that have a dynamic domain defined.

3. Select the classes for which you want to update the enumerated domain and then click **Finish**.
4. Save the changes in the BOM.

## Results

If a value is no longer used, the value is set to deprecated, and a warning is displayed in the Problems view.

**Parent topic:** [Working with domains](#)

### Related tasks:

[Updating a single dynamic domain](#)

[Defining domains](#)

[Creating dynamic domains from Excel](#)

# Translating an existing business vocabulary

To translate an existing vocabulary that is defined on the BOM, you can copy the vocabulary file or edit the verbalizations.

## About this task

You can translate an existing business vocabulary by either making a translated copy of the .voc file or editing the class and member verbalizations.

## Procedure

1. To translate a vocabulary by using the .voc file:

- a. In Rule Designer, switch to the Resource Perspective by clicking **Window > Open Perspective > Other > Resource**.

In the Navigator view, the BOM is displayed as a set of three files:

- model\_en.voc
- model.b2x
- model.bom

- b. Right-click the .voc file, and then click **Copy**.
- c. Right-click the .bom folder, and then click **Paste**.

A Name Conflict dialog opens, prompting you to rename the file.

- d. In the Name Conflict dialog, replace the language identifier of the file with the language identifier of the translation language. For example, if your BOM entry is named model and you want to translate it to French, name the file model\_fr.voc.
- e. Click **OK**.
- f. In the Navigator view, right-click the new .voc file, and then click **Open With > Text Editor**.

The text definition of the vocabulary opens in the Text Editor.

- g. You can now translate the vocabulary by translating the values of the vocabulary keys into the target language.
- h. Save the BOM.

Now when you launch Rule Designer in your target locale, the specified locale becomes the locale for both the vocabulary and the business rule project items.

2. To translate a vocabulary by using the class and member verbalizations:

- a. Open Rule Designer in the target locale.
- b. Open the BOM editor.
- c. Edit the class and member verbalizations.
- d. Save the file.

You now have a vocabulary for the target locale.

**Parent topic:** [Configuring the BOM for rule authoring](#)

## Related information:

[Options for customizing rule authoring](#)

[Concrete syntax properties](#)

[Business Action Language \(BAL\)](#)

# Authoring business rules

You use Rule Designer to create and edit different types of rules. You can use automatic variables, ruleset variables, templates, and categories to simplify the rule creation process.

## **Overview: Ways to express business rules**

You write business rules such as action rules and decision tables by using the Business Action Language (BAL).

## **Working with action rules**

You can create action rules by using the Intellirule editor or the guided editor.

## **Working with decision tables**

You use the decision table editor to create and update decision tables.

## **Working with variables**

You can use automatic variables and ruleset variables to define variables for use in business rules.

## **(Deprecated) Working with decision trees**

You use the decision tree editor to create a workflow for the execution of your rules.

## **(Deprecated) Working with templates**

You can create partially written rules and partially defined decision tables for use as templates, to simplify the task of creating rules and decision tables.

## **Working with technical rules**

Use the Technical Rule Editor to write technical rules in the ILOG® Rule Language.

## **Working with functions**

You can create a function in a rule project to share code procedures across more than one element of a ruleset. You express a function in ILOG Rule Language (IRL), and its code is evaluated when the ruleset runs.

## **Applying a category filter**

You can apply a category filter to specify which categories of elements can be used in action rules, decision tables, and decision trees.

## **Correcting errors by using Quick Fix**

You can use the Eclipse Quick Fix feature in the Intellirule Editor, the decision table editor, and the decision tree editor. Quick Fix messages offer suggestions to automatically correct detected errors.

## **Applying verbalization changes to business rules**

If you change the verbalization of a business element used in a rule, you can refactor the business rules that use the business element to take your changes into account.

**Parent topic:** [Designing projects for rule authoring](#)

## Overview: Ways to express business rules

You write business rules such as action rules and decision tables by using the Business Action Language (BAL).

The BAL provides a simple if-then syntax that you use with a vocabulary to write business rules. The BAL defines the syntax and provides constructs for expressing business rule conditions and actions, and the vocabulary defines the terms that you use in the business rules.

Modifiable building blocks make up the business rules. The building blocks represent vocabulary elements, ruleset parameters, ruleset variables, and BAL constructs and operators. For example, in the following action rule statement, the building block the `current load` uses a business term from the vocabulary, `is more than` is a BAL operator, and `5000` is a value:

```
if
 the current load is more than 5000
```

A phrase in a business rule can use a business term. For example, to complete the following phrase, you must select the business term `<a customer>`:

```
the age of <a customer>
```

You can create and maintain business rules in Rule Designer or Decision Center. You can deploy the business rules through decision services.

Operational Decision Manager provides various ways to express business rules:

### Action rules

Action rules are sentence-like statements that express a set of conditions followed by the actions to take if the conditions are true. With action rules, you can state business policies in a predefined business vocabulary that a computer can interpret.

For example, you might state the policy "change customers in the Gold category to the Platinum category when they spend more than \$1,500 in a single transaction" as follows in an action rule:

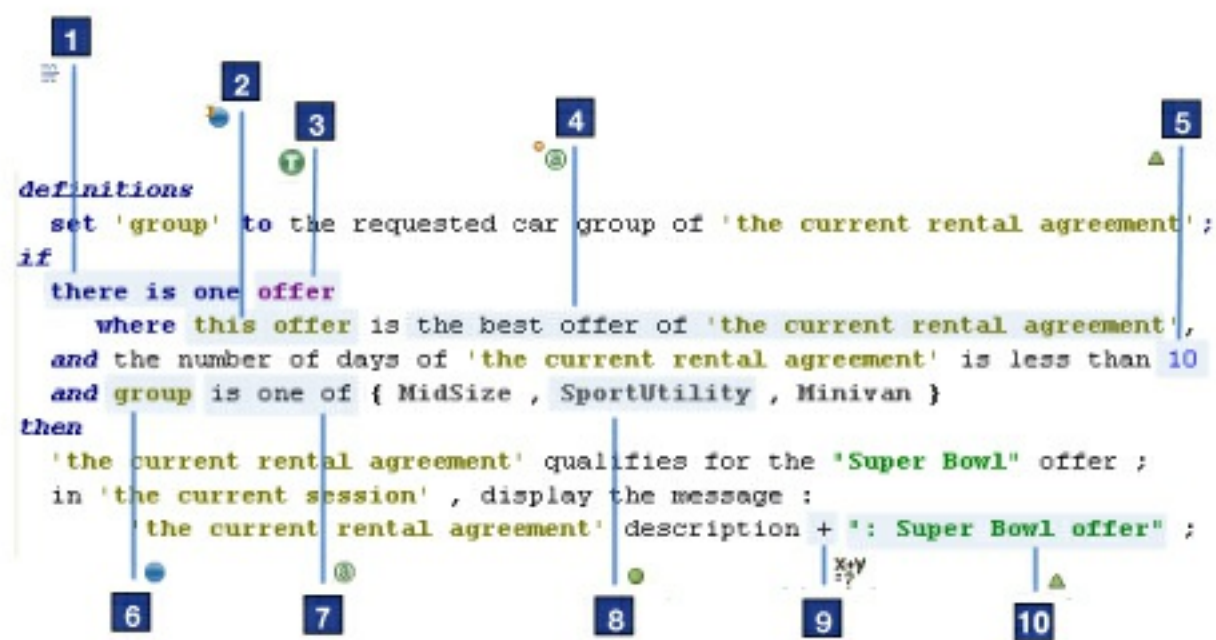
```
If
 All of the following conditions are true:
 - the customer category is Gold
 - the value of the shopping cart is more than $ 1,500
Then
 Change the customer category to Platinum
```

In an action rule, you can use all the BAL constructs and operators, and all the elements in the vocabulary.

You can find these elements in the following figure:

- BAL construct **1**
- Implicit variable **2**
- Term **3**
- Phrase **4**
- Number value **5**
- Rule variable **6**
- BAL operator **7**
- Constant **8**
- Arithmetic operator **9**
- Text value **10**





## Decision tables

Decision tables represent decision logic as a table. Each row in a decision table corresponds to an action rule.

The rows and columns of the tables show the possible situations that a business decision might encounter, and specify which action to take in each situation. You use decision tables to view and manage large sets of action rules with identical conditions.

As shown in the following decision table, you can see business terms, navigation phrases, BAL operators **1**, and action phrases **2** when you edit the condition columns. You can see constants **3**, number values **4**, and text values **5** in the cells of the table.

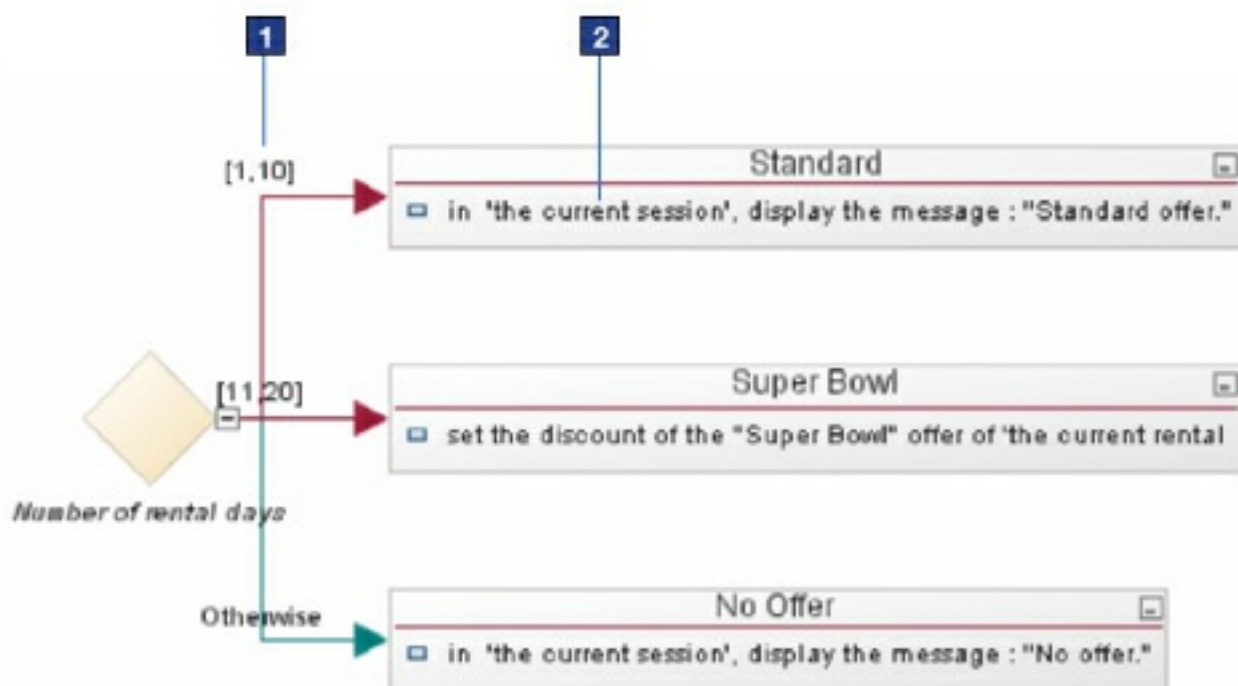
	Coverages	Car group	Surcharge	Comment
1	CDW	Economy,Compact,MidSize	\$15	Add surcharge of 15 for CDW [Economy, Compa.
2		FullSize,Luxury,SportUtility,Minivan	\$25	Add surcharge of 25 for CDW [FullSize, Luxury, .
3	LDW	Economy,Compact,MidSize	\$21	Add surcharge of 21 for LDW [Economy, Compa.
4		FullSize,Luxury,SportUtility,Minivan	\$21	Add surcharge of 21 for LDW [FullSize, SportUtili.
5	PAI		\$3	Add surcharge of 3 for PAI
6	PEI		\$2	Add surcharge of 2 for PEI
7	ALI	Economy,Compact,MidSize	\$11	Add surcharge of 11for ALI [Economy, Compact.
8		FullSize,Luxury,SportUtility,Minivan	\$12	Add surcharge of 12 for ALI [FullSize,SportUtilit..
9				

## Decision trees

Decision trees graphically represent decision logic. In a decision tree, the branches represent conditions, and the leaves represent the actions to take if the conditions are true.

Decision trees provide the same function as decision tables. However, with decision trees, you can manage a large set of rules with some common conditions, but not all.

When you edit branches in a decision tree, you can see business terms, phrases, BAL operators, values, and constants. The following diagram shows values **1**, and constants and action phrases **2**:



## Technical rules

You write technical rules in IRL, which is similar to Java™ code. You use technical rules to represent specific loops in the action part of your rules.

In the following example, the technical rules contain IRL keywords **1**, BOM elements in their code form **2**, values **3**, and constants **4**.

```

1
when {
 financialEvent: FinancialEvent(type equals 2 EventType.ACKNOWLEDGEMENT);
}
1
then {
 dispatcher.send(financialEvent, "Global Ledger");
 financialEvent.processedDate = Helper.getCurrentDate();
 3 update financialEvent;
}

```

## Simplifying rule editing for business users

When you design a rule project in Rule Designer, set up management and rule authoring environments that meet the needs of your business users. You want to avoid differences between the system architecture and the needs of business users.

For example, rule packages can reflect decision points in your application, but not the geography to which the business rules apply. The business rule vocabulary is ultimately implemented as an object model. The constraints that are involved in developing an object model can result in a verbose vocabulary. For example, your model might contain inheritance relationships or utility classes that business users cannot understand. You can hide these classes in the business object model.

## Filtering the vocabulary with categories

When you have a large vocabulary, you can use categories to filter the terms and phrases that are available to you when you edit rules in Rule Designer or Decision Center. When a category is assigned to a business rule, only the vocabulary terms and phrases of that category are visible in the business rule.

## Adding custom properties to rule artifacts

To add properties to existing types of rule artifacts, you must extend the rule model. You can set up BAL and rule model extensions, and then deploy them to both Rule Designer and Decision Center. For example, to identify the country for which a rule is applicable, you can create a property that is called geography.

Business users can use custom properties to manage business rules. For example, they can use queries in Decision Center to set up views that are based on a property. With these views, business users have a different view of the rule project that is compared to the physical rule package organization that you set up in Rule Designer.

**Parent topic:** [Authoring business rules](#)



**Related concepts:**[Action rules](#)[Technical rules](#)[\(Deprecated\) Decision tree overview](#)**Related tasks:**[Applying verbalization changes to business rules](#)[Applying a category filter](#)[Creating an action rule type](#)**Related information:**[Decision tables](#)[Rule project item naming conventions](#)[Working with action rules](#)[Working with decision tables](#)[\(Deprecated\) Working with decision trees](#)[Working with technical rules](#)[Rule languages](#)

# Working with action rules

You can create action rules by using the Intellirule editor or the guided editor.

## Action rules

You express business policies in rules that match actions to conditions.

## Action rule editing errors and warnings

Rule Designer searches for problems when you edit an action rule. Errors and warnings are displayed in the Problems view.

## Creating an action rule

When you create a new action rule, you specify the rule project to which it belongs. You can also store the rule in a specific package.

## Selecting an editing mode for your action rules

You can choose a predefined editing profile or define a custom profile that best suits your preferences for editing action rules.

## Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

## Building action rules using the guided editor

When you use the guided editor to build rules, you construct predefined rule fragments by entering text or numbers in fields or by selecting fragments from drop-down lists.

## Creating an action rule type

When you create a new rule model class to define a new type of action rule, the new rule type becomes available in the creation wizards.

**Parent topic:** [Authoring business rules](#)

## Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
 the credit score of 'the borrower' is less than 200
then
 in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

### Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

### Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

### Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

### Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

#### Rule variables

You create a rule variable to define the scope of a rule.

#### Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

#### Rule actions

Rule actions define what to do when the if part of the rule is true or false.

**Parent topic:** [Working with action rules](#)

**Related concepts:**

[Technical rules](#)  
[\(Deprecated\) Decision tree overview](#)

**Related tasks:**

[Creating an action rule](#)  
[Creating an action rule template](#)  
[Defining a folder structure for rule project items](#)

**Related reference:**

[Business Action Language \(BAL\)](#)

**Related information:**

[Overview: Ways to express business rules](#)  
[Decision tables](#)  
[Building rules using the Intellirule editor](#)  
[Building action rules using the guided editor](#)

# Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

## Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
 the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
 set 'the cart' to the shopping cart of customer;
if
 the value of 'the cart' is less than $100
then...
```

## One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
 set Smith to a customer;
if
 the category of Smith is Gold
then
 apply 10 % discount to the shopping cart of Smith;
```

## When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

## Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
( )	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

## Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation

marks:

<div>R u l e d i t o r</div>	<div></div> <div>Description</div>
<div>In t e l l i r u l e e d i t o r</div>	<div>If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.</div>
<div>G u i d e d e d i t o r</div>	<div>When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.</div>

**Types of rule variables**

You can assign different types of values to your rule variables.

**Parent topic:** [Action rules](#)

**Related concepts:**

[Dependency of rule actions on rule variables](#)

[Rule actions](#)

## Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, customer). Once you set a variable, you can use it in any part of the rule that declares the variable.

### Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and then parts of the rule use the same value:

```
definitions
 set maxAmount to 1000000;
if
 the amount of 'the loan' is at least maxAmount
then
 in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

### Restrictions on rule variables

You can further restrict a variable in the definitions part of a rule by using the operator `where`.

#### Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
 set 'loyal customer' to a customer
 where the category of this customer is Gold;
if
 the value of the shopping cart of 'loyal customer' is more than $200
then
 apply the super discount;
```

#### Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
 set 'senior Gold customer' to a customer
 where all the following conditions are true:
 - the category of this customer is Gold
 - the age of this customer is at least 65;
```

### Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the definitions part of the rule, for example:

```
definitions
 set applicant to a customer;
 set 'loyal customer' to a customer;
if
 all of the following conditions are true:
 - applicant is married to 'loyal customer'
```

```
 - 'loyal customer' is insured
then
 upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
 'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
 set 'customer 1' to a customer;
 set 'customer 2' to a customer;
if
 'customer 1' is married to 'customer 2'
 and 'customer 2' is insured
then
 upgrade 'customer 1''s rating;
```

**Note:** When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

### Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
 set 'gold customers' to all customers
 where the category of this customer is gold;
 set 'junior gold customer' to a customer in 'gold customers'
 where the age of this customer is at most 15;
 set 'senior gold customer' to a customer in 'gold customers'
 where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

**Parent topic:** [Rule variables](#)



## Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
 the customer category is Gold
then
 redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

### Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

### Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

### Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

### Combinations of conditions

You can apply conditions to groups, and test nested groups.

### Condition negation

You can set a rule to perform an action when a condition is not true.

**Parent topic:** [Action rules](#)

## Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

### starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
 the customer's maintenance number starts with "TX"
then
 redirect the call to call center A;
```

### is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
 the purchase value of the shopping cart is more than $100
then
 apply a 10% discount to the value of the shopping cart
```

### after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
 the return date of 'rented car' is after 'pickup date' and before
 'scheduled return date'
then...
```

**Parent topic:** [Rule conditions](#)

#### Related concepts:

[Conditions that test for existence](#)

[Combinations of conditions](#)

#### Related reference:

[BAL operators](#)

## Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

### there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
 there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
 there are 10 customers where the category of each customer is Gold,
then...
```

### there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
 there are at most 3 customers
 where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
 set 'silver customers' to all customers
 where the category of each customer is Silver;
 set 'silver count' to the number of 'silver customers'
if
 there are at least ('silver count' + 1) customers
 where the category of each customer is Gold,
then...
```

### there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:

```
if
 there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
 set 'gold customers' to all customers
 where the category of each customer is Gold;
if
 there is at least one customer in 'gold customers'
 where the age of this customer is at least 65,
then...
```

**Parent topic:** [Rule conditions](#)

**Related concepts:**

[Combinations of conditions](#)

[Conditions that compare business terms and values](#)

**Related reference:**

[BAL operators](#)

## Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
 the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
 set 'luxury car groups' to all car groups where the daily rate of each
 car group is at least 50;
if
 'luxury car groups' contain the car group of 'the current rental
 agreement'
```

**Parent topic:** [Rule conditions](#)

## Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting
longer than 5 minutes
```

### Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if
 the customer is older than 60
 or the customer is younger than 21
 and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

### Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if
 the customer is older than 60
 or (the customer is younger than 21 and the category of the customer
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if
 (the customer is older than 60 or the customer is younger than 21)
 and the category of the customer is Student
```

### Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true`: This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true`: This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if
 all of the following conditions are true:
 - the category of the customer is Gold
 - a member of the Gold team is available
then
 redirect the call to a member of the Gold team;
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
 any of the following conditions is true:
 - the category of the customer is Gold
 - the customer has been waiting longer than 5 minutes
then
 redirect the call to a member of the Gold team;
```

**Parent topic:** [Rule conditions](#)

**Related concepts:**

[Conditions that test for existence](#)

[Conditions that compare business terms and values](#)

**Related reference:**

[BAL operators](#)

## Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

### **it is not true that**

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
 the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
 it is not true that the category of the customer is Gold
then...
```

### **none of the following conditions are true**

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
 all the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
 none of the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

**Parent topic:** [Rule conditions](#)



## Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

### Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
 the value of the shopping cart is more than $100
then
 apply a 15% discount on the shopping cart;
```

### Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
 'the loan report' is approved
 and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
 in 'the loan report', accept the loan with the message
 "Congratulations! Your loan has been approved";
else
 in 'the loan report', refuse the loan with the message "We are sorry.
 Your loan has not been approved";
```

### Example of an action that calls methods

An action statement can invoke a method on a business term. You perform the method by using an action phrase from the rule vocabulary. The method can be a static method or a member of a class.

The following action statement uses `apply` to call the `applyDiscount` method, and `display` to call the `displayMessage` method.

```
then
 apply a 10% discount;
 display the message: "You received a discount!";
```

#### [Rule actions for lists of business terms](#)

You can define actions that a rule executes for each item in a list.

#### [Dependency of rule actions on rule variables](#)

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

**Parent topic:** [Action rules](#)

**Related concepts:**

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

## Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
 for each item in expensive items:
 - apply 5% discount to this item;
 - display the message: "A 5% discount has been applied";
```

**Parent topic:** [Rule actions](#)

## Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
 all of the following conditions are true:
 the value of the customer's shopping cart is more than $100
 the category of the customer is Gold
then
 apply a 15% discount
else
 apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

**Parent topic:** [Rule actions](#)

**Related concepts:**

[Rule variables](#)

[Rule actions](#)

# Action rule editing errors and warnings

Rule Designer searches for problems when you edit an action rule. Errors and warnings are displayed in the Problems view.

The types of errors and warnings that Rule Designer reports when you edit an action rule, where the error or warning is reported, and the remedial steps you can take are described in the following sections.

## Lexical errors

Lexical errors occur when a word in the rule is not recognized and are reported when you type a rule in the Intellirule editor.

Modify the text of the rule to follow the syntax specified by the language grammar and by the vocabulary defined in the application. Error messages might help, but not all times. Code completion and vocabulary use might be used to learn the exact syntax of phrase of the vocabulary.

If you have lexical errors in your rules, you cannot use the guided editor.

## Syntactic errors

Syntactic errors are reported when a rule statement is not well formed as soon as you type a rule in the Intellirule editor.

Modify the text of the rule to follow the syntax specified by the language grammar and by the vocabulary defined in the application. Error messages might help, but not all times. Code completion and vocabulary use might be used to learn the exact syntax of phrase of the vocabulary.

If you have syntactic errors in your rules, you cannot use the guided editor.

## Semantic errors

Semantic errors are reported when the text of the rule is syntactically correct but its interpretation is wrong as soon as you type a rule in the Intellirule editor.

For semantic errors, you should consider revising the text of the rules to comply with the language semantics and with semantic rules that might have been set up (for example, BOM domains).

## Consistency errors

Consistency errors are reported by the consistency checker at build time. For example, a consistency error is raised if a rule contains incompatible tests, or a test that is always false.

You can disable the consistency checker in the Rule Designer build preferences.

## Ambiguities

Ambiguities can occur in the following cases:

- A part of the rule has at least two interpretations that are semantically correct. You can use parentheses, intermediate variables, or punctuation if it is available in the statement, to make the rule non-ambiguous.
- There are two identical terms or phrases in the vocabulary. Changing one of the terms or phrases in the vocabulary can remove the ambiguity.

The following tables list error messages that you can see when editing an action rule.

Table 1. Action rule editing errors quick reference

Message	Description
A ruleset parameter <parameter name> is already declared.	<b>Ruleset Parameter Already Declared</b>  A declared variable has the same name as a ruleset parameter declared on a type.
A ruleset variable <variable name> is already declared.	<b>Ruleset Variable Already Declared</b>  A declared variable has the same name as a ruleset variable declared on a type.
Ambiguous sentence.	<b>Ambiguous Sentence</b>  Part of the rule has at least two interpretations that are semantically correct. Use parentheses, intermediate variables, or punctuation if available in the statement.

	There are two identical terms or phrases in the vocabulary. Change one of the terms or phrases in the vocabulary.
An automatic variable <variable name> is already declared.	<b>Automatic Variable Already Declared</b>  A declared variable has the same name as the automatic variable declared on a type.
Can't specify else clause without if clause.	<b>Else Without If</b>  An <b>else</b> part has been specified, but there is no <b>if</b> part in the rule.
Invalid cardinality <cardinality>, it is incompatible with <cardinality>	<b>Invalid Cardinality</b>  The given cardinality (single or multiple) is not compatible with the one requested.
Invalid type <type name>, it is not assignable from type <type name>	<b>Not Assignable From</b>  A given type <b>is</b> cannot be assigned from a requested type. For example: <ul style="list-style-type: none"><li>the title of a CD is one of {1}</li><li>The requested type to be assigned String (the title of a CD) cannot be assigned from Number (1).</li></ul>
Invalid value (<type name>) <value text>	<b>Invalid Value</b>  An error on the value has been reported by a value descriptor associated to the value type (checking of value).
Invalid variable <variable name> for expected type <a type>.	<b>Invalid Variable</b>  The type of a variable is not compatible with the requested type.
Number must be integer.	<b>Non Integer Value</b>  An integer value must be specified.
Value is not in domain.	<b>Value Not In Domain</b>  A BOM domain has been specified and the value used in the rule is violating this domain.
Value is out of the range	A range BOM domain has been specified and the value used in the rule is out of this range.
Only values are expected.	<b>Invalid Expression</b>  Expressions such as sentences, variables and grammar constructions are not valid. Only value expressions are accepted.
Incompatible sentence, due to incompatible precedence	Some operators are incompatible with the types used in the sentence.  For example, in the sentence print "Hello world!" +2 -3, the operators are incompatible, because "Hello world!" +2 is evaluated as a String, and the operator - is incompatible with the String type. To solve the problem, use explicit parentheses: print "Hello world!" + (2 - 3)".
Origin type: <type name> is not a super type of target type: <type name>.	<b>Not Super Type Of</b>  A type is not a subtype of the requested type. For example:  the name of the customer is 3  The requested type is String (the name of the customer) and the given type is Number (3). String is not a super type of Number.

The variable <variable name> cannot be used, a variable of that type is already declared.	<b>Automatic Variable Bound</b>  Cannot use the automatic variable of a type, where there is a global variable of that type already declared.
The variable <variable name> is not assignable.	<b>Variable Not Assignable</b>  The variable cannot be assigned.
The word <word> is expected in place of <word>.	<b>Token Mutation</b>  A word in the rule is wrong, and the error recovery mechanism suggests another one.
The word <word> is missing.	<b>Missing Token</b>  A word is missing in the rule.
The word <word> is not required.	<b>Inserted Token</b>  A word has been inserted in the wrong place in the rule.
Unknown word: <word>.	<b>Unknown Token</b>  A word is not recognized. This usually occurs with a malformed value.
The phrase <phrase> is not required.	<b>Not Implemented</b>  A complete phrase is in the wrong location and must be removed to correct the rule.
The phrase <phrase> is missing.	<b>Not Implemented</b>  A complete phrase is missing in the text. Complete the phrase to correct the rule.
No term <term> found.	<b>Term Does Not Exist</b>  There is a reference to a term that is not in the vocabulary.
No navigation phrase <phrase> (of subject <subject>) found on term <term>.	Navigation sentence on a given term does not exist in the vocabulary.
No action phrase <phrase> (of subject <subject>) found on term <term>.	Action sentence on a given term does not exist in the vocabulary.
No automatic variable <variable> found.	The automatic variable is not declared automatic on a term, or a term does not exist in the vocabulary.
Too many errors. Cannot recover from errors.	The rule contains too many errors to correct.
Invalid value for <value>: <value>	<b>Invalid Value</b>  Invalid value for the Guided Editor.
Invalid variable name: <variable>, the characters: <character> are not allowed.	<b>Invalid Name/Character</b>  The variable name is invalid and contains one or more of the following invalid characters:

	' " \n \r \t / ( ) , ;
Unknown phrase, did you mean: <phrase>?	<b>Unknown Phrase</b>  A sentence in the text is incorrect but similar to an existing correct sentence. The message suggest the correct sentence.
Value (<value type>) <value text> is incorrect. <a message>	<b>Value Error</b>  An error on the value has been reported by a value info associated to a role in the phrase (checking of value). If the value is used in a place where a BOM domain is defined, this error is generated if the value violates the BOM domain.
Variable <variable name> already declared.	<b>Variable Already Declared</b>  A variable of that name is already declared.
Variable <variable name> is not declared.	<b>Variable Not Declared</b>  A reference to a variable that it is not declared.

The following table lists the most current warnings you might see when editing an action rule.

Table 2. Action rule editing warnings quick reference

Message	Description
<an element> is deprecated	<b>Deprecated Element</b>  A deprecated element (type, constant or phrase) has been used in the rule. Deprecated is a property defined on a business element.
Can not bind value types.	A value type such as number, string, and so on cannot be used in a binding.
Category(ies) (<categories>) defined on phrase <a phrase> does not match the rule category filter.	<b>Invalid Phrase Category</b>  The phrase is not intended to be used on the rule, because the categories of that phrase do not match the rule category filter.
Category(ies) (<categories>) defined on term <a term> does not match the rule category filter.	<b>Invalid Term Category</b>  The term is not intended to be used on the rule, because the categories of that term do not match the rule category filter.
Category(ies) (<categories>) defined on type of variable <variable name> does not match the rule category filter.	<b>Invalid Variable Category</b>  The variable is not intended to be declared on the rule, because the categories of the variable type do not match the rule category filter.
Category(ies) (<categories>) defined on value <a constant> does not match the rule category filter.	<b>Invalid Constant Category</b>  The constant is not intended to be used on the rule, because the categories of that constant do not match the rule category filter.
Predicate not applicable to value types.	<b>Predicate Not Applicable</b>  The predicate is not applicable to the value type, such as number, string, and so on.
The construct <construct> is	

The construct <construct> is deprecated.	<b>Construct Deprecated</b>  A deprecated grammar construct (issued from the XML schema definition of the language) has been used in the rule.
The (part of the) rule execution may be unsafe: <message>.	<b>Rule Execution Server</b>  The rule tries to set an attribute to a value outside its domain.
The (part of the) rule is never applicable because <message>.	The rule never applies because its conditions can never be met. This is usually caused by simple logic errors in the rule.
The rule is incomplete, fill all the placeholders.	<b>Incomplete Rule</b>  A rule has at least one empty placeholder.
Value (<value type>) <value text> is incorrect. <a message>	<b>Value Warning</b>  A warning on the value has been reported by a value info associated to a role in the phrase (checking of value).
Variable <variable name> is not used.	<b>Variable Not Used</b>  A variable is declared but not used.

**Parent topic:** [Working with action rules](#)

**Related concepts:**  
[Consistency checking](#)  
[Action rules](#)  
[Technical rules](#)

**Related information:**  
[Vocabulary errors and warnings](#)  
[Working with action rules](#)  
[Business Action Language \(BAL\)](#)



# Creating an action rule

When you create a new action rule, you specify the rule project to which it belongs. You can also store the rule in a specific package.

## Procedure

To create an action rule:

1. Select your rule project.
2. On the **File** menu, click **New** > **Action Rule**.

The New Action Rule wizard opens.

3. If you want to store the rule in a rule package, click **Browse** in the **Package** field, and then select the required package.
4. In the **Name** field, type the name of your action rule.

**Note:** Avoid using any of the following characters in the name: \* ? " < > | \ /

5. In the **Template** field, click **Browse** if you want to use an action rule template. (Rule templates are deprecated.)
6. Click **Finish**.

The new action rule is displayed in the Rule Explorer view and the Intellirule Editor opens. This is the default editor: any time you create or open an action rule, you open the Intellirule editor.

## Results

Rule Designer provides the following editors for you to edit action rules:

- **Intellirule Editor:** a text editor that enables you to build rules using a Content Assist system.
- **Guided Editor:** a point-and-click editor that enables you to build rules by entering text or numbers in fields, or by selecting items from drop-down lists.

You can choose which editor you want to use when you build a rule.

**Parent topic:** [Working with action rules](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

### Related reference:

[Business Action Language \(BAL\)](#)

### Related information:

[Building rules using the Intellirule editor](#)

[Building action rules using the guided editor](#)

# Selecting an editing mode for your action rules

You can choose a predefined editing profile or define a custom profile that best suits your preferences for editing action rules.

## About this task

You can choose the appropriate editing mode for your profile when you create and edit action rules.

## Procedure

To select your mode for editing action rules:

1. On the **Window** menu, click **Preferences**. (On Mac, click **Eclipse** > **Preferences**.)
2. In the Preferences dialog, click **Rule Designer** > **Action Rule Editing**.
3. On the Action Rule Editing page, select a user profile. Each profile has its own set of editing options:
  - **Guided**: Provides assistance by enabling most of the editing options automatically.
  - **Developer**: Enables some options automatically, which you can deselect.
  - **Custom**: You choose the options that you want to enable.
4. Select the editing options that you need.
5. Click **Apply**, and then click **OK**.

**Parent topic:** [Working with action rules](#)

# Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

## **Overview: Intellirule rule editor**

You build rules with the assistance of a completion menu.

## **Creating rule parts**

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

## **Using punctuation to clarify rules**

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

## **Setting completion menu options**

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

## **Highlighting and correcting errors**

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

## **Using action rules to open business elements**

You can use the text in an action rule or in the vocabulary browser to open a business element for editing in the BOM editor.

**Parent topic:** [Working with action rules](#)

### **Related concepts:**

[Action rules](#)

### **Related tasks:**

[Creating an action rule](#)

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

### **Related reference:**

[Business Action Language \(BAL\)](#)

### **Related information:**

[Building action rules using the guided editor](#)

## Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

When you create or open an action rule, the Intellirule Editor opens by default. If you are currently using the guided editor in a Designer, you can switch to the Intellirule editor: close the guided editor, right-click the action rule you want to edit in the Rule Explorer view, and then click **Open With > Intellirule Editor**.

You can add terms and phrases to a rule by typing or pasting in text, or by selecting them from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, value, or other phrase.

### Note:

See the [Rule editor flash demo](#) on the IBM® Customer Support site for an English-only tour of the rule-editing features.

## Terms and phrases

The rule editor provides an editing area for building rules. You can add terms and phrases in the following ways:

- Type directly in the editing area.
- Copy text from another editor or application, and paste it into the editing area.
- Select predefined terms and phrases from a completion menu.

## Completion menu

The business vocabulary of your company defines the terms and phrases in the completion menu, and the ways you can combine them.

## Placeholders

Placeholders indicate places in a term or phrase that you must complete. You must insert the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain more placeholders that you must also complete to construct a valid expression.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

### About this task

An action rule can comprise some or all of the following parts. The parts must be defined in the following order: definitions, if, then, and else.

### Procedure

1. Click anywhere in the editing area and press Ctrl+Spacebar. The completion menu displays a list of available insertion options that are based on your current position in the rule.
2. Select an insertion option from the completion menu.
3. Press Esc or click anywhere in the editing area to hide the completion menu.

**Parent topic:** [Building rules using the Intellirule editor](#)


## Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Ambiguities occur when part of a rule has at least two interpretations that are semantically correct, or when there are two identical terms or phrases in the vocabulary.

For example, the following statement is ambiguous:

```
if
all of the following conditions are true :
 - the category of the customer is Gold
 - the age of the customer is at most 15
and
 the salary of the customer is more than 100
```

Rule Designer raises an ambiguity error (which appears in green ) because it cannot tell whether the salary of the customer is more than 100 is associated with the age of the customer is at most 15, or whether it is the third condition of the rule.

To remove the ambiguity, you can add a comma at the end of the the age of the customer is at most 15 condition statement:

```
if
all of the following conditions are true :
 - the category of the customer is Gold
 - the age of the customer is at most 15,
and
 the salary of the customer is more than 100
```

An alternative way to remove ambiguity is to use parentheses to group expressions.

In cases where there are two identical terms or phrases in the vocabulary, change one of the terms or phrases to remove the ambiguity.

**Parent topic:** [Building rules using the Intellirule editor](#)

### Related concepts:

[Rule variables](#)

[Condition negation](#)

[Action rules](#)

### Related tasks:

[Creating rule parts](#)

[Correcting errors by using Quick Fix](#)

### Related information:

[Action rule editing errors and warnings](#)

[Building rules using the Intellirule editor](#)

[Business Action Language \(BAL\)](#)

[Using action rules to open business elements](#)

# Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

## Procedure

1. In the toolbar above the editing area, click **Completion Menu Options**. A window opens.
2. Select the check boxes for the options you want to set.

Table 1. Completion menu options

Option	Description
Enable on Spacebar	<p>Select to open the completion menu whenever you press Spacebar while you are typing in the editing area.</p> <p>Clear this check box if you want the completion menu to open only when you press Ctrl+Spacebar.</p>
Enable on double-click	<p>Select to open the completion menu by double-clicking a word in the editing area.</p> <p>Clear this check box if you want double-clicking to select the current word instead.</p>
Enable auto-restart	<p>Select to automatically reopen the completion menu when a term or phrase is selected.</p> <p>Clear this check box if you want to open the completion menu by pressing Ctrl+Spacebar.</p>
Enable smart mode	<p>Select to filter the completion menu entries in a smart way to reduce the number of entries.</p>
Enable template mode	<p>Select if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable this mode if you want to insert longer phrases, and then substitute placeholders.</p> <p>Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate this mode if you prefer to build a complete rule in the same way that you might compose an email message.</p>
Use Hierarchical View	<p>Select if you want the completion menu to group terms and phrases by type, for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.</p> <p>Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.</p>
Filter unreachable phrases	<p>Select if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable.</p>
Display toolbar	<p>Select if you want the completion menu toolbar to be displayed above the list of available terms and phrases.</p>

	Clear this check box if you want to hide the completion menu toolbar.
<b>Display documentation</b>	<p>Select if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.</p> <p>Clear this check box if you want to hide the hover help.</p>

3. Click outside the pop-up window to save your settings.

**Parent topic:** [Building rules using the Intellirule editor](#)



## Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

### About this task

Correct errors manually in rule and business model files. You can use the Problems view to view the error messages and navigate to the errors in your files.

### Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
  - Hover the mouse over a highlighted error in the rule editor to display an error tip icon.
  - Double-click the text that is displayed in the error description in the Problems view to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Using action rules to open business elements

You can use the text in an action rule or in the vocabulary browser to open a business element for editing in the BOM editor.

### Editing a business element from the text of a rule

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements from the text of the rule.

### Editing a business element from the Vocabulary Browser

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements by using a Vocabulary Browser, which shows the vocabulary elements available in the current rule.

**Parent topic:** [Building rules using the Intellirule editor](#)

# Editing a business element from the text of a rule

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements from the text of the rule.

## Procedure

To edit a business element from the text of a rule:

1. In the Intellirule editor, press Ctrl.  
The terms and phrases used in the rule are displayed as links.
2. Click the term or phrase corresponding to the business element you want to edit.



The BOM Editor opens on the selected business element.

3. Make the required changes, and then from the **File** menu, click **Save**.

## Results

Business elements can be edited in the Vocabulary Browser.

**Parent topic:** [Using action rules to open business elements](#)

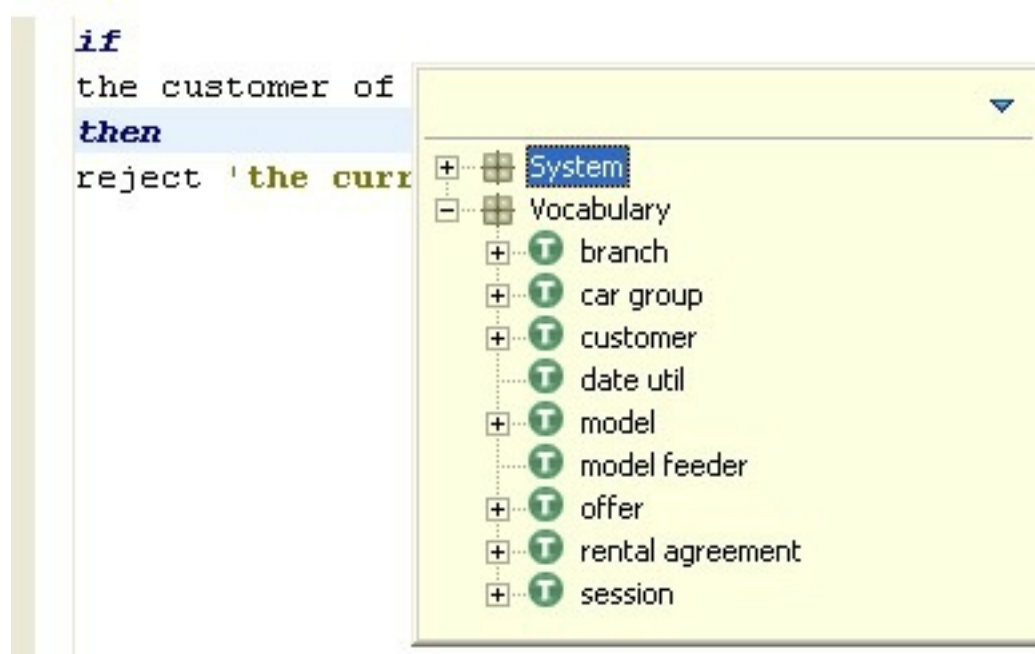
## Editing a business element from the Vocabulary Browser

You can open the BOM Editor from within the Intellirule Editor to view and edit business elements used in rules. You can access business elements by using a Vocabulary Browser, which shows the vocabulary elements available in the current rule.

### Procedure

To edit a business element from the Vocabulary Browser:

1. In the Intellirule editor, press Ctrl+B.
2. In the Vocabulary Browser, either type the first letters of the business element you want to access, or browse to it.



3. Double-click the business element.

The BOM Editor opens on the selected business element.

4. Make the required changes, and then from the **File** menu, click **Save**.

**Parent topic:** [Using action rules to open business elements](#)

# Building action rules using the guided editor

When you use the guided editor to build rules, you construct predefined rule fragments by entering text or numbers in fields or by selecting fragments from drop-down lists.

## [Switching to the guided editor](#)

When you create or open an action rule, the Intellirule editor opens by default. If you want to work in a more guided environment, you can switch to the guided editor.

## [Controls in the guided editor](#)

The guided editor displays controls that you can click to build rule statements.

## [Creating rule parts](#)

You can create action rule definitions, conditions, and actions.

## [Adding or removing rule statements](#)

You can add or remove statements in any part of an action rule.

## [Defining a variable](#)

You define variables in the definitions part of an action rule.

## [Choosing whether to enter values or select items](#)

You can build fragments by entering values in fields or by selecting items from a drop-down list.

## [Inserting an arithmetic expression](#)

Use the arithmetic operator icon to insert arithmetic expressions.

## [Adding a statement to a group of statements](#)

Use the hyphen icon to add a statement to a group of statements.

## [Controlling how condition statements are combined](#)

Use logical operators to control the way condition statements are combined.

## [Changing the grouping order of condition statements](#)

Use opening and closing parentheses to control the grouping order of condition statements.

## [Negating a condition statement](#)

Use the Negate command to negate the sense of a condition statement.

**Parent topic:** [Working with action rules](#)

### **Related concepts:**

[Action rules](#)

### **Related tasks:**

[Creating an action rule](#)

[Creating an action rule template](#)

[Defining a folder structure for rule project items](#)

### **Related reference:**

[Business Action Language \(BAL\)](#)

### **Related information:**

[Building rules using the Intellirule editor](#)

## Switching to the guided editor

When you create or open an action rule, the Intellirule editor opens by default. If you want to work in a more guided environment, you can switch to the guided editor.

### Procedure

To switch from the Intellirule editor to the guided editor:

1. Save any changes that you made to your rule and close the Intellirule editor.
2. In the Rule Explorer view, right-click the action rule that you want to edit, and then click **Open With > Guided Editor**.



The guided editor opens.

**Parent topic:** [Building action rules using the guided editor](#)

# Controls in the guided editor

The guided editor displays controls that you can click to build rule statements.

The following table displays the controls that you can click to add fragments and build rule statements.

Control	Description
[definitions], [if], [else]	Add a new rule part. See <a href="#">Creating rule parts</a> .
	Add a definition, condition, or action statement. See <a href="#">Adding or removing rule statements</a> .
	Select a value from a drop-down list. See <a href="#">Choosing whether to enter values or select items</a> .
[#]	Insert arithmetic operators into arithmetic expressions. See <a href="#">Inserting an arithmetic expression</a> .
[ ]	Add a statement to a group of statements. See <a href="#">Adding a statement to a group of statements</a> .
and, or	Control how condition statements are combined. See <a href="#">Controlling how condition statements are combined</a> .
[where]	Add restrictions to definition or condition statements. See <a href="#">where &lt;test&gt;</a> .
[from/in]	Use from to specify the origin of objects of a one-to-many relation used in a definition statement (see <a href="#">from &lt;object&gt;</a> ).  Use in to specify the source of objects used in a definition and condition statements (see <a href="#">in &lt;list&gt;</a> ).

**Parent topic:** [Building action rules using the guided editor](#)

**Related tasks:**  
[Freezing and unfreezing parts of an action rule template](#)

# Creating rule parts

You can create action rule definitions, conditions, and actions.

## Overview

An action rule can include some or all of the following parts. The parts must be defined in the order listed.

- definitions
- if
- then
- else

Square brackets [ ] around the title of a part of a rule means that it is empty and optional. The then part of the rule is not shown between brackets because it is a required part of an action rule.

## Building an optional part of a rule

To build an optional part of a rule:

1. Click the title between brackets. The part expands to display the placeholders and fragments that you can use.
2. Click placeholders and select the elements that you require to build the rule.

## Building the mandatory then part of a rule

To build the mandatory then part of a rule:

1. Click <select an action>.
2. Select the required action.

The following example shows a complete rule:

```
definitions
 set minimum score to ▼ the number: 200 [4]
 [where]
 ↩
if
 the credit score of the borrower [4] is less than minimum score [4]
 ↩
then
 in the loan report , refuse the loan with the message ▼ "Credit score below" [4] + minimum score [4] [4]
 ↩
[else]
```

**Parent topic:** [Building action rules using the guided editor](#)

### Related information:

[Building action rules using the guided editor](#)  
[Business Action Language \(BAL\)](#)




## Adding or removing rule statements

You can add or remove statements in any part of an action rule.

### Procedure

You can do this as follows:

1. To add a statement after the last existing statement, click the blue arrow icon  below the last existing statement.
2. To add a statement before an existing statement, right-click the first building block of an existing statement and click **Insert Definition**, **Insert Condition**, or **Insert Action**.
3. To delete a statement:
  - a. Select the statement or statements you want to delete.
  - b. Right-click the first building block of the statement and click **Delete Definition**, **Delete Condition**, or **Delete Action**.

**Parent topic:** [Building action rules using the guided editor](#)

## Defining a variable

You define variables in the definitions part of an action rule.

### Procedure

To define a variable:

1. Open the definitions part of the rule.
2. Click the **variable1** field and type a name for the variable.
3. Select an item from the **<select a choice >** drop-down list. You can choose whether you want to bind the variable to:
  - a sentence from the vocabulary
  - **an <object >** : an instance of an object. If you choose **an <object >** , the placeholder **<select an object >** is shown. Click the placeholder and select the type of object you want to bind.
  - **all <objects >** : all instances of the same type. If you choose **all <objects >** , the placeholder **<select objects >** is shown. Click the placeholder and select the type of object you want to bind.

### Results

Your variable is now defined. You can use it in all parts of the action rule.

**Parent topic:** [Building action rules using the guided editor](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Adding or removing rule statements](#)

[Inserting an arithmetic expression](#)

[Adding a statement to a group of statements](#)

[Controlling how condition statements are combined](#)

[Negating a condition statement](#)

### Related information:

[Changing the grouping order of condition statements](#)

[Building action rules using the guided editor](#)

[Business Action Language \(BAL\)](#)

[Choosing whether to enter values or select items](#)

[Creating rule parts](#)


## Choosing whether to enter values or select items

You can build fragments by entering values in fields or by selecting items from a drop-down list.

### Selecting items from a drop-down list

By default, whenever a value can be typed, a placeholder **<enter a string>** is displayed with a black arrow next to it.

Follow these steps to select from a drop-down list:

1. Click the black arrow icon  next to the placeholder to display the list of options.
2. Select an item from the drop-down list.

### Entering values in fields

Follow these steps to change from a drop-down list back to a field:

1. Click the building block and select **<enter a string>** from the drop-down list. A field is inserted instead of the building block at that location in your rule statement.
2. Click the field and enter a value.

**Parent topic:** [Building action rules using the guided editor](#)

## Inserting an arithmetic expression


Use the arithmetic operator icon to insert arithmetic expressions.

### About this task

You can build complete arithmetic expressions.

### Procedure

To insert an arithmetic operator:

1. Click the arithmetic operator icon  in a rule statement. The following expression are shown next to the first statement, preceded by the addition operator [+]:

+ <enter a string>

2. If you want to change the addition operator to another operator, click the addition operator and select the one you want from the drop-down list.
3. To complete the arithmetic operation, click **<enter a string>** and choose whether you want to enter a string or select an item from the drop-down list (click the arrow next to it to display the list).

**Parent topic:** [Building action rules using the guided editor](#)

## Adding a statement to a group of statements

Use the hyphen icon to add a statement to a group of statements.


### About this task

You can create groups of condition statements and action statements by using the following building blocks:

- Condition statements:
  - all of the following conditions are true
  - any of the following conditions is true
  - none of the following conditions are true
- Action statements:
  - for each

You can then add or remove statements to or from a group.

### Procedure

1. To add a statement to a group of statements:
  - a. Click the hyphen icon . A new statement is added to the group of statements.
2. To remove a statement from a group of statements:
  - a. Right-click the hyphen icon next to the statement, and then click **Delete Definition**, **Delete Condition**, or **Delete Action**.

**Parent topic:** [Building action rules using the guided editor](#)

## Controlling how condition statements are combined

Use logical operators to control the way condition statements are combined.

### About this task

When there is more than one condition statement in the definitions or if part of a rule, you can control whether the rule conditions are met if all (represented by the logical operator and) or any (represented by the logical operator or) of its condition statements are valid.

### Procedure

To change how condition statements are combined:

1. Click the **and** or the **or** building block.
2. Select the appropriate logical operator from the drop-down list.

**Parent topic:** [Building action rules using the guided editor](#)

# Changing the grouping order of condition statements

Use opening and closing parentheses to control the grouping order of condition statements.

By default, the and logical operator takes precedence over the or logical operator when you have several condition statements.

However, you can use parentheses to group condition statements to change the order in which they are processed.

Action	Procedure
Add an opening parenthesis to a condition statement	Right-click the first building block of the statement and click <b>... (</b> .
Add a closing parenthesis to a condition statement	Right-click the last building block of the statement, or the next operator, and click <b>) ...</b> .
Remove a parenthesis	Right-click the parenthesis and click <b>Delete</b> .
Insert both parentheses in one step	Select the entire phrase, right-click it, and then click <b>(...)</b> .

**Parent topic:** [Building action rules using the guided editor](#)

## Negating a condition statement

Use the Negate command to negate the sense of a condition statement.

### About this task

You can negate the sense of a condition statement by adding `it is not true that` at the beginning of the statement.

### Procedure

1. Right-click the first building block of the statement.
2. Click **Negate**.

#### Note:

To remove `it is not true that` from the statement, right-click it and select **Delete**.

**Parent topic:** [Building action rules using the guided editor](#)



## Creating an action rule type

When you create a new rule model class to define a new type of action rule, the new rule type becomes available in the creation wizards.

### Procedure

To create an action rule of type MyRule:

1. Select your rule project and on the **File** menu click **New** > **Action Rule**.
2. In the New Action Rule wizard, type the name of your action rule in the **Name** field.
3. In the **Type** field, select the custom rule type, in this example, **MyRule**.
4. Click **Finish**.

The new action rule is shown in the Rule Explorer view and the Intellirule editor opens.

**Parent topic:** [Working with action rules](#)

# Working with decision tables

You use the decision table editor to create and update decision tables.

## [Decision tables](#)

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

## [Creating a decision table from scratch](#)

Decision tables represent in tabular form all possible situations that a business decision might encounter.

## [\(Deprecated\) Creating a decision table from a template](#)

You can create decision tables using a decision table template.

## [Defining columns](#)

You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.

## [Adding and removing columns](#)

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.

## [Sorting and moving columns](#)

You can sort condition columns and move action columns.

## [Adding and populating rows](#)

You can add blank rows, partially completed rows, and otherwise rows to a decision table. You can also move, split, merge and delete rows, and populate them with values from an enumerated domain.

## [Specifying values and operators](#)

You can specify values and operators in condition statements.

## [Disabling action cells](#)

You can disable an action cell by clicking an icon on the menu bar or by using the Enable / Disable Action command.

## [Populating a decision table from an Excel spreadsheet](#)

You can copy the contents of an Excel spreadsheet into a decision table provided the structure of the Excel spreadsheet and the decision table are the same.

## [Defining preconditions](#)

You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.

## [Formatting a decision table](#)

You can define format settings for a table, a column, or individual cells.

## [Defining checking options](#)

You can specify tests that Rule Designer conducts to identify problems with a decision table.

## [Value test conditions](#)

When you construct conditions that test values, the way you complete the associated input fields depends on the type of test.

## [Using decision table locking facilities](#)

You can apply a variety of locks to protect your decision tables against editing by others.

## [Viewing rule details](#)

You can view the details of a rule in a decision table row by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision table editor.

## [Viewing the decision table as a spreadsheet](#)

You can display a decision table in a spreadsheet view if the decision table has two condition columns and one action column and the structure of the decision table is symmetric.

## [Documenting a decision table](#)

You can document each row in a decision table.

**Parent topic:** [Authoring business rules](#)

# Decision tables

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥600,000		true	0.005
5	B	<100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		≥600,000		true	0.0075

When an application calls a decision table, the action rules are executed. If the conditions in a row are met, the rule that is formed by the row performs the actions in the row.

You can add rows to the decision table and enter values in their cells to create new rules. You can also define preconditions that apply to all the rules in a table. Error markers help you find overlaps and gaps in your rules.

## Columns

Each column in a decision table represents a condition or an action.

## Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

## Preconditions

You can pretest data before using it with a decision table.

## Decision table errors and warnings

Rule Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.

**Parent topic:** [Working with decision tables](#)

**Related concepts:**

[Action rules](#)

[Technical rules](#)

[\(Deprecated\) Decision tree overview](#)

**Related information:**

[Overview: Ways to express business rules](#)

# Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. The top cell of each column identifies the object of a condition or the target of an action.

Row	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001

Each numbered row in the table forms a rule. The rule performs the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of the loan is between 100000 and 300000
then
 set the Insurance required to true
 set the Insurance rate to 0.001
```

## Condition operators

You can split a condition across columns in a decision table when a rule statement contains more than one value. For example, the following condition requires you to specify values for <min> and <max>:

```
if
 the age of the customer is between <min> and <max>
```

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [Decision tables](#)

# Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

## Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

**Note:** If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In Rule Designer, you group cells by merging them.

In the following table, the Grade A cells of rows 1 and 2 are grouped into one cell. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
then
 - set the insurance required to true
 - set the loan rate to 0.001
```

- Rule 2

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row that has the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3		300,000	600,000	true	0.003

The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.005
5	B	< 100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		600,000	800,000		
9		≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003
3	A			12	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.004

Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003
3		Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3		Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
then
 - set the insurance required to true
 - set the loan rate to 0.001
```

- Rule 2

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.002
```

- Rule 3

```
if
 all of the following conditions are true:
 - the loan grade is A
 - it is not true that the amount of loan is between 300000 and
600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

Parent topic: [Decision tables](#)





## Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
 set 'wealthy customer' to a customer
 where the average monthly balance of this customer
 is more than $1 000 000
if
 the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.


**Parent topic:** [Decision tables](#)

# Decision table errors and warnings

Rule Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.

As you submit values to the cells of your decision table, Rule Designer checks the column automatically for errors. As shown in the following table, when there is a problem, the column header displays a warning symbol **1** and the affected cells are highlighted:

- A red line for an error **2**
- A yellow line for a warning **3**

State	Age of the customer 	
	<min>	<max>
Rhode Island	18	22
	21	25

The errors and warnings are also listed in the Problems view.

You can define different checking properties for different condition columns.

## Overlap warnings

Rule Designer can verify that the values you enter for a given condition throughout the different rows do not overlap or are not identical.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 29 and 40

In this example, customers that are 29 years old satisfy the condition of both rows. Rule Designer reports this as an overlap warning.

Rule Designer checks for overlaps for all the entries in a column, and within partitioned groups of cells.

The overlap checker supports the following types:

- Number (discrete and continuous): int, long, float and double.
- The various types of Date defined in the Boot BOM: Date, SimpleDate, UniversalDate, Time, UniversalTime, DayOfWeek, Month, and Year.

The extended BAL also supports:

- Percentage (ilog.rules.brl.Percentage)
- Currency (ilog.rules.brl.Currency)

The following operators are supported for all types of object: is, is not, is one of, is not one of.

## Hierarchical overlap warnings

Rule Designer checks for hierarchical overlaps (that is, overlaps involving more than one column) and distinguishes between real overlaps (duplications in multiple columns) and local overlaps (duplications of a value in one column).

In the following example, Rule Designer identifies a real overlap because the adjoining cells in the Age and Category columns contain identical values:

Age	Category
18	Gold
18	Gold

For real overlaps, Rule Designer does the following:

- Displays a warning triangle in the Age column header to highlight the duplication of the value 18.
- Displays a wavy yellow line on the side of each row.
- Adds a warning in the Problems view.

In the following example, Rule Designer identifies a local overlap because although the adjoining cells in the Age column are identical, the Category column contains distinct values (Gold in row 1 and Silver in row 2).

Age	Category
18	Gold
18	Silver

For local overlaps, Rule Designer does the following:

- Displays a warning triangle in the Age column header to highlight the duplication of the value 18.
- Displays a wavy blue line on the side of each row.
- Does not add a warning in the Problems view.

Gap warnings

Rule Designer can verify that the cells in a condition column consider all possible cases, which helps you make sure there are no gaps in your tables.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 32 and 40

In this example, customers that are 31 years old are never be taken into account. Rule Designer reports this as a gap warning.

Rule Designer evaluates gaps for all the entries in a column, and within partitioned groups of cells.

The gap checker supports the following types:

- Number (discrete and continuous): int, long, float and double.
- Types that describe concepts. For example, type Category, which might have items such as Silver, Gold and Platinum.
- The various types of Date defined in the Boot BOM: Date, SimpleDate, UniversalDate, Time, UniversalTime, DayOfWeek, Month, and Year.

There are two modes for numbers: interval and infinity.

- Interval checks for gaps from the smallest value to the largest value.
- Infinity checks for gaps from minus infinity to plus infinity.

Gap and overlap warning limitations

The detection of gaps and overlaps in decision tables is too complex to do under all circumstances. Rule Designer consequently applies limits in the following areas:

- Complex column expressions
- Hierarchical overlaps

**Complex column expressions:** Rule Designer only detects gaps and overlaps for columns with simple expressions. If an expression is too complex (for example, expressions that involve operators), no gaps or overlaps are computed.

The following expressions are considered to be simple:

- `the amount of the loan is <a number>`
- `myParam is <a number>`

where myParam is a ruleset parameter or variable.

- `var is <a number>`

where var is a variable defined as a precondition and assigned a simple expression.

The following expressions are considered to be too complex for gap and overlap calculations:

- `the amount of the loan + 1,000 is <a number>`
- `var is <a number>`

where var is a variable defined as a precondition and assigned a complex expression.

**Hierarchical overlaps:** Rule Designer checks for hierarchical overlaps on a best-effort basis to avoid the high computational cost of detecting all possibilities. Hierarchical overlaps might not be computed accurately when

conditions are added or removed; that is, when structural changes occur on the partition where a hierarchical overlap is computed.

## **Symmetry information**

Rule Designer can verify that all partitions use the same set of items. You can check symmetry for the whole table or for specified columns. Such checking is useful to make sure that all choices are covered at the same level in the decision table.

When enabled, symmetry checking is just given as information, allowing you to create non symmetric tables.

**Parent topic:** [Decision tables](#)

### **Related information:**

[Vocabulary errors and warnings](#)

[Business Action Language \(BAL\)](#)

# Creating a decision table from scratch

Decision tables represent in tabular form all possible situations that a business decision might encounter.

## About this task

Decision tables comprise rows and columns, and each row corresponds to a rule. They are used to represent in tabular form all possible situations that a business decision might encounter, and to specify which action to take in each of these situations.

## Procedure

To create a decision table from scratch:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Table**.

The New Decision Table wizard opens.

3. Optional: Add the decision table to an existing rule package:
  - In the **Package** field, click **Browse** and click the required rule package.
4. In the **Name** field, type the name of the decision table.
5. Click **Finish**.

The new decision table is displayed in the Rule Explorer view and the decision table editor opens.

The default decision table has three condition columns and one action column. Action columns have a shaded background.

## Results

You can now use the decision table editor to define the condition and action columns for the decision table, and to specify values for each row, that is, for each rule.

**Parent topic:** [Working with decision tables](#)

### Related tasks:

[Correcting errors by using Quick Fix](#)  
[Setting up a decision table template](#)  
[Populating a decision table from an Excel spreadsheet](#)  
[Defining preconditions](#)  
[Formatting a decision table](#)  
[Defining checking options](#)  
[Using decision table locking facilities](#)  
[Viewing rule details](#)  
[Viewing the decision table as a spreadsheet](#)  
[Documenting a decision table](#)  
[\(Deprecated\) Creating a decision table from a template](#)  
[Defining columns](#)

### Related information:

[Decision tables](#)  
[Business Action Language \(BAL\)](#)  
[Adding and populating rows](#)  
[Specifying values and operators](#)

## (Deprecated) Creating a decision table from a template

You can create decision tables using a decision table template.

### About this task

If you have a number of sets of rules that are similar, rather than creating a new decision table each time, you can use a decision table template.

### Procedure

To create a decision table from a template:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Table**.

The New Decision Table wizard opens.

3. Optional: Add the decision table to an existing rule package:
  - In the **Package** field, click **Browse** and then click the required rule package.
4. In the **Template** field, click **Browse**. The Select a template dialog opens.
5. Click the template you want to use and then click **OK**.
6. In the **Name** field, type the name of the decision table.
7. Click **Finish**.

### Results

The new decision table is displayed in the Rule Explorer view and the decision table editor opens. You can now use the Decision Table Editor to define the condition and action columns for the decision table and to specify values for each row; that is, for each rule.

**Parent topic:** [Working with decision tables](#)

# Defining columns

You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.

## About this task

A decision table must have at least one condition column and one action column. You can define condition and action columns using the decision table editor.

In a condition column, you enter a condition statement that defines a comparison but not its values.

A condition statement is an incomplete BAL predicate expression whose placeholders are mapped to specific subcolumns. Placeholders are specific tokens enclosed within < and >. They can represent a concept from the vocabulary (for example, <an object>, <a loan>, or <a string>), or empty text defined in the vocabulary phrase, such as is between <min> and <max>.

If you create a condition statement that has more than one placeholder, the required subcolumns are automatically created. If the expression does not contain any placeholders, Rule Designer completes the expression with an ... is a <boolean> statement so that it has a placeholder.

**Note:** An expression can have placeholders anywhere. However, any ambiguity about the expected definition might trigger a warning.

In an action column, you specify each action to take in each of the specified situations.

## Procedure

To define condition and action columns:

1. Use the decision table editor to define the required columns, as summarized in the following table:

To define a condition column:	To define an action column:
<div><div>a. Double-click the top cell of the condition column. The Condition Column window opens.</div><div>b. In the <b>Title</b> field, type the required column title.</div><div>c. In the <b>Test</b> field, double-click &lt;condition&gt; and construct your condition statement in the Column Expression Editor using the available elements from the vocabulary.</div></div>	<div><div>a. Double-click the top cell of the action column. The Action Column window opens.</div><div>b. In the <b>Title</b> field, type the required column title.</div><div>c. In the <b>Action</b> field, double-click &lt;action&gt; and construct your action statement in the Action Expression Editor using the available elements from the vocabulary.</div></div> <div>To define an action column in the decision table editor edit bar, click a cell in the action column you want to define, and select the required action.</div>

**Note:** Press Ctrl+Spacebar to open the Content Assist box when constructing your condition or action statement.

2. When you have constructed your condition or action statement, click **OK**.

## Results

Your columns are now defined. If you need to redefine a column at any time, double-click the column header and change the required details.

When you have defined at least one condition column and an action column, you can begin to specify values for the placeholders in the condition and action definitions. See [Specifying values and operators](#) for details.

**Parent topic:** [Working with decision tables](#)

## Related tasks:

- [Formatting a decision table](#)
- [Documenting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)
- [Adding and removing columns](#)
- [Sorting and moving columns](#)

## Related information:

- [Decision tables](#)
- [Adding and populating rows](#)



# Adding and removing columns

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.





## About this task

You can insert additional condition and action columns into a decision table, and remove any columns you do not want.

## Procedure

To add or remove columns:

1. Use the decision table editor to add or remove the required columns, as summarized in the following table:

To add a condition column:	To add an action column:	To remove a column:
<div><div>a. Right-click a condition column next to the place you want to insert the new condition column.</div><div>b. Click  <b>Insert Condition Column Before</b> or  <b>Insert Condition Column After.</b></div><div>A new condition column is added to the decision table.</div></div>	<div><div>a. Right-click an action column next to the place you want to insert the new action column.</div><div>b. Click  <b>Insert Action Column Before</b> or  <b>Insert Action Column After.</b></div><div>A new action column is added to the decision table.</div></div>	<div><div>a. Right-click the column you want to remove.</div><div>b. Click <b>Remove Condition Column</b> or <b>Remove Action Column.</b></div><div>The column and any dependent cells are removed from the decision table.</div><div>If you want to remove an action column that is visible, but all other action columns are hidden, you must first make at least one other action column visible: right-click the header of the column that you want to make visible, click <b>Edit action column</b> and check the <b>Visible</b> check box.</div></div>

**Note:** If you insert a column at the first position in the table, you create a new Root partition whose children are all non-empty conditions of the existing first column.

2. Save your changes.

**Attention:** You can remove or add sub columns by changing the definition of the column. When you do, there is no automatic refactoring of the data contained in the sub columns. To avoid losing data, you can copy the content of the sub columns in an excel sheet, add a column with the new definition in Rule Designer, and paste the data in the new sub columns.

Parent topic: [Working with decision tables](#)

## Related tasks:

- [Formatting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)

## Related information:

- [Adding and populating rows](#)



# Sorting and moving columns

You can sort condition columns and move action columns.

## About this task





You can sort only condition columns. You can sort an entire column or a group of related cells. Sorting a condition column impacts the order in which the rules in the corresponding rows are executed.

You can move only action columns. When you move an action column, it changes the order in which the actions are executed.

## Procedure

To sort or move columns:

- 1. Use the decision table editor to sort or move the required columns, as summarized in the following table:

To sort a condition column:	To move an action column:
<div>a. In the decision table editor, right-click a cell in a condition column.</div> <div>b. Click  <b>Sort Ascending</b> or  <b>Sort Descending</b>.</div>	<div>a. In the decision table editor, right-click the action column you want to move.</div> <div>b. Click  <b>Move Action Column Left</b> or  <b>Move Action Column Right</b>.</div>

- 2. Save your changes.

**Parent topic:** [Working with decision tables](#)

### Related tasks:

- [Formatting a decision table](#)
- [Creating a decision table from scratch](#)
- [\(Deprecated\) Creating a decision table from a template](#)
- [Adding and removing columns](#)
- [Defining columns](#)

### Related information:

- [Decision tables](#)
- [Adding and populating rows](#)

# Adding and populating rows

You can add blank rows, partially completed rows, and otherwise rows to a decision table. You can also move, split, merge and delete rows, and populate them with values from an enumerated domain.

## Adding a blank row

You add and populate a row for each rule you want to define.

## Adding a partially completed row

You can add partially completed rows to a decision table.

## Adding an otherwise row

You can add an otherwise row to a condition column for when none of the other conditions in the column are correct.

## Populating rows with values from an enumerated domain

You can add new rows and populate them with values from an enumerated domain.

## Adding rows that contain values

You can add rows that already contain values to a decision table to create or extend a series of rules.

## Moving rows

You can move rows up or down in a decision table, using the decision table editor.

## Splitting rows

You can split a row if one of its columns has two subcolumns.

## Merging rows

You can merge rows in a decision table.

## Deleting rows

You can delete decision table rows.

**Parent topic:** [Working with decision tables](#)

**Related information:**



[Working with decision tables](#)

# Adding a blank row

You add and populate a row for each rule you want to define.

## Procedure

To add a blank row to a decision table:

- 1. In the decision table editor, right-click a cell in the first column.
- 2. Click **Add**, and then click either **Insert New Row Before**  or **Insert New Row After** .

A new row is added either above or below the row containing the cell you selected. All the values for the row are empty.

	State	Age of the customer		Rental Rejected
		Minimum Age	Maximum Age	
1	New York	18	21	true
2		21	25	false
3				
4	Rhode Island	16	21	true
5		21	25	true

Parent topic: [Adding and populating rows](#)



Related information:  
[Working with decision tables](#)

# Adding a partially completed row

You can add partially completed rows to a decision table.

## Procedure

To add a partially completed row:

- 1. In the decision table editor, right-click the condition cell immediately next to the cell containing the value or values you want to copy to the new row.
- 2. Click **Add**, and then click either **Insert New Row Before**  or **Insert New Row After** .

A new row is added either above or below the row containing the condition cell you selected. The values next to the cell you right-clicked are automatically copied into the new row.

	State	Age of the customer		Rental Rejected
		Minimum Age	Maximum Age	
1	New York	18	21	true
2		21	25	false
3				
4	Rhode Island	16	21	true
5		21	25	true

Parent topic: [Adding and populating rows](#)

Related information:  
[Working with decision tables](#)

# Adding an otherwise row

You can add an otherwise row to a condition column for when none of the other conditions in the column are correct.

## Procedure

To add an otherwise row:

Right-click the condition cell and click **Add > Otherwise**.

A new, partially completed row is added below the row you right-clicked. The cell below the condition cell you selected contains the text otherwise.

	State
1	New York
2	
3	
4	Rhode Island
5	
6	Massachusetts
7	
8	New Hampshire
9	
10	Otherwise

The otherwise row runs when the conditions next to the otherwise statement are true, but none of the cases in the same group as the otherwise cell are true.

## Results

In a condition column, you can automatically create and complete rows for all the values from an enumerated domain.

**Parent topic:** [Adding and populating rows](#)

**Related information:**  
[Working with decision tables](#)

# Populating rows with values from an enumerated domain


You can add new rows and populate them with values from an enumerated domain.

## About this task

You can add new rows and populate them with the values provided through a BRL Value Provider or from an enumerated domain, if you have set one up. An enumerated domain is a list of predefined elements (classes assigned to a particular group), defined in the BOM.

## Procedure

To add new rows and populate them with values from an enumerated domain:

1. In the decision table editor, right-click a cell in a condition column whose values are enumerated domain items.
2. Click **Add** >  **All Domain Values**.

A row is created for each value of the enumerated domain.

	length of rental (days)	pickup branch state
1	10	Massachusetts
2		New Hampshire
3		New York
4		Rhode Island

## Results

**Note:** The decision table editor can detect missing domain values, and allows you to add them automatically using the Quick Fix function. Look for the light bulb icon  in the editor marker bar. See [Correcting errors by using Quick Fix](#).

You can use a drag-and-drop function to add a number of rows at a time and have them automatically populated with values. Rule Designer automatically calculates the content of the cells for days of the week, months, intervals, and numbers. For any values next to the selected cell, it copies the value of the cells in the first row into each of the new rows.

**Parent topic:** [Adding and populating rows](#)

**Related information:**  
[Working with decision tables](#)

# Adding rows that contain values

You can add rows that already contain values to a decision table to create or extend a series of rules.

## Procedure

To add rows containing values:

- 1. In the decision table editor, select the first value in the series you want to create or extend.
- 2. Press Ctrl and place your cursor over the cell handle.

The cursor changes into a hand with a + symbol.

age	
number	number
15	30

- 3. Drag the hand down to select the number of rows that you want to insert.

A tooltip displays the values that Rule Designer places in the cells.

pickup branch state	age	
	number	number
New York	15	30
Rhode Island		
Massachusetts		
New Hampshire		

+ 47 - 62

- 4. Release the cursor.

The rows are inserted and populated with the automatically computed values, and any values copied from the original row.

length of rental (days)	pickup branch state	age	
		number	number
10	New York	15	30
		31	46
		47	62
	Rhode Island		

**Note:** If you do not want the values of the cells next to the inserted values to be automatically filled in, do not press Ctrl when clicking the cell handle. If you just drag the cell handle, values are inserted only in the selected column.

**Parent topic:** [Adding and populating rows](#)



**Related information:**  
[Working with decision tables](#)

## Moving rows

You can move rows up or down in a decision table, using the decision table editor.

### Procedure

To move a row in a Decision Table:

1. In the decision table editor, right-click a cell in the row you want to move.
2. Click  **Move Up** or  **Move Down**.

### Results

The row swaps places with the row above or below it.

**Parent topic:** [Adding and populating rows](#)

### Related information:

[Working with decision tables](#)



# Splitting rows

You can split a row if one of its columns has two subcolumns.

## Procedure

To split a row:

1. Right-click a cell in the row you want to split. The chosen cell must be in a condition column that has subcolumns.

State	Age of the customer	
	Min	Max
Rhode Island	18	21
	21	25

2. Click  **Split**.

When a row made up of grouped cells is split, the values of the column are split across two rows. The other values in the row are duplicated in the new row:

State	Age of the customer	
	Min	Max
Rhode Island	18	21
	21	25

3. To specify the empty values, double-click an empty cell and enter or select the required value, as appropriate.

**Parent topic:** [Adding and populating rows](#)

**Related information:**  
[Working with decision tables](#)

# Merging rows

You can merge rows in a decision table.

## Procedure

To merge rows:

- 1. Right-click a cell in the row you want to merge. The chosen cell must be in a condition column.

length of rental (days)	pickup branch state
10	New York
	Rhode Island
	New Hampshire

- 2. Click  **Merge**.

When you merge the rows, you place the values of the merged condition cells together. However, this action does not merge the values of the action cells in the rows.

length of rental (days)	pickup branch state
10	in "New York","Rhode Island"
	New Hampshire

**Parent topic:** [Adding and populating rows](#)


**Related information:**  
[Working with decision tables](#)

## Deleting rows

You can delete decision table rows.

### Procedure

To delete rows:

1. In the decision table editor, right-click the row you want to delete.
2. Click  **Remove selected row(s)**.

**Parent topic:** [Adding and populating rows](#)

### Related information:

[Working with decision tables](#)

## Specifying values and operators

You can specify values and operators in condition statements.

### Specifying values

You can enter values in the cell itself, or by using the edit bar.

### Setting default operators for columns

When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

**Parent topic:** [Working with decision tables](#)

# Specifying values

You can enter values in the cell itself, or by using the edit bar.

## About this task

You must complete at least one condition cell for a rule before you can complete the associated action cell.

**Note:** When you edit a cell directly in the table, you enter values, not complex expressions. You can enter complex expressions using the edit bar located above the Decision Table.

## Procedure

To specify a value in a cell:

1. In the decision table editor, double-click the required cell and specify a value using one of the following options.

To specify a value using a value editor:	To specify a value without using a value editor:
<p>If the cell type allows for the use of a dedicated value editor, a button is displayed next to the cell.</p> <ol style="list-style-type: none"><li>a. Click the button to open the value editor.</li><li>b. Enter the required value, and then press Enter.</li></ol> <p>Move the cursor away from the cell you are editing to validate, or press Esc to cancel</p>	<ol style="list-style-type: none"><li>a. In the decision table editor, double-click a cell.</li></ol> <p>A cursor or a list of allowed values displays, depending on the type of the value you specify.</p> <ol style="list-style-type: none"><li>b. Type or select the required value, and press Enter.</li></ol> <p>Move the cursor away from the cell you are editing to validate.</p>

2. Save your changes.

**Parent topic:** [Specifying values and operators](#)

**Related tasks:**  
[Setting default operators for columns](#)

**Related information:**  
[Decision tables](#)  
[Business Action Language \(BAL\)](#)  
[Adding and populating rows](#)  
[Working with decision tables](#)

## Setting default operators for columns

When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

## About this task

When you define a condition column statement, you set the default operator for all the cells in that column.

For example, if you define the following condition column statement, you set the default operator to is more than (>):

the age of the customer is more than <enter a string>



The operators you can choose from match the value type for the cell. You can change an operator later by editing the cell. You can edit a cell using the decision table editor, or in the edit bar.

You can set the cell value to an expression. You can specify expressions only by using the edit bar.

## Procedure

To define a condition column statement:

1. Set the default operator for all the cells in that column in one of the following ways:

To change an operator using the decision table editor:	To change an operator using the edit bar:	To set the cell value to an expression:
<p>a. In the decision table editor, right-click the cell.</p> <p>b. Click <b>Operator</b>, and then click an operator.</p>	<p>a. In the decision table editor, click the cell.</p> <p>b. In the edit bar, modify the operator part of the statement.</p> <p>c. Click <b>Enter</b>  next to the edit bar.</p> <p>The operator displays as a symbol next to the cell value.</p>	<p>a. In the decision table editor, click the cell.</p> <p>b. In the edit bar, modify the value part of the statement.</p> <p>c. Click <b>Enter</b> .</p> <p>The expression is shown in italics in the cell, as in the example.</p>

2. Save your changes.

## Results

The following example of a decision table shows an expression in italics in the cell:

<div> <div>✕</div> <div>✓</div> </div>		the state of the pickup branch of 'the current rental agreement' is "Rhode Island" + ", my favorite state!"			
	State	Age of the customer		Rental Rejected	Reason
		Minimum Age	Maximum Age		
1	New York	18	21	true	In New Yo.
2		21	25	false	
3					
4	"Rhode Island" + ", my favorite state!"	16	21	true	In Rhode .
5		21	25	true	In Rhode .

**Parent topic:** [Specifying values and operators](#)

**Related tasks:**  
[Specifying values](#)

**Related information:**  
[Business Action Language \(BAL\)](#)  
[Working with decision tables](#)

# Disabling action cells

You can disable an action cell by clicking an icon on the menu bar or by using the Enable / Disable Action command.


## About this task

When you define an action column, you specify the default action phrase for all the cells in that column. You cannot change the action phrase for a specific cell.

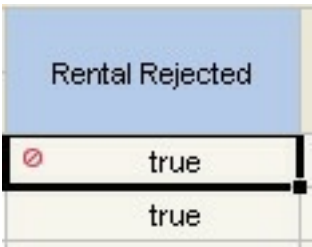
However, you can disable action cells so that the action is not carried out for that particular row. This is equivalent to clearing the contents of the cell, except that you keep the information it contains so that it can be re-enabled later if required.

## Procedure

To disable an action cell:

- 1. In the decision table editor, right-click the action cell you want to disable.
- 2. Click  **Enable / Disable Action**.

The cell shows the disabled icon.



## Results

This is a toggle function, which means that you do the same action to re-enable the action cell. When you re-enable the cell, the cell no longer shows the disabled icon.

**Parent topic:** [Working with decision tables](#)

## Related tasks:

[Populating a decision table from an Excel spreadsheet](#)  
[Defining columns](#)

## Related information:

[Adding and populating rows](#)



## Populating a decision table from an Excel spreadsheet

You can copy the contents of an Excel spreadsheet into a decision table provided the structure of the Excel spreadsheet and the decision table are the same.

### Procedure

To populate a decision table with the contents of an Excel spreadsheet:

1. Define the columns of your decision table. See [Defining columns](#).
2. Make sure that the columns of the Excel spreadsheet correspond to the columns of your decision table.
3. Copy the cells in the Excel spreadsheet.
4. Right-click the first cell of the decision table, that is, the cell on row 0 in the first column, and click **Paste**. If you are inserting a new line and not overwriting an existing line of information, right-click and use **Paste Special**.

**Note:** This feature is not available in the Decision Center decision tables.

### Results

Rule Designer automatically merges cells that have the same value. For example, if the content of your Excel spreadsheet is as follows:

	A	B	C	D
1	CDW	Economy,Compact,MidSize	15	Add surcharge of 15 for CDCW
2	CDW	FullSize,Luxury,SportUtility,Minivan	25	Add surcharge of 25 for CDW [FullSize,Luxury,SportUtility,
3	LDW	Economy,Compact,MidSize	21	Add surcharge of 21 for LDW
4	LDW	FullSize,Luxury,SportUtility,Minivan	21	Add surcharge of 21 for LDW [FullSize,SportUtility,Luxury,
5	PAI		3	Add surcharge of 3 for PAI
6	PEI		2	Add surcharge of 2 for PEI
7	ALI	Economy,Compact,MidSize	11	Add surcharge of 11 for ALI
8	ALI	FullSize,Luxury,SportUtility,Minivan	12	Add surcharge of 12 for ALI

The resulting decision table looks like this:

	Coverages	Car group	Surcharge	Comment
1	CDW	Economy,Compact,MidSize	\$15	Add surcharge of 15 for CDW...
2		FullSize,Luxury,SportUtility,Mi...	\$25	Add surcharge of 25 for CDW...
3	LDW	Economy,Compact,MidSize	\$21	Add surcharge of 21 for LDW ...
4		FullSize,Luxury,SportUtility,Mi...	\$21	Add surcharge of 21 for LDW ...
5	PAI		\$3	Add surcharge of 3 for PAI
6	PEI		\$2	Add surcharge of 2 for PEI
7	ALI	Economy,Compact,MidSize	\$11	Add surcharge of 11for ALI [E...
8		FullSize,Luxury,SportUtility,Mi...	\$12	Add surcharge of 12 for ALI [...

If you specify a range of values, the range is represented by two subcolumns in the decision table. For example, if you specified the following range in your Excel spreadsheet:

	A	B	C	D
1	CDW	Economy,Compact,MidSize	[15..21]	Add surcharge of 15 for CDCW

The Surcharge column in the decision table would look like this:

Surcharge	
[15	21]
# 25	

**Parent topic:** [Working with decision tables](#)

#### Related tasks:

[Creating a decision table from scratch](#)

[\(Deprecated\) Creating a decision table from a template](#)

[Defining columns](#)

#### Related information:

[Decision tables](#)

[Business Action Language \(BAL\)](#)





## Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.

### About this task

You can define preconditions for a decision table. Preconditions can contain the following definitions:

- Variables that are available throughout the decision table
- Conditions that must be satisfied before the decision table is executed

### Procedure

To define a precondition:

1. In the decision table editor, click the **General** tab.
2. In the **Preconditions** section, press Ctrl+Spacebar.

The Content Assist box opens. You define variables and conditions in the Preconditions section in the same way that you would create variables and conditions for an action rule in the Intellirule editor. See [Building rules using the Intellirule editor](#) for details.

3. Save the decision table.

The variables you defined are now available for creating condition definitions in the decision table, which means that the rows of the decision table cannot be evaluated before the conditions defined in the precondition are satisfied.

**Parent topic:** [Working with decision tables](#)

# Formatting a decision table

You can define format settings for a table, a column, or individual cells.

## About this task

- Column format settings override decision table format settings.
- Cell format settings override decision table and column format settings.

## Procedure

To format a decision table, a column, or an individual cell:

1. Follow one of the following procedures, depending on what you want to do:

To format a decision table:	To format a column:	To format a cell:
<div><div><div>a. Right-click the header part of the decision table, and then click <b>Decision Table Properties</b>.</div><div>b. In the Decision Table Properties dialog, click the <b>Format</b> tab.</div><div>c. Apply the required formatting and font settings.</div></div><div><div>You can set the foreground color and font of the column headers and row headers. You can also set the foreground color, background color, and font for all cells throughout the table.</div></div></div>	<div><div><div>a. Right-click the header part of the decision table, and then click <b>Format</b>.</div><div>b. In the Format Column dialog, apply the required alignment, colors, font, and format.</div></div><div><div>To format the contents of the cells in the column, do one of the following tasks:</div><div><div>■ From the <b>Format Statement</b> list, select one of the preconfigured formats. The formats available depend on the column data type: string, number, date, or time.</div><div><div>Use the <b>Preview</b> field to verify the selected format.</div><div>For example, in a column of cells that contain numbers, select the format {0, number, currency} to display the number in currency format using the currency of the currently active locale.</div></div><div><div>■ Enter the required format in the <b>Format Statement</b> field. Use a zero enclosed in curly braces to represent the cell contents. For example, to display numbers in currency format</div><div>using a specific currency symbol such as £, enter</div></div></div></div></div>	<div><div><div>a. Right-click the cell you want to format, and then click <b>Format cell</b>.</div><div>b. In the Format Cell dialog, apply the cell tooltip, alignment, colors, font, and format. You can override the formatting options at subcell level.</div></div><div><div>To format the cell contents, do one of the following tasks:</div><div><div>■ From the <b>Format Statement</b> list, select one of the preconfigured formats. The formats available depend on the cell data type: string, number, date, or time.</div><div><div>Use the <b>Preview</b> field to verify the selected format.</div><div>For example, in a cell displaying a number, select the format {0, number, currency} to display the number in currency format using the currency of the currently active locale.</div></div><div><div>■ Enter the required format in the <b>Format Statement</b> field. Use a zero enclosed in curly braces to represent the cell contents. For example, to display the number in currency format</div><div>using a specific currency symbol such as £, enter</div></div></div></div></div>

	<p>the following format: £{0}.</p> <p>Rule Designer applies the settings you specify to all the cells in the column and its subcolumns.</p>	<p>the following format: £{0}.</p> <p>Rule Designer applies the settings you specify to the cell.</p>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------

2. Click **OK** to save your settings.

**Parent topic:** [Working with decision tables](#)

**Related information:**  
[Business Action Language \(BAL\)](#)

# Defining checking options

You can specify tests that Rule Designer conducts to identify problems with a decision table.

## About this task

You can define global checking options for condition columns, or you can set up your own value checks to verify cell values in individual condition or action columns.


**Attention:**

Checking is an intense CPU activity and might increase processing time for large decision tables and, as a result, for the build process.

## Procedure

To define checking options for condition columns or cell values:

1. Use the decision table editor to define the type of checking you require, as outlined in the following table:

To define global checking options	To define your own cell value checks:
<div><div>a. In the decision table editor, right-click a column or a cell and then click  <b>Decision Table Properties</b>.</div><div>b. Click the General tab of the Decision Table Properties dialog.</div><div>c. In the “Table checks” area, select the options you want to check.</div><div>You can choose which checks you want to be done against which column, and the severity of the checking reports.</div></div>	<div><div>a. In the decision table editor, double-click the required column.</div><div>b. In the Condition Column or Action Column dialog, in the <b>Properties</b> section, select a placeholder in the Expression Parameters list, and then select the <b>Check Value</b> check box.</div><div>c. Select the required test condition from the drop-down list next to the check box.</div><div>The contents of the drop-down list differs, depending on the type of value check. See <a href="#">Value test conditions</a> for details.</div><div>d. When you make your selection, an input field opens (or more than one, depending on your selection) for you to enter the value(s) to check against.</div><div>The check value input field provides you with a content assist facility. To activate it, click Ctrl+Spacebar.</div><div>Note that you can enter only values, not complex expressions.</div><div>The check applies to all the cells in the column. Warnings are reported if any values do not match the test.</div></div>

2. Click **OK** to save your settings.

**Parent topic:** [Working with decision tables](#)

## Value test conditions

When you construct conditions that test values, the way you complete the associated input fields depends on the type of test.

A numeric value check displays a list of logical operators (greater than, less than and so on), to which you add the required numeric value in the associated input field. A true or false check displays a choice of equal to or not equal to.

You might also see these options. You should complete the associated input fields as follows:

- `is in`: enter a comma-delimited set of values, enclosed in curly brackets. For example: {Massachussets, New York, Rhode Island}
- `is not in`: enter a comma-delimited set of values, enclosed in curly brackets.
- `is between`: enter a range of values in square brackets, separated by three periods. For example: [19...25]
- `match`: enter the value, character or string to be matched.

Values in a set, range or match might be numeric or non-numeric.

**Parent topic:** [Working with decision tables](#)

### Related information:

[Decision tables](#)

[Business Action Language \(BAL\)](#)

# Using decision table locking facilities

You can apply a variety of locks to protect your decision tables against editing by others.

## About this task

Decision tables have locking facilities that allow you to protect your decision tables against editing by others. The locking rules should be specified in the context of a template library to be applied on instances of this decision table. For both condition columns and action columns, you can lock the structure, data, or both.

## Procedure

To define decision table locking requirements:

1. Right-click any cell in your decision table, and then click **Decision Table Properties**. The Decision Table Properties Editor opens.
2. Click the Lock tab.
3. Choose which locks you want to apply:

### Apply locking rules to this table

The default is off to allow you to edit your decision table and lock rules. Set to on for decision tables used in a package on which you want to enforce locking rules.

### Show lock icons on columns

If on, the Lock icon is displayed on any locked condition and action items.

### Prevent addition and removal of columns

Prevents condition or action columns from being added or deleted.

4. In the **Condition columns** section, select any of the following locks:

#### Lock test

The test expression associated with the condition column cannot be changed. Note that cell content, including cell-specific test operators, can still be changed.

#### Lock partitions

Partition items cannot be added to or removed from the selected condition column. The values of existing partition items can still be changed, but no rows can be added to or removed from this column.

Locking partitions locks partitions in all preceding columns because adding a new partition item to a previous column can create a new row in the locked column.

#### Lock values

The values of existing partition items or the cell operator cannot be changed.

5. In the **Action columns** section, select any of the following locks:

#### Lock action

The expression associated with the column cannot be changed.

#### Lock status

The status of the action cell cannot be changed. The action status specifies whether an action is enabled or disabled. See [Disabling action cells](#) for details on disabling action columns.

#### Lock values

The values in the action cell of the selected column cannot be changed.

6. Click **OK** to apply your selections.

**Parent topic:** [Working with decision tables](#)

# Viewing rule details

You can view the details of a rule in a decision table row by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision table editor.


## About this task

In a decision table, a row corresponds to one rule.

## Procedure

To view the rule details:

Use the decision table editor to view the rule details in a tooltip or in a viewing pane, as follows:

To view the rule details in a tooltip:	To view rules in a dedicated viewing pane:
<div>a. In the decision table editor, place the cursor over the row header number on the side.</div>	<div><div>a. In the decision table editor, right-click a column or a cell, and then click  <b>Decision Table Properties</b>.</div><div>b. In the Decision Table Properties dialog, go to the <b>Table view</b> section of the General page.</div><div>c. Select <b>Show rules</b>.</div><div>d. Click <b>OK</b>.</div><div>A new viewing pane opens at the bottom of the decision table editor.</div><div>e. In the decision table editor, click the row header number on the left.</div></div>

## Results

The rule is shown in a tooltip or the viewing pane, depending on which procedure you followed. If you defined preconditions for the decision table, these are also shown.

**Parent topic:** [Working with decision tables](#)



# Viewing the decision table as a spreadsheet

You can display a decision table in a spreadsheet view if the decision table has two condition columns and one action column and the structure of the decision table is symmetric.

## About this task

You can display a decision table in a spreadsheet view, if the decision table meets the following conditions:

- The decision table has two condition columns and one action column.
- The structure of the decision table is symmetric. A decision table is symmetric if, for each row in the first condition column, the same rows are defined for the second condition column.

	length of rental (days)	pickup branch state	message
0	≤ 10	New York	City fare
1		Rhode Island	
2		New Hampshire	Rent this car one more day and get a discount!
3	≥ 11	New York	
4		Rhode Island	Special discount
5		New Hampshire	You get a discount!

The spreadsheet view of a decision table presents the first condition column as the row header, and the second condition column as the column header. The action column values are represented as cells in the spreadsheet.



**Note:** This feature is not available in the Decision Center decision tables.


## Procedure

To view a decision table as a spreadsheet:

1. In the decision table editor toolbar, click  **Switch to Spreadsheet View.**

The decision table is displayed as a spreadsheet.

 			
	"New York"	"Rhode Island"	"New Hampshire"
10	City fare		Rent this car one more day and get a ...
11		Special discount	You get a discount!

2. To switch back to the decision table view, in the spreadsheet view toolbar, click  **Switch to Decision Table View.**

**Parent topic:** [Working with decision tables](#)

## Documenting a decision table

You can document each row in a decision table.


### About this task

You can create a column to document the rows in a decision table.

<b>Note:</b> This feature is not available in the Decision Center decision tables.
------------------------------------------------------------------------------------

### Procedure

To document a decision table:

1. In the decision table editor, right-click a column or a cell, and then click **Decision Table Properties** .
2. In the Decision Table Properties dialog, go to the **Table view** section of General page.
3. Select **Show comments**.
4. Click **OK**.

A Comments column is added to the side of the table. You can use this column to document the rows.

**Parent topic:** [Working with decision tables](#)

## Working with variables

You can use automatic variables and ruleset variables to define variables for use in business rules.

### Variables available for rule authoring

A variable is any entity that can have different values.

### Setting up automatic variables

You set up automatic variables to define a variable that can be accessed anywhere in business rules.

### Defining ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project.

**Parent topic:** [Authoring business rules](#)

# Variables available for rule authoring

A variable is any entity that can have different values.

A variable comprises three elements:

- Name
- Type
- Value

You define variables so that you can use the name independently of the value it represents. Variables can make your business rules less ambiguous and easier to understand.

A variable name must not contain any of the following characters:

Character type	Example
Tabulation	TAB
Line break	\n
Single quotation mark	'
Double quotation mark	"
Opening and closing parentheses	()
Slash	/
Comma	,
Semicolon	;

A variable can have one of the following value types:

Variable value type	Example
Constant	<ul style="list-style-type: none"><li>• A number: 12</li><li>• A literal character string: "Gold customer"</li><li>• A Boolean, an object domain element: STATUS_APPROVED</li></ul>
Expression	<ul style="list-style-type: none"><li>• An arithmetic expression whose result is a number: the price of the product * 1.20</li><li>• A text concatenation or formatting expression: "The loan request for " + the name of the applicant + " is rejected"</li><li>• A Boolean expression: the customer is married</li><li>• A navigation expression to an object: the last transaction of the checking account of the customer</li></ul>
Object	For example, a reference to a predefined business term: the customer
Collection of values of the types already listed	For example, a list of numbers { 1, 3, 5 }, a list of objects { CHINA, FRANCE, UK, USA }

**Important:** The type of value that you use must match or be compatible with the variable type.

The following tables list the different types of variables that you can use in your business rules.

An automatic or implicit variable has a predefined name and type, and is set automatically or implicitly by the system:

Variabl e	Scope	Definition and use
Aut om atic vari able	A variable that is available in all the business rules that use the BOM class where the variable is declared.	
Impl icit vari able	A variable that is automatically declared by a language construct like a binding. For example:  set 'var-name' to <a type> where ..., there is a ..., there are ..., for each....	Built into the Business Action Language. The name, type and value are controlled by the system.  You use implicit variables to refer to a current instance of an object that is being tested or declared, or to a collection that is being

		<p>iterated.</p> <p>Implicit variables are visible only in the code completion menu of the Intellirule editor.</p>
--	--	--------------------------------------------------------------------------------------------------------------------

You can define your own rule variables, ruleset variables, and ruleset parameters:

V a r i a b l e	Scope	Definition and use
R u l e v a r i a b l e	A variable that can be referenced only in the action rule in which it is defined.	<p>You define a rule variable in the definitions part of the rule.</p> <p>Rule variables are local to the rule in which they are defined. They are not available in other rules. The variable name must be unique within the rule.</p>
R u l e s e t v a r i a b l e	<p>A variable that can be referenced by all the business rules in a rule project.</p> <p>Ruleset variables are not available outside of the rule project scope.</p>	<p>You define a ruleset variable in a variable set in the rule project. Before you define a ruleset variable therefore, you must create a variable set in which to group your ruleset variables.</p> <p>You define a ruleset variable by its name, type, and verbalization. Ruleset variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.</p> <p>Make sure that you set default values to ruleset variables before you use them in rules.</p> <p>You can also use ruleset variables to pass data between tasks in a ruleflow.</p>
R u l e s e t p a r a m e t e r	<p>A ruleset parameter is a variable that is defined as the interface between the calling application of a ruleset and the ruleset itself.</p> <p>It has a direction: in, out, or in-out.</p> <p>The direction indicates whether the caller passes data in, expects data out of the ruleset, or passes data in and expects some modifications to the data passed in.</p>	<p>You define a ruleset parameter in the signature of a decision operation in a decision service.</p> <p>You use ruleset parameters to exchange data between a ruleset and an application, and as the main data elements on which the rule operates.</p> <p>Ruleset variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.</p>

Parent topic: [Working with variables](#)

## Setting up automatic variables

You set up automatic variables to define a variable that can be accessed anywhere in business rules.

### About this task

To define a variable that can be accessed anywhere in business rules, you can set up automatic variables. Automatic variables are bound to the instance of the class on which it is defined. Automatic variables are variables that you declare as an instance of a specific BOM class. They are available in all the business rules that use the BOM class where the variable was declared.

When you specify an automatic variable, it is defined for the corresponding business term, and identified with the article “the”. For example, if customer is a business term, a corresponding variable called the customer is available in the rule editors.

<b>Note:</b> This is valid only for the locales that use this type of article.
--------------------------------------------------------------------------------

You cannot define one variable to be the same as another variable of the same type. If your rule requires you to refer to more than one occurrence of customer, you have to define the other variables explicitly, in the definitions part of the rule. In this case, the automatic variable is no longer available in this rule.

The automatic variable is known internally as customer. The variable the customer is the automatic variable in a given verbalization context (with definite article the) and is the form that you can use in rules to reference this variable. Consequently, you cannot declare another variable named customer in the definition part of a rule:

```
definitions set 'customer' to a customer;
```

This statement generates the following error:

"An automatic variable 'customer' is already declared"

For example, in the following rule, ceiling is an explicit variable declared in the definitions part of the rule, and the customer is an automatic variable.

```
definitions set ceiling to 10,000;
if the value of the shopping cart of the customer is more than ceiling
then...
```

You cannot declare a ruleset parameter or variable with the same name as an automatic variable. If you do, Rule Designer redirects references to automatic variables such as the customer to the ruleset parameter. For example, if the rule project contains a ruleset parameter called customer, in the following condition, the customer references the ruleset parameter instead of the automatic variable:

```
if
 the salary of the customer is more than 100 ...
```

### Procedure

To generate an automatic variable for a business class:

1. In the Outline view, click the class for which you want to generate an automatic variable.
2. In the Class Verbalization section of the BOM Editor, select the **Generate automatic variable** box.
3. Save the BOM.

An instance of the class is now available as an automatic variable from business rules.

**Parent topic:** [Working with variables](#)

### Related concepts:

[Rule variables](#)  
[Vocabulary](#)

### Related information:

[Overview: Ways to express business rules](#)  
[Working with action rules](#)  
[Business Action Language \(BAL\)](#)

## Defining ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project.

### Creating a variable set

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables.

### Defining a ruleset variable

After you have created a variable set, you can define a ruleset variable.

### Modifying the verbalization of a ruleset variable

After defining a ruleset variable, you can modify its verbalization.

**Parent topic:** [Working with variables](#)

**Related concepts:**

[Action rules](#)

**Related information:**

[Overview: Ways to express business rules](#)

[Decision operations](#)

## Creating a variable set

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables.

### About this task

Ruleset variables are variables that you can use in all the business rules of the ruleset. You can also use ruleset variables to pass data between tasks in a ruleflow. Ruleset variables are accessible from business rules only if you verbalize them.

### Procedure

To create a variable set:

1. Select your rule project.
2. On the **File** menu, click **New** > **Variable Set**.

The New Variable Set wizard opens.

3. If you want to select a rule package, click **Browse** next to the **Package** field.
4. In the **Name** field, type the name of your variable set.
5. Click **Finish**.

The new variable set is shown in the Rule Explorer view and the Variable Set Editor opens.

### Results

You can now define your ruleset variables.

**Parent topic:** [Defining ruleset variables](#)



# Defining a ruleset variable

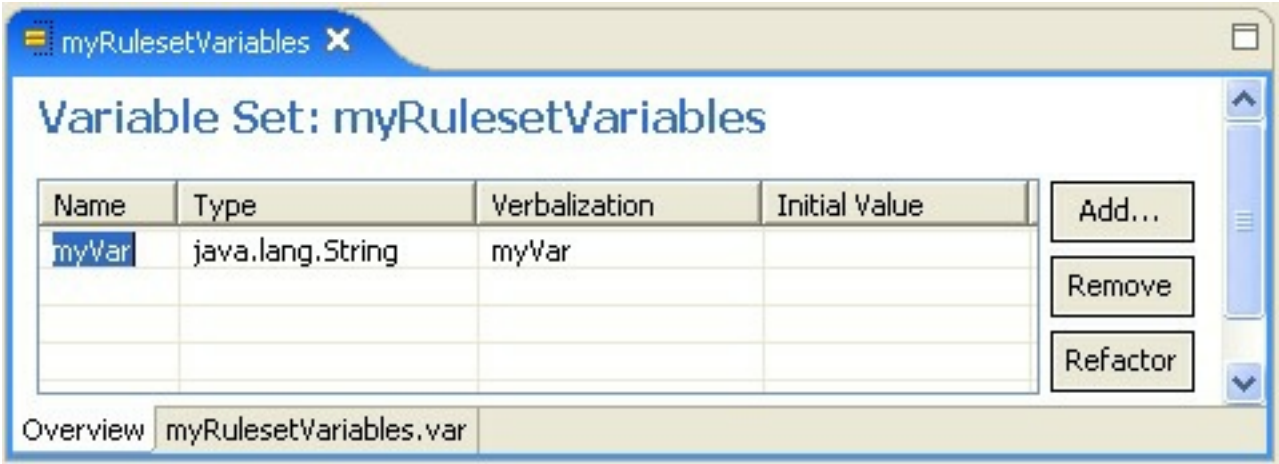
After you have created a variable set, you can define a ruleset variable.

## Procedure

To define a ruleset variable:

1. In the Variable Set editor, click **Add**.

Default values for a new ruleset variable are displayed in a table row.



2. Specify the name, type and (optionally) the initial value of the ruleset variable.
3. If you do not want the ruleset variable to be visible from business rules, delete the default verbalization from the Verbalization column. Otherwise, modify the verbalization.

## Results

The ruleset variable is now defined and you can start using it in business rules.

**Parent topic:** [Defining ruleset variables](#)

# Modifying the verbalization of a ruleset variable

After defining a ruleset variable, you can modify its verbalization.

## Procedure

To modify the verbalization of a ruleset variable:

1. In the Variable Set editor, select the variable in the table and click **Refactor**.

The Refactor Variable window opens.

2. In the **Variable verbalization** field, type a new verbalization for the variable.

**Restriction:**

Do not use the following characters in the verbalization of ruleset variables:

Variable	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
( )	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon

3. If you want to see the impact of the change on the business rules, click **Preview**.

The Refactor dialog shows which business rules (if any) need to be refactored.

4. Click **OK**.

The business rules are refactored.

## What to do next

**Note:** You must refactor the rows individually. If you change the verbalization in a row without refactoring and then refactor another row, the new value of the first row is not refactored.

**Parent topic:** [Defining ruleset variables](#)

## **(Deprecated) Working with decision trees**

You use the decision tree editor to create a workflow for the execution of your rules.

### **Decision trees**

Decision trees provide a concise view of a set of action rules that are not sufficiently symmetric to present in a decision table.

### **Creating a decision tree**

You create a decision tree by specifying a name and optionally specifying a package.

### **Labeling and defining condition nodes**

You label condition nodes to make condition tree diagrams easier to read. You define a condition node by constructing a condition statement in the edit bar of the decision tree editor.

### **Labeling and defining branches**

When you define or insert a new branch, you must label it. You can insert Otherwise branches, and branches on nodes that define an enumerated domain. You can also duplicate and delete branches.

### **Labeling and defining actions**

Actions are made up of two elements: the action set header (title) and the action set.

### **Defining preconditions**

You define preconditions to create variable definitions that are available throughout the decision tree and conditions that must be satisfied before the decision tree is executed.

### **Defining checking options**

You can specify the tests that you want Rule Designer to conduct to identify problems with a decision tree.

### **Viewing rules**

You can view the details of a rule in a decision tree by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision tree editor.

**Parent topic:** [Authoring business rules](#)

## Decision trees

Decision trees provide a concise view of a set of action rules that are not sufficiently symmetric to present in a decision table.

### [\(Deprecated\) Decision tree overview](#)

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

### [Decision tree editing errors and warnings](#)

Rule Designer checks for overlaps and gaps in decision tree conditions, and indicates problems by using visual cues.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

### **Related information:**

[Overview: Ways to express business rules](#)

[\(Deprecated\) Working with decision trees](#)

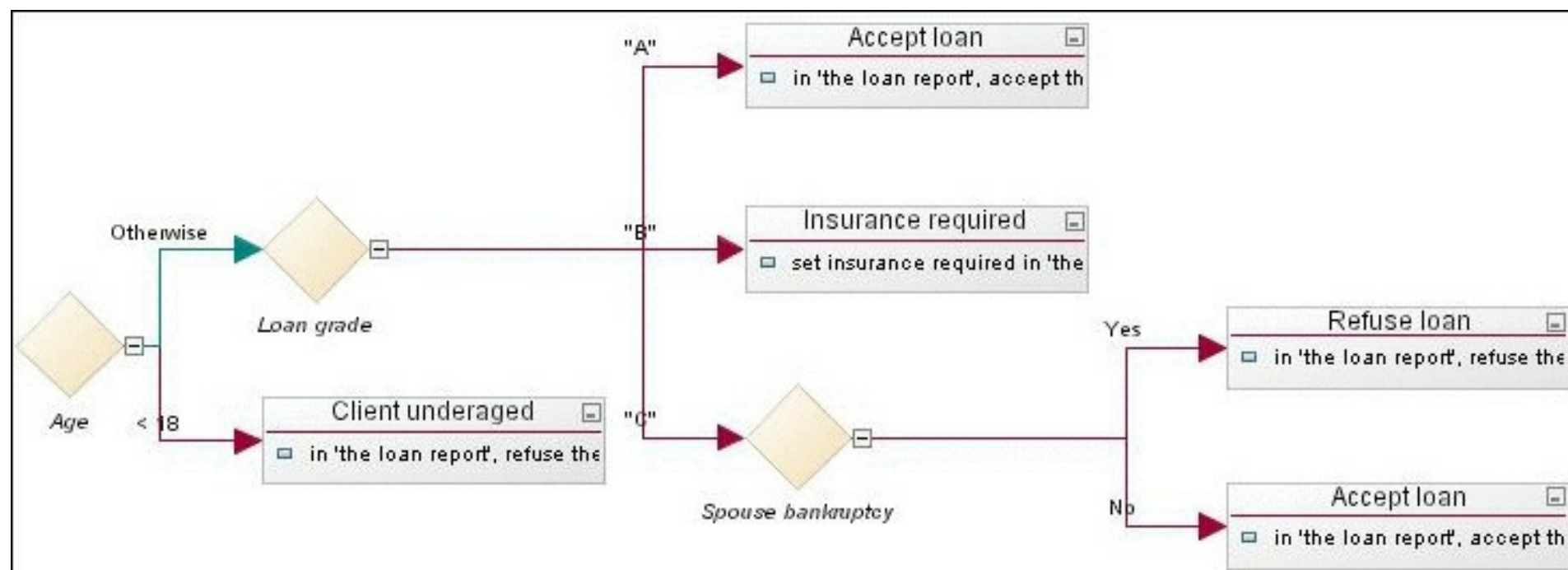
[Business Action Language \(BAL\)](#)

## (Deprecated) Decision tree overview

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

Decision trees provide a way to view and manage large sets of business rules in diagrams.

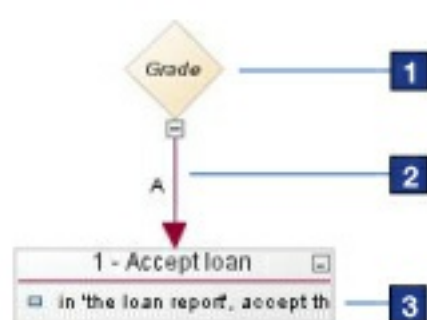
Decision trees make the interaction of nonsymmetrical rules easier to understand. The path from the first condition to the end of the actions along any branch represents one rule.



Looking at the following figure, you can see why decision trees are easy to understand:

- A condition is declared in its diamond-shaped node **1**.
- The possible values for the condition are represented by branches **2**.
- The actions are declared at the end of each branch **3**.

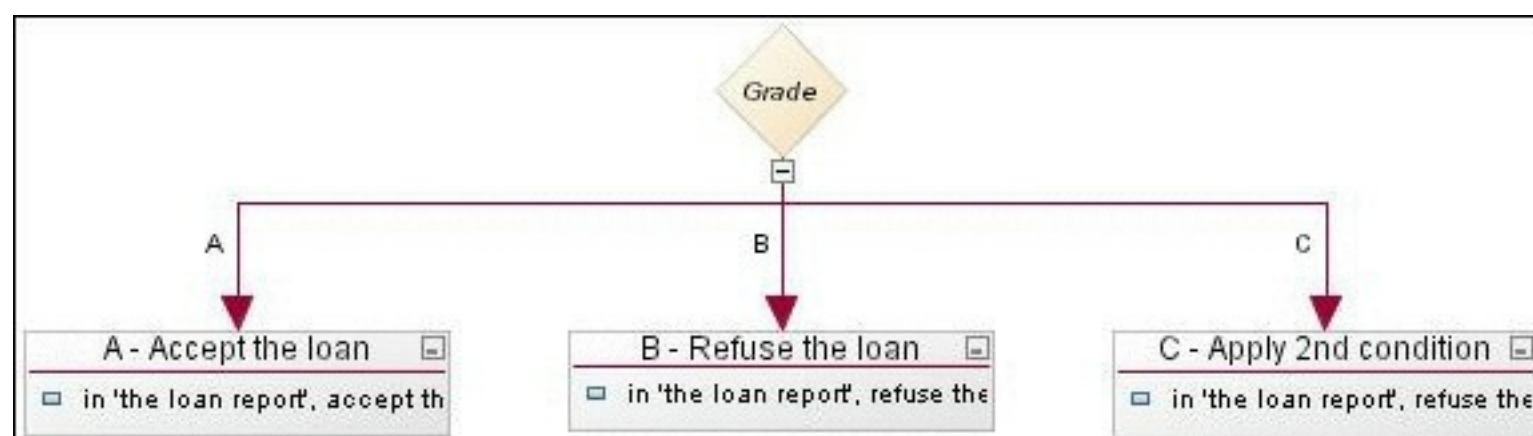
This simple decision tree corresponds to the following rule:



**if**  
the grade in the loan report is 'A'  
**then**  
in the loan report, accept the loan with the message "Loan accepted"

By adding branches, you add new rules that have different values for the condition.

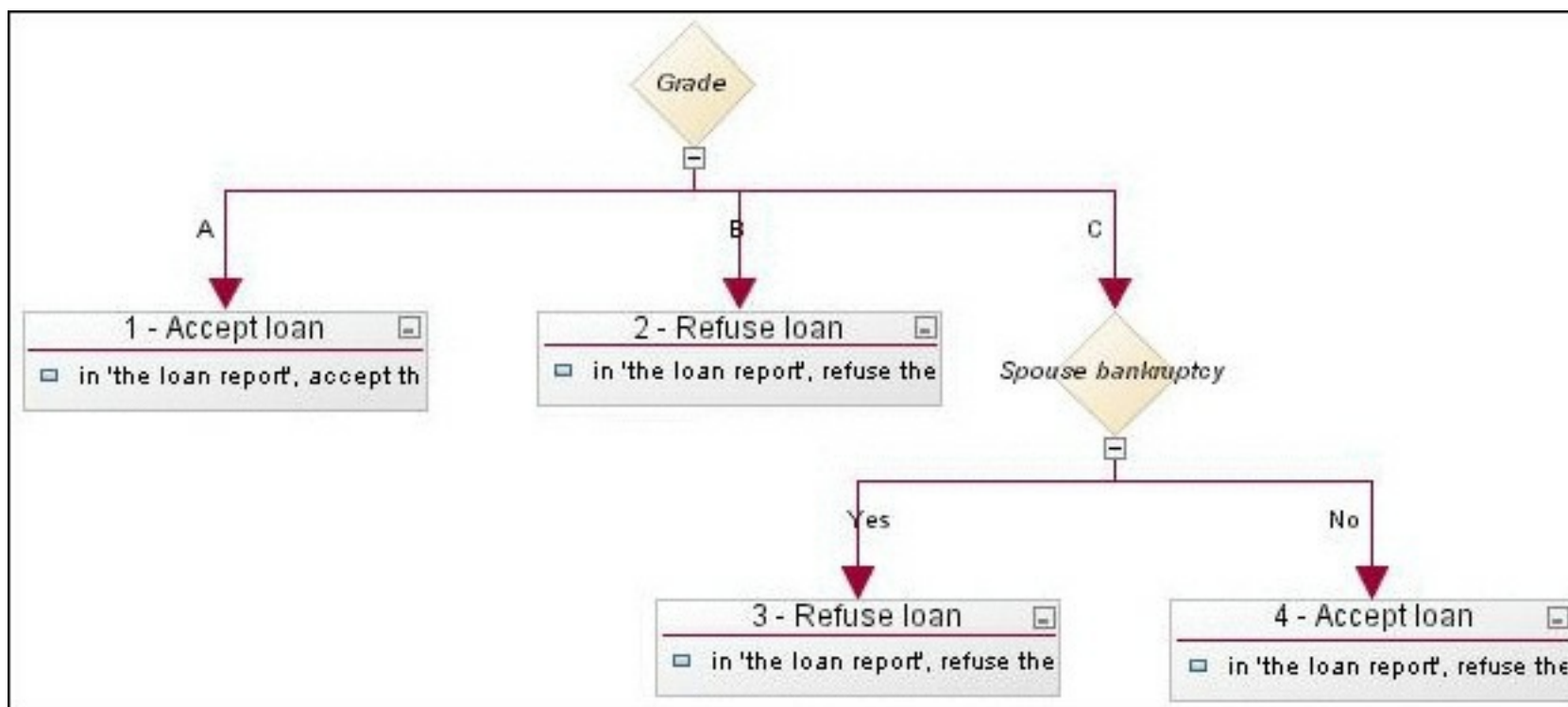
For example, the following decision tree forms three rules for a loan application (A, B, and C):



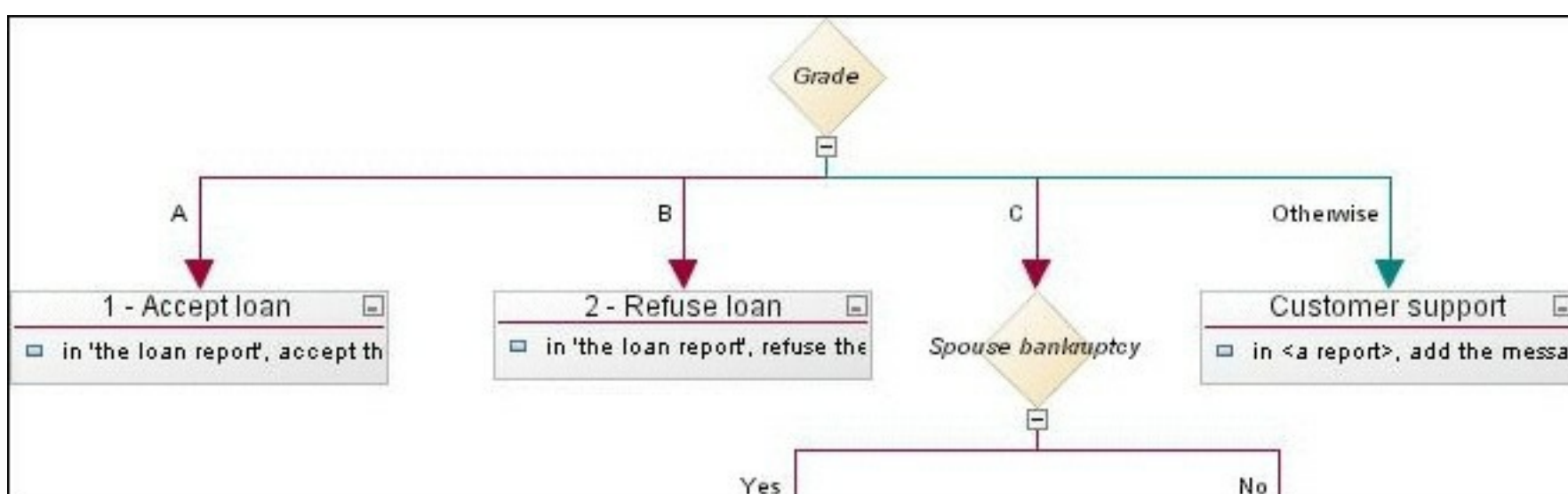
You can put as many actions as you want at the end of a branch, and add another condition to a branch.

For example, rules 1 and 2 in the following decision tree have only one condition (Grade), while rules 3 and 4

have a second condition to check (Spouse bankruptcy) before they do the actions:



Finally, you can add an Otherwise branch for condition values that are not covered by any of the other branches:



You can lay out your decision tree vertically or horizontally for optimal viewing, and set or remove consistency checking.

## Preconditions

You can set the following elements in a precondition section of a decision tree:

- Variables that can be used in the decision tree.
- Condition that is applied to an entire decision tree.

If the precondition is not satisfied, none of the rules in the decision tree can be evaluated.

For example, you can apply the following precondition to a decision tree:

### definitions

set 'wealthy customer' to a customer  
where the average monthly balance of this customer  
is more than \$1,000,000

### if

the state of residence of 'wealthy customer' is NY

Applying this precondition to a decision tree limits the scope of the decision tree rules to only those customers who have an average monthly balance of \$1,000,000 and live in New York. You can also use the variable `wealthy_customer` in the tree.

Preconditions are tested before each rule in a tree is run.

**Parent topic:** [Decision trees](#)

**Related concepts:**

[Action rules](#)

[Technical rules](#)

**Related reference:**

[BAL operators](#)


**Related information:**

[Overview: Ways to express business rules](#)



## Decision tree editing errors and warnings

Rule Designer checks for overlaps and gaps in decision tree conditions, and indicates problems by using visual cues.

If there is an error or warning in a decision tree, the warning symbol  identifies the affected nodes and branches.

When you save the decision tree, the Problems view also displays the errors and warnings.

A decision tree that has a checking error cannot be executed, unless you disable checking.

The rules that make up your decision tree can be checked for gaps and overlap. You can specify your checking preferences at node level and at tree level.

### Gaps

Rule Designer checks that the branches in the decision tree are uninterrupted. For example, if one branch defines the interval [21 - 30] and another branch defines the interval [33 - 40], an error is generated since the interval [31 - 32] is not accounted for.

### Overlap

Rule Designer checks that no branches overlap in the decision tree. For example, if one branch defines the interval [21 - 30] and another branch defines the interval [28 - 40], an error is generated since the interval [28 - 30] overlaps in that node.

#### Note:

You cannot check dates for gaps and overlap.

**Parent topic:** [Decision trees](#)

#### Related concepts:

[\(Deprecated\) Decision tree overview](#)

#### Related tasks:

[Applying verbalization changes to business rules](#)

[Applying a category filter](#)

#### Related reference:

[BAL operators](#)

#### Related information:

[Overview: Ways to express business rules](#)

[Vocabulary errors and warnings](#)

[\(Deprecated\) Working with decision trees](#)

[Business Action Language \(BAL\)](#)



# Creating a decision tree

You create a decision tree by specifying a name and optionally specifying a package.

## About this task

Decision trees are composed of branches that have a condition node as their root, and end with actions. Every node is a condition node, except for leaf nodes. Decision trees allow you to manage a large set of rules with some conditions in common but not all.

## Procedure

To create a decision tree:

1. In the Rule Explorer view, click your rule project.
2. On the **File** menu, click **New** > **Decision Tree**. The New Decision Tree wizard opens.
3. Optional: Add the decision tree to a rule package:
  - a. In the **Package** field, click **Browse**.
  - b. Expand the folder structure, click the package to which the decision tree should be added, and then click **OK**.
4. In the **Name** field, type a name for the decision tree.
5. Click **Finish**. The new decision tree is displayed in the Rule Explorer view and the Decision Tree Editor opens. The default decision tree has one condition node, one branch, and one action.

## Results

You can now use the decision tree editor to label and define the condition node and action, and to create further nodes, branches and actions as appropriate.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

### Related tasks:

[Correcting errors by using Quick Fix](#)

[Defining preconditions](#)

[Defining checking options](#)

[Viewing rules](#)

### Related information:

[Decision trees](#)

[Business Action Language \(BAL\)](#)

[Labeling and defining condition nodes](#)

[Labeling and defining branches](#)

[Labeling and defining actions](#)

## Labeling and defining condition nodes

You label condition nodes to make condition tree diagrams easier to read. You define a condition node by constructing a condition statement in the edit bar of the decision tree editor.

### Labeling a condition node

You label condition nodes to make condition tree diagrams easier to read.

### Defining a condition node

When you have created a label for the condition node, you can define the node.

### Inserting a condition node

You can insert condition nodes before or after an existing condition node.

### Inserting a new condition node after a branch

You can insert condition nodes after an existing branch.

### Deleting a condition node

When you delete a condition node, all its branches are also deleted.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

# Labeling a condition node

You label condition nodes to make condition tree diagrams easier to read.

## About this task

Before you can define a condition node, you must give it a name (label). Labels make the condition tree diagrams easier to read.

## Procedure

To label a condition node:

1. In the decision tree editor, double-click the condition node. The Node Editor dialog opens.
2. In the **Label** field, type a title for the condition node.
3. Click **OK**.

## Results

Your title replaces the default label.

**Parent topic:** [Labeling and defining condition nodes](#)

### Related tasks:

[Defining a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

## Defining a condition node

When you have created a label for the condition node, you can define the node.

### Procedure

To define a condition node:

1. In the decision tree editor, click the condition node.
2. In the edit bar, build a condition statement.

The process is the same as using the guided editor, except that you cannot use the edit bar to edit the values.

3. To save the definition, click the **Enter** button  next to the edit bar.

### Results

The condition node is defined. You can now add other condition nodes to the tree and create branches to specify different values for the condition statement defined for the node.

**Parent topic:** [Labeling and defining condition nodes](#)

### Related tasks:

[Labeling a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

## Inserting a condition node


You can insert condition nodes before or after an existing condition node.

### About this task

You can insert condition nodes before an existing condition node, or after it. This procedure describes how to insert a condition node before an existing condition node. After you insert a condition node, you must label and define it.

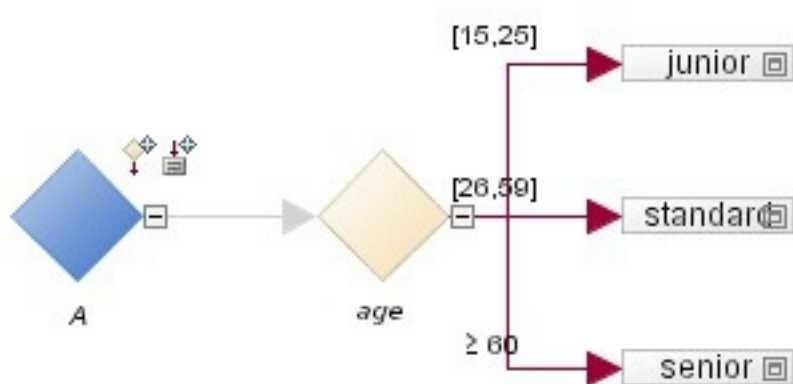
### Procedure

To insert a condition node before an existing condition node:

1. Right-click the condition node before which you want to insert a new condition node.
2. Click  **Insert Condition Node**.

### Results

The new condition node is added to the decision tree. You must now label and define it, as described in [Labeling a condition node](#) and [Defining a condition node](#).



**Parent topic:** [Labeling and defining condition nodes](#)

### Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a new condition node after a branch](#)

[Deleting a condition node](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

## Inserting a new condition node after a branch


You can insert condition nodes after an existing branch.

### About this task

After you insert a condition node, you must label and define it.

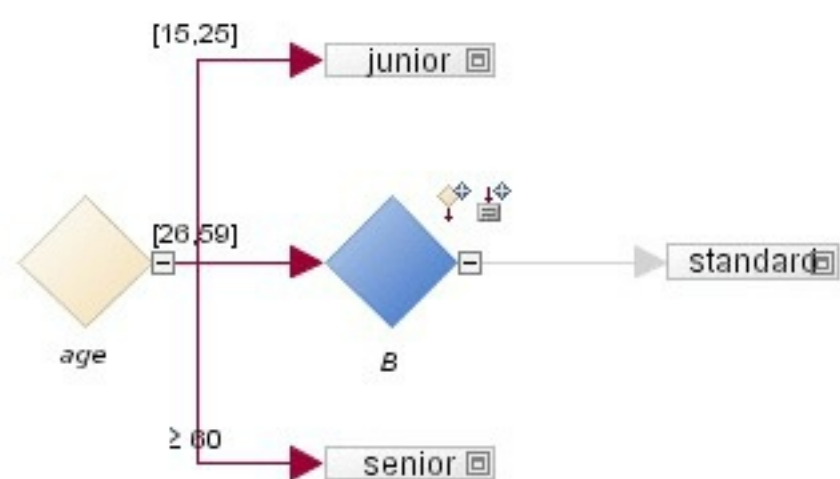
### Procedure

To insert a new condition node after a branch:

1. Right-click the branch after which you want to insert a new condition node.
2. Click  **Insert Condition Node**.

### Results

The new condition node is added to the decision tree. You must now label and define it, as described in [Labeling a condition node](#) and [Defining a condition node](#).



**Parent topic:** [Labeling and defining condition nodes](#)

### Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a condition node](#)

[Deleting a condition node](#)

### Related information:

[Decision trees](#)


[\(Deprecated\) Working with decision trees](#)

## Deleting a condition node

When you delete a condition node, all its branches are also deleted.

### Procedure

To delete a condition node:

Right-click the node in the decision tree editor and click  **Delete**.

### Results

The condition, together with its associated branches, is deleted.

**Parent topic:** [Labeling and defining condition nodes](#)

### Related tasks:

[Labeling a condition node](#)

[Defining a condition node](#)

[Inserting a condition node](#)

[Inserting a new condition node after a branch](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

# Labeling and defining branches

When you define or insert a new branch, you must label it. You can insert Otherwise branches, and branches on nodes that define an enumerated domain. You can also duplicate and delete branches.

## Defining a branch

Branches are represented by links. You must define the branch before you can label it.

## Labeling a branch

You label a branch.

## Inserting a new branch

You can attach new branches to condition nodes.

## Inserting an Otherwise branch

You use an Otherwise branch if none of the other possibilities for a condition are correct.

## Changing a branch to and from Otherwise

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch.

## Inserting and defining branches for enumerated domain values

You can insert and define branches automatically on nodes that define an enumerated domain.

## Duplicating a branch

You duplicate a branch by using drag and drop in the Decision Tree Editor.

## Deleting a branch

When you delete a branch, you also delete all its associated actions.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

### **Related tasks:**

[Correcting errors by using Quick Fix](#)

### **Related information:**

[Decision trees](#)

[Business Action Language \(BAL\)](#)

[Labeling and defining condition nodes](#)




## Defining a branch

Branches are represented by links. You must define the branch before you can label it.

### Procedure

To define a branch:

1. In the decision tree editor, click the branch.
2. In the edit bar, edit the operator and values of the condition statement in the same way as in the guided editor.
3. Click the **Enter** button  next to the edit bar to save the definition.

### Results

The newly-defined branch shows up in dark red.

**Parent topic:** [Labeling and defining branches](#)

#### Related tasks:

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

# Labeling a branch

You label a branch.

## Procedure

To label a branch:

1. In the decision tree editor, double-click the required branch. The Branch Editor dialog opens.
2. In the **Label** field, type a label for the branch.
3. Click **OK**.

## Results

Your label replaces the default label. You can add other branches, duplicate and delete branches, and insert and define branches on nodes that define an enumerated domain.

**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)


[Deleting a branch](#)

## Inserting a new branch

You can attach new branches to condition nodes.

### Procedure

To insert a new branch:

In the decision tree editor, right-click the condition node to which you want to attach a new branch, and then click **Add** >  **New Branch**.

### Results

A new branch is displayed under the node. You can now label and define it, as described above.

**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

## Inserting an Otherwise branch

You use an Otherwise branch if none of the other possibilities for a condition are correct.

### About this task

You can also change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch. Note that each node can have only one Otherwise branch.

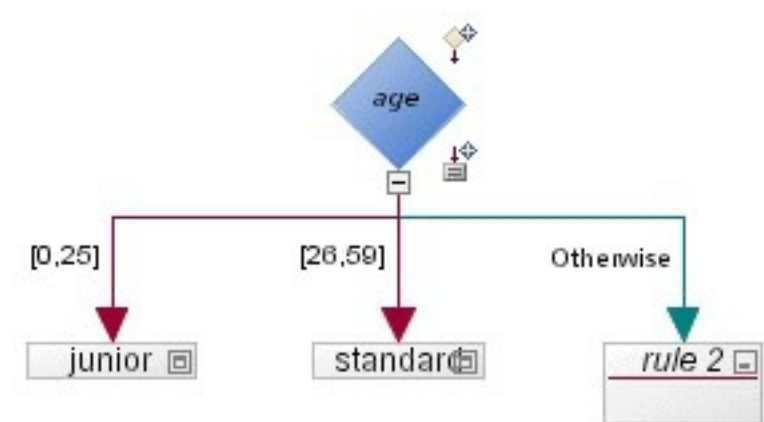
### Procedure

To insert an Otherwise branch:

In the decision tree editor, right-click the condition node to which you want to attach an Otherwise branch and then click **Add > Otherwise**.

### Results

A new Otherwise branch is added under the node. The Otherwise branch is green.



**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)

[Deleting a branch](#)

# Changing a branch to and from Otherwise

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch.

## About this task

You can change an existing branch to an Otherwise branch, or change an Otherwise branch back to a normal branch. Note that each node can have only one Otherwise branch.

## Procedure

To change a branch to and from Otherwise:

1. In the decision tree editor, right-click the Otherwise branch.
2. Click **Set / Unset as Otherwise**.

## Results

The status of the branch changes to or from an Otherwise branch, as appropriate.

**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Duplicating a branch](#)

[Deleting a branch](#)

# Inserting and defining branches for enumerated domain values


You can insert and define branches automatically on nodes that define an enumerated domain.

## About this task

In the decision tree editor, create branches for the enumerated domain values.

## Procedure

To insert and define branches for all the values of an enumerated domain:

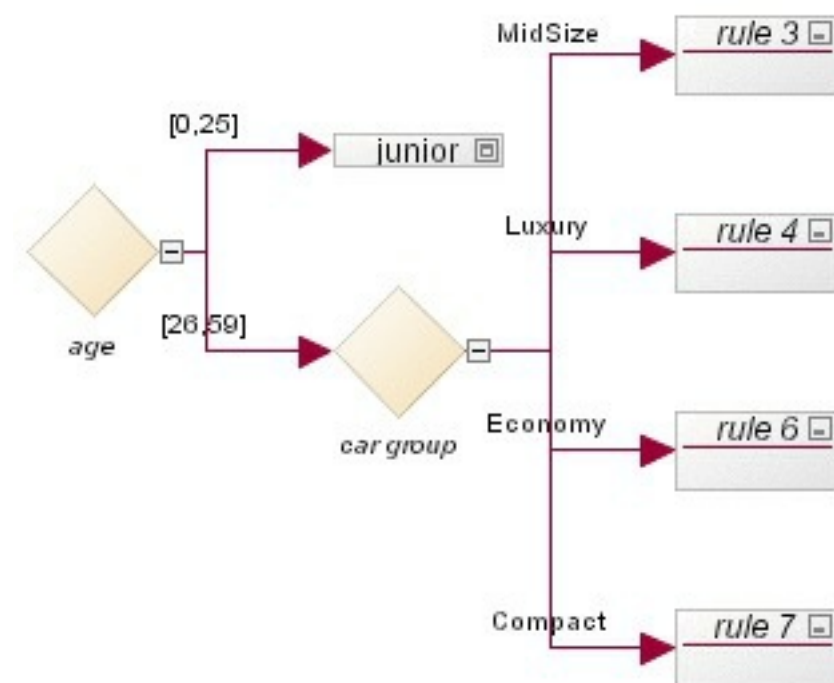
1. In the decision tree editor, right-click the condition node to which you want to attach the branches.
2. Click **Add** >  **All Domain Values**.

### Note:

The Decision Tree Editor can detect missing domain values. You can add missing values automatically using the quick fixes. See [Correcting errors by using Quick Fix](#).

## Results

The new branches are displayed under the node.



**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)  
[Labeling a branch](#)  
[Inserting a new branch](#)  
[Inserting an Otherwise branch](#)  
[Changing a branch to and from Otherwise](#)  
[Duplicating a branch](#)  
[Deleting a branch](#)

# Duplicating a branch

You duplicate a branch by using drag and drop in the Decision Tree Editor.

## Procedure

To duplicate a branch:

1. In the decision tree editor, click the branch that you want to duplicate.
2. Press Ctrl and drag the branch to a location at the same level.
3. Release Ctrl.

## Results

If overlap checking is active on the parent condition node, an error is displayed on the duplicated branch.

Without using the Ctrl key, you can use drag and drop to reorder nodes at the same level. A drag and drop without using the Ctrl key is equivalent to a cut and paste. You can also drop a branch onto a node to replace it with a copy of the selected branch.

**Parent topic:** [Labeling and defining branches](#)

### Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)


[Deleting a branch](#)

## Deleting a branch

When you delete a branch, you also delete all its associated actions.

### Procedure

To delete a branch:

1. In the decision tree editor, right-click the branch.
2. Click **Delete** .

### Results

The branch, together with its associated actions, are deleted.

**Parent topic:** [Labeling and defining branches](#)

#### Related tasks:

[Defining a branch](#)

[Labeling a branch](#)

[Inserting a new branch](#)

[Inserting an Otherwise branch](#)

[Inserting and defining branches for enumerated domain values](#)

[Changing a branch to and from Otherwise](#)

[Duplicating a branch](#)



## Labeling and defining actions

Actions are made up of two elements: the action set header (title) and the action set.

### Labeling an action set

You label actions by entering a title for the actions in the action set header.

### Defining an action

You define an action in the action set part of the action.

### Adding an action to an action set

When you add an action to an action set, you must then define it.

### Changing the order of actions

You change the order of actions in an action set by moving them up or down the list of actions.

### Deleting actions

You can delete individual actions from an action set, or the whole action set.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

# Labeling an action set

You label actions by entering a title for the actions in the action set header.

## About this task

You label action set headers. The label (title) must match the name of the corresponding rule.

## Procedure

To label an action set:

1. In the decision tree editor, double-click the action set header.
2. In the Action Set Editor dialog, type a title for the actions in the **Title** field.
3. Click **OK**.

## Results

Your title replaces the default title.

**Parent topic:** [Labeling and defining actions](#)

### Related tasks:

[Defining an action](#)

[Adding an action to an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

# Defining an action


You define an action in the action set part of the action.

## About this task

The action set can contain one or more actions.

## Procedure

To define an action:

1. In the decision tree editor, in the action set part of the action, click **edit action**.
2. In the edit bar, define the action. The edit bar operates in the same way as the guided editor.
3. Click **Enter** .

## Results

The action is now defined, and the action set displays the action text.

**Parent topic:** [Labeling and defining actions](#)

### Related tasks:

[Labeling an action set](#)

[Adding an action to an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

# Adding an action to an action set


When you add an action to an action set, you must then define it.

## About this task

When you add an action to an action set, you define it by using the **edit action** entry.

## Procedure

To add an action to an action set:

1. In the decision tree editor, right-click the action set header.
2. Click **Add Action** .

## Results

The action set displays a new **edit action** entry, for you to define the new action.

**Parent topic:** [Labeling and defining actions](#)

### Related tasks:

[Defining an action](#)

[Labeling an action set](#)

[Changing the order of actions](#)

[Deleting actions](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

# Changing the order of actions



You change the order of actions in an action set by moving them up or down the list of actions.

## About this task

Move the actions up or down to change their order.

## Procedure

To change the order of actions:

1. In the action set, right-click the action you want to move.
2. Click  **Move Up** or  **Move Down**.
3. Save your changes.

## Results

Actions are now listed in the new order.

**Parent topic:** [Labeling and defining actions](#)

### Related tasks:

[Defining an action](#)

[Adding an action to an action set](#)

[Labeling an action set](#)

[Deleting actions](#)

### Related information:

[Decision trees](#)

[\(Deprecated\) Working with decision trees](#)

# Deleting actions

You can delete individual actions from an action set, or the whole action set.



## About this task

When you delete the whole action set, you also delete the associated branch.

## Procedure

To delete actions:

- 1. Decide whether you want to delete an individual action, or an action set:

<div><div>a. In the action set, right-click the action.</div><div>b. Click  <b>Delete</b>.</div><div>The action is removed from the action set.</div></div>	<div><div>a. In the decision tree editor, right-click the action set header.</div><div>b. Click  <b>Delete</b>.</div><div>The action set and its associated branch is deleted.</div></div>

- 2. Save your changes.

**Parent topic:** [Labeling and defining actions](#)

## Related tasks:

- [Labeling an action set](#)
- [Defining an action](#)
- [Adding an action to an action set](#)
- [Changing the order of actions](#)

## Related information:

- [Decision trees](#)
- [\(Deprecated\) Working with decision trees](#)

# Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision tree and conditions that must be satisfied before the decision tree is executed.

## About this task

You can define preconditions for a decision tree. Preconditions can contain the following definitions:

- Variables that are available throughout the decision tree
- Conditions that must be satisfied before the decision tree is executed

## Procedure

To define a precondition:

1. In the decision tree editor, click the **General** tab.
2. In the **Preconditions** section, press Ctrl+Spacebar.

The Content Assist box opens. You define variables and conditions in the Preconditions section in the same way as you would create variables and conditions for an action rule in the Intellirule editor. See [Building rules using the Intellirule editor](#) for details.

3. Save the decision tree.

The variables you defined are now available for creating condition nodes and branches in the decision tree, which means the decision tree cannot be evaluated before the conditions defined in the precondition are satisfied.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

# Defining checking options

You can specify the tests that you want Rule Designer to conduct to identify problems with a decision tree.


## About this task

You can define checking options at tree level, or at individual condition node level.

## Procedure

To define checking options for the whole tree:

Choose whether you want to check options for the whole tree, or just a condition node:

To define checking options for the whole tree:	To define checking options on a condition node:
<div><div>a. Right-click anywhere in the decision tree editor, and then click  <b>Decision Tree Properties</b>. The Decision Tree Properties dialog opens.</div><div>b. In the <b>Checking</b> section, select the checks you require.</div><div>You can select whether you want gaps and/or overlaps to be checked, and the severity of checking reports.</div></div>	<div><div>a. In the decision tree editor, double-click the condition node. The Node Editor dialog opens.</div><div>b. In the <b>Checking</b> section, select which checks you require.</div><div>You can select whether you want gaps and/or overlaps to be checked.</div></div>

Parent topic: [\(Deprecated\) Working with decision trees](#)

## Related information:

[Decision trees](#)

[Business Action Language \(BAL\)](#)



# Viewing rules

You can view the details of a rule in a decision tree by viewing a tooltip or by enabling the display of a dedicated viewing pane below the decision tree editor.


## About this task

You can view the text of a rule corresponding to an action. You can view the rule text in a tooltip, or in a dedicated viewing pane.

## Procedure

To view rule text:

Choose whether you want to view the rule text in a tooltip or in a viewing pane:

To view rule text in a tooltip:	To view rules in a dedicated viewing pane:
<div><div>a. In the decision tree editor, place the cursor over the action set.</div><div>The rule is displayed in a tooltip.</div></div>	<div><div>a. In the decision tree editor toolbar, click  <b>Decision Tree Properties</b>. The Decision Tree Properties dialog opens.</div><div>b. In the <b>Tree view</b> section, select <b>Show rules</b>.</div><div>c. Click <b>OK</b>.</div><div>A new viewing pane opens at the bottom of the decision tree editor.</div><div>d. In the decision tree editor, click the header of an action set.</div><div>The corresponding rule is shown in the viewing pane.</div></div>

## Results

Whichever way you choose to view the rules, if you defined preconditions for the decision tree, these are also shown.

**Parent topic:** [\(Deprecated\) Working with decision trees](#)

## **(Deprecated) Working with templates**

You can create partially written rules and partially defined decision tables for use as templates, to simplify the task of creating rules and decision tables.

### **(Deprecated) Templates**

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

### **Setting up an action rule template**

You use the guided editor to create action rule templates. You leave placeholders to cater for variations, and you freeze the parts that should remain fixed.

### **Setting up a decision table template**

You create decision table templates to enable the efficient creation of multiple decision tables that have similar content and structure.

**Parent topic:** [Authoring business rules](#)

## **(Deprecated) Templates**

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

By defining it as such in the template, parts of rules that are created from a template can be frozen so that they cannot be edited.

Templates simplify the task of writing rules and decision tables, and are useful for:

- Creating action rules and decision tables that are already partially written.
- Restricting users from editing certain parts of partially completed rules.

A template applies only at the moment you create a rule or a decision table. If you subsequently modify the template, this will have no impact on rules or decision tables previously created from this template. You would need to update these rules or decision tables manually.

**Parent topic:** [\(Deprecated\) Working with templates](#)

**Related reference:**

[Business Action Language \(BAL\)](#)

**Related information:**

[Setting up an action rule template](#)

# Setting up an action rule template

You use the guided editor to create action rule templates. You leave placeholders to cater for variations, and you freeze the parts that should remain fixed.

## Overview

If you want to create a number of action rules with similar content and structure, you can create an action rule template, and protect (freeze) parts or all of a template so that they cannot be edited in the action rules created from the template.

## Creating an action rule template

You can create an action rule template if you want to create a number of action rules with similar content and structure.

## Freezing and unfreezing parts of an action rule template

You can protect certain parts of the template so that they cannot be edited in the action rules created from the template.

## Freezing or unfreezing the whole template

How to freeze or unfreeze the whole template.

## Propagating changes to a template's freeze/unfreeze settings

How to propagate changes to a template's freeze/unfreeze settings.

**Parent topic:** [\(Deprecated\) Working with templates](#)

**Related concepts:**

[\(Deprecated\) Templates](#)

**Related reference:**

[Business Action Language \(BAL\)](#)

## Overview

If you want to create a number of action rules with similar content and structure, you can create an action rule template, and protect (freeze) parts or all of a template so that they cannot be edited in the action rules created from the template.

If you change the freeze/unfreeze settings in the template, you can use a specific command to reflect the change in the rules created from it. However, if you change the template content, you can reflect these changes only by manually updating the associated action rules.

You build action rules in a template in the same way that you create a normal action rule, except that:

- You leave parts of the rule empty to cater for variations in the action rules that are created from it
- You can use only the guided editor to build template rules and the action rules created from the template, as only this editor has the capability to set and apply freeze information. However, if text in a rule template or a rule derived from a template needs to be edited at a later date, you must use the Intellirule Editor.

**Parent topic:** [Setting up an action rule template](#)

# Creating an action rule template

You can create an action rule template if you want to create a number of action rules with similar content and structure.

## Before you begin

### Procedure

To create an action rule template:

1. Select your rule project and on the **File** menu click **New** > **Action Rule Template**.

The New Action Rule Template wizard opens.

2. The default directory for templates to be saved is the templates folder in your rule project. If you want to save your template to a folder inside the templates folder, click **Browse** in the **Folder** field and navigate to the required folder.
3. Type the name of your action rule in the **Name** field.
4. Click **Finish**.

The new action rule template is shown in the folder you specified, and the Action Rule Template guided editor opens.

5. Using the guided editor, build the contents of the template, leaving those parts of the rule empty where variations are likely to occur.

In the Documentation section, you can enter information specific to the template. This information is not copied to the rules created from the template. To enter documentation that applies to all the rules created from the template, use the **documentation** property in the Properties view.

6. Save your work.

## Results

You freeze parts of the rule template through the Action Rule Template guided editor.

**Parent topic:** [Setting up an action rule template](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Creating an action rule](#)

[Defining a folder structure for rule project items](#)

### Related reference:

[Business Action Language \(BAL\)](#)

### Related information:

[Building rules using the Intellirule editor](#)

[Building action rules using the guided editor](#)

# Freezing and unfreezing parts of an action rule template

You can protect certain parts of the template so that they cannot be edited in the action rules created from the template.

## About this task

In the guided editor, you can define a specific structure for the rules by freezing some controls in the template.

## Procedure

To freeze and unfreeze controls:

1. In the Action Rule Template guided editor, right-click each control that you want to freeze, and click **Freeze**.

The controls that you have frozen are grayed out. Note that place holders are still shown in blue, indicating that you can still edit them.

### Note:

If you keep the same structure, the freeze information is kept. However, the freeze information is lost if you change the structure of the rule.

2. To unfreeze a control, in the Action Rule Template Guided Editor, right-click the control that you want to unfreeze, and click **Unfreeze**.

The controls that you have unfrozen are shown in black.

## Results

You can also freeze the whole template. This action freezes everything except the place holders. These are still shown in blue, indicating that you can still edit them.

**Parent topic:** [Setting up an action rule template](#)

### Related information:

[Controls in the guided editor](#)

# Freezing or unfreezing the whole template

How to freeze or unfreeze the whole template.

## About this task

You can protect (freeze) the whole template so that it cannot be edited in the action rules created from the template.

## Procedure

To freeze or unfreeze the whole template:

1. Right-click anywhere in the Action Rule Template Guided Editor, and click **Freeze All**.
2. To unfreeze the whole template, right-click anywhere in the Action Rule Template guided editor, and click **Unfreeze All**.

### Note:

The **Freeze** and the **Freeze All** functions also freeze the black arrow icon ▼ that allows policy managers to change whether they edit a value or select an item. To allow them to make this choice, unfreeze the symbol.

**Parent topic:** [Setting up an action rule template](#)



# Propagating changes to a template's freeze/unfreeze settings

How to propagate changes to a template's freeze/unfreeze settings.

## About this task

If you change the freeze/unfreeze settings in the template, you can use a specific command to reflect the change in the rules created from it.

However, if you change the template content, you can reflect these changes only by manually updating the associated action rules.

## Procedure

To propagate changes to a template's freeze/unfreeze settings:

1. In the Rule Explorer, select the template.
2. Right-click the template and click **Action Rule Template > Apply Freeze Information**.

The new freeze or unfreeze setting in the template is propagated through to the action rules created from the template.

**Parent topic:** [Setting up an action rule template](#)

# Setting up a decision table template

You create decision table templates to enable the efficient creation of multiple decision tables that have similar content and structure.

## About this task

You build decision tables in a template in the same way that you create a normal decision table, except that you leave parts of the decision table empty to cater for variations in the decision tables that are created from it.

## Procedure

To create a decision table template:

1. Select your rule project and on the **File** menu click **New > Decision Table Template**.

The New Decision Table Template wizard opens.

2. The default directory for templates to be saved is the templates folder in your rule project. If you want to save your template to a folder inside the templates folder, click **Browse** in the **Folder** field and navigate to the required folder.
3. Type the name of your decision table in the **Name** field.
4. Click **Finish**.

The new decision table template is shown in the folder you specified, and the decision table editor opens.

5. Using the decision table editor, build the contents of the template, leaving those parts of the decision table empty where variations are likely to occur.

In the Documentation section, you can enter information specific to the template. This information is not copied to the decision tables created from the template. To enter documentation that applies to all the decision tables created from the template, use the **documentation** property in the Properties view.

6. Save your work.

**Parent topic:** [\(Deprecated\) Working with templates](#)

## Related tasks:

[Creating a decision table from scratch](#)

## Working with technical rules

Use the Technical Rule Editor to write technical rules in the ILOG® Rule Language.

### Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping.

### Creating a technical rule

You create a technical rule by specifying a name and source folder, and optionally specifying a package.

### Using the Technical Rule Editor

You use the Technical Rule Editor to build technical rules.

### Writing technical rules

Your technical rules must conform to the ILOG Rule Language (IRL) structure. Technical rules can contain variable definitions, conditions, and actions.

**Parent topic:** [Authoring business rules](#)

## Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping.

### Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

### Variables in technical rules

You can define variables in the condition part of a technical rule and use it subsequently.

### Technical rule conditions

Conditions are expressed using the class of object being tested followed by an operator. Actions are only executed if all the tests are satisfied.

### Technical rule actions

Technical rule actions control execution, make loops, branch, handle exceptions and discriminate between execution branches.

**Parent topic:** [Working with technical rules](#)

## Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
  - `evaluate`
  - `exists`
  - `from`
  - `in`
  - `instanceof`
  - `not`
  - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

<b>Note:</b> Rule Designer does not refactor technical rules. If you rename, move, or delete a technical rule, the references in other rule project artifacts are not updated.
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Parent topic:** [Technical rules](#)

**Related concepts:**

[Action rules](#)

[\(Deprecated\) Decision tree overview](#)

**Related information:**

[Overview: Ways to express business rules](#)

[Decision tables](#)

[Action rule editing errors and warnings](#)

## Variables in technical rules

You can define variables in the condition part of a technical rule and use it subsequently.

```
when
{
 ?x: Fish(color==yellow; type==angel);
}
then
{
 retract ?x;
 insert Fish(yellow, shark);
}
```

The presence of the ?x variable at the beginning of the condition serves as a marker, which identifies any object that matches this condition. You can then refer to the matched object in other conditions or in the action part of the rule. Consequently, when the first action asks for the removal of the object referenced by the ?x rule variable, the yellow angel fish object is removed from the set of objects provided to the rule engine for execution.

**Parent topic:** [Technical rules](#)

**Related concepts:**  
[Technical rule conditions](#)

**Related tasks:**  
[Applying verbalization changes to business rules](#)

**Related information:**  
[Technical rule actions](#)  
[Working with technical rules](#)  
[ILOG Rule Language \(IRL\)](#)

# Technical rule conditions

Conditions are expressed using the class of object being tested followed by an operator. Actions are only executed if all the tests are satisfied.

The condition part of a technical rule is composed of a set of tests on Java™ objects. Tests are applied to each object provided to the rule engine for execution, and an object is said to match the condition when it passes all the tests in the condition.

## Syntax of conditions

A condition starts with the name of the class concerned by the condition, such as `Film`. This class name is followed by tests, enclosed in parentheses, that express constraints on the attribute values for objects of this class.

For example, suppose that you want the rule to apply only to films showing after 8 in the evening. This condition is expressed as follows:

```
Film(showTime > 20.00);
```

This condition matches objects of the `Film` class whose `showTime` attribute is greater than `20.00`. Suppose that you then require the location of the film to be in Paris. This compound condition is expressed as follows:

```
Film(showTime > 20.00; location == Paris);
```

## Operators

Each test starts by naming the attribute, followed by one of the operators in the following table.

Table 1. Condition operators

Operator	Description
<code>==</code>	equals
<code>!=</code>	does not equal
<code>&lt;</code>	is less than
<code>&gt;</code>	is greater than
<code>&lt;=</code>	is less than or equal to
<code>&gt;=</code>	is greater than or equal to
<code>in</code>	is in
<code>!in</code>	is not in
a method call	method invocation

The test ends with an expression whose value is compared with that of the attribute.

A condition can contain several tests involving the same object attribute. For example, the following condition tests for films whose show times are between 8 and 10 in the evening:

```
Film(showTime > 20.00; showTime < 22.00);
```

## Combination sign: &

The ampersand sign `&` is used as a combination sign, to associate several tests with the same object attribute involved in a condition. Therefore, to test for films whose show times are between 8 and 10 in the evening, you can express the condition as follows:

```
Film(showTime > 20.00 & < 22.00);
```

Here are two equivalent conditions, the second of which uses the `&` sign:

```
Film(produced > 1980; produced < 1990; showTime > 20.00; showTime < 22.00);
Film(produced > 1980 & < 1990; showTime > 20.00 & < 22.00);
```

## Conjunction

Implicit AND relationships exist between all the tests in a rule. In other words, all the tests in the condition part of a rule must be satisfied before actions can be executed.

For example, consider the following conditions: “a film whose spoken language is English” and “a cinema whose location is Paris.” These conditions might be written in the following way:

```
Film(language == English);
Cinema(location == Paris);
```

These two conditions are matched by pairs of Film and Cinema objects for which the language of the film object is equal to English AND the location of the cinema object is equal to Paris.

**Parent topic:** [Technical rules](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)



## Technical rule actions

Technical rule actions control execution, make loops, branch, handle exceptions and discriminate between execution branches.

### Action part of a technical rule

Actions can create, remove, and modify objects.

### If/Else control

Use `if` statements to apply actions selectively. Use `else` statements to do a different set of actions when the expression is false.

### Loops

You can add loops in actions to repeat the execution of action statements. You control the repetitions using `for` and `while` statements.

### Branching statements

You can add `break` and `continue` branching statements in actions.

### Exception handling statements

You can add statements that handle exceptions in actions.

### Else clause with an evaluate statement

Technical rules with an evaluation statement in the condition part can have an optional `else` statement in the action part.

**Parent topic:** [Technical rules](#)

## Action part of a technical rule

Actions can create, remove, and modify objects.

The keyword `then` is used to specify the start of the action part of a rule. The action part of a rule is composed of IRL statements that can, for example, create, remove, and modify objects.

For controlling the flow of the action part of a rule, the ILOG® Rule Language (IRL) includes statements for:

- controlling execution (`if/else`)
- making loops (`for`, `while`)
- branching (`break`, `continue`)
- handling exceptions (`try`, `catch`, `finally`)

These statements follow the Java™ language specification. You can also add an optional `else` clause in the action part of a technical rule.

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[If/Else control](#)

[Loops](#)

[Branching statements](#)

[Exception handling statements](#)

[Else clause with an evaluate statement](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## If/Else control

Use `if` statements to apply actions selectively. Use `else` statements to do a different set of actions when the expression is false.

The `if` statement enables rule actions to selectively execute other statements based on specific criteria.

For example, suppose that your rule prints debugging information based on the value of a Boolean variable named `DEBUG`. If `DEBUG` is `true`, your rule prints debugging information, such as the value of a variable `x`. Otherwise, your rule proceeds normally. A segment of code to implement the action might look like this:

```
then {
 if (?DEBUG) {
 System.out.println("DEBUG: x = " + ?x);
 }
}
```

This is the simplest version of the `if` statement: the block governed by the `if` is executed if a condition is true. Generally, the simple form of `if` can be written like this:

```
then {
 if (test) {
 statements
 }
}
```

If you want to do a different set of statements if the expression is false, use the `else` statement. For example, suppose your rule needs to do different actions depending on whether the user clicks the OK button or another button in an alert window. Your rule could do this by using an `if` and an `else` statement:

```
then {
 int ?i = 1;
 if (?i == 1) {
 System.out.println(" i = 1 ") ;
 }
 else {
 System.out.println(" i <> 1 ") ;
 }
}
```

The `else` block is executed only if the `if` part is false.

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[Branching statements](#)

[Exception handling statements](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Loops

You can add loops in actions to repeat the execution of action statements. You control the repetitions using `for` and `while` statements.

IRL includes statements for making loops in the action part of technical rules.

### For loop

The `for` statement provides a compact way to iterate over a range of values.

The general form of the `for` statement can be expressed like this:

```
then{
 for (initialization; test; increment) {
 statements
 }
}
```

The expression `initialization` is used to initialize the loop. It is executed once at the beginning of the loop. The test expression determines when to terminate the loop. This expression is evaluated at the top of each iteration of the loop. When the expression evaluates to false, the loop terminates. Finally, `increment` is an expression that is invoked after each iteration through the loop. Here is an example:

```
then {
 int ?i = 1;
 int ?j = 1;
 for (?i = 1; ?i <= 3; ?i++) {
 for (?j = 1; ?j <= 3; ?j++) {
 System.out.println(" i =" + ?i + " j = " + ?j) ;
 }
 }
}
```

Often `for` loops are used to iterate over the elements in an array, or the characters in a string. The following example uses a `for` statement to iterate over the elements of an array and print them:

```
then {
 for (?i = 0; ?i < ?arrayOfInts.length; ?i++) {
 System.out.println(?arrayOfInts[?i] + " ");
 }
}
```

### While loop

You use a `while` statement to continually execute a block of statements while a condition remains true. The general syntax of the `while` statement is:

```
then {
 while (test) {
 statements
 }
}
```

First the `while` statement evaluates the `test` expression, which must return a boolean value. If the test returns true, then the `while` statement executes the statements associated with it. The `while` statement continues testing the test expression and executing its block until the test returns false. Here is an example of a `while` loop:

```
then {
 int ?i = 0 ;
 while (?i <= 3) {
 System.out.println(?i) ;
 ?i++ ;
 }
}
```

The following example uses a while statement in the action part of the rule to iterate over an array of elements in a variable named elements. The variable elements is bound to an attribute of a variable c, also named elements. The variable c is declared in the condition part of the rule as a collection of ManagedObject().

```
rule foundManagedObject {
 when {
 ?c: collect ManagedObject();
 }
 then {
 Enumeration ?elements = ?c.elements();
 while (?elements.hasMoreElements()) {
 System.out.println(elements.nextElement());
 }
 }
}
```

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[Branching statements](#)

[Exception handling statements](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Branching statements

You can add break and continue branching statements in actions.

IRL supports two branching statements: the break statement and the continue statement.

### Break

The break statement terminates a for or a while loop when the statement is found. In a for loop, for example, the flow of control transfers to the statement following the enclosing for, as shown here:

```
then {
 int ?i = 1;
 int ?j = 1;
 for (?i = 1; ?i <= 3; ?i++) {
 System.out.println("\ni= " + ?i + ": ");
 for (?j = 1; ?j <= 3; ?j++) {
 System.out.println(" j = " + ?j) ;
 if (?i == ?j) {
 break;
 }
 }
 }
}
```

The break statement ends only the loop in which it exists. If two loops are nested, a break in the inner loop exits the inner loop but not the outer loop. The output of the above example is shown here:

```
i= 1:
 j = 1

i= 2:
 j = 1
 j = 2

i= 3:
 j = 1
 j = 2
 j = 3
```

### Continue

You use the continue statement to skip the current iteration of a for or while loop. Instead of ending the loop like the break statement, the continue statement skips all following statements in the loop body and executes the next iteration of the loop. The continue statement is demonstrated in the following example.

```
then {
 StringBuffer ?whitePaper = new StringBuffer(
 "The Case for Business Users of Information Technology");
 int ?max = ?whitePaper.length();
 int ?numSs = 0;
 int ?i = 0;
 for (?i = 0; ?i < ?max; ?i++) {
 if (?whitePaper.charAt(?i) != 's'){
 continue;
 }
 ?numSs++;
 ?whitePaper.setCharAt(?i, 'S');
 }
 System.out.println("Found " + ?numSs + " s's in the string.\n");
 System.out.println(?whitePaper);
}
```

Here is the output:

```
Found 6 s's in the string.
```

## The CaSe for BuSineSS USeRS of Information Technology

The example steps through a string buffer checking each letter. If the current character is not an s, the continue statement skips the rest of the statements in the loop and proceeds to the next iteration to test the next character. If it is an s, the rule increments a counter and converts the s to uppercase.

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[Loops](#)

[Exception handling statements](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Exception handling statements

You can add statements that handle exceptions in actions.

IRL provides mechanisms for reporting and handling exceptions. When an error occurs, the rule throws an exception. This means that the normal flow of the rule is interrupted, and the rule engine attempts to find an exception handler, that is, a block of code that handles a particular type of error. The exception handler generally does the necessary actions to recover from the error.

### Exception setup

Before you can catch an exception, code somewhere must throw one. Any Java™ code can throw an exception.

A throw expression is any kind of expression whose type is assignable to the Java Throwable type or subclass. A throw expression can be specified either in the API or in a rule. When an exception is thrown it can be caught by the API or a rule—with a try-catch-finally statement or by any Java code—which causes the IRL code to execute.

#### Note:

A thrown exception is an `IlrUserRuntimeException` subtype of `IlrRunTimeException` which encapsulates the exception thrown by the throw statement.

- `IlrSystemRuntimeException` is thrown when Decision Server detects an error at runtime. This kind of exception cannot be recovered; it is used only for debugging purposes.
- `IlrUserRuntimeException`, on the other hand, could be used to reset the context and relaunch the rules.

### The try-catch-finally statements

The try-catch-finally statements are:

- The try statement identifies a block of statements within which an exception might be thrown.
- The catch statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block. There can be as many statements following a try statement as needed. Each statement handles any and all exceptions that are instances of the class listed in parentheses, of any of its subclasses, or of a class that implements the interface listed in parentheses.
- The finally statement must be associated with a try statement and identifies a block of statements that is executed regardless of whether or not an error occurs within the try block. The finally statement is generally used to clean up after the code in the try statement. If an exception occurs in the try block and there is an associated catch block to handle the exception, control transfers first to the catch block and then to the finally block.

Here is the general form of these statements:

```
try {
 statements
}
catch (ExceptionType1 name) {
 statements
}
catch (ExceptionType2 name) {
 statements
}
...
finally {
 statements
}
```

IRL allows general exception handlers that handle multiple types of exceptions. However, more specialized exception handlers can determine what type of exception occurred and assist in the recovery of these errors. Handlers that are too general can make code more error prone by catching and handling exceptions that were not anticipated and for which the handler was not intended.

The following try statements demonstrate how to create an exception and throw it.

```
then {
```



```

try {
 method1(1) ;
 System.out.println("Call method1(1) was OK") ;
}
catch (Exception e) {
 System.out.println("Catch from method1(1) call") ;
}
try {
 method1(2) ;
 System.out.println("Call method1(2) was OK") ;
}
catch (Exception e) {
 System.out.println("Catch from method1(2) call") ;
 System.out.println("Exception details :-") ;
 System.out.println("Message: " + e) ;
}
}

```

The try-catch statements use a method call from a Java class to throw an exception when the variable passed into method1 is not equal to 1. Here is method1:

```

public static void method1(i) throws Exception{
 if (i == 1) {
 System.out.println("method1 - Things are fine \n") ;
 }
 else {
 System.out.println("method1 - Somethings wrong! \n") ;
 throw new Exception("method1 - Its an exception! \n") ;
 }
}

```

Exceptions thrown by any rule or Java code within the scope of the rule engine is caught by the try-catch statements for that specific exception.

The following CatchCustomer rule provides an example of a try-catch statement block capable of catching exceptions thrown by the ILOG Rule language and/or Java code. The rule uses two classes, Customer and Item, and two subclasses of the Java Exception class, TooHighExpense and IsNotMember.

```

rule CatchCustomer
{
 when {
 ?c: Customer();
 ?item: Item();
 }
 then {
 try {
 System.out.println("Customer " + ?c.getName() + " wants to buy "
 + " item " + ?item.getLabel() + " for a price of " +
 ?item.getPrice() + "$");

 IsMember(?c);
 ?c.buy(?item);
 }
 catch (TooHighExpense ex) {
 System.out.println("M/Mrs/Mz " + ?c.getName() + " you have already
 bought:");

 int j = 0;
 for(j = 0; j<?c.getItems().size(); j++) {
 java.util.Enumeration ?i = ?c.getItem(j);
 System.out.println(" Item: " + ?i.getLabel() + " costs " +
 ?i.getPrice() + "$");
 }
 System.out.println("You have at your disposal " + ?c.getBudget() + "$");
 System.out.println("The current purchase is not allowed");
 }
 catch (IsNotMember ex) {
 System.out.println("M/Mrs/Mz " + ?c.getName() +

```

```

 ", you are not a member; would you like to
 subscribe?");
 }
}
}

```

In the example, a customer is given a budget. The customer is not allowed to buy items that exceed the allocated budget. If the customer attempts to buy items that exceed the budget, a TooHighExpense exception is thrown. In addition, to buy items the customer must be a member. If a nonmember attempts to buy an item, an IsNotMember exception is thrown.

The two Java classes IsNotMember and TooHighExpense used in the CatchCustomer rule are shown here:

```

public class IsNotMember extends Exception
{
 public IsNotMember(String name){
 super();
 this.customer = name;
 }
 public void printStackTrace(PrintWriter s){
 s.println("An IsNotMember exception has been caused by customer "
 + customer + ". We are sorry but you can not make a purchase "
 + "without being a member.");
 super.printStackTrace(s);
 }
 String customer;
};

public class TooHighExpense extends Exception
{
 public TooHighExpense(String name, int expense, int limit, int price){
 super();
 this.expense = expense;
 this.limit = limit;
 this.price = price;
 this.customer = name;
 }
 public void printStackTrace(PrintWriter s){
 s.println("A TooHighExpense exception has been caused by customer "
 + customer + ". For this customer, the current expense is " +
expense +
 " and the authorized budget is " + limit +
 ". The purchase of the item whose price is " + price
 + " is not allowed.");
 super.printStackTrace(s);
 }
 private int price;
 private int limit;
 private int expense;
 String customer;
};

```

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[Loops](#)

[Branching statements](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Else clause with an evaluate statement

Technical rules with an evaluation statement in the condition part can have an optional `else` statement in the action part.

When the condition part of a rule finishes with an `evaluate` statement, the action part of the rule might include an `else` clause. The `else` clause is optional. The last `evaluate` expression in the condition part acts as a discriminator, to choose between the `then` or the `else` execution branches. If the `evaluate` expression is true, the `then` part is executed; if it is false, the `else` part is executed.

If the rule ends with several `evaluate` statements, only the last one is used to discriminate.

Here is an example of a rule with an `else` statement:

```
rule myrule
{
 when
 {
 ?s : String(startsWith("Irl"));
 evaluate(?s.length() > 30);
 }
 then
 {
 out.println("Very long string");
 }
 else
 {
 out.println("String length is reasonable");
 }
};
```

**Parent topic:** [Technical rule actions](#)

**Related concepts:**

[Branching statements](#)

[Exception handling statements](#)

**Related tasks:**

[Applying verbalization changes to business rules](#)

**Related information:**

[Technical rule actions](#)

[Working with technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Creating a technical rule

You create a technical rule by specifying a name and source folder, and optionally specifying a package.

### About this task

Technical rules are rules written in the ILOG® Rule Language. They allow you to write specific loops in the action part of your rules.

### Procedure

To create a technical rule:

1. Select your rule project and on the **File** menu click **New** > **Technical Rule**.

The New Technical Rule wizard opens.

2. Select the required Source folder.
3. If you want to select a rule package, click **Browse** in the **Package Name** field.
4. Type the name of the technical rule in the **Name** field.
5. Click **Finish**.

The new technical rule is shown in the Rule Explorer view and the Technical Rule Editor opens.

**Parent topic:** [Working with technical rules](#)

# Using the Technical Rule Editor

You use the Technical Rule Editor to build technical rules.

## **Using Content Assist in a technical rule**

Use Content Assist to build technical rules by selecting valid code from a list.

## **Errors in the Technical Rule Editor**

The Technical Rule Editor dynamically checks the correctness of the code in a technical rule.

## **Organizing import statements**

You organize import statements to check that all BOM classes that are used in the technical rule are correctly referenced. The Technical Rule Editor inserts missing import statements and removes redundant ones.

## **Opening declarations from the Technical Rule Editor**

You can use the text in a technical rule to browse through BOM classes, methods, attributes, and functions.

## **Defining IRL editing settings**

You can define preferences for editing in IRL. This applies to the editors for functions and ruleflow transitions.

**Parent topic:** [Working with technical rules](#)

## Using Content Assist in a technical rule

Use Content Assist to build technical rules by selecting valid code from a list.

### About this task

You use the Technical Rule Editor to write rules in the ILOG® Rule Language (IRL). To open the Technical Rule Editor, double-click a technical rule in the Rule Explorer.

The Technical Rule Editor lets you use Content Assist for the following items:

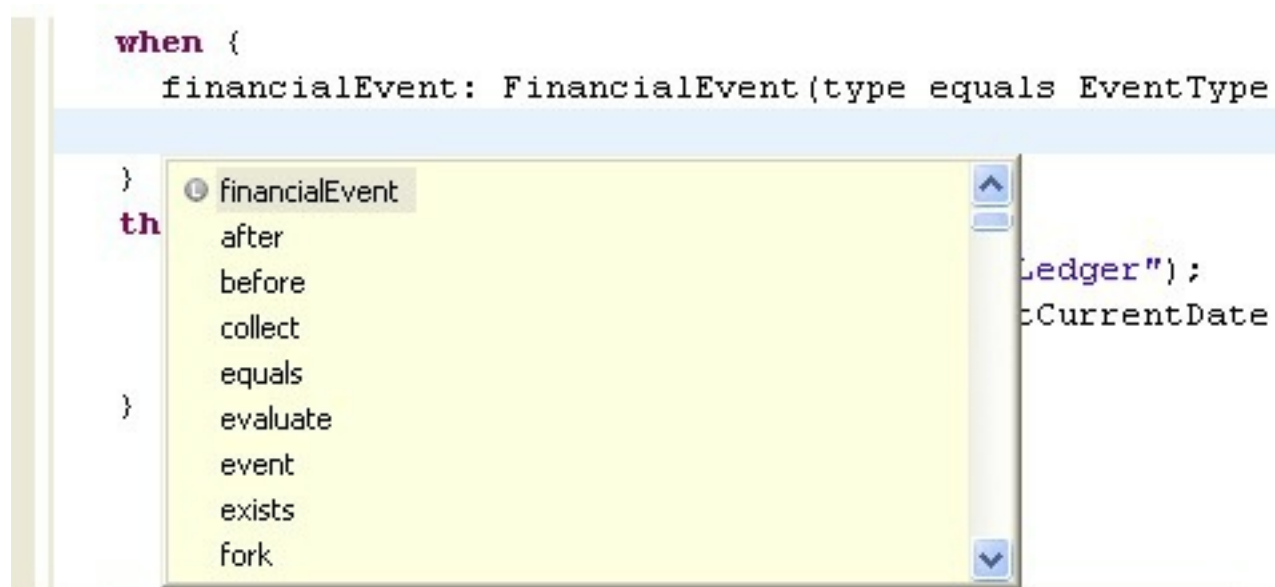
- IRL keywords
- Class names and package names
- BOM attributes, properties and methods
- Functions
- Rules and tasks
- Ruleset variables and parameters

### Procedure

To complete your code:

1. Type the first letters of a statement and press Ctrl+Spacebar.

This opens a list from which you can select an item.



2. Select an item and press Enter. The item is added to the statement in the Technical Rule Editor.
3. Continue until you have created your technical rule.

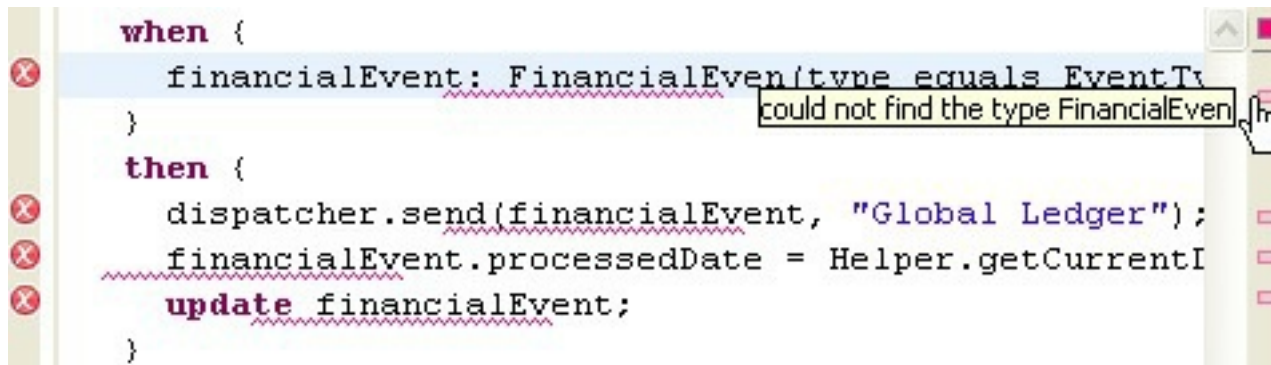
**Parent topic:** [Using the Technical Rule Editor](#)

## Errors in the Technical Rule Editor

The Technical Rule Editor dynamically checks the correctness of the code in a technical rule.

As you type in the Technical Rule Editor, it shows errors and warnings:

- Invalid code is underlined.
- Errors are displayed in the Technical Rule Editor margin.
- Both the Technical Rule Editor margin and the underlined code have tooltips that contain the error messages.



Errors are underlined in red. Warnings are underlined in yellow.

**Parent topic:** [Using the Technical Rule Editor](#)

### Related tasks:

[Organizing import statements](#)

[Opening declarations from the Technical Rule Editor](#)

[Defining IRL editing settings](#)

### Related information:

[Technical rules](#)

[Using the Technical Rule Editor](#)

[ILOG Rule Language \(IRL\)](#)

# Organizing import statements

You organize import statements to check that all BOM classes that are used in the technical rule are correctly referenced. The Technical Rule Editor inserts missing import statements and removes redundant ones.

## About this task

The Technical Rule Editor can help you to write the required import statements for a technical rule. The import statements reference BOM classes that are used in the technical rule.

## Procedure

To update the import statements:

Do one of the following actions:

- Select **Source** > **Organize Imports**.
- Press Ctrl+Alt+O.

## Results

Either action adds any missing import statements and removes the redundant ones in the Imports section of the Technical Rule Editor.

**Parent topic:** [Using the Technical Rule Editor](#)

## Related tasks:

[Opening declarations from the Technical Rule Editor](#)

[Defining IRL editing settings](#)

## Related information:

[Errors in the Technical Rule Editor](#)

[Technical rules](#)

[Using the Technical Rule Editor](#)

[ILOG Rule Language \(IRL\)](#)



## Opening declarations from the Technical Rule Editor

You can use the text in a technical rule to browse through BOM classes, methods, attributes, and functions.

### Procedure

To browse from an item in the Technical Rule Editor to its declaration in another file:

Do one of the following actions:

- Select the item, and then select **Navigate > Open Declaration**.
- Press F3.
- Press Ctrl and click to switch to link mode.

In link mode, the words in the Technical Rule Editor are displayed as links when you hover over them. When you click a link, the element declaration opens in a separate editor.

**Parent topic:** [Using the Technical Rule Editor](#)

## Defining IRL editing settings

You can define preferences for editing in IRL. This applies to the editors for functions and ruleflow transitions.

### Procedure

To modify IRL editing preferences:

1. From the **Window** menu, click **Preferences**. (On Mac, click **Eclipse** > **Preferences**.)
2. Expand **Rule Designer** , and then click **IRL Editing**.
3. Select the required appearance and syntax coloring settings.
4. Select the required options for automatic text modifications.

**Parent topic:** [Using the Technical Rule Editor](#)

## Writing technical rules

Your technical rules must conform to the ILOG® Rule Language (IRL) structure. Technical rules can contain variable definitions, conditions, and actions.

### Writing the basic structure of a technical rule in IRL

When you write a technical rule, use the ILOG Rule Language structure.

### Defining a variable

When you define a variable, use the ILOG Rule Language syntax.

### Writing conditions in IRL

You can write simple, complex, existence, collection, nonexistence, and evaluate conditions. To write conditions, use the ILOG Rule Language syntax.

### Adding tests to a condition

You can write standard tests, tests on attribute values, tests on values that are returned by methods, and tests on values that belong to sets of values. You can also write conditions in join tests.

### Using objects that are not in memory in a condition

When you write conditions, you can use objects that are not in memory by referring to relations from objects that are available.

### Writing actions in IRL

When you write rule actions, use the same syntax as a Java™ method body, except for the syntax for anonymous inner classes.

### Writing a rule with an else action clause

When you write a rule with an else action clause, use the ILOG Rule Language syntax.

### Notifying the rule engine of object changes

You can notify the rule engine of the availability of new objects, or when an object is updated or retracted.

**Parent topic:** [Working with technical rules](#)

# Writing the basic structure of a technical rule in IRL

When you write a technical rule, use the ILOG® Rule Language structure.

## About this task

In technical rules, when introduces conditions, and then introduces actions.

## Procedure

To write a technical rule, use the following pattern:

```
rule rulename {
 when {
 // conditions:
 <condition>;
 [<condition>;]*
 }
 then {
 // actions
 [<action>;]*
 }
};
```

**Note:** Use a valid identifier name in Java™ syntax for the rulename, except when you use dots, which are accepted.

## Example

```
rule financial.rules.CheckBalance {
 when {
 ...
 }
 then {
 ...
 }
};
```

**Parent topic:** [Writing technical rules](#)

### Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

### Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Defining a variable

When you define a variable, use the ILOG® Rule Language syntax.

### Procedure

- To define a simple class condition variable, use the following syntax in the condition part of the rule:  
`variable : ClassName();`

It defines a variable of type `ClassName`. This variable is never null and its value is shown in the rule instance tuple. The scope of this variable is the whole rule.

- To define a simple variable in a condition, use the following syntax in the test part of a condition:  
`variable : <value>`

To use this syntax in a class condition, follow this pattern: `ClassName (variable : <value>)`. To use this syntax in an evaluate condition, follow this pattern: `evaluate (variable : <value>)`. It defines a variable that has the type of the value. The value of the variable can be null. The scope of this variable depends on the condition in which it is defined: If the variable is defined in an existential condition, its scope is limited to this condition. If the variable is defined in any other type of condition, its scope is the whole rule.

- To define a simple variable in the action part of the rule, use the following syntax: `int variable = <value>;`

It defines a variable. The scope of the variable follows the Java™ language scoping rules.

### Example

In this example, the account and contract variables refer to objects in the working memory, objects of the Account BOM class and objects of the Contract BOM class. The balance is defined as the balance of an account in the condition part of the code. Similarly, the expiration is defined as the expiration date of the contract. The variable `b` is an example of a variable defined in the action part of the rule.

```
when {
 account: Account(balance : getBalance());
 contract: Contract(expiration : getExpirationDate());
}
then {
 int b = balance;
 System.out.println(contract + expiration);
 System.out.println(account + b);
}
```

This rule generates a list of all the available instances of each contract and its expiration date, and each account and its balance.

**Parent topic:** [Writing technical rules](#)

**Related information:**  
[ILOG Rule Language \(IRL\)](#)

# Writing conditions in IRL

You can write simple, complex, existence, collection, nonexistence, and evaluate conditions. To write conditions, use the ILOG® Rule Language syntax.

## About this task

You can define several types of conditions.

Table 1. Conditions you can write in IRL

Condition name	Condition role
Simple condition	Checks for every instance of a class and runs the rule for every instance.
Existence condition	Checks that at least one instance of a class is available. If it is the case, the existence condition runs the rule once.
Collection condition	Gathers all instances of a class and runs the rule once per collection found.
Nonexistence condition	Checks that no instance of a class is available. If it is the case, the nonexistence condition runs the rule once. This condition is the negation of simple conditions, existence conditions, and collection conditions.
Evaluate condition	Runs global tests on one or several bound variables. <div><b>Note:</b> You cannot put an evaluate condition at the beginning of the condition part of a rule. You must add one or more conditions that are not evaluate conditions before the evaluate condition.</div>

## Procedure

- To write a simple condition, use the following pattern: `variable: ClassName(<tests>);`

In this example, the condition matches for all the accounts in the working memory and the rule runs all instances that match this condition. You can reuse the Account variable in other conditions or actions.

```
when {
 account : Account();
}
```

- To write an existence condition, use the following pattern: `exists ClassName(<tests>);`

In this example, the condition checks that at least one account exists inside the working memory. If it is the case, the rule runs once.

```
when {
 exists Account();
}
then {
 ...
}
```

- To write a collection condition, use the following pattern: `collection: collect ClassName(<tests>);`

In this example, the condition gathers all the accounts in a collection and the rule runs once for all the accounts. You can reuse the *accounts* variable as a collection in other conditions or actions.

```
when {
 accounts : collect Account();
}
then {
 out.println(accounts.size());
 ...
}
```

- To write a nonexistence condition, use the following pattern: `not ClassName(<tests>);`

In this example, the condition checks that no report exists in the working memory. If it is the case, the rule runs once.

```
when {
 not Report();
}
then {
 ...
}
```

- To write an evaluate condition, use the following pattern: `evaluate (<test> [<test>]*);`

In this example, the condition gathers all the sales and purchases of a stock. If the result of a statistics computation of these two groups is less than two percent of the stock value, the condition runs the rule.

```
when {
 stock : Stock();
 purchases : collect Purchase(getStock()==stock);
 sales : collect Sales(getStock()==stock);
 evaluate (Statistics.variance(purchases, sales) < 0.02 *
 stock.getValue());
}
```

**Parent topic:** [Writing technical rules](#)

**Related tasks:**

[Defining a variable](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

**Related information:**

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Adding tests to a condition

You can write standard tests, tests on attribute values, tests on values that are returned by methods, and tests on values that belong to sets of values. You can also write conditions in join tests.

### About this task

You can add tests to conditions when you write technical rules in IRL.

### Procedure

- To add standard tests to a condition, use the following pattern:

```
[variable:] ClassName(<test> [; <test>]*);
not ClassName(<test> [; <test>]*);
collect ClassName(<test> [; <test>]*);
exists ClassName(<test> [; <test>]*);
```

- To write a test is based on the attribute value of an object that is bound in the current condition, use the following pattern:

```
attribute <operator> <value>
attribute booleanmethod <value>
attribute.method (<arguments>)
<operator> <value>
```

In this example, the rule applies to all accounts whose balance is over 2,000, with a currency in Euros (EUR) and a date greater than the current date.

```
when {
 account: Account(
 balance > 2000;
 currency equals Currency.EUR;
 date.compareTo(currentDate) > 0);
}
```

- To write a test that is based on the return value of the method that is called on an object that is bound in the current condition, use the following pattern:

```
non-void-method(<arguments>) <operator> <value>
non-void-method(<arguments>) booleanmethod <value>
// only in case of a method with a single argument, returning a Boolean:
attribute.method(<arguments>)<operator> <value>
```

In this example, the rule applies to all accounts whose balance is over 2,000, with a currency in Euros and a date greater than the current date. This test is the same as the test on attribute values, but it uses methods instead.

```
when {
 account: Account(
 getBalance()>2000;
 getCurrency() equals Currency.EUR;
 getDate().compareTo(currentDate) > 0);
}
```

- To write a test based on a value that belongs (in) or not (!in) to a set of values, use the following pattern:

```
<value> in|!in <group>
<value> in|!in {value1 [,value]*}
```

Where <group> is a collection or an array.

In this example, the rule applies for all accounts whose currency is in Euros or U.S. dollars (USD), and for all contracts whose status does not belong to the list of returned methods.

```
when {
```



```
account : Account(getCurrency() in { Currency.EUR,
 Currency.USD});
contract : Contract(status !in getBlockedStatusList());
}
```

- To write a condition with a join test, use the following pattern:

```
variable1: ClassName1(<tests>);
variable2: ClassName2(<tests using variable1>);
[variableP: ClassNameP(<tests using variable1, 2, ...>);]
```

**Note:** A join test uses one or more variables that were defined in previous conditions.

In this example, the rule applies for all purchases that are related to a contract, and for which an account exists in the same currency as the purchase.

```
when {
 purchase : Purchase(pcurrency : getCurrency());
 contract : Contract(getID() equals
 purchase.getContractID());
 Account(getContractID() equals contract.getID();
 getCurrency() == pcurrency);
}
```

**Parent topic:** [Writing technical rules](#)

**Related tasks:**

[Writing conditions in IRL](#)

## Using objects that are not in memory in a condition

When you write conditions, you can use objects that are not in memory by referring to relations from objects that are available.

### Procedure

To use objects that are not in memory when you write a condition, use the following pattern:

```
<condition> in <group>
<condition> from <instance>
```

In this pattern, *<group>* is an expression that returns an object array, a collection, or an enumeration, and *<instance>* is an object.

### Example

In this example, `contract` is a group of objects that is in the working memory. The condition can apply to the object `account.getCurrency` because of its relation with `contract.getAccounts`. With this condition, the rule executes all the objects.

```
when {
 contract : Contract();
 account : Account() in contract.getAccounts();
 currency : Currency() from account.getCurrency()
}
```

**Parent topic:** [Writing technical rules](#)

#### Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Writing a rule with an else action clause](#)

[Notifying the rule engine of object changes](#)

#### Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Writing actions in IRL

When you write rule actions, use the same syntax as a Java™ method body, except for the syntax for anonymous inner classes.

All bound variables that are defined in the rule, all ruleset parameters, all public static methods and attributes, and all functions are visible in the action scope. Some specific variables are always available in the action scope:

- `?context` is the current rule engine
- `?instance` is the current rule instance

As in a Java method body, all public classes of the current class loader are visible (through an import or with their fully qualified name).

### Example

```
then {
 System.out.println("The account " + account + " has no
 money in it");
 Report report = new Report(account);
 dispatcher.send(report); // dispatcher is a ruleset parameter
}
```

**Parent topic:** [Writing technical rules](#)

**Related tasks:**

[Writing conditions in IRL](#)

[Writing a rule with an else action clause](#)

## Writing a rule with an else action clause

When you write a rule with an else action clause, use the ILOG® Rule Language syntax.

### About this task

When the condition part of a rule finishes with an evaluation, the action part of the rule might include an else clause. The last evaluation within the condition part acts as a discriminator to choose between the then or the else execution branches. If the evaluation expression is true, the then part is executed. If it is false, the else part is executed.

If the rule ends with several evaluations, only the last one is used as the discriminator. The else clause is optional, and cannot be used when there is no evaluation at the end of the condition part.

### Procedure

To write a rule with an else action clause, use the following pattern:

```
rule rulename {
 when {
 // conditions:
 <condition>;
 [<condition>;]*
 evaluate (<then/else discrimination test>);
 }
 then {
 // "then" actions
 [<action>;]*
 }
 else {
 // "else" actions
 [<action>;]*
 }
}
```

### Example

In this example, if the evaluation expression is true, the rule executes the then action and charges a penalty. If the evaluation is false, the rule executes the else action and computes a bonus.

```
rule financial.rules.CheckBalance {
 when {
 account: Account();
 }
 evaluate (account.getBalance() < 0);
 then {
 account.chargePenalty();
 }
 else {
 account.computeBonus();
 }
};
```

**Parent topic:** [Writing technical rules](#)

#### Related tasks:

[Defining a variable](#)

[Writing conditions in IRL](#)

[Adding tests to a condition](#)

[Using objects that are not in memory in a condition](#)

[Notifying the rule engine of object changes](#)

#### Related information:

[Writing actions in IRL](#)

[Technical rules](#)

[ILOG Rule Language \(IRL\)](#)

## Notifying the rule engine of object changes

You can notify the rule engine of the availability of new objects, or when an object is updated or retracted.

### About this task

Notification of object changes depends on the execution mode:

- When the rule engine runs in RetePlus mode, it instantly recomputes the matching rules and reschedules an internal agenda.
- When the rule engine runs in sequential or Fastpath mode, it uses the new status of the objects when you enter a new rule task.

### Procedure

1. Use the following code phrase structure: `insert object;`

This statement notifies the RetePlus algorithm that a new object is available. Rules that depend on this kind of object run when the objects meet the rules conditions.

2. Use the following code structure: `insert ClassName(<constructor_arguments>) [{statement1; ... statement}] ;`

3. If the rule engine runs in RetePlus mode, use the following code phrase structure: `update object;`

This statement notifies the RetePlus algorithm that rules that depend on this object must be reevaluated. Some of them no longer run, and others start running.

4. If the rule engine runs in RetePlus mode, use the following code phrase structure: `retract object;`

This statement notifies the RetePlus algorithm that this object is no longer available. The rules that depend on this object no longer run.

### Results

The following code presents the global pattern to notify the rule engine of object changes:

```
insert object;
insert ClassName(<constructor_arguments>)
 [{statement1; ... statement}] ;
update object;
retract object;
```

### Example

In the then action part of this example, the RetePlus algorithm is notified that the new object penalty is available, and that the account object is no longer available. As a result, rules can run for penalty objects, but not for account objects anymore. In the else action part, the RetePlus algorithm is notified that the rules that depend on the account object must be reevaluated. As a result, some of these rules stop running and others start running.

```
rule financial.rules.CheckBalance {
 when {
 account: Account();
 evaluate (account.getBalance() < 0);
 }
 then { // compute a penalty
 Penalty penalty = new Penalty(account);
 insert penalty;
 retract account;
 }
 else { // compute a bonus
 account.computeBonus();
 update account;
 }
};
```

**Parent topic:** [Writing technical rules](#)

#### Related tasks:

[Adding tests to a condition](#)

#### Related information:

[Technical rules](#)



## Working with functions

You can create a function in a rule project to share code procedures across more than one element of a ruleset. You express a function in ILOG® Rule Language (IRL), and its code is evaluated when the ruleset runs.

### Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

### Creating functions

You write a function by using the ILOG Rule Language (IRL).

**Parent topic:** [Authoring business rules](#)

# Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG Rule Language (IRL) and its code is evaluated when the ruleset runs.

**Note:** Rule Designer does not refactor functions. If you rename, move, or delete a function, the references in other rule project artifacts are not updated.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)
{
 code
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the return keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of return that does not return a value:

```
return;
```

**Note:** A function is not required to declare in its throws statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

**Parent topic:** [Working with functions](#)

**Related tasks:**

[Creating functions](#)

**Related reference:**

[function](#)

**Related information:**

[Default constructor dynamic methods](#)



# Creating functions

You write a function by using the ILOG® Rule Language (IRL).

## About this task

A function is a procedural item in a ruleset. You write functions in the IRL by using the Function Editor in Rule Designer.

## Procedure

To write a function:

1. Right-click an item in the Rule Explorer, and then on the **New** menu, select **Function**.
2. In the New Function dialog box, give the following information:
  - **Function Name**
  - **Source Folder**
  - **Package Name**

Decision Server creates the function file and opens the Text view of the new function. The Rule Explorer tree displays the new function name.

3. Click **Finish**.
4. Click the **Edit Signature** link to open and edit the function signature. You can add parameters and change the function type.
5. Click the **Text** tab to return to the function text view.
6. Enter the function code in the **Content** area.

**Parent topic:** [Working with functions](#)

**Related concepts:**  
[Functions](#)

**Related reference:**  
[function](#)

**Related information:**  
[Default constructor dynamic methods](#)

# Applying a category filter

You can apply a category filter to specify which categories of elements can be used in action rules, decision tables, and decision trees.

## Before you begin

You must first define categories at the rule project level before you can apply them to rule artifacts.

## About this task

Apply a category filter to specify the categories to use on a specific rule artifact.

## Procedure

1. Open the action rule editor, the decision table editor, or the decision tree editor.
2. On the first page of the editor, in the Category Filter section, click **Edit**.
3. In the Category Filter dialog, select a category in the **All categories** field, and then click > > to move it to the **Selected categories** field. You can also double-click it to move it from one field to the other.
4. Click **OK**.

The categories you selected are now applied to the business rules and are listed in the Category Filter section of the editor.

**Parent topic:** [Authoring business rules](#)

**Related concepts:**  
[Categories](#)

**Related tasks:**  
[Defining and assigning categories](#)

# Correcting errors by using Quick Fix

You can use the Eclipse Quick Fix feature in the Intellirule Editor, the decision table editor, and the decision tree editor. Quick Fix messages offer suggestions to automatically correct detected errors.

## About this task

The Quick Fix feature in the Intellirule editor is supported only in the English version of the product.

In some cases, Quick Fix messages let you make the following changes:

- Replace one element by another
- Remove incorrect elements
- Declare new variables in the definitions part of the rule
- Add elements to the BOM

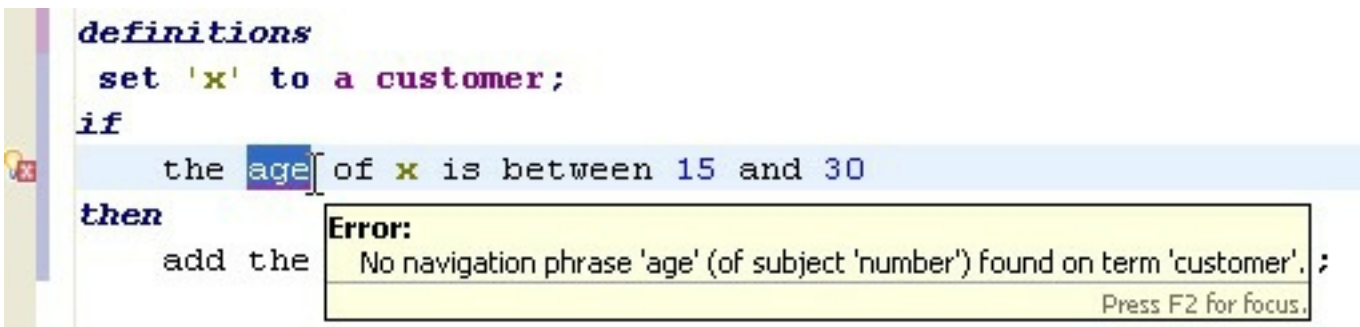
In the decision table editor and the decision tree editor, Quick Fix can also detect and let you add missing domain values.

To access the proposed quick fixes, click the light bulb icon  or place your cursor on the error and press Ctrl-1. The messages provide a list of possible corrections from which you can select the appropriate fix.

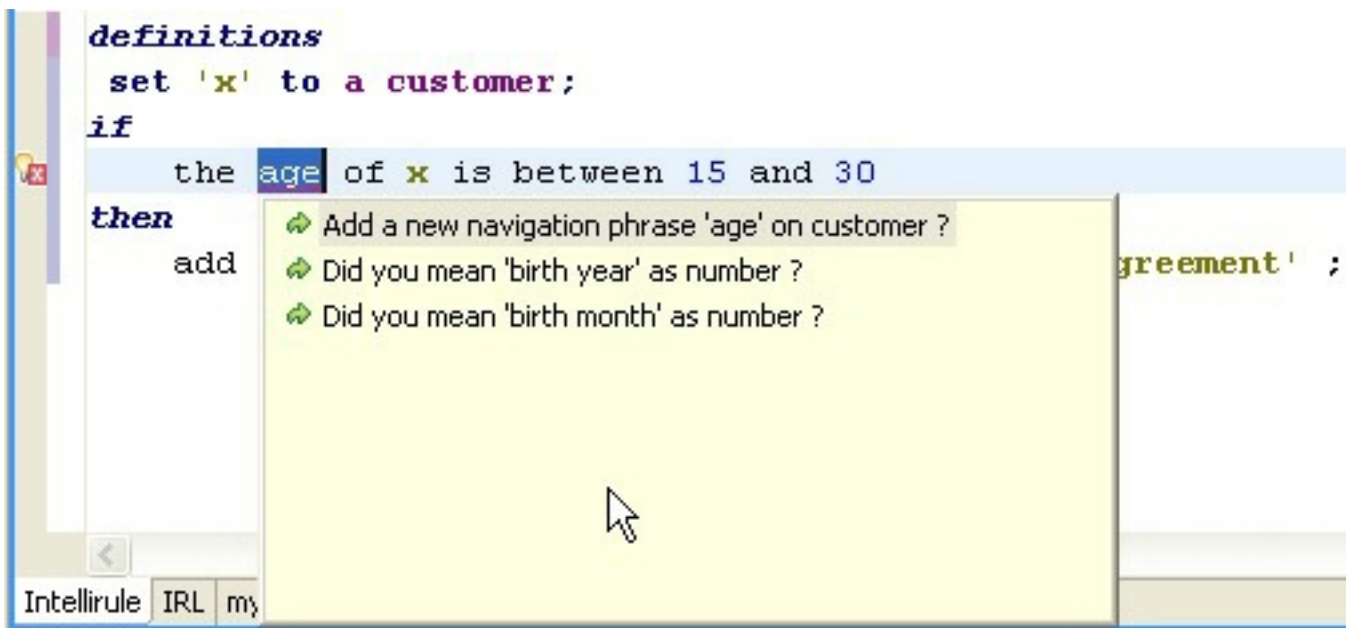
## Procedure

To auto-correct an error from the quick fix message:

1. Put your cursor on the error indicator to display the error messages.



2. Click the light bulb icon or move your cursor to the error in the rule, and then press Ctrl-1 to open the quick fix message.



In this example, the **customer** object does not have an **age** attribute, so it must be added to the BOM in order for it to be used in the rule.

**Note:** To access quick fixes, you must correct errors in the order in which they are shown.

3. From the list of suggestions, click the appropriate fix.

In this example, selecting **Add a new navigation phrase 'age' on customer** opens the Add New Phrase wizard. In the wizard, you can choose to create a navigation phrase or an action phrase for the new element when you add it to the BOM.

**New Phrase Wizard**

**Add New Phrase Wizard**  
Add a new Phrase to the selected BOM.

BOM Entry:

Class:

Type:

Attribute:

☒ Navigation Phrase  
i Create a navigation phrase corresponding to the selected attribute.

☐ Action Phrase  
i Create an action phrase corresponding to the selected attribute.

When you finish, the error is resolved and you can continue building your rule.

**Parent topic:** [Authoring business rules](#)

# Applying verbalization changes to business rules

If you change the verbalization of a business element used in a rule, you can refactor the business rules that use the business element to take your changes into account.

## About this task

When you change the verbalization of a business element used in a rule, Rule Designer prompts you to specify whether you want the rules that use the business element to be refactored to take your changes into account. If you decide not to do refactoring, the affected rules are not modified and syntax errors are reported in the Problems view.

If you want to change the subject of a phrase (referenced in curly braces in the navigation phrase template in the Member Verbalization area on the Member page of the BOM Editor) and benefit from the refactoring capability to update the rules that use them, you should make the change in the Edit Term dialog box, which you access by clicking **Edit the subject used in phrases** in the Member Verbalization area. If you make the change directly in the **Template** field, Rule Designer treats it as a change of the phrase template, which might cause an incorrect refactoring that leads to erroneous rules.

Refactoring is also executed after you modify ruleset parameters and variables that are used in business rules.

**Note:** Rule refactoring relies on business rules being compiled by the Eclipse build after modification. If you disable the Eclipse automatic build option, refactoring might not be executed when you modify the vocabulary.

Refactoring does not preserve rule formatting such as white spaces, tab spaces and new lines. Refactoring results in the text in a rule being regenerated, and this process might not preserve existing formatting.

Rules that have errors or that have not been saved might not have been refactored properly.

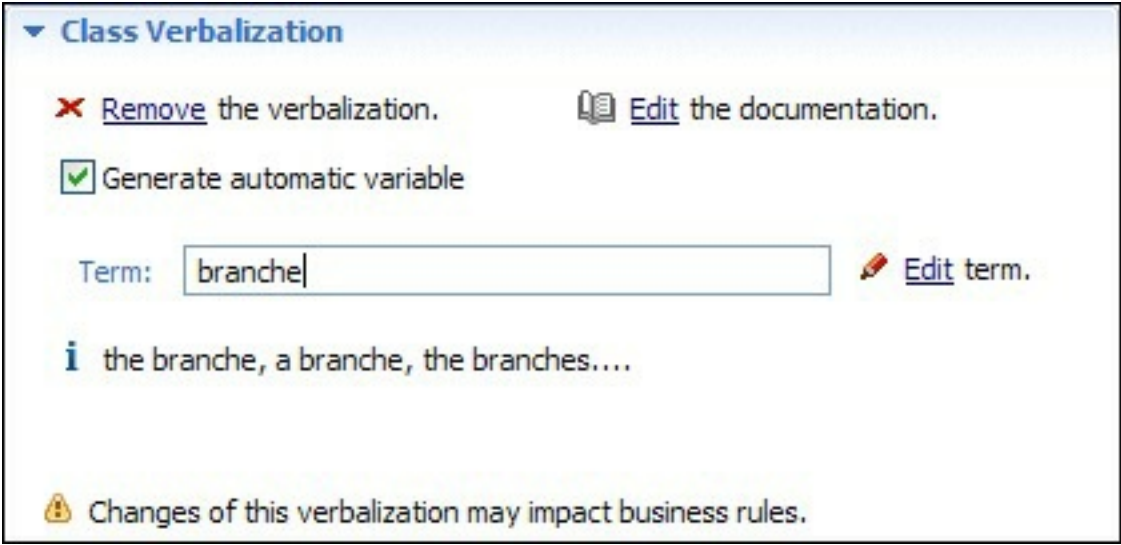
Rule refactoring is automatically activated when you save the BOM. You can also explicitly execute it from the Package page of the BOM Editor.

## Procedure

To execute rule refactoring:

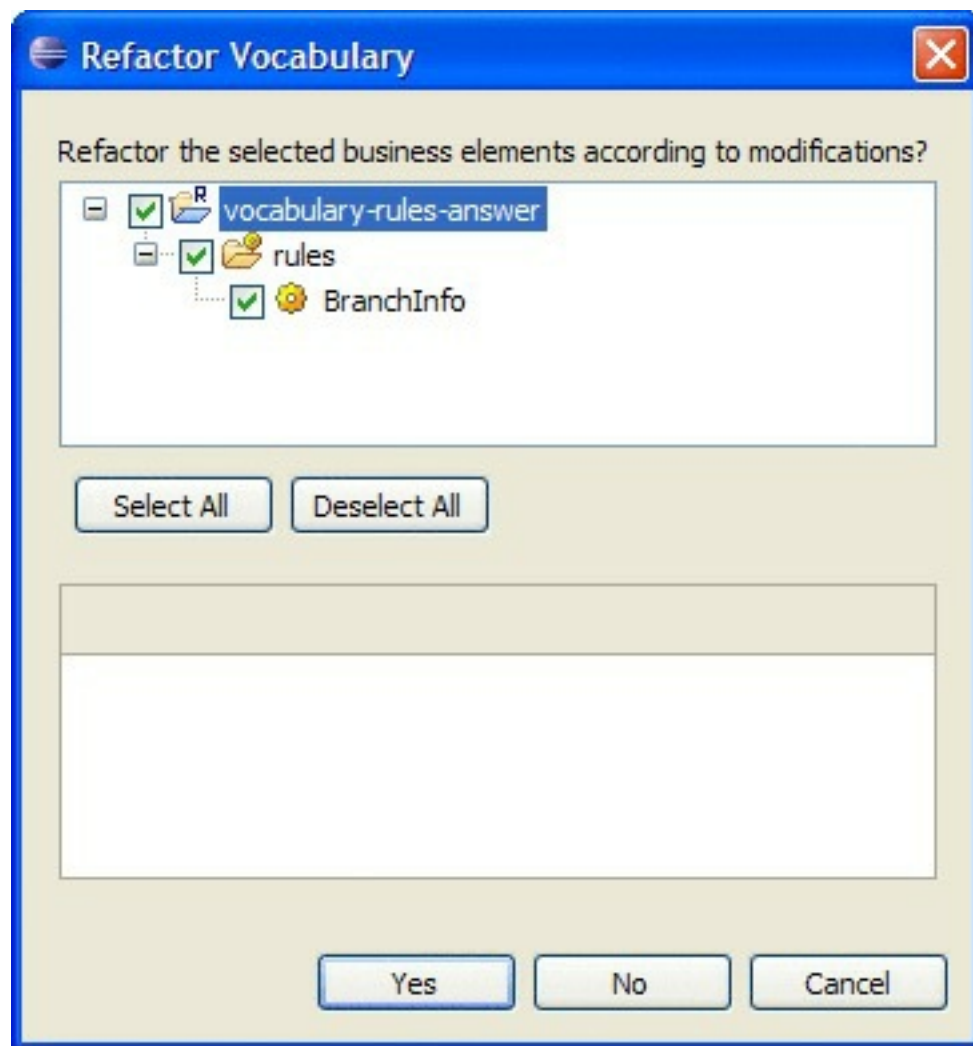
1. In the BOM Editor, modify the verbalization of a term or phrase that is used in a business rule.

The BOM Editor displays the following warning:



2. Save the BOM.

The Refactor Vocabulary dialog opens.



3. In the Refactor Vocabulary dialog, select the business rules you want to be refactored, and click **Yes**.

### Results

If you want to see refactoring history, in the **Project** menu click **Properties > Refactoring History**.

**Parent topic:** [Authoring business rules](#)

### Related information:

[Overview: Ways to express business rules](#)

[Updating the BOM when the XOM changes](#)

# Reviewing a rule project

Manage rule project items and run queries and reports using the features provided in Rule Designer.

## **Overview: Querying, analyzing and reporting**

Rule Designer provides functionality to search, query, and analyze rule projects, and generate reports on them.

## **Searching**

Using the Search feature of Rule Designer, you can search for rule artifacts, XOM elements, and BOM elements of a selected project. You can view the results in the Search view.

## **Querying**

You can filter your rules by creating and running queries in Rule Designer.

## **Analyzing**

Rule analysis relies on consistency checking and completeness analysis. Analyze your rule projects to find ambiguities, conflicts, and missing rules, and fix them. Using Rule Analysis view options, you can fine tune the process.

## **Generating Rule Project Statistics reports**

You can easily generate a report to get an overview of rule projects and of the artifacts that they contain or reference.

**Parent topic:** [Designing projects for rule authoring](#)

## Overview: Querying, analyzing and reporting

Rule Designer provides functionality to search, query, and analyze rule projects, and generate reports on them.

After you have created your rule project or if a rule project has been modified, you can review it by exploring and analyzing its contents.

In Rule Designer, you can do handle rule projects in the following way:

- Search through your project: see [Searching](#) and [Querying](#).
- Analyze the project rules using the Rule Analysis view: see [Analyzing](#).
- Generate reports using the three BIRT report templates provided in Rule Designer.
- Obtain statistics on the rule projects that are opened in your workspace, and on the artifacts that the rule projects contain: see [Generating Rule Project Statistics reports](#).

**Parent topic:** [Reviewing a rule project](#)

**Related tasks:**

[Creating a query](#)

[Running a query](#)

**Related information:**

[Analyzing a rule project](#)



# Searching

Using the Search feature of Rule Designer, you can search for rule artifacts, XOM elements, and BOM elements of a selected project. You can view the results in the Search view.

## [Searching rule project elements](#)

To search for packages, rule files, or rule project elements in a rule project, you can specify some scoping options such as case sensitiveness, project dependencies, and type of rule artifact.

## [Searching the execution object model](#)

You can search for XOM classes and packages or limit the search to the current XOM.

## [Searching the business object model](#)

You can search through BOMs or for a specific BOM.

**Parent topic:** [Reviewing a rule project](#)

## Searching rule project elements

To search for packages, rule files, or rule project elements in a rule project, you can specify some scoping options such as case sensitiveness, project dependencies, and type of rule artifact.

### [Searching for packages and rule files](#)

Using the search feature, you can search for packages and rule files throughout a rule project.

### [Searching for a project element](#)

Using the search feature, you can define a scope to search only some rule project elements.

**Parent topic:** [Searching](#)

**Related tasks:**

[Generating Rule Project Statistics reports](#)

[Finding rule dependencies](#)

# Searching for packages and rule files

Using the search feature, you can search for packages and rule files throughout a rule project.

## About this task

You can use the Search dialog to search packages and rule files for some rule artifacts that you specify by selecting the appropriate options.

## Procedure

1. In the **Search** menu, select **Search** to open the Search dialog, and then click the **Rule Search** tab to open the Rule Search page.

The Rule, BOM, and XOM searches are Eclipse-specific.

2. Specify patterns to find rules, tasks, functions, and rule packages in the **Search string** field.

Use \* for a multi-character wildcard and ? for a single-character wildcard. Using wildcards, you can specify all or part of the name of the rule, task, function, or rule package that you are searching for.

3. Select the **Case sensitive** box if you want your search to differentiate lowercase from uppercase characters in names.
4. Select **Include referenced projects** if you want to conduct the search on the current project and its dependencies.

If you do not select this option, the search is only done on the current project.

5. In the **Search for** area, select the type of element that you want to locate.
6. In the **Limit to** area, click **Name** if you want the search to be carried out on the names of rule, functions, tasks, and rule packages, or click **Body** for rules and functions.
7. Click **Search**.

## Results

The results are displayed in the Search view of the Rule Designer Eclipse window.

**Parent topic:** [Searching rule project elements](#)

### Related tasks:

[Searching for a project element](#)

### Related information:

[Searching](#)

## Searching for a project element

Using the search feature, you can define a scope to search only some rule project elements.

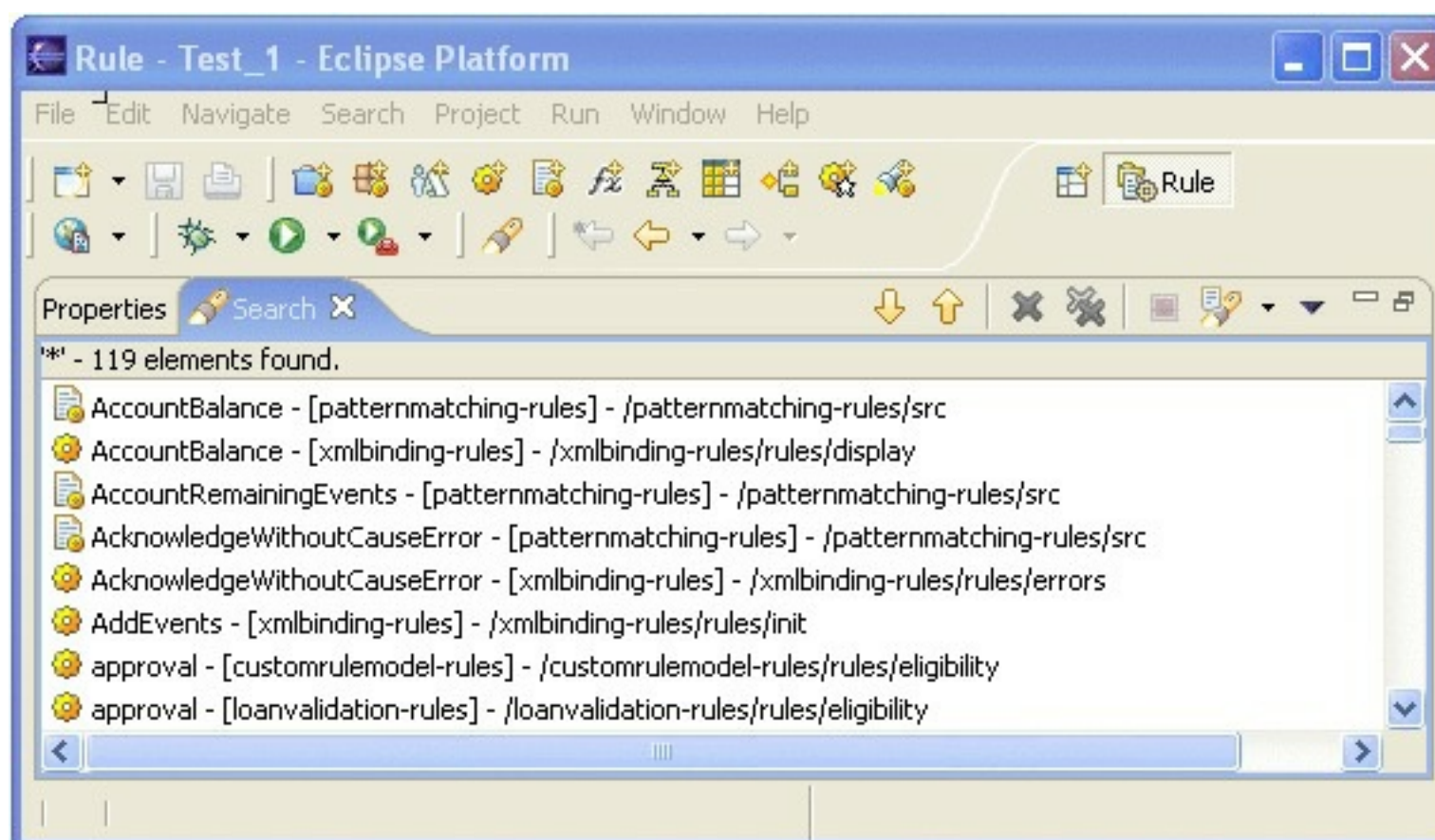
### About this task

In the Search dialog, you can define a scope to search specific rule project elements in a project.

### Procedure

1. In the Rule Explorer, select the rule project element.
2. Open the Rule Search page in the Search dialog and set up the search.
3. Select the scope in the Scope area.
  - To search in all the rule projects in the workspace, select **Workspace**.
  - To search on specific rule projects, select **Selected Resources**.
4. Click **Search**.

The results are displayed in the Search view.



For every match, the Search view shows the full name and the path to the element found.

5. To open the file where an element is defined, double-click the item in the Search view.

**Parent topic:** [Searching rule project elements](#)

### Related tasks:

[Searching for packages and rule files](#)

### Related information:

[Searching](#)

# Searching the execution object model

You can search for XOM classes and packages or limit the search to the current XOM.

## [Searching for XOM classes and packages](#)

In the Search dialog, you can search for XOM classes and packages in the workspace.

## [Searching the current XOM](#)

You can limit the search to the current XOM.

**Parent topic:** [Searching](#)

### **Related concepts:**

[Overview: BOM and execution object model \(XOM\)](#)

### **Related tasks:**

[Generating Rule Project Statistics reports](#)

### **Related information:**

[Searching the business object model](#)

[Searching rule project elements](#)

## Searching for XOM classes and packages

In the Search dialog, you can search for XOM classes and packages in the workspace.

### About this task

You can search for XOM classes and packages.

### Procedure

1. In the **Search** menu, select **Search**, and then click the **XOM Search** tab.
2. Specify patterns to find classes and packages in the **Search string** field.

Use \* to specify any string and ? to specify any character. Using wildcards, you can specify all or part of the name of the classes or packages that you want to find.

3. Select the **Case sensitive** box if you want your search to differentiate lowercase from uppercase characters in names.
4. Click **Search**.

The results are displayed in the Search view.

**Parent topic:** [Searching the execution object model](#)

### Related tasks:

[Searching the current XOM](#)

### Related information:

[Searching](#)

## Searching the current XOM

You can limit the search to the current XOM.

### About this task

If you do not want to search for all the XOM classes and packages in the workspace, you can search only the current XOM.

### Procedure

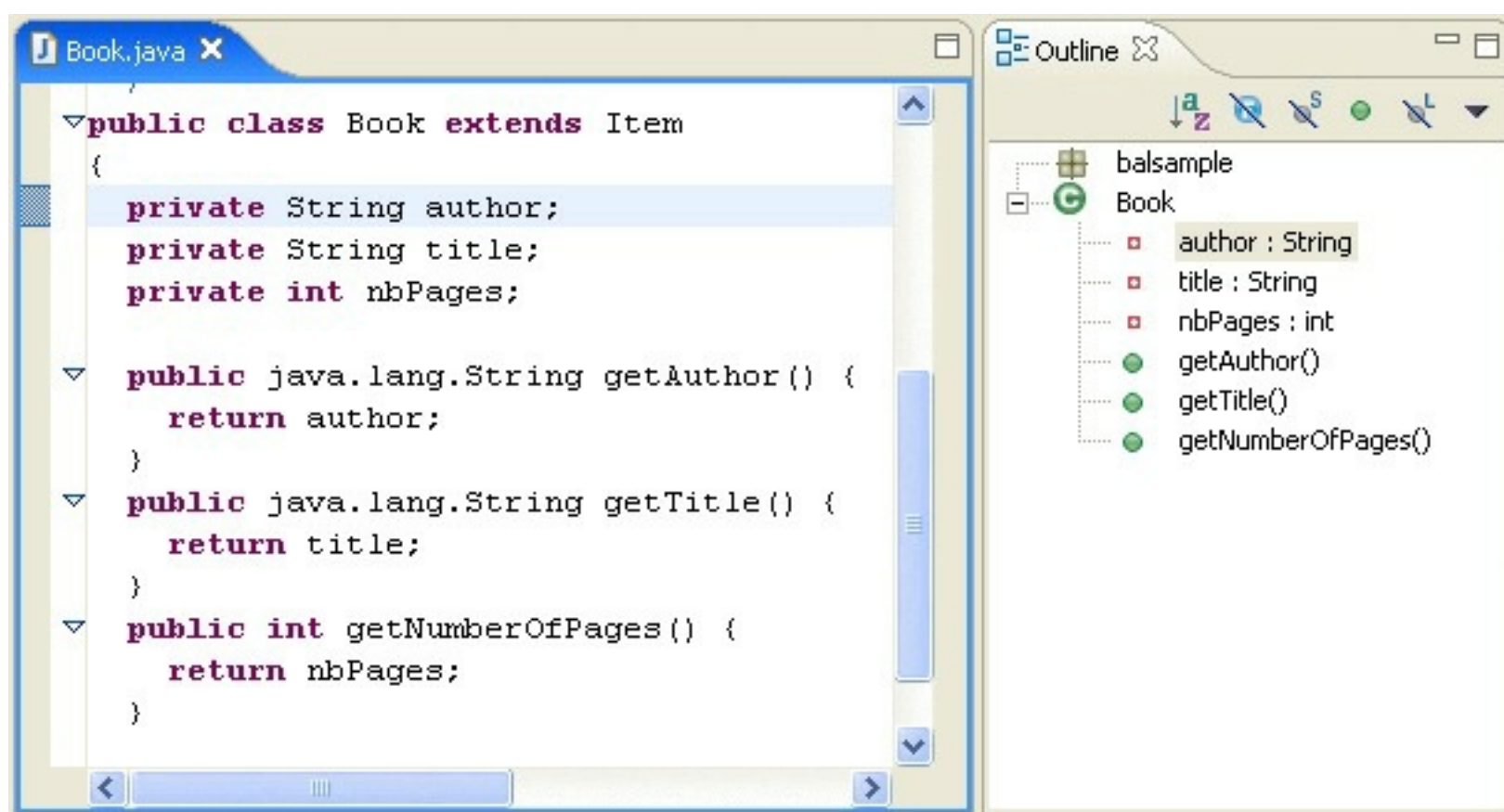
1. Highlight the rule project in the Rule Explorer.
2. Open the XOM Search page in the Search dialog and set up the search.
3. Select the scope in the Scope area:
  - To search on all XOMs in the workspace, select **Workspace**.
  - To search on XOMs for specific rule projects, select **Selected Rule Projects**.
4. Click **Search**.

The results display in the Search view.

For every match, the Search view shows the fully qualified name, the rule project, and the JAR file or class folder where the match was found.

5. To open the source file of a match for a class, double-click the match item in the Search view.

If the source is attached to the corresponding XOM reference, the source file is opened. Otherwise, an editor opens for you to set the source.



**Parent topic:** [Searching the execution object model](#)

### Related tasks:

[Searching for XOM classes and packages](#)

### Related information:

[Searching](#)

# Searching the business object model

You can search through BOMs or for a specific BOM.

## [Searching the BOM classes and packages](#)

You can search for BOM classes and packages or limit the search to the current BOM.

## [Searching the current BOM](#)

In the Search dialog, you can limit the search to the BOM of the selected rule project.

**Parent topic:** [Searching](#)

### **Related tasks:**

[Generating Rule Project Statistics reports](#)

### **Related information:**

[Searching the execution object model](#)

[Searching rule project elements](#)

[Business object model \(BOM\)](#)



## Searching the BOM classes and packages

You can search for BOM classes and packages or limit the search to the current BOM.

### About this task

You can search the BOM for classes and packages.

### Procedure

1. In the **Search** menu, select **Search**, and then click the **BOM Search** tab.
2. Specify patterns to find classes and packages in the **Search string** field.

Use \* to specify any string and ? to specify any character. Using wildcards, you can specify all or part of the names of the classes or packages that you want to find.

3. Select the **Case sensitive** box if you want your search to differentiate between lowercase and uppercase characters in names.
4. Click **Search**.

### Results

The results display in the Search view.

**Parent topic:** [Searching the business object model](#)

## Searching the current BOM

In the Search dialog, you can limit the search to the BOM of the selected rule project.

### About this task

You can search for classes and packages in the BOM of a specific rule project, or in all the rule projects contained in the workspace.

### Procedure

1. Highlight the rule project in the Rule Explorer.
2. Open the **BOM Search** page in the Search dialog and set up the search.
3. Select the scope in the Scope area:
  - To search on BOMs in all the rule projects in the workspace, select **Workspace**.
  - To search on BOMs for specific rule projects, select **Selected Resources**.
4. Click **Search**.

The results display in the Search view.

For every match, the Search view shows the name of the class or package, its container, and the BOM file where it is located.

5. To open the BOM editor corresponding to the match, double-click the match.
  - If the match is a class, the BOM editor opens on the Class tab with the selected class.
  - If the match is a package, the BOM editor opens on the Package tab with the selected package in the BOM tree.

**Parent topic:** [Searching the business object model](#)

# Querying

You can filter your rules by creating and running queries in Rule Designer.

## Queries

Using queries, you can extract rule artifacts or project elements by defining criteria as statements in the query condition part and display the found artifacts or, optionally, modify them with Do statements in the action part. Queries synchronize between Rule Designer and Decision Center, except for module-specific constructs.

## Creating a query

To create a query, you select the project folder, name the query, and select the appropriate project element, conditions, and actions in the Query Editor. To make the query synchronizable, avoid any module-specific construct.

## Running a query

You run your own queries or predefined queries on the rule project and, optionally, on referenced rule projects.

## Finding rule dependencies

You can run predefined queries to find rules that can enable or disable another rule or ruleflows that can select a rule.

## Customizing queries

To create business-specific query predicates that process customized rule project items and properties, you extend the query BOM files, and then integrate your extensions into Rule Designer and Decision Center.

## Automating queries with the Rule Designer API

You can run a query in four steps using the Rule Designer API.

**Parent topic:** [Reviewing a rule project](#)

# Queries

Using queries, you can extract rule artifacts or project elements by defining criteria as statements in the query condition part and display the found artifacts or, optionally, modify them with Do statements in the action part. Queries synchronize between Rule Designer and Decision Center, except for module-specific constructs.

## Introducing queries

Using queries, you can search through your workspace for rule artifacts and enforce actions on the artifacts found. You enter the search criteria in the condition part and, in the action part, the modification to carry out.

## Query conditions

In the condition part of a query, you specify the type of project element that you want to search and, if applicable, the “such that” filters to apply to the selected project elements.

## Query actions

A query action specifies the action to be taken if the conditions of a query are met.

## Query synchronization between Rule Designer and Decision Center

Queries work both in Rule Designer and Decision Center. When you synchronize a rule project between Rule Designer and Decision Center, queries in both modules are also synchronized, except for certain module-specific query constructs.

**Parent topic:** [Querying](#)

## Introducing queries

Using queries, you can search through your workspace for rule artifacts and enforce actions on the artifacts found. You enter the search criteria in the condition part and, in the action part, the modification to carry out.

For example, if you have a modification to apply to a certain type of rule, you can create a query that searches for the rules corresponding to this type and modifies them. You can also create queries to find which rules have been affected by any modification you made.

Queries can be categorized according to the rule part that they search:

- The condition part of a rule
- The action part of a rule
- Both the condition and action parts of a rule

You can use queries to evaluate the impact of changes to the object model or changes to rules.

You construct queries in the same way that you construct rules with a condition and an action. As the query condition, you define the criteria to match (see [Query conditions](#)) and as the action, you specify what to do. The default query action is to display the results in the Search window, but it is possible to have the query carry out other actions such as move the results of a query to another package (see [Query actions](#)).

You can use queries in Rule Designer and Decision Center. However, if you are synchronizing your rule projects between the two modules, you must be aware of some differences (see [Query synchronization between Rule Designer and Decision Center](#)).

**Parent topic:** [Queries](#)

**Related tasks:**

[Creating a query](#)

[Running a query](#)

**Related information:**

[Automating queries with the Rule Designer API](#)

## Query conditions

In the condition part of a query, you specify the type of project element that you want to search and, if applicable, the “such that” filters to apply to the selected project elements.

When you write a query condition, you start by selecting a project element type, then you refine the search by filtering on the project element properties, definition, or semantic content.

By default, queries are run on business rules, but you can also search on other project elements:

- Specific types of business rule:
  - Action rules
  - Decision tables
  - Decision trees (Decision trees are deprecated)
  - Your own rule class if you extended the rule model
- Other project elements:
  - Rule packages
  - Rule artifacts
  - Technical rules
  - Ruleflows
  - Templates (Templates are deprecated)
  - Functions
  - Variables
  - Variable sets

After you have selected the project element that you want to query, you can refine the query by adding a filter in the form of a `such that` statement, followed by an appropriate condition statement, or statements. The statements available for you to select depend on the type of project element that you have chosen and on the elements defined in the BOM.

### Examples

*Find all ruleflows  
such that each ruleflow is in package "accounts"*

Or:

*Find all business rules  
such that the effective date of each business rule is after 3/31/2016*

You can further refine queries by specifying more than one statement:

*Find all business rules  
such that the effective date of each business rule is after 3/31/2016  
and the effective date of each business rule is before 4/30/2016*

<b>Note:</b> Queries on rule artifacts such as “Find all rule artifacts” do not include functions.
----------------------------------------------------------------------------------------------------

#### Queries on properties

You can refine a query by filtering on one of the properties of the project elements being queried, such as name, status, or effective date.

#### Queries on definitions

You can refine a query by filtering on project element definitions.

#### Queries on semantic content

You can refine a query by filtering on the semantic content of a rule, that is on its behavior.

#### Queries on rule dependencies

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule affects the applicability of other rules and how a rule is affected by the execution of

other rules.

### **[Queries on categories](#)**

When you write a business rule, you can specify a category filter on the rule. The default category is Any.

**Parent topic:** [Queries](#)

**Related concepts:**

[Query actions](#)

**Related tasks:**

[Creating a query](#)

[Running a query](#)

**Related information:**

[Automating queries with the Rule Designer API](#)

[Query synchronization between Rule Designer and Decision Center](#)

## Queries on properties

You can refine a query by filtering on one of the properties of the project elements being queried, such as name, status, or effective date.

Name is a standard rule property. Status and effective date are extended properties, provided by the default extension model.

You can run this type of query against any type of project element. This type of query is useful for carrying out a rule audit.

### Example 1

The following query returns all business rules in your workspace with a status of new.

```
Find all business rules
such that the status of each business rule is new
```

### Example 2

The following query returns all variables in a package called accounts.

```
Find all variables
such that each variable is in package "accounts"
```

**Parent topic:** [Query conditions](#)

#### **Related concepts:**

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

[Queries on categories](#)



## Queries on definitions

You can refine a query by filtering on project element definitions.

Queries on definitions find rules that use or modify the value of a member or call a method. You can run this type of query against any type of rule. It is useful for identifying rules that are affected by policy change.

You can run queries on definitions against any type of rule. It is useful for identifying rules that are affected by policy change:

- The [uses the phrase](#) and [uses the phrase ... where](#) predicates find rules that call a method.

Although their naming is similar, the `uses the phrase` and `uses the phrase ... where` queries behave differently. The first query (`uses the phrase`) looks at the syntactic form of the rule, whereas the second query (`uses the phrase ... where`) looks at the behavior of a rule. That is to say that the `uses the phrase` query checks if the textual representation of a rule contains a method call; and the `uses the phrase ... where` query checks whether the execution of a rule calls the method and checks that the arguments of the method satisfies the constraints. The first query can return never-applicable rules, whereas the second query does not return never-applicable rules as they are not executed.

- The [uses the value of](#) and [modifies the value of](#) predicates find rules that use or modify the value of a member.
- The [is using the BOM class](#) and [is using the BOM member](#) predicates find rules that use a BOM class or BOM member.

Queries with the **is using BOM class** or the **is using BOM member** predicate work only if the workspace has been rebuilt since the last modification. If not, the results might not be accurate. The selected BOM class or member must be verbalized.

### uses the phrase

The predicate `uses the phrase <a method verbalization>` returns rules that call the specified method.

#### Example

The following query returns all the action rules that call the method that adds 'a number' to the corporate score in 'a report':

```
Find all action rules
such that each action rule uses the phrase [add 'a number' to the
corporate
score in 'a report']
```

### uses the phrase ... where

The predicate `uses the phrase ... where <a method verbalization with a constraint on method arguments>` returns rules that call a given method to which a constraint on arguments is applied.

This query filters on the project element definition **and** on the semantic content of a rule.

#### Example 1

The following query returns all the business rules that call the method that sets the credit score of 'a borrower' to 'a number', where this number equals 100:

```
Find all business rules
such that each business rule uses the phrase [set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100]
```

This query could return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

#### Example 2

The following query returns all the business rules that call the method that sets the credit score of 'a borrower' to 'a number', where this number is at least 100:

```
Find all business rules
such that each business rule uses the phrase [set the credit score of 'a
borrower'
to 'a number' , where 'a number' is at least 100]
```

This query could return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 200;
```

### Example 3

The following query returns all the business rules that call the method that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a number'
to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When a rule does not show the BOM elements involved in the constraint, this rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query could return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10 to
'the borrower';
```

### Example 4

The following query returns all the business rules that call the method that sets the loan amount range to <min> and <max>, with a maximum rate of <rate> where the value of the <min> and <max> parameters are lower than 10000, and the <rate> parameter is greater than 15%:

```
Find all business rules
such that each business rule uses the phrase [set the loan amount
range to 'a number' and
'another number', with a maximum rate of 'a number 3', where 'a number' is
lower than 10000,
and 'another number' is lower than 10000, and 'a number 3' is greater than 15
%]
```

You can select a predefined variable for each placeholder. Each variable has the type of the corresponding placeholder. If the type of the variable is String or Number, you can select up to 10 variables: 'a number', 'another number', 'a number 3', 'a number 4', and so on. For any other type, you can select up to three variables.

You can also select the predefined variables in the constraint part of the query and apply a constraint to them.

### uses the value of

The predicate uses the value of <an attribute, ruleset parameter, or variable> returns rules that use the values of certain members.

### Example

The following query returns all the business rules that use loan grade values.

```
Find all business rules
such that each business rule uses the value of the loan grade in 'a report'
```

### modifies the value of

The predicate modifies the value of <an attribute, ruleset parameter, or variable> returns rules that modify certain members.

### Example

The following query returns all the decision tables that modify the insurance rate of the loan report ruleset

parameter:

*Find all decision tables  
such that each decision table **modifies the value of** the insurance rate of  
'the loan report'*

### **is using the BOM class**

The predicate **is using the BOM class** returns the rules that reference a certain class either by using a binding to an instance of this class or by using the automatic variable linked to this class (Rule Designer only).

This query returns only business rule artifacts, it does not return technical rules.

#### **Example**

The following query returns all the business rules that use the class Borrower:

*Find all business rules  
such that each business rule **is using the BOM class** "loan.Borrower"*

You can also use the predicate **is not using BOM class**.

### **is using the BOM member**

Returns the rules that reference a certain BOM member (Rule Designer only).

This query returns only business rule artifacts, it does not return technical rules.

#### **Example**

The following query returns all business rules that read or modify the amount attribute of the Loan class:

*Find all business rules  
such that each business rule **is using the BOM Member** "loan.Loan.amount"*

You can also use the predicate **is not using BOM member**.

**Note:** You can use other queries such as **uses the value of**, **modifies the value of**, and **uses the phrase** to obtain more accurate results.

**Parent topic:** [Query conditions](#)

#### **Related concepts:**

[Queries on properties](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

[Queries on categories](#)

## Queries on semantic content

You can refine a query by filtering on the semantic content of a rule, that is on its behavior.

This type of query uses the phrases of the vocabulary that refer to members. It finds rules that could be applicable when certain conditions are true or could become true. It is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can run this type of query against the following project elements:

- **Action rules:** a type of business rule whose purpose is to do an action
- **Business rules:** any rule written using the Business Action Language (BAL)
- **Decision tables:** business rules written in table format, where each row corresponds to a single rule
- **Decision trees:** business rules written in the form of a tree
- **Technical rules:** rules written using the ILOG® Rule Language (IRL)
- **All rules:** any type of rule, including decision tables or decision trees

You can run a query on all rules of the current project, or all rules of the current project and its dependent projects.

### Note:

- When you create a semantic query, for dates as well as for strings, only the “is” and “is not” constructs are valid. The following constructs are not available: “is before”, “is after”, “is between”, “contains”, “starts with”, and “ends with”.
- Semantic queries run on the IRL code generated from the BAL rule. If a value translator is attached to a BOM member, the generated IRL uses this translation. Therefore, the semantic queries cannot find the BOM member because it is not shown in the IRL code.

You can use the following query predicates:

- [may apply when](#)
- [may become applicable when](#)
- [may lead to a state where](#)

### may apply when

The predicate `may apply when <a condition>` returns all rules whose condition part could meet the query condition, or the rules in which nothing in the rule condition contradicts the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

### Example

#### Rule 1:

If the score of the borrower is at least 10 then...

#### Rule 2:

If the age of the borrower is at least 21 then...

#### Rule 3:

If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

#### Query 1:

*Find all business rules  
such that each business rule **may apply when** [the score of the borrower is 20]*

#### Query 2:

*Find all business rules  
such that each business rule **may apply when** [the score of the borrower is 5]*

Query 1 returns Rule 1 and Rule 2. It returns Rule 1 because if the borrower's score is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the borrower's score could well be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

Query 2 returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to below 10. In Rule 2, there is nothing that specifically stops the rule from being applicable when the score is 5. In Rule 3, at least one of the conditions could apply and there is nothing in the other condition that negates the fact that the score could be 5.

### **may become applicable when**

The predicate `may become applicable when` <a condition> is a more specific query that returns only those rules in which the condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition. The query returns rules that would be applicable if the query condition was true. For example, if a change occurs in the elements of the rule condition, and the changed rule condition matches the query condition.

#### **Example**

##### **Rule 3:**

If the category of the customer is Platinum then...

##### **Rule 4:**

If the category of the customer is not Platinum then...

##### **Rule 6:**

If the age of the customer is at most 65 and the category of the customer is not Platinum...

##### **Query 1:**

*Find all business rules  
such that each business rule may become applicable when [the category of  
'a customer' is Gold]*

This query returns Rule 4 and Rule 6. It returns Rule 4 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 6 for the same reason. The additional condition relating to the customer's age does not contradict the condition in which the category can be Gold.

The query does not return Rule 3 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 5 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer's category is Gold.

##### **Query 2:**

*Find all business rules  
such that each business rule may become applicable when [the age of  
'a customer' is at least 21]*

This query does not return Rule 5 because it searches for rules that could "become" applicable when the customer's age is over 21. Rule 5 is applicable even if the customer's age is under 21. Therefore Rule 5 does not "become" applicable, but it "remains" applicable even if the customer's age changes from under 21 to over 21.

### **may lead to a state where**

The predicate `may lead to a state where` <a condition> returns the rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of the rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

#### **Example**

##### **Rule 7:**

If the age of the borrower is at least 25 then set the credit score of the borrower to 60

**Rule 8:**

If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

**Query:**

*Find* all business rules  
such that each business rule **may lead to a state where** [the credit score of the borrower is more than 50]

This query returns Rule 7 but not Rule 8 because, after the age of the borrower has been checked and the credit score set, only Rule 7 shows a result of over 50.

**Note:** You can also run a query on the action part of a rule using the `uses the` phrase predicate, see [uses the phrase](#).

**Parent topic:** [Query conditions](#)

**Related concepts:**

[Queries on properties](#)

[Queries on definitions](#)

[Queries on rule dependencies](#)

[Queries on categories](#)

## Queries on rule dependencies

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule affects the applicability of other rules and how a rule is affected by the execution of other rules.

You can use the following predicates:

- [may select](#)
- [may enable rule](#)
- [may disable](#)
- [may be enabled by](#)
- [may be disabled by](#)

### may select

The predicate `may select <a rule>` returns the ruleflows and rule tasks that might select a given rule.

When you define a ruleflow, you can specify the rules that make up a rule task either explicitly or by using a filtering function. The rule overriding mechanism is also likely to determine how rules are selected for execution at run time (for more information, see [Rule overriding](#)).

#### Example

The following query returns the ruleflows and rule tasks that might select "Rule 1".

```
Find all ruleflows
such that each ruleflow may select "Rule 1"
```

### may enable rule

The predicate `may enable rule <a rule>` returns rules that might make a given rule applicable.

#### Example

##### Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

##### Rule 2:

```
if the category of the customer is Bronze
then give Champagne to the shopping cart of the customer with message:
"Congratulations" ;
```

#### Query:

```
Find all business rules
such that each business rule may enable "Rule 2"
```

This query returns Rule 1 because this rule sets the category of the customer to Bronze and therefore makes the condition of Rule 2 valid.

### may disable

The predicate `may disable <a rule>` returns rules that might make a given rule inapplicable.

#### Example

##### Rule 1:

```
if the age of the customer is 18
then set the category of the customer to Copper ;
```

##### Rule 2:

```
if the category of the customer is Bronze
then give Champagne to the shopping cart of the customer with message:
"Congratulations" ;
```

##### Rule 3:



```
if the age of the customer equals 25 and the category of the customer is Gold
then set the category of the customer to Diamond ;
```

#### Query:

```
Find all business rules
such that each business rule may disable "Rule 2"
```

This query returns Rule 1 because if the category of the customer is set to Copper, it cannot be Bronze. The query does not return Rule 3 because the action is to set the category of the customer to Diamond and this rule can be executed only from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

#### may be enabled by

The predicate may be enabled by <a rule> returns rules that might be made applicable by a given rule.

#### Example

##### Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

##### Rule 2:

```
if the category of the customer is not Copper
then set the discount of the shopping cart of the customer to 5;
```

##### Rule 3:

```
if the age of the customer equals 25 and the category of the customer is
Bronze
then set the category of the customer to Diamond ;
```

#### Query:

```
Find all business rules
such that each business rule may be enabled by "Rule 1"
```

This query returns Rule 2 and Rule 3. It returns Rule 2 because if the category of the customer is not Copper, it could be Bronze. It returns Rule 3 because the condition specifies that the category of the customer should be Bronze.

#### may be disabled by

The predicate may be disabled by <a rule> returns rules that might be made inapplicable by a given rule.

#### Example

##### Rule 1:

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

##### Rule 2:

```
if the age of the customer equals 25 and the category of the customer is
Copper
then set the category of the customer to Gold
```

#### Query:

```
Find all business rules
such that each business rule may be disabled by "Rule 1"
```

This query returns Rule 2 because this rule is made inapplicable by Rule 1. In Rule 1, the category of a customer whose age is 25 is set to Bronze, therefore it cannot be Copper.



**Note:**

You can run these queries by right-clicking a rule in the Rule Explorer and clicking:

- **Find Rule Dependencies > Rules which may enable or disable this rule**
- **Find Rule Dependencies > Rules which may be enabled or disabled by this rule**
- **Find Rule Dependencies > Ruleflows which may select this rule**

**Parent topic:** [Query conditions](#)

**Related concepts:**

[Queries on properties](#)

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on categories](#)

## Queries on categories

When you write a business rule, you can specify a category filter on the rule. The default category is Any.

You can query rules that use a specific category or rules that use the default category Any.

**Attention:** In the localized versions of the product, the Any category is translated in the editor. However, to create a query on the rules containing the category Any, you must use the English word Any as the name of the category, instead of the translated name even if the query is in a different locale.

**Parent topic:** [Query conditions](#)

**Related concepts:**

[Queries on properties](#)

[Queries on definitions](#)

[Queries on semantic content](#)

[Queries on rule dependencies](#)

## Query actions

A query action specifies the action to be taken if the conditions of a query are met.

The default action of a query is to display the results of the query in the Search window. However, you can extend the actions of a query by adding a Do statement to it.

You can manage rules by specifying the following actions:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add a category to, or remove a category from, the displayed elements
- Set any of the properties (such as effective date, status, priority) of the displayed elements to a value that you specify

The actions available depend on the project element specified in the query. For example, there are more actions to choose from for a query on business rules than for a query on variables.

### Example 1

In the following example, you change the status of the business rules from new to validated.

```
Find all business rules
such that the status of each business rule is new
Do set the status of each business rule to validated
```

### Example 2

In the following example, you collect the decision tables that modify the value of 'the insurance rate' and group them into the 'insurance' package.

```
Find all decision tables
such that each decision table modifies the value of 'the insurance rate'
Do move each decision table to package "insurance"
```

### Example 3

In the following example, you delete the rules that could lead to a certain credit score and in which the expiration date is after a certain date.

```
Find all business rules
such that each business rule may lead to a state where [the credit score of
'a borrower' is less than 5]
and the expiration date of each business rule is after 6/21/2007
Do delete each business rule
```

**Parent topic:** [Queries](#)

**Related concepts:**

[Query conditions](#)

**Related tasks:**

[Creating a query](#)

[Running a query](#)

**Related information:**

[Automating queries with the Rule Designer API](#)

[Query synchronization between Rule Designer and Decision Center](#)

# Query synchronization between Rule Designer and Decision Center

Queries work both in Rule Designer and Decision Center. When you synchronize a rule project between Rule Designer and Decision Center, queries in both modules are also synchronized, except for certain module-specific query constructs.

You can create and run queries in either Rule Designer or Decision Center. However, you do not have to write the same query twice: when you synchronize rule projects between Rule Designer and Decision Center, queries in both modules are also synchronized.

The query BOM and vocabulary files are shared between Rule Designer and Decision Center. However, certain query constructs are not: if you write queries using module-specific constructs, these queries are synchronized but you cannot run them in the other module. If this is the case, you receive an error message. To make sure that a query written in one module can be run in the other, avoid using module specific constructs.

## Rule Designer query constructs

The following condition predicates are specific to Rule Designer:

- {this} is editable
- {this} is not editable
- {rule artifact} is using BOM class
- {rule artifact} is not using BOM class
- {rule artifact} is using BOM member
- {rule artifact} is not using BOM member
- the parent package of {a rule package}
- {rule package} is in package {rule package}
- {variable} is in package
- the rule query class

The following action predicate is specific to Rule Designer:

set the editable status of {this} to {0}

## Decision Center query constructs

The following condition predicates are specific to Decision Center:

- the creator of {this}
- the creation date of {this}
- the last modifier name of {this}
- the last modification date of {this}
- the view class
- the group of

The following action predicate is specific to Decision Center:

set the group of {project element} to {group}

**Parent topic:** [Queries](#)

### Related tasks:

[Creating a query](#)

[Running a query](#)

### Related information:

[Automating queries with the Rule Designer API](#)

[Synchronization commands in Designer](#)

## Creating a query

To create a query, you select the project folder, name the query, and select the appropriate project element, conditions, and actions in the Query Editor. To make the query synchronizable, avoid any module-specific construct.

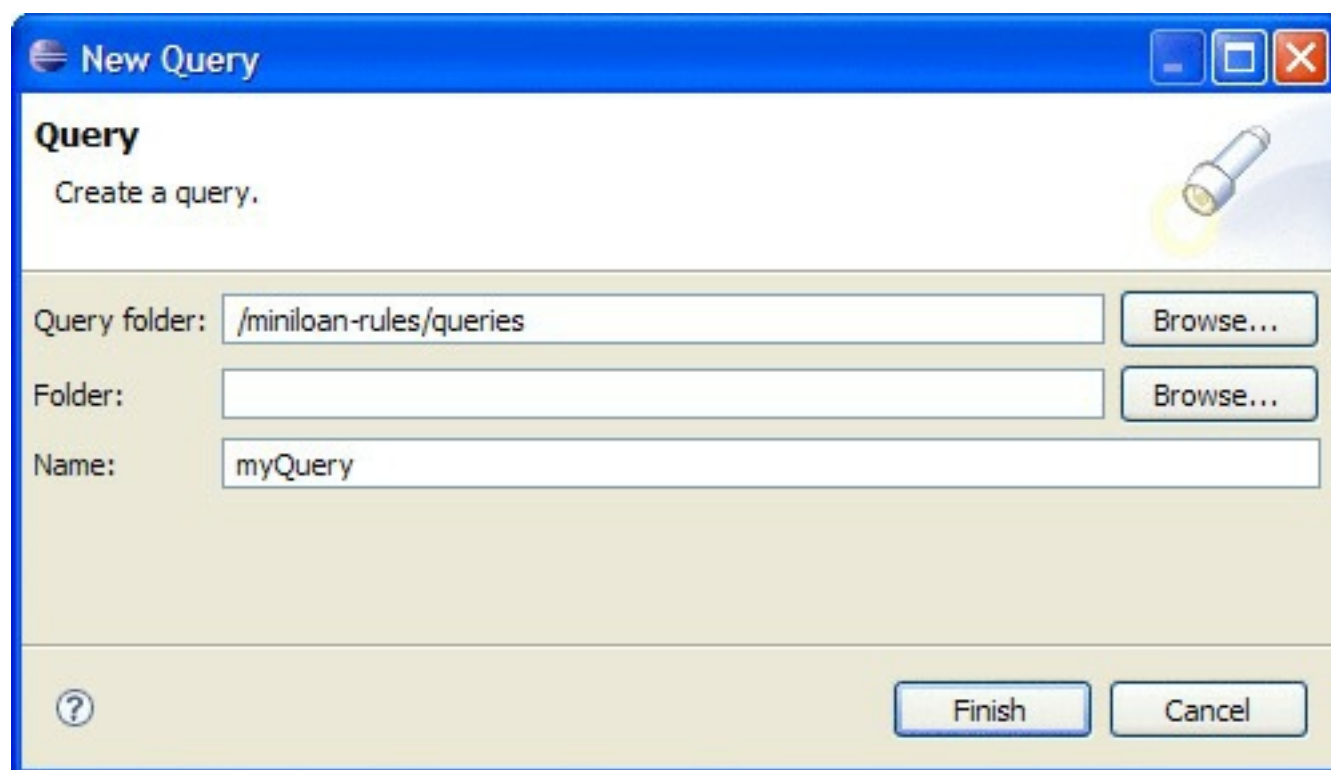
### About this task

Not all query constructs are available in Rule Designer and Decision Center. To make sure that a query can run in both Rule Designer and Decision Center, use only query constructs that are available in both. For more information, see [Query synchronization between Rule Designer and Decision Center](#).

### Procedure

To create a query:

1. In the Rule Explorer, right-click the queries project folder and click **New** > **Query** to open the New Query wizard.



Use the **Folder** field to specify a subfolder within the queries folder. Click **Browse** to select this subfolder or create it.

2. Type a name for the query in the **Name** field.
3. Click **Finish**.

You have now created an entry for the query in the Rule Explorer in the location that you specified. The Query Editor opens.

4. In the Content part of the Query Editor, define your query by selecting the required project elements, conditions, and actions.

You build a query in the same way as you build a rule. The project elements, conditions, and actions are displayed in drop-down lists when you click the building blocks. The conditions and actions available in the lists depend on the elements that you select (see [Query conditions](#) and [Query actions](#)).

5. If you want to enter a description or other information about the query, expand the Documentation section and type the required comments.
6. Save the query.

#### Note:

Some queries are based on a rule or a package. In this case, a tree view lets you select the rule or package to use in your query. After you have selected the rule or package, its fully qualified name is shown in the query, for example:

Find all rules  
such that each rule may enable "validation.loan.checkAmount"

To modify the query and select another rule, double-click the existing rule.

**Parent topic:** [Querying](#)

**Related concepts:**  
[Query conditions](#)

**Related information:**



# Running a query

You run your own queries or predefined queries on the rule project and, optionally, on referenced rule projects.

## About this task

After you have written a query, you can run it on the rule project that contains it and, optionally, on rule projects referenced from the current rule project.

## Procedure

To run a query:

1. In the Rule Explorer, double-click the query to open it.
2. If you want to run the query on the current rule project and on the elements of any rule project on which the current project depends, select **Include dependencies**.
3. In the Query Editor, click **Run query**.

All rules that meet the query search criteria are listed in the Search window.

**Note:** In some cases, when you run queries that return decision tables or ruleflows, the rows or rule tasks that match the search criteria are highlighted.

4. If you have created queries with actions, you can run them using the **Run query actions** option.

## Results

**Note:** Queries that include the **is using BOM member** or the **is using BOM class** predicate work only if the workspace has been rebuilt since the last modification. If it has not, the results might not be accurate.

**Parent topic:** [Querying](#)

**Related concepts:**

[Query conditions](#)

**Related information:**

[Automating queries with the Rule Designer API](#)

## Finding rule dependencies

You can run predefined queries to find rules that can enable or disable another rule or ruleflows that can select a rule.

### About this task

In Rule Designer, you can run predefined queries on rule dependencies. Rule Designer provides predefined queries so that you do not have to write the queries that enable you to search for rules that affect or are affected by other rules. You can also run a predefined query to find the ruleflows that select a given rule.

You can search for the following artifacts:

- Rules that are affected by the execution of other rules
- Rules that affect the execution of other rules
- Ruleflows that can select a given rule

For more information, see [Queries on rule dependencies](#).

### Procedure

To run queries on rule dependencies:

1. Right-click a rule in the Rule Explorer and click **Find Rule Dependencies**.
2. Click one of the following queries:
  - **Rules which may enable or disable this rule**
  - **Rules which may be enabled or disabled by this rule**
  - **Ruleflows which may select this rule**

### Results

The rules or ruleflows that meet the query search criteria are listed in the Search window.

**Parent topic:** [Querying](#)



## Customizing queries

To create business-specific query predicates that process customized rule project items and properties, you extend the query BOM files, and then integrate your extensions into Rule Designer and Decision Center.

### [Integrating query extensions into Rule Designer](#)

You use query extensions to create business-specific query predicates that process customized rule project items and properties. You integrate them into Rule Designer by creating a Rule Designer plug-in with a query model extension point.

**Parent topic:** [Querying](#)

# Integrating query extensions into Rule Designer

You use query extensions to create business-specific query predicates that process customized rule project items and properties. You integrate them into Rule Designer by creating a Rule Designer plug-in with a query model extension point.

## Before you begin

Before you integrate query extensions into Rule Designer, make sure you have all the necessary files:

- A BOM file
- A vocabulary file
- A BOM extender mapping file
- One or more extender classes

## About this task

You use query extensions to create business-specific query predicates that process customized rule project items and properties. To integrate them into Rule Designer, you create a Rule Designer plug-in with a query model extension point.

## Procedure

To integrate query extensions into Rule Designer:

1. Create a plug-in project that depends on the plug-in: `ilog.rules.studio.model.query`.
2. In this plug-in, create an extension for `ilog.rules.studio.model.query.queryModelExtension`.
3. Place your query extension files and classes in the `src` directory of the plug-in.
4. In the extension details, set the following attributes:

`bom_url` - the location of the BOM file

`voc_url` - the base name of the vocabulary files (without the locale suffix)

`bom_extender_mapping_file` - the location of the BOM extender mapping file

For example, if you create three extension files named `ext.bom`, `ext.voc`, and `ext.properties` and place them at the root of the `src` directory in the plug-in project, the extension details are as follows:

- **`bom_url`**: `ext.bom`
  - **`voc_url`**: `ext.voc`
  - **`bom_extender_mapping_file`**: `ext.properties`
5. Save the plug-in project.
  6. Deploy the plug-in.

## Results

Your query extensions are now available in your Rule Designer installation.

**Parent topic:** [Customizing queries](#)

## Related information:

[Business object model \(BOM\)](#)  
[Queries](#)

## Automating queries with the Rule Designer API

You can run a query in four steps using the Rule Designer API.

Services on top of the rule model API allow you to automate queries. You can automate queries by API with the entry point `IlrQueryService`.

You run a query in two or three steps, depending on whether you want to run the actions of the query or not:

1. Fetch the query service.
2. Initialize the query.
3. Run the query (this can be done repeatedly).
4. Run the actions of the query.

### Fetching the query service

Use the following to fetch the query service:

```
IlrQueryService queryService = (IlrQueryServiceImpl)
IlrStudioQueryPlugin.getQueryService(ruleProject);
```

If you want to set a BOM parser other than the default, use:

```
queryService.setBom(IlrBOMGetter.getInstance().getBOM());
```

### Initializing the query service

Depending on the elements you have, you can initialize the query service from:

- an `IlrQuery` project element
- a string containing the text of the query (for example, `Find all business rules such that...`)

### Initializing from `IlrQuery`

The following example shows how to initialize a query service with an `IlrQuery` element obtained from an `IlrProject`:

```
IlrQuery myQuery;
queryService.initQuery(myQuery, null);
```

To pass your own `IlrRuleQueryMatchCollector` as second argument:

```
IlrQuery myQuery;
IlrRuleQueryMatchCollector matchCollector;
queryService.initQuery(myQuery, matchCollector);
```

### Initializing with query text

The following example shows how to initialize a query service with query text:

```
String myQuery = "Find all business rules such that the name of each business
rule contains \"foobar\"";
boolean useLocalScope = false;
IProject currentProject;
queryService.initQuery(myQuery, useLocalScope, currentProject, null);
```

The second Boolean parameter determines whether the query scope is restricted to the specified project, or if it includes project dependencies.

As with an `IlrQuery`, you can also pass an `IlrRuleQueryMatchCollector` as last parameter:

```
queryService.initQuery(myQuery, useLocalScope, currentProject,
matchCollector);
```

## Running the query

To run a query, use:

```
List results = queryService.runQuery();
```

The contents of the list returned depends on the implementation of `IlrRuleQueryMatchCollector`.

Using the default provided, a list of objects of one of the following type is returned:

- `IlrQueryProjectElementWrapper`
- `IlrQueryParameterWrapper`
- `IlrQueryVariableWrapper`

## Running query actions

If the query has actions (checked using `IlrQueryService.queryHasActions()`), you can run the actions with:

```
queryService.runQueryActions(results)
```

The results are those returned by `runQuery()`. If needed, you can change the list before passing it to `runQueryActions()`.

**Parent topic:** [Querying](#)

**Related concepts:**

[Query conditions](#)

**Related tasks:**

[Creating a query](#)

[Running a query](#)

**Related information:**

[Queries](#)

# Analyzing

Rule analysis relies on consistency checking and completeness analysis. Analyze your rule projects to find ambiguities, conflicts, and missing rules, and fix them. Using Rule Analysis view options, you can fine tune the process.

## **Rule analysis**

Running in the background, rule analysis carries out completeness and consistency checking on a rule project to warn you of semantically redundant, conflicting or missing rules and let you fix them interactively.

## **Analyzing a rule project**

To check the integrity of a rule project, you generate an analysis report on it, you resolve the ambiguities and conflicts, you instantiate the rules that were found missing, and you check for remaining errors.

## **Fine tuning rule project analysis**

Use the advanced options of the Rule Analysis view to customize the process by filtering out some of the rules, by locking the results display of the latest analysis while another one is running, or by creating multiple views to run several concurrent cycles.

**Parent topic:** [Reviewing a rule project](#)

### **Related tasks:**

[Creating a query](#)

### **Related information:**

[Rule analysis](#)

# Rule analysis

Running in the background, rule analysis carries out completeness and consistency checking on a rule project to warn you of semantically redundant, conflicting or missing rules and let you fix them interactively.

## [Introducing rule analysis](#)

Rule analysis is a feature for checking that your rule project contains consistent rules with no conflict or redundancy.

## [Consistency checking](#)

The consistency analysis mechanism finds semantically conflicting rules such as rules that are never selected, rules that never apply, equivalent rules, redundant rules, conflicting rules, and conflicts between decision tables or decision trees.

## [\(Deprecated\) Completeness analysis](#)

The completeness analysis mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

**Parent topic:** [Analyzing](#)

### **Related information:**

[Analyzing a rule project](#)

[Fine tuning rule project analysis](#)

## Introducing rule analysis

Rule analysis is a feature for checking that your rule project contains consistent rules with no conflict or redundancy.

Using the rule analysis feature, you can analyze your rule project to check that it contains no ambiguities or conflicts, and that no rules are missing.

Rule analysis can be very useful at different stages in the elaboration of your rules. For instance, after you have created your rule project, you want to make sure that the ruleset is consistent and that there are no conflicting or redundant rules. You also want to check that every possible case is covered, for example, that there is a rule for every age range.

If your project has been modified, you can use rule analysis to make sure that no inconsistencies or gaps have been introduced.

### Rule Analysis view

The Rule Analysis view can run as a background task. It raises a warning when an ambiguity, a conflict, or a missing rule is identified. It lets you access the rules directly and fix the errors found and fill the gaps interactively.

If you do not want to analyze the whole rule project, the Rule Analysis view provides different options that filter the rules in your rule project.

**Note:** You can run an analysis of the rules only on classic rule projects that were built with the classic rule engine.

**Parent topic:** [Rule analysis](#)

**Related concepts:**

[Consistency checking](#)

[\(Deprecated\) Completeness analysis](#)

## Consistency checking

The consistency analysis mechanism finds semantically conflicting rules such as rules that are never selected, rules that never apply, equivalent rules, redundant rules, conflicting rules, and conflicts between decision tables or decision trees.

Consistency checking is a mechanism for checking whether rules do not contain semantically conflicting elements.

Ambiguities can be found either in a single rule or in a set of rules. For example:

- A single rule can contain self-contradictory conditions and therefore never apply.
- Two rules can apply to the same object and set a given attribute to two different values. Such rules are conflicting.

Consistency checking goes beyond syntax aspects to consider semantics as well. That is, how the rule behaves during execution. Using Rule Designer , you can choose which checks are carried out.

Consistency checks belong to one of the following categories:

- Checks that analyze an individual rule. These checks are activated when you build the rule and when you run the Consistency checking analysis:
  - [Rules that are never selected](#)
  - [Rules that never apply](#)
  - [Rules with range violation](#)
- Checks that analyze rules in relation to other rules. These checks are activated only when you run the Consistency checking analysis.
  - [Rules with equivalent conditions](#)
  - [Equivalent rules](#)
  - [Redundant rules](#)
  - [Conflicting and self-conflicting rules](#)

Consistency checking reports problems on rules:

- If there is a ruleflow in your rule project, the consistency checking mechanism reports problems on the rules that are included in a rule task and that might be selected at run time.

The mechanism compares only rules that could be in the same task. In the case of a rule task with dynamic selection filtering, the mechanism takes into account the rules that are potentially selected by this task. A rule is considered potentially selectable when it cannot be established that it definitely cannot be selected.

- If there is no ruleflow in your rule project, all the rules in the project are likely to be selected.

For more information, see [Runtime rule selection](#) and [Rule overriding](#).

**Note:** Consistency checking gives an indication of the consistency of your rules but cannot identify all potential problems. An empty consistency checking report is therefore not a guarantee that there are no problems in the analyzed rules.

### Rules that are never selected

Rules are reported as "never selected" when they are not part of a rule task and cannot be selected at run time. For more information, see [Runtime rule selection](#) and [Rule overriding](#).

### Rules that never apply

A rule is analyzed as never applicable when its conditions can never be met.

Typically, the syntax of such rules is correct but the rules contain common logic errors. For example:

- The wrong operator is used to combine condition statements, for example and instead of or: the category of the customer is Gold and the category of the customer is Platinum.
- Values are inverted, for example, in the following rule: the age of the customer is between 70 and 50.
- Values in the conditions are not within the permitted range.

### Rules with range violation

In order to reduce the risk of errors, some members can only be assigned values within a specified range. For example, the yearly interest rate on a loan might be limited to values between 0 and 10.

If a rule contains an action that tries to assign a value that is not within the permitted range, Rule Designer



displays a range violation error in the report and in the Rule Editor.

## Rules with equivalent conditions

The consistency checking mechanism also reports rules in which the condition parts have the same meaning, and where the action parts are different but not in conflict.

Rules with equivalent conditions do not necessarily represent an error situation, but they could be good candidates to be merged.

## Equivalent rules

Equivalent rules are reported when both their conditions and actions are the same.

In the following example, **Rule1** and **Rule2** are equivalent:

### Rule1

```
definitions
 set minDiscount to 5
 set ageDiscount to 10
if
 the age of the borrower is more than 65
then
 set the discount to minDiscount + ageDiscount
```

### Rule2

```
if
 the age of the borrower is at least 66
then
 set the discount to 15
```

Although the syntax of these two rules is different, rule analysis evaluates the numeric expressions and reports that the rules are equivalent. You can therefore delete one of them.

<b>Note:</b> Equivalent rules often arise between a decision table that you create and an existing rule.
----------------------------------------------------------------------------------------------------------

## Redundant rules

When two rules have the same actions, one of them becomes redundant when its conditions are included in the conditions of the other.

In the following example, the Else part of **Rule2** makes **Rule1** redundant:

### Rule1

```
if
 the category of the customer is Gold
then
 set the discount to 10
```

### Rule2

```
if
 the category of the customer is Platinum
then
 set the discount to 15
else
 set the discount to 10
```

Although **Rule1** is correct, it is redundant and can therefore be deleted.

<b>Note:</b> Redundant rules often arise between a decision table that you create and an existing rule.
---------------------------------------------------------------------------------------------------------

## Conflicting and self-conflicting rules

Rules **conflict** when the actions of two different rules set a different value for the same business term (member). Conflicts occur between two rules when the conditions are equivalent or cover the same values. For example, consider the following two rules:

## Rule1

```
if
 the loan report is approved
 and the amount of the loan is at least 300 000
then
 set the category of the borrower to Gold
```

## Rule2

```
if
 the age of the latest bankruptcy of the borrower is less than 1
 and the category of the borrower is not Platinum
then
 set the category of the borrower to No Category
```

**Rule1** and **Rule2** conflict when the following situation arises:

- The loan report is approved.
- The amount of the loan is 300 000 or more.
- The borrower has not had a bankruptcy in the last year.
- And the category is anything but Platinum.

In these specific circumstances, the rules sets the category of the borrower to different values. You can correct conflicting rules by changing the conditions, deleting one of the rules, or setting different priorities on the rules.

A rule is **self-conflicting** when two executions of a rule assign different values to the same member. For example, a rule that can apply twice on a given working memory (and ruleset parameters) and sets different values to a common attribute is self-conflicting. For example:

```
if
 the customer category is Gold
then
 set the discount of the cart to the bonus points of the customer
```

With this rule, if there are two customer objects with different bonus points in the working memory, the rule is executed twice and a conflict occurs because the two executions of the rule set different values to the discount of the cart.

## Decision table conflicts

To check decision tables and decision trees, you must enable the option **Include decision tables and decision trees in the inter-rule checks**.

When this option is enabled, you can check rules between and within decision tables or decision trees. The rule analyzer checks for conflicts, redundancies, and equivalences between the following elements:

- Two rows of the same decision table
- Two rows of two different decision tables
- Two leaves of the same decision tree
- Two leaves of two different decision trees
- A row of a decision table and a leaf of a decision tree
- A row of a decision table and an action rule
- A leaf of a decision tree and an action rule

When this option is disabled, these checks are not done on decision tables and decision trees, but only on action rules.

However, decision tables and decision trees are checked for never-applicable rules and rules with domain violation even if the option is disabled. This option does not impact overlapping and gap checks which are detected when you write the rules.

**Parent topic:** [Rule analysis](#)

**Related concepts:**[Introducing rule analysis](#)[\(Deprecated\) Completeness analysis](#)**Related information:**[Action rule editing errors and warnings](#)[Querying](#)

## (Deprecated) Completeness analysis

The completeness analysis mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

Completeness analysis is a mechanism for checking whether there are rules missing from a set of rules.

A ruleset is considered complete if for any possible case at least one rule applies.

You run completeness analysis for the following purposes:

- Check whether a ruleset is complete.
- Obtain a report of the missing rules.
- Add the missing rules to the ruleset.

For more information, see [Analyzing a rule project](#).

### Completeness mechanism

The completeness mechanism analyzes the condition part of each rule in a ruleset and detects whether there are cases where no rule applies.

To check that a ruleset is complete, the completeness mechanism proceeds as follows to identify the cases that are missing:

- If the rule project contains a ruleflow, the completeness mechanism analyzes the ruleflow task by task, to check that each rule task is complete.

For example, if a rule is missing in several rule tasks, the report lists the rule tasks where the rule is missing.

- If the rule project does not contain any ruleflow, the completeness mechanism takes into account all the rules in the project.

#### Note:

A ruleflow must contain at least one rule task.

### Example of completeness analysis

In the following example, the ruleset is composed of five rules that define the `customer` and that set the category of the `customer` to Silver, Platinum, or Gold.

#### Rule 1

```
if
 the value of the customer is at least 500 and the value of the
customer is less than 1000
then
 set the category of the customer to "Silver";
```

#### Rule 2

```
if
 the age of the customer is at least 65 and the value of the customer
is less than 500
then
 set the category of the customer to "Platinum";
```

#### Rule 3

```
if
 the value of the customer is at least 1000 and the value of the
customer is less than 1500
then
 set the category of the customer to "Gold";
```

#### Rule 4

```
if
 the value of the customer is at least 1500
then
```

```
 set the category of the customer to "Platinum";
```

#### Rule 5

```
if
 the value of the customer is less than 500 and the age of the customer
is less than 50
then
 set the category of the customer to "Platinum";
```

Ruleset completeness analysis detects that no rule applied in the case where the customer is between 50 and 65 and has a value of less than 500.

This missing case is presented as a rule skeleton that you can use to create the missing rule:

```
definitions
 set 'Customer1' to a customer ;
if
 the value of Customer1 is less than 500
 and the age of Customer1 is less than 65
 and the age of Customer1 is at least 50
then

<action>
```

#### Missing rules

If completeness analysis shows that your ruleset is not complete, it proposes additional rules to complete the ruleset. These new rules are skeleton rules. You must edit the missing rule to define the action to be done.

The completeness mechanism analyzes the conditions of each rule in the ruleset to define the set of conditions for the proposed rule. If the rules match a Customer object but test different attributes, for example age, value, and shopping cart, the set of conditions in the proposed rule takes into account the attributes tested in the analyzed rules. Therefore, the condition part in the suggested rule might seem complex and include many condition statements. It can be simplified to reflect a more natural way of expressing the rule conditions.

##### Note:

The analysis is based on the objects used in the existing rules and that match the working memory. For more information, see [Working memory](#).

In the following example, the suggested rule tests the value and the age attributes. You can simplify the rule by removing either condition.

#### Example

If completeness analysis proposes the following rule:

```
definitions
 set 'Customer1' to a customer ;
if
 the value of Customer1 is less than 500
 and the age of Customer1 is less than 65
 and the age of Customer1 is at least 50
then

<action>
```

you can remove the condition on the age of the customer, and keep only the condition on the value:

```
definitions
 set 'Customer1' to a customer ;
if
 the value of Customer1 is less than 500
then

<action>
```

If the ruleset contains only action rules, the proposed missing rules are verbalized. However, if the ruleset includes technical rules or decision tables, or if there is too little information on verbalization, the missing rules are presented in IRL.

**Note:**

The completeness mechanism might generate rules that overlap. You can modify the conditions of these rules.

## Performance

The performance of completeness analysis depends on several factors such as the number of rules in the ruleset, the number of rule tasks, and most importantly the number of missing rules.

To improve performance and reduce execution time of completeness analysis, set a limit to the number of missing rules to propose. For example, limit the results to ten missing rules. Once you have integrated the first ten missing rules, run completeness analysis again to identify other missing rules.

**Parent topic:** [Rule analysis](#)

**Related concepts:**

[Introducing rule analysis](#)

[Consistency checking](#)

# Analyzing a rule project

To check the integrity of a rule project, you generate an analysis report on it, you resolve the ambiguities and conflicts, you instantiate the rules that were found missing, and you check for remaining errors.

## Analyzing a rule project

You run a rule analysis cycle to check the integrity of a rule project.

## Resolving ambiguities and conflicts

From the Rule Analysis view, you can directly access the rules that contain ambiguities and conflicts.

## Instantiating the missing rules

When a missing rule is reported in the Rule Analysis view, you can create the missing rule and define a package for the new rule.

## Checking for remaining errors

You look for new or remaining errors.

**Parent topic:** [Analyzing](#)

**Related information:**

[Rule analysis](#)

[Fine tuning rule project analysis](#)

# Analyzing a rule project

You run a rule analysis cycle to check the integrity of a rule project.


## About this task

When you run a rule analysis cycle to check the integrity of a rule project, you go through the following steps:

1. Generate an analysis report by selecting the rule project and the extraction options.
2. Resolve each ambiguity and conflict listed.
3. Instantiate the missing rules found, if applicable: select a package, name the rule, and edit the conditions and actions as necessary.
4. Rerun the analysis cycle to check for any remaining errors and repeat the process as appropriate.

## Procedure

To run a report on a rule project:

1. In the Rule Explorer, right-click a rule project and click **Open Rule Analysis**.
2. Click  **Select Rule Project and Extraction Type** in the Rule Analysis view toolbar.
3. Select an extraction option in the Type of Extraction section.

The default option is **Use all rules contained in project**. If you want to streamline the results of the analysis, see [Filtering the results](#).

4. Click **OK** to close the Select Rule Project and Extraction Type dialog.
5. In the Analysis Options section, select the type of checks that you want to activate.

To check decision trees and decision tables, select the **Include decision trees and decision tables in the inter-rule checks** check box.

For more information on the checks available, see [Consistency checking](#) and [\(Deprecated\) Completeness analysis](#).

### Note:

If you start the rule analysis and modify the checks selected before the analysis is complete, the changes are not taken into account until the next time you run the analysis.

6. To search for missing rules in your rule project, select the **Completeness** check box.

By default, the analysis stops looking for missing rules when it has found 100 missing rules. If you have a large rule project and if you want to make sure that the analysis does not take too long, you can modify this number and start searching for 10 missing rules, for example.

7. To start the analysis, click  **Run Rule Analysis** in the view toolbar.

The report is shown in the Results section of the Rule Analysis view.

### Note:

- You can also analyze rules by clicking the **Analyze rule project** link in the Rule Project Map. This link activates the analysis and refreshes the results.
- You can cancel the analysis by clicking the **Cancel Operation** icon in the Progress view (**Window > Show view > General > Progress**).

**Parent topic:** [Analyzing a rule project](#)

## Related tasks:

[Resolving ambiguities and conflicts](#)

[Instantiating the missing rules](#)

[Checking for remaining errors](#)



## Resolving ambiguities and conflicts


From the Rule Analysis view, you can directly access the rules that contain ambiguities and conflicts.

### About this task

The Rule Analysis view displays the ambiguities or conflicts found in your rule project and provides a link to the rules that contain these potential errors.

### Procedure

To resolve ambiguities or conflicts:

1. Expand each ambiguity or click  **Expand all** on the top of the **Results** section.
2. To open the rule, click the link that points to it.
3. Modify the rules as appropriate and save your rule project.

Rule analysis reports ambiguities or conflicts. You can choose to fix them, or to ignore them if you think they are not relevant. For more information on the type of ambiguities and how to resolve them, see [Consistency checking](#).

<b>Note:</b> If you close or delete your rule project, the results are no longer shown in the Rule Analysis view.
-------------------------------------------------------------------------------------------------------------------

**Parent topic:** [Analyzing a rule project](#)

### Related tasks:

[Analyzing a rule project](#)  
[Instantiating the missing rules](#)  
[Checking for remaining errors](#)

# Instantiating the missing rules

When a missing rule is reported in the Rule Analysis view, you can create the missing rule and define a package for the new rule.

## About this task

The Rule Analysis view identifies rules that are potentially missing in the rule project and enables you to instantiate the missing rules. For more information, see [\(Deprecated\) Completeness analysis](#).

## Procedure

To instantiate missing rules:

1. In the Results section of the Rule Analysis view, expand the reported missing rule.  
If the rule project contains a ruleflow, rule analysis lists all the rule tasks in which the rule is missing.
2. Click **Instantiate missing rule**.
3. In the New Action Rule dialog, in the **Package** field, select the package to which you want to add the new rule.
4. In the **Name** field, enter a name for the new rule and click **OK**.  
The new rule is created in the package that you selected.
5. Edit the new rule to define the action, and modify the condition statements if appropriate.

<b>Note:</b> If the ruleset contains a ruleflow, make sure the missing rule is included in all the rule tasks that report the rule as missing.
------------------------------------------------------------------------------------------------------------------------------------------------

**Parent topic:** [Analyzing a rule project](#)

## Related tasks:

[Analyzing a rule project](#)

[Resolving ambiguities and conflicts](#)

[Checking for remaining errors](#)

## Checking for remaining errors


You look for new or remaining errors.

### About this task

After you have resolved the ambiguities or conflicts and included the missing rules, check whether there are any new errors or remaining errors.

### Procedure

To rerun the analysis:

1. To start the analysis, click  **Run Rule Analysis** in the view toolbar.
2. Check if there are new or remaining errors and repeat the correction process as necessary.

**Parent topic:** [Analyzing a rule project](#)

### Related tasks:

[Analyzing a rule project](#)

[Resolving ambiguities and conflicts](#)

[Instantiating the missing rules](#)

## Fine tuning rule project analysis

Use the advanced options of the Rule Analysis view to customize the process by filtering out some of the rules, by locking the results display of the latest analysis while another one is running, or by creating multiple views to run several concurrent cycles.

### Filtering the results

You can use the Rule Analysis view to filter your rule project using extractors.

### Displaying the results

You can lock the results display and choose when you want to view the new results.

### Creating multiple views

You can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

**Parent topic:** [Analyzing](#)

#### **Related tasks:**

[Creating a query](#)

#### **Related information:**

[Analyzing a rule project](#)

[Rule analysis](#)

## Filtering the results

You can use the Rule Analysis view to filter your rule project using extractors.

### About this task


If you are working on a large rule project for which a full analysis might take some time to complete, you can use the advanced options of Rule Analysis view to filter the results.

- You can select an extractor to retain a subset of rules and filter out the rest of the project.
- In automatic run mode, you can lock the display to the results of a given analysis cycle while another is running in the background.
- To work on different analyses concurrently, you can create multiple views.

When you analyze your rule project, you might not want results for all the rules in your rule project. If you prefer to analyze only a subset of your rules, you can use the Rule Analysis view to filter your rule project using extractors. Extractors are often based on the queries in your rule project.

### Procedure

To select a filter:

1. Click  **Select Rule Project and Extraction Type** in the view toolbar.
2. In the Select Rule Project and Extraction Type dialog, select **Use an extractor to filter rules contained in project** and click **Browse**.
3. In the Extractor Selection dialog, select an extractor from the list, and then click **OK**.

**Parent topic:** [Fine tuning rule project analysis](#)

### Related tasks:


[Displaying the results](#)

[Creating multiple views](#)

## Displaying the results

You can lock the results display and choose when you want to view the new results.

### About this task


If you activate the  **Set Automatic Run** option, the Rule Analysis view updates the results every time you save your rule project. However, you can lock the results display and choose when you want to view the new results. This is useful to start fixing the existing errors while another analysis is running.

#### Note:

Do not use the **Set Automatic Run** option during the development of a rule project.

### Procedure

To lock the results display:

1. In the view toolbar, click  **Lock Results Display**.  
This option is available only in Set Automatic Run mode.
2. To view the latest results when the analysis is complete, click the **New results available: refresh** link.

**Parent topic:** [Fine tuning rule project analysis](#)

#### Related tasks:

[Filtering the results](#)

[Creating multiple views](#)

# Creating multiple views

You can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

## About this task

While an analysis is running, you cannot change the rule project or select a different extractor. If you modify the selected checks, they are be taken into account until the next time you run the analysis. However, you can create as many views as you like and use each view to run a rule analysis on different projects and with different options.

It is useful to create multiple views for the following reasons:

- You can reduce the time needed to complete the analysis by activating only the checks in which you are interested.
- You can activate different checks for the same rule project. For example, you can have a view displaying consistency problems while another view displays missing rules found in the rule project.
- You can run rule analysis on different rule projects at the same time.
- You can compare the number of errors between two analysis reports.

### Note:

- If you have different views set on Automatic Run for the same project, each view is updated each time you save your rule project.
- In some cases, you might get more errors even when you select fewer checks. Some checks, such as equivalent rules and redundant rules, are aggregated.

## Procedure

To create multiple views:

1. In the view toolbar, click **Add New Rule Analysis View**.
2. To select a rule project and a type of extraction, click **Select Rule Project and Extraction Type** as described in [Analyzing a rule project](#).

**Parent topic:** [Fine tuning rule project analysis](#)

### Related tasks:

[Filtering the results](#)

[Displaying the results](#)

# Generating Rule Project Statistics reports

You can easily generate a report to get an overview of rule projects and of the artifacts that they contain or reference.

## About this task

The Rule Project Statistics report provides an overview of the rule projects that are opened in your workspace and indicates their scale.

The report shows statistics for each rule project and for the rule artifacts that it contains, for example:

- The number of ruleset parameters and categories defined in the rule project
- The number of BOM classes, methods and attributes
- The minimum and maximum numbers of rows in decision tables
- The number of rule tasks in a ruleflow, and the execution mode they use
- The number of phrases and business terms in the vocabulary

## Procedure

To generate a Rule Project Statistics report:

1. Click **File > Export**.
2. In the Export wizard, select **Rule Designer > Rule Project Statistics**, and then click **Next**.
3. To save the report in the workspace directory, keep the default location and click **Finish**.

The report opens as an HTML page in your browser.

**Tip:** Alternatively, you can right-click a rule project, and click **Export > Rule Project Statistics**.

## Results

Rule Designer creates a <rule-statistics-report>.xml file in the workspace directory, along with a rs4j-export.xsl file to view the report in HTML. The generated report uses the locale of Rule Designer.

- The .xml file contains the statistics.
- The .xsl file is based on the eXtensible Stylesheet Language standard and converts the XML information into a series of HTML tables. You can customize the .xsl file or replace it with your own .xsl file to fit specific needs. If you modify this file manually, it cannot be regenerated by exporting the rule project statistics in Rule Designer. To override the modified file with a new generated one, you must first delete the file.

**Note:** If you want to open the report after closing it, go to the directory where you saved the report and open the <rule-statistics-report>.xml file in a web browser.

**Parent topic:** [Reviewing a rule project](#)

## Related information:

[Developing rule projects](#)

[Searching](#)



# Building and running rules

In Operational Decision Manager, building rules refers to the compilation of your rule projects and, depending on your level of automation, can include creating the corresponding rulesets or RuleApps. Running rules refers to the use of launch configurations to run and debug rulesets in Rule Designer. Rule execution refers specifically to topics related to the engine.

## **Overview: Building and running rules**

Building rules involves the compilation of your rule projects and can include creating the corresponding rulesets or RuleApps.

## **Building rule projects interactively**

Each time you modify a rule and save it, Rule Designer builds the rule project by compiling its content.

## **Automating builds**

You can automate the build of your projects in a continuous deployment pipeline by building your rule and Java projects as rule applications.

## **Running and debugging**

Rule Designer provides a running and debugging environment to test the execution of a ruleset.

## **Executing rulesets in the decision engine**

In Rule Designer, you can extract and package your rules for execution on different platforms.

## **Optimizing execution**

You can improve performance and scalability of your application through the way Decision Server integrates with Java™, and settings for execution modes, algorithms, and various associated configuration files.

## Overview: Building and running rules

Building rules involves the compilation of your rule projects and can include creating the corresponding rulesets or RuleApps.

Rule Designer proposes launch configurations to build a rule project into a ruleset archive, which you can run and debug in Rule Designer using a basic predefined engine.

You can use Rule Designer to build rule projects into RuleApps, which contain sets of ruleset archives that you can deploy to Rule Execution Server.

You can also automate the build process using Rule Designer in headless mode.

**Parent topic:** [Building and running rules](#)

## Building rule projects interactively

Each time you modify a rule and save it, Rule Designer builds the rule project by compiling its content.

### About this task

During the build process, Rule Designer detects problems and reports them in the Problems view:

- Errors: Syntax or structural defects
- Warnings: Indicates awkward or deprecated syntax in the valid parts of the rules

Some errors result from the way in which rules interact in a ruleset archive, which is the standard artifact that a rule engine can read. For example, if a project contains two rule with the same name, the resulting ruleset would contain an error because there cannot be two rules with the same name in a single ruleset. A typical case of this is when a rule generated from a decision table conflicts with a BAL rule of the same name.

### Procedure

1. To build a rule project, click **Save**.

When you modify a rule and save it, Rule Designer builds the project.

2. To rebuild a rule project, click **Project > Clean**.

### Results

After a successful build, you can use a launch configuration to generate a ruleset archive, and run and debug the ruleset.

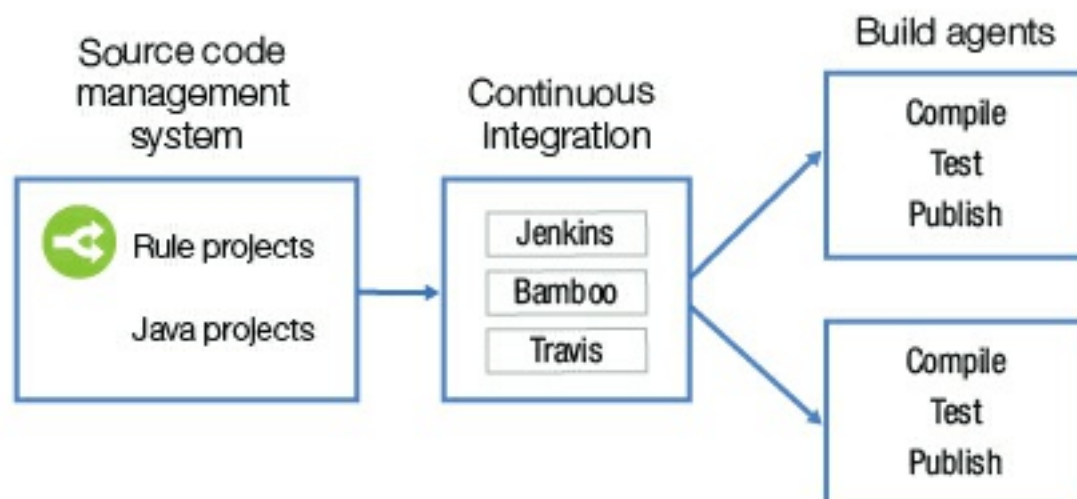
**Parent topic:** [Building and running rules](#)

## Automating builds

You can automate the build of your projects in a continuous deployment pipeline by building your rule and Java projects as rule applications.

The following diagram shows an example of a continuous deployment pipeline architecture that uses Jenkins, Bamboo and Travis for continuous integration, and that describes the tasks that build agents do.

Figure 1. Diagram of a continuous deployment architecture



You can use Rule Designer in headless mode to automate the compilation tasks in build agents.

### Adding global variables

Describes how to add a global variable.

### Rule Designer build automation tool

Rule Designer provides a tool to automate builds and ruleset extraction from the command line. The build automation tool starts Rule Designer in headless mode, that is, with no user interface for the development environment.

**Parent topic:** [Building and running rules](#)

## Adding global variables

Describes how to add a global variable.

### About this task

You can use the `ilog.rules.studio.javascript` plug-in in conjunction with Eclipse to automate tasks on a preconfigured workspace or a folder containing the rule projects.

The following example demonstrates how to add a global variable named `out` to replace a fully qualified call to:

```
java.lang.System.out.println(<text message>);
```

### Procedure

To add global variables:

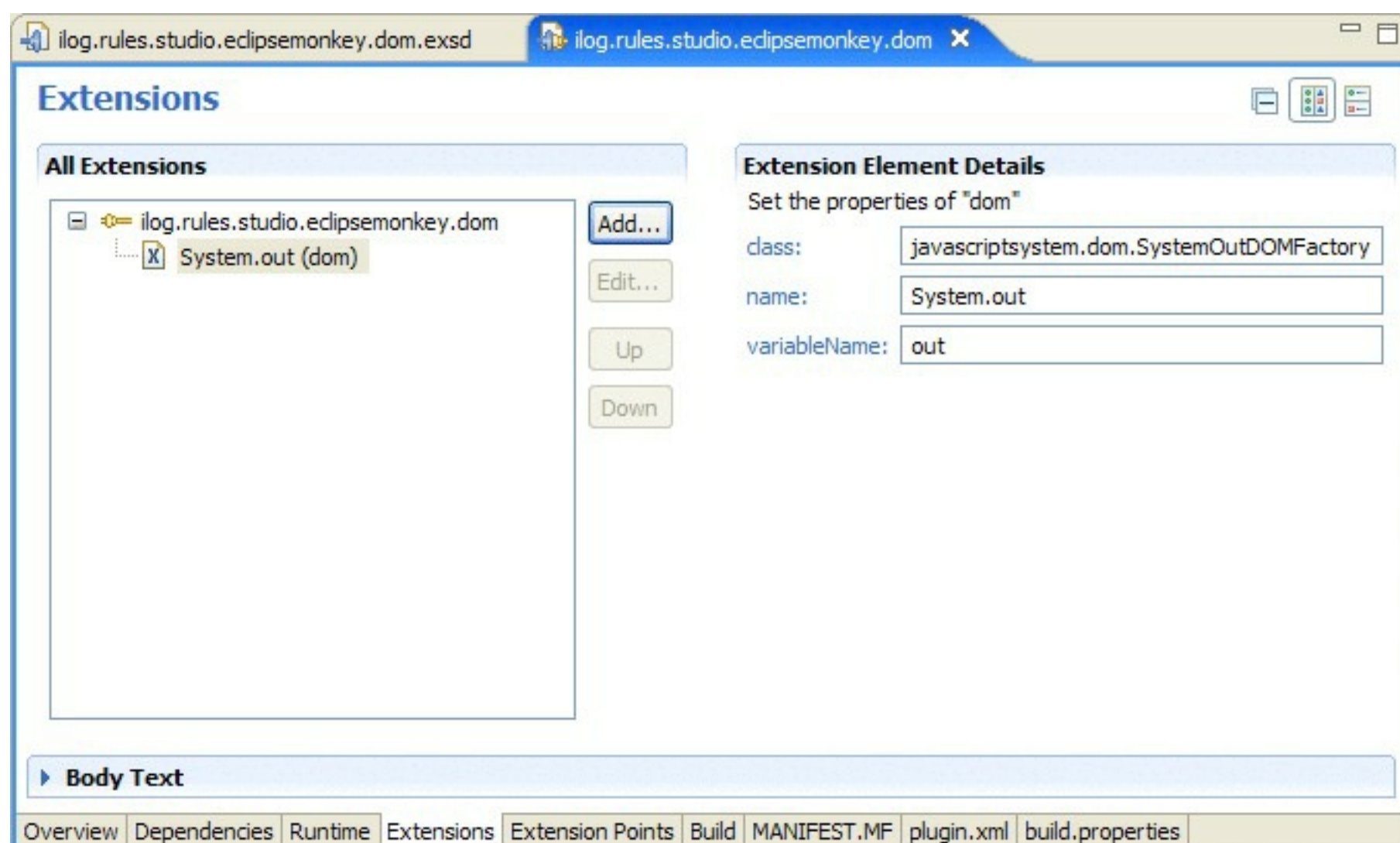
1. On the **File** menu, click **New** > **Project**.
2. Under the **Plug-in Development** category, select **Plug-in Project** and then click **Next**.
3. In the **project name** field, enter a name and then click **Next**.
4. Click **Finish**.
5. Click **Yes** to open the Plug-in Development perspective.
6. Click the Dependencies tab, and under **Required Plug-ins** click **Add**.
7. Add `ilog.rules.studio.eclipsemonkey` to the dependencies.
8. Click the Extension Points tab and then click **Add**.
9. In the **Extension Point ID** and **Extension Point Name** fields, enter `ilog.rules.studio.eclipsemonkey.dom` and then click **Finish**.
10. Click the `plugin.xml` tab and then enter the following text:

```
<extension point="ilog.rules.studio.eclipsemonkey.dom">
 <dom
 class="javascriptsystem.dom.SystemOutDOMFactory"
 name="System.out"
 variableName="out"/>
</extension>
```

The extension declares the factory that creates the Java™ instance on which the JavaScript method calls are forwarded when used with the `out` variable.

### Results

The plug-in Extensions tab now includes the new plug-in project:



You must now add the implementation of the factory. Factories contain a single method named `getDOMroot`.

This method creates the Java instance of the class to which the method calls are forwarded.

The following SystemOutDOMFactory example returns System.out static instances:

```
package javascriptsystem.dom;

import ilog.rules.studio.eclipsemonkey.dom.IMonkeyDOMFactory;

public class SystemOutDOMFactory implements IMonkeyDOMFactory {
 public Object getDOMroot() {
 return System.out;
 }
}
```

By adding this global variable, you simplify each print command to the standard output. You can also create a more complex object that provides methods such as formatting text messages with parameters, as shown in the following example:

```
function main() {
 // instead of java.lang.System.out.println("my message");
 out.println("my message");
}
```

**Parent topic:** [Automating builds](#)

## Rule Designer build automation tool

Rule Designer provides a tool to automate builds and ruleset extraction from the command line. The build automation tool starts Rule Designer in headless mode, that is, with no user interface for the development environment.

For IBM® AIX® platforms, only headless Eclipse mode is supported. Installing Rule Designer on IBM AIX is useful if you want to run automated builds or ruleset extraction from rule projects that are maintained on a platform where Rule Designer runs in full mode. You can also automate routine operations such as building rule projects or running queries. For advanced automation tasks, you can customize the automation using JavaScript.

In the same build, you import the projects into the workspace, apply an extension model, build the projects, and generate ruleset archives.

You can use the build automation tool to test builds in Rule Designer, and then run automated builds from the command line. The `ilog.rules.studio.automation` plug-in contains the `ilog.rules.studio.automation.builder` application.

You use arguments in the command line to specify the different steps that are involved in the build. For example, you use arguments to define the paths to the projects to import and to specify whether to remove existing projects from the workspace before building. All arguments are optional, but you must at least specify the name of the rule project to build.

### Note:

You can use an extra argument on the command line: the name of the decision operation to use when you generate the ruleset archive for a decision service.

The list of rule projects to build is mandatory except when you use the help option **-h**.

## Running the build automation tool from the command line

The build automation tool supports command-line arguments, which you can use to set up automated builds.

In the command line, you specify the arguments and the rule projects to build. You can choose the decision operation that you want to use for generating the ruleset archive. All arguments and extractors are optional, but you must at least specify the name of the rule projects to build.

To filter the rules when you generate a ruleset archive, use a question mark (?) between the rule project name and the extractor name or decision operation. For example, if you are using a decision service, write `<rule_project_name>?<operation_name>`.

To run the build automation tool for different operating systems, write command lines similar to the following examples:

### Windows

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
```

You can then add the arguments, the rule projects to build, and the extractor to filter the rules. If your rule project name contains spaces, use double quotation marks as separators.

If you are using a decision service, use the following syntax:

```
-importPath "<path_to_xom_project>;<path_to_rule_project>"
<rule_project_name>?<operation_name>
```

Here is an example of a complete command line with no filter:

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
-importPath
"my_workspace/rulesetextraction/rulesetextraction_xom;my_workspace/rulesetextraction/rulesetextraction_rules"
-cleanBuild
-buildOptions BUILD_CHECK_IRL|BUILD_INCLUDES_RULE_VALIDATION
rulesetextraction_rules
```

### UNIX

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
```

You can then add the arguments, the rule projects to build, and the extractor to filter the rules. If your rule project name contains spaces, use double quotation marks as separators.

If you are using a decision service, use the following syntax:

```
-importPath <path_to_xom_project>:<path_to_rule_project>
<rule_project_name>?<operation_name>
```

Here is an example of a complete command line with no filter:

```
eclipse -application ilog.rules.studio.automation.builder
-noSplash
-nl en_US
-importPath
"my_workspace/rulesetextraction/rulesetextraction_xom:my_workspace/rulesetextraction/rulesetextraction_rules"
-cleanBuild
-buildOptions BUILD_CHECK_IRL|BUILD_INCLUDES_RULE_VALIDATION
rulesetextraction_rules
```

Command-line arguments for the automated build

The following table lists the command line arguments that you can use to define your automated build.

Table 1. Build command line arguments

Argument name	Description	Default
-? (-h)	Prints usage information and then exits.	-
-autoRefresh	Refreshes the workspace automatically if Eclipse determines that the contents of the workspace are changed.  For example, set this argument to true when modify the contents of the workspace outside Eclipse.	false
-brdx <file>	Specifies the path to the .brdx extension model file to use.  If you specify the .brmx file, you must also specify a .brdx file.	-
-brmx <file>	Specifies the path to the .brmx extension model file to use.  If you specify the .brdx file, you must also specify a .brmx file.	-
-buildOptions	Specifies the build options to use, separated by the pipe ( ) character. Build options that you do not specify are set to false, except for <b>BUILD_GENERATE_IRL</b> , which is always set to true.  You can use the following build options:	<b>BUILD_GENERATE_IRL</b> is always set to true



	<ul style="list-style-type: none"> <li>• <b>BUILD_GENERATE_IRL:</b> Generates IRL during build</li> <li>• <b>BUILD_CHECK_IRL:</b> Performs IRL checks during build</li> <li>• <b>BUILD_INCLUDES_B2X_VALIDATION:</b> Performs BOM to XOM checks during build</li> <li>• <b>BUILD_INACTIVE_RULES:</b> Builds rules for which the property "active" is false</li> <li>• <b>BUILD_INCLUDES_RULE_VALIDATION:</b> Performs rule analysis during build</li> </ul> <p>The build is incremental.</p>	
<b>-cleanBuild</b>	Cleans the results of prior workspace builds before it rebuilds.	false
<b>-clearWorkspace</b>	Removes all projects from the workspace before it imports projects. Project contents remain intact.	false
<b>-failFast</b>	Stops the build if any rule project contains errors.	false
<b>-importPath &lt;val&gt;</b>	<p>Specifies the list of projects to import into the workspace before building. The list uses the <code>File.separator</code> system property.</p> <p>The paths of projects to import are separated by a semicolon (;) for Windows and by a colon (:) for UNIX.</p>	-
<b>-nl &lt;language&gt;</b>	Sets the language for the applications, for example, <b>-nl en_US</b> for American English.	-
<b>-noSplash</b>	Bypasses the splash screen on startup.	-
<b>-output</b> or <b>-o &lt;val&gt;</b>	<p>Specifies the name of the output directory to store the generated ruleset archive.</p> <p>You must specify an existing folder. If you do not specify an output directory, the ruleset archive is stored in the output directory of the rule project (<code>./output</code>). If you specify a relative path, the path is relative to the parent rule project that is being processed.</p>	-
<b>-refresh</b>	Refreshes the entire workspace prior to building projects.	false

For more information about the arguments supported by the [llrAutomator](#) class, refer to [llrBuildCommandLineArguments](#).

**Parent topic:** [Automating builds](#)



# Running and debugging

Rule Designer provides a running and debugging environment to test the execution of a ruleset.

## [Running and debugging decision operations](#)

To run a decision operation in standard or debugging mode, you must create a launch configuration by selecting the project in which the operation is stored.

## [Debugging rulesets](#)

Rule Designer provides a set of tools to inspect the execution of the rules and the state of the engine, and set breakpoints on classes, objects, rules, decision tables, and ruleflows. The debugger monitors rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and running all the rules.

**Parent topic:** [Building and running rules](#)

## Running and debugging decision operations

To run a decision operation in standard or debugging mode, you must create a launch configuration by selecting the project in which the operation is stored.

### About this task

To run or debug a decision operation, the rule engine needs values for all the IN and IN\_OUT parameters for your ruleset.

### Procedure

To start creating a run or debug configuration:

- On the **Run** menu, click **Run Configurations**.
- On the **Run** menu, click **Debug Configurations**.

To complete the new launch configuration:

- Double-click the **Decision Operation** header to further configure the new run or debug configuration, and set up the parameters.

To run the decision operation in standard or debugging mode:

- According to your initial choice, click **Run** to start the execution, or click **Debug** to start the execution in debug mode.

### Results

The ruleset is run in either standard or debugging mode, and you now have a launch configuration that is ready to be reused.

If a particular decision operation already contains values for all IN and IN\_OUT parameters, you can also run or debug it with default settings as follows:

1. Select the decision operation or the rule project in which it is stored.
2. Choose **Run As > 2 Decision Operation** or **Debug As > 2 Decision Operation** from the **Run** menu.

If you select a rule project that has more than one decision operation, you are prompted to select one in the wizard.

**Parent topic:** [Running and debugging](#)

## Debugging rulesets

Rule Designer provides a set of tools to inspect the execution of the rules and the state of the engine, and set breakpoints on classes, objects, rules, decision tables, and ruleflows. The debugger monitors rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and running all the rules.

### [Ruleset debugging tools](#)

Use the Rule Designer debug tools to make sure that your rule application projects run as expected.

### [Starting a debugging session](#)

You start a debugging session to test your rules and evaluate the runtime performance of a ruleset.

### [Setting breakpoints in rules](#)

You can set breakpoints on rule expressions to do a step-through analysis of the rules.

### [Setting breakpoints in the working memory](#)

You can set breakpoints on objects in working memory if you want execution to stop when the objects are retracted or updated.

### [Setting breakpoints in the BOM](#)

You can set breakpoints on business object model (BOM) classes to stop execution when an object is added, retracted, or updated.

### [Stepping through the rule execution code](#)

You can use the Debug Perspective to step into the code to debug a rule.

### [Inspecting the working memory](#)

You can inspect the objects in working memory when the execution mode is RetePlus.

### [Inspecting the agenda](#)

You can inspect the rules that are placed in the agenda when you debug a rule or Java™ project for rules.

### [Viewing and evaluating expressions](#)

You can inspect expressions in the context of a stack frame when the virtual machine (VM) suspends a thread at a breakpoint or it is stepping through code.

### [Viewing variable values](#)

You can view the variables in a stack frame.

### [Viewing breakpoints](#)

You can view the breakpoints that you inserted.

**Parent topic:** [Running and debugging](#)

## Ruleset debugging tools

Use the Rule Designer debug tools to make sure that your rule application projects run as expected.

### [Debug mode](#)

You can execute a ruleset in debug mode to test rules, inspect execution of rules and engine state, and set breakpoints on classes, objects, rules, decision tables and trees, and rule flows.

### [Debug view](#)

The Debug view displays the stack frame for the suspended threads for each target you are debugging. You can get a stack trace of the execution history.

### [Working memory](#)

The Working Memory view display the objects in working memory for the RetePlus execution mode. Use it with the Agenda view to easily check whether the rules behave as expected.

### [Agenda](#)

The Agenda view displays any rule instances scheduled for execution in RetePlus mode. Use it with the Working Memory view to quickly debug your applications.

### [Variables](#)

The Variables view displays the names of the variables currently bound to a business rule and the parameters currently visible in the engine.

### [Breakpoints](#)

The Breakpoints view displays the breakpoints that stop the execution of rules at any point to examine the state of variables, the agenda, and working memory.

### [Console](#)

The Console view displays messages that have been sent to the output stream associated with the engine.

**Parent topic:** [Debugging rulesets](#)

## Debug mode

You can execute a ruleset in debug mode to test rules, inspect execution of rules and engine state, and set breakpoints on classes, objects, rules, decision tables and trees, and rule flows.

In Rule Designer, you can debug both rule code and Java™ code in a project.

The Debug views are useful when you start testing and debugging the execution of your rules. The debugger monitors business rule execution by checking the ruleset, connecting to a rule engine, sending the ruleset to the engine, resetting the engine, and executing all the rules.

### Debug actions

With the debugger, you can control the process of business rule execution:

- Step into, step over, and step return in any rule or code statement:
  - Step over: to go to the next rule statement.
  - Step into: to go to the next rule statement or stop in the next Java statement.
  - Step return: to go to the next rule statement that is not defined in the current rule artifact.
- Drop to frame or use step filters. Use **Window > Preferences > Java > Debug > Step Filtering** to set debug filters. (On Mac, click **Eclipse > Preferences > Java > Debug > Step Filtering**.)
- Resume, suspend, and terminate.
- Set breakpoints on rules, classes, and objects.

### Debug mode performance

You can improve the performance of ruleset parsing when you are debugging a rule project in Eclipse by disabling this option: **Window > Preferences > Java > Debug > Suspend execution on compilation error** (On Mac, click **Eclipse > Preferences > Java > Debug > Suspend execution on compilation error**)

**Note:** The improvement factor depends on your Java virtual machine. Even when this preference is turned off, the debug mode remains slower than the regular run mode.

**Parent topic:** [Ruleset debugging tools](#)

**Related information:**

[Debugging rulesets](#)

## Debug view

The Debug view displays the stack frame for the suspended threads for each target you are debugging. You can get a stack trace of the execution history.

From the Debug view, you can manage the debugging of a rule project or any Java™ program in the workbench. It displays the stack frame for the suspended threads for each target you are debugging. Each thread is shown as a node in the tree.

The Debug view does not provide a history of all the events occurring in the rule engine. However, a stack trace is a useful debugging tool, particularly when an exception has been thrown in the code or when a method is called from a variety of places within your code and you want to learn from where it is called when a particular problem occurs.

You can get stack trace information of the execution history. The stack lists the classes, and the methods within those classes, that are called at the point where the exception occurs through programmatic access. For example, you can use the `StackTraceElement` class and the `getStackTrace` method of the `Throwable` class.

You can also generate a partial Java stack trace by using the static `Thread.dumpStack` method or the `printStackTrace` method of the `Throwable` class.

If the JVM experiences an internal error, it calls its own signal handler to print out the threads and monitors information.

**Parent topic:** [Ruleset debugging tools](#)



## Working memory

The Working Memory view display the objects in working memory for the RetePlus execution mode. Use it with the Agenda view to easily check whether the rules behave as expected.

The view can display the value and type of the various fields of the object. From that view, you can dynamically inspect the objects as they affect the rules. This is important to check whether a rule behaves correctly. For example, you can easily check the behavior of the rules currently in the agenda that use a specific value.

**Parent topic:** [Ruleset debugging tools](#)

**Related information:**

[RetePlus algorithm](#)

[Debugging rulesets](#)

# Agenda

The Agenda view displays any rule instances scheduled for execution in RetePlus mode. Use it with the Working Memory view to quickly debug your applications.

The values in the Working Memory view are updated when the engine is updated. The view can display the priority of a rule in the agenda and the objects used by it. In conjunction with the Working Memory view, it provides you with a powerful way to debug an application quickly.

**Parent topic:** [Ruleset debugging tools](#)

**Related concepts:**

[RetePlus mode](#)

**Related information:**

[RetePlus algorithm](#)

# Variables

The Variables view displays the names of the variables currently bound to a business rule and the parameters currently visible in the engine.

For example, if the ShoppingCart object is bound to a variable called ?shoppingCart in one of the rules, the Variables view shows this relationship. The values in the Variables view are updated after the evaluation of each expression.

**Parent topic:** [Ruleset debugging tools](#)

**Related information:**  
[Debugging rulesets](#)

## Breakpoints

The Breakpoints view displays the breakpoints that stop the execution of rules at any point to examine the state of variables, the agenda, and working memory.

You can set breakpoints on the following elements:

- Rule expressions
- XOM classes and class members
- Objects in the working memory

You can set breakpoints in the following rule artifacts:

- BAL rules (in the action part)
- Technical rules
- Decision tables and decision trees (in an action cell)
- Functions
- Ruleflows
- Rule tasks (in the Ruleflow Editor)
- Working memory of the rule engine (when executing with the RetePlus execution mode)
- Business object model (BOM)
- BOM-to-XOM mapping code

**Parent topic:** [Ruleset debugging tools](#)

## Console

The Console view displays messages that have been sent to the output stream associated with the engine.

The Console shows the following text:

- Standard output
- Standard error
- Standard input

**Parent topic:** [Ruleset debugging tools](#)

**Related information:**  
[Debugging rulesets](#)

## Starting a debugging session

You start a debugging session to test your rules and evaluate the runtime performance of a ruleset.

### About this task

Before deploying your rules, you should always test and debug them, and evaluate the runtime performance of a ruleset.

### Procedure

To start a debug session in Rule Designer:

1. On the **Run** menu, select **Debug Configurations**.
2. In the **Debug Configurations** dialog, click a previously defined configuration in the **Configurations** pane.
3. Click **Debug**.
4. Click **Yes** to confirm the switch to the Debug perspective.

### Results

You can then add and remove breakpoints, inspect the agenda and working memory, and take other debugging actions. This way, you can test and fix bugs in a decision operation for decision services in a Java™ application.

**Parent topic:** [Debugging rulesets](#)

**Related information:**  
[Ruleset debugging tools](#)

## Setting breakpoints in rules

You can set breakpoints on rule expressions to do a step-through analysis of the rules.

### Setting breakpoints

You can set breakpoints on rule expressions to facilitate a step-through analysis of the rules.

### Removing a breakpoint

When you no longer need a breakpoint, you can remove it.

### Removing all breakpoints

You can remove all the breakpoints together.

**Parent topic:** [Debugging rulesets](#)

## Setting breakpoints

You can set breakpoints on rule expressions to facilitate a step-through analysis of the rules.

### About this task

You can set breakpoints only on the parts of the rule that are run, that is, function statements and rule actions. If you try to set a breakpoint on a line where breakpoints are meaningless, the breakpoint is set at the next valid line.

### Procedure

To set a rule breakpoint in a rule:

1. In the Intellirule editor, select the line where you want to set a rule breakpoint.
2. Do one of the following actions:
  - Select **Toggle Breakpoint** either from the **Run** menu or after right-clicking in the shaded border at the rule.
  - Double-click the Intellirule editor margin.
  - Press Ctrl+Shift+B.

The margin of the Intellirule editor displays a blue dot.

**Parent topic:** [Setting breakpoints in rules](#)




## Removing a breakpoint

When you no longer need a breakpoint, you can remove it.

### Procedure

To remove a breakpoint:

1. Select **Toggle Breakpoint** either from the **Run** menu or by right-clicking the breakpoint in the shaded border at the rule.
2. Double-click the breakpoint in the Intellirule editor margin.
3. In the Breakpoints view, select the breakpoint that you want to remove and click **Remove Selected Breakpoints** .

**Parent topic:** [Setting breakpoints in rules](#)

## Removing all breakpoints

You can remove all the breakpoints together.

### Procedure

To remove all the breakpoints:

Select **Remove All Breakpoints**.

You can do this operation in only the Breakpoints view.

**Parent topic:** [Setting breakpoints in rules](#)

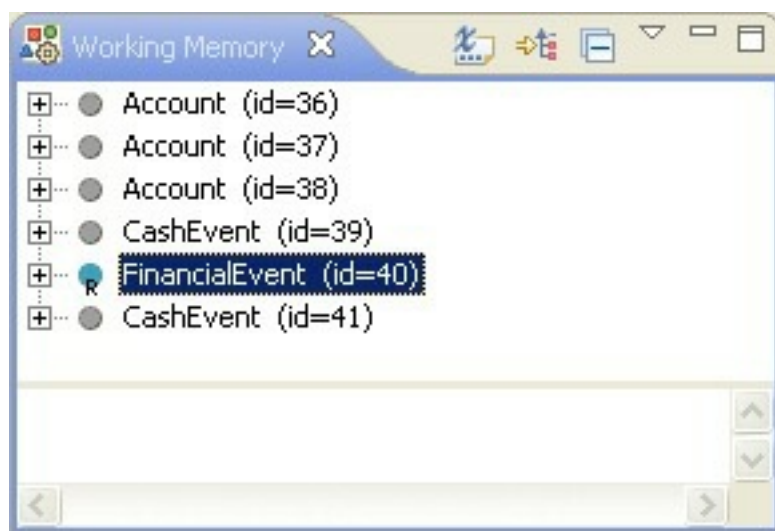
## Setting breakpoints in the working memory

You can set breakpoints on objects in working memory if you want execution to stop when the objects are retracted or updated.

### Procedure

To set a breakpoint:

1. Start a debug session.
2. In the **Working Memory** view, right-click the object on which you want to set a breakpoint, and then select one of the following options in the pop-up menu:
  - If you want execution to stop when you retract this object, click **Add/Remove Working Memory Breakpoint > Retract**.
  - If you want execution to stop when you update this object, click **Add/Remove Working Memory Breakpoint > Update**.



3. Continue debugging.

### Results

Execution stops when the object is retracted or updated.

**Parent topic:** [Debugging rulesets](#)

**Related information:**  
[Ruleset debugging tools](#)

## Setting breakpoints in the BOM

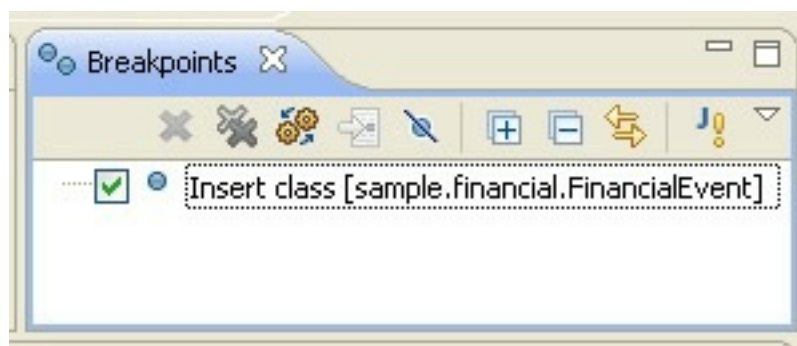
You can set breakpoints on business object model (BOM) classes to stop execution when an object is added, retracted, or updated.

### Procedure

To set a breakpoint:

1. Choose the type of BOM class breakpoint you want to set:
  - To stop execution when adding an object of the class, click **Run** > **Add BOM Class Breakpoint** > **Insert**.
  - To stop execution when retracting an object of the class, click **Run** > **Add BOM Class Breakpoint** > **Retract**.
  - To stop execution when updating an object of the class, click **Run** > **Add BOM Class Breakpoint** > **Update**.
2. A dialog opens. In the **Class name** field, type the fully qualified name of the class on which you want to set a breakpoint, then click **OK**.
3. On the **Window** menu, click **Show View** > **Breakpoints**.

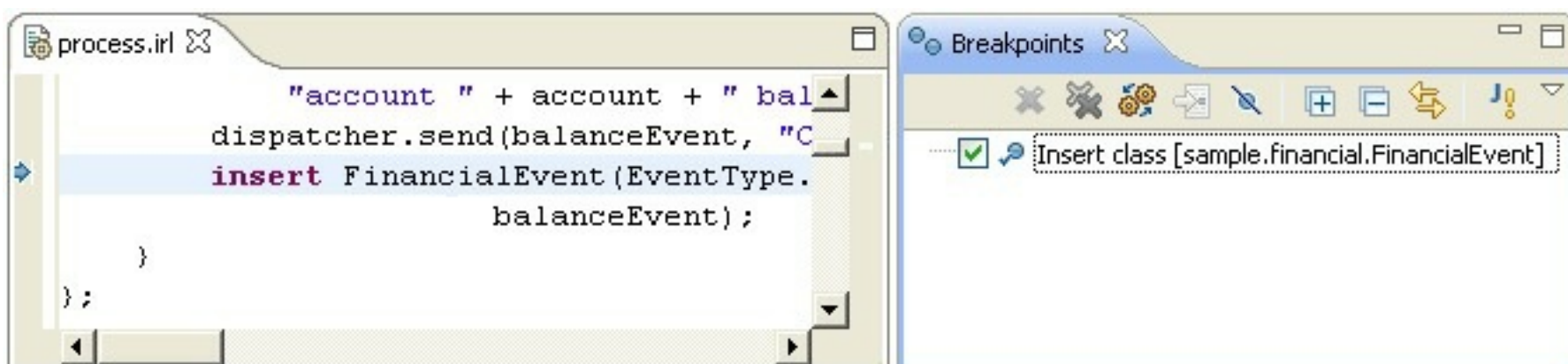
The Breakpoints view opens. You can see that a BOM class breakpoint has been added to the list of breakpoints.



4. Start debugging.

### Results

Execution stops when an object of the BOM class is added, retracted, or updated.



**Parent topic:** [Debugging rulesets](#)

## Stepping through the rule execution code

You can use the Debug Perspective to step into the code to debug a rule.

### About this task

Several stepping options are available.

#### Note:

You cannot step into Java™ code if the rules are executed in sequential mode. When debugging, you can step into the BOM-to-XOM code, even if the sequential mode is used.

### Procedure

To step into your IRL code:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

2. Select your configuration in the Configurations pane.
3. Make sure that you select the **Stop at first rule statement** check box.
4. Click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be executed.

5. Several stepping options are available to you:

- To call the next expression and suspend execution at the next unfiltered executable line, click **Run** > **Use Step Filters** or click the **Use Step Filters** button. You can also press Shift+F5.
- To call the next Java or rule statement and suspend execution at the next Java or rule line, click **Run** > **Step Into** or click the **Step Into** button. You can also press F5.
- To call the next Java or rule statement and suspend execution at the next rule line in the next rule, function, or rule task, click **Run** > **Step Over** or click the **Step Over** button. You can also press F6.
- To call the next Java or rule statement and suspend execution at the first rule statement of the next rule, function, or rule task, select **Run** > **Step Return** or click the **Step Return** button. You can also press F7.

**Parent topic:** [Debugging rulesets](#)

# Inspecting the working memory

You can inspect the objects in working memory when the execution mode is RetePlus.

## About this task

When you debug a rule project or a Java™ project for rules, and the execution mode is RetePlus, you can inspect the objects in the working memory.

## Procedure

To inspect the working memory:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

2. Select your configuration in the Configurations area.
3. Make sure that the **Stop at first rule statement** box is selected, and then click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be run.

4. Step into the execution statements and check that the Working Memory view is open.

## Results

When objects are added into the working memory, they are shown in the Working Memory view. When you expand objects, you can inspect their values.

**Parent topic:** [Debugging rulesets](#)

**Related information:**  
[Ruleset debugging tools](#)

## Inspecting the agenda

You can inspect the rules that are placed in the agenda when you debug a rule or Java™ project for rules.

### Procedure

To inspect the agenda:

1. Click **Run** > **Debug Configurations**.

The Debug Configurations dialog opens.

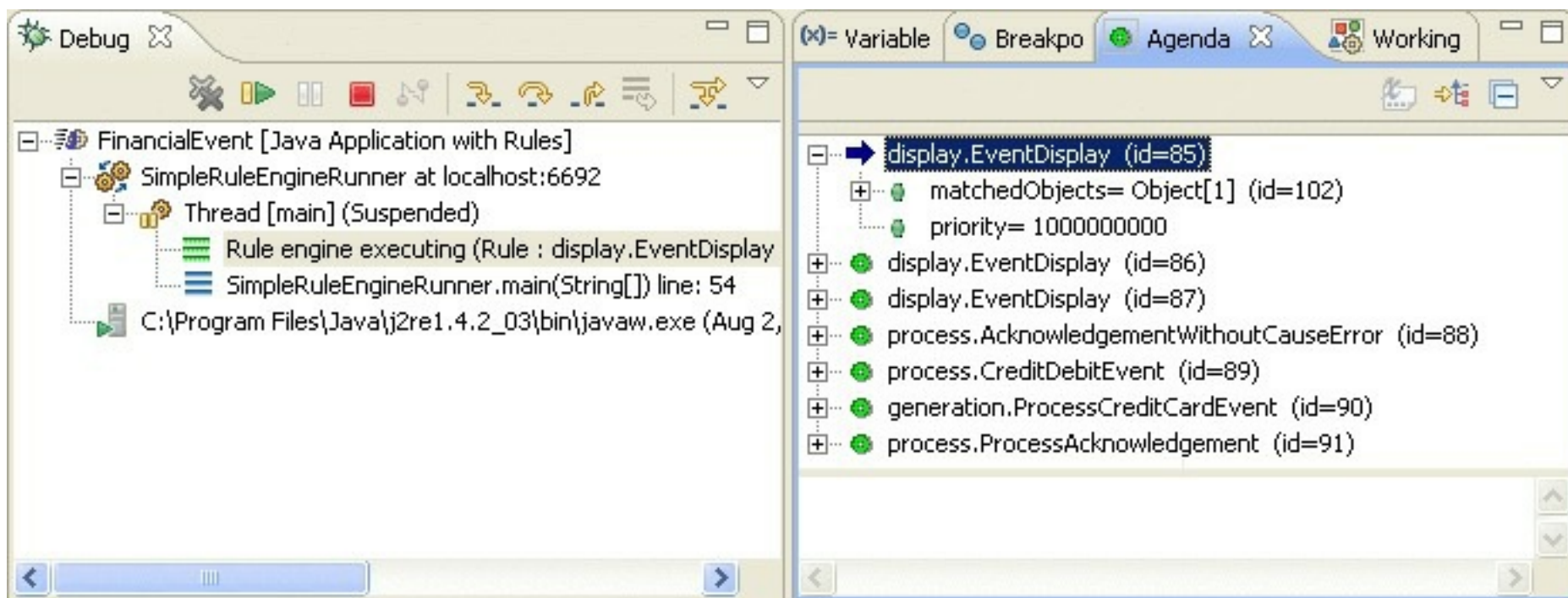
2. In the Configurations area of the Debug Configurations dialog, select your configuration.
3. Make sure the **Stop at first rule statement** box is selected, and then click **Debug**.

The Debug Perspective opens and execution stops at the first rule statement to be executed.

4. Step into the execution statements and check that the Agenda view is open.

### Results

When objects in the working memory match the conditions of a rule, the rule is added to the agenda. The rule is shown in the Agenda view, and when you expand it, you can inspect the matched objects and their values.



Parent topic: [Debugging rulesets](#)

## Viewing and evaluating expressions

You can inspect expressions in the context of a stack frame when the virtual machine (VM) suspends a thread at a breakpoint or it is stepping through code.

### About this task

When the VM suspends a thread, you can inspect expressions in the context of a stack frame.

### Procedure

To view an expression:

1. On the **Windows** menu, click **Show View > Expressions**.

You can then select the expression to do an evaluation.

2. In the **Detail** pane of the **Expressions View**, right-click the expression and click one of the options: **Display**, **Inspect**, or **Execute**.

### Results

The result of a Display or Inspect evaluation is shown in a pop-up window. The **Execute** option does not display a result; the expression is run.

**Parent topic:** [Debugging rulesets](#)

### Related information:

[Ruleset debugging tools](#)



## Viewing variable values

You can view the variables in a stack frame.

### About this task

When the stack frame is selected, you can see the visible variables in that stack frame in the Variables View.

### Procedure

To view the variables:

1. On the Windows menu, click **Show View > Variables**.

The Variables View shows the value of primitive types.

2. Expand the variables to examine their values and display their members.

**Parent topic:** [Debugging rulesets](#)

## Viewing breakpoints

You can view the breakpoints that you inserted.

### Procedure

To view the breakpoints:

1. On the **Windows** menu, click **Show View > Breakpoints**.
2. Right-click a breakpoint, and then click one of the following options:
  - **Go to File**: Opens the file in which the breakpoint is placed.
  - **Enable/Disable**: Makes the breakpoint active or inactive.
  - **Remove/Remove All**: Removes the selected breakpoint or all the breakpoints.

**Parent topic:** [Debugging rulesets](#)

## Executing rulesets in the decision engine

In Rule Designer, you can extract and package your rules for execution on different platforms.

You define the rule engine to execute your rules in the properties of the rule project. When you change the **Rule Engine** property for a rule project, it also applies to the projects that it references. For decision services, you select the rule engine in the properties of the decision service main project.

In Operational Decision Manager on Cloud, decision services use the decision engine. When you create a decision service in the cloud portal, make sure that the rule project is set to the decision engine.

### Decision engine

The decision engine optimizes the execution performance of your ruleset. Before deployment, Decision Center compiles the rules into executable code (Java™ bytecode) by default.

Rule Designer compiles the rules into executable code by default. Clear the **Optimize ruleset loading (Java bytecode generation)** option on the Export a Ruleset Archive page to compile the rules to intermediate code.

You can choose the Fastpath, sequential, or RetePlus execution mode. With the working memory and agenda features, you can store and manipulate application objects. The working memory contains references to the application objects. The agenda lists and orders the rule instances that are eligible for execution.

### Workflow for running rules on the decision engine

To run rules with the decision engine, do the following steps:

1. Create a rule project to encapsulate the business logic of your legacy applications.
2. If necessary, change the **Rule Engine** property of the rule project to **Decision engine** instead of **Classic rule engine**. When the rule project contains a BOM-to-XOM mapping in ILOG Rule Language (IRL), the mapping migrates to a BOM-to-XOM mapping in ARL. The migration happens only once. For more information, see [BOM-to-XOM mapping migration](#).
3. Test the execution of the rules in Rule Designer.
4. After you create decision operations and deployment configurations, you can publish the decision service to Decision Center on the cloud.
5. After you update the decision service in Decision Center on the cloud, you can deploy the modified decision service to the Rule Execution Server in the development environment.
6. After you test and validating the decision service, you can deploy it to the production environment.

### BOM-to-XOM mapping migration

The migration of the BOM-to-XOM mapping is carried out the first time you change the **Rule Engine** property of the rule project to **Decision engine** instead of **Classic rule engine**. The original BOM-to-XOM mapping that is written in IRL is migrated to the Advanced Rule Language (ARL). In the BOM editor, you can work directly with ARL to define the BOM-to-XOM mapping for rule projects that are designed for decision engine.

The migration does not modify the original BOM-to-XOM mapping in IRL (.b2x file), but creates a new file (.b2xa) to store the migrated code instead, which enables the user to switch back and forth between engines.

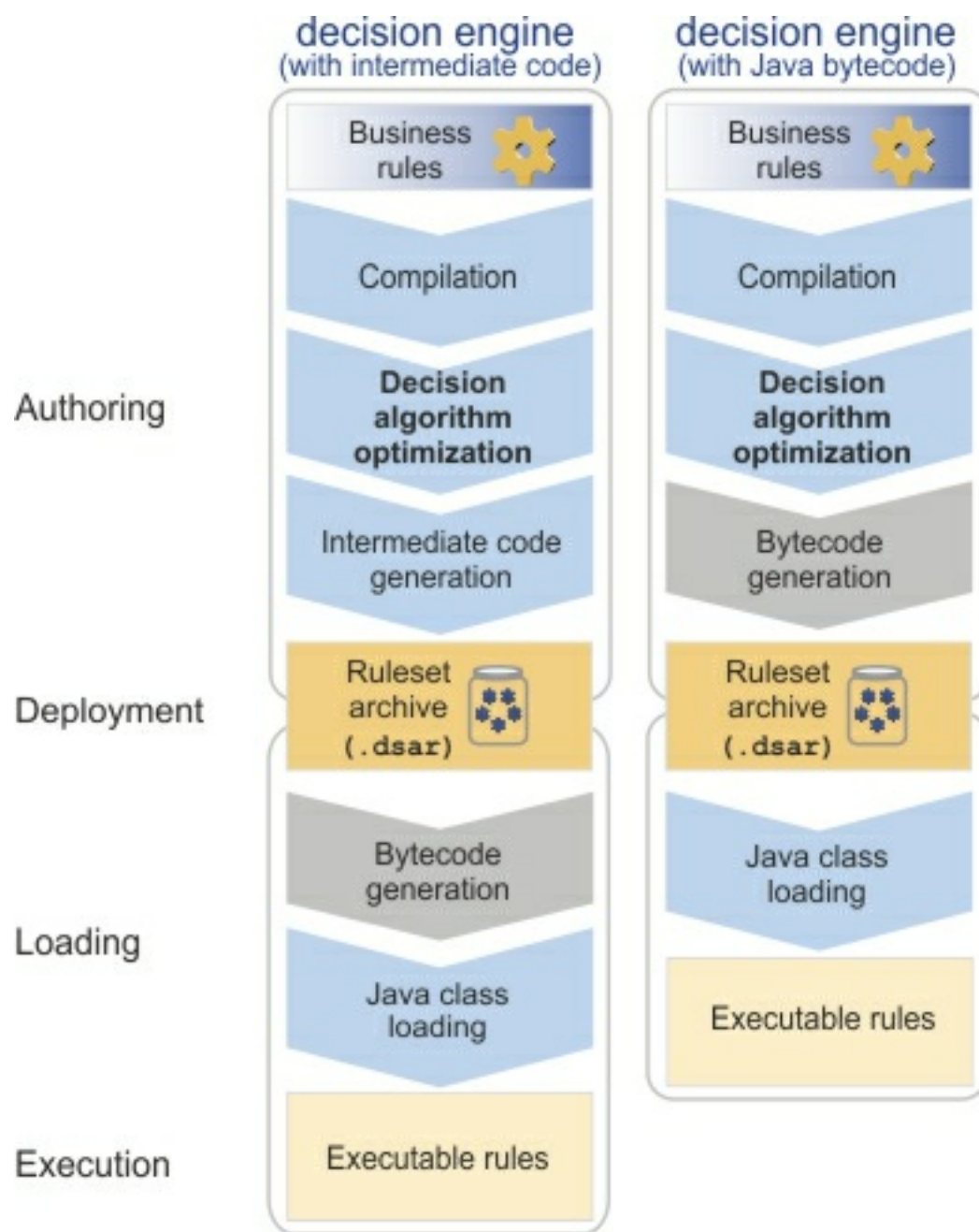
When created, each file separately retains the changes that you make for a particular engine, but there is no synchronization of the changes between the two files.

Migrated code can contain an empty body if the original body contains statements that cannot be migrated.

### Compilation and execution

The decision engine compiles rule artifacts into an archive that contains compiled and optimized code that becomes executable when converted to Java bytecode. The ruleset archive .dsar file consists of binary files that contain execution code for the rules and ruleflows. You either deploy intermediate code or Java bytecode.

The following figure shows the process of compilation and execution for the decision engine, with or without Java bytecode generation. This process goes through different stages from the initial compilation of rules until the execution of rules.



**Important:** Bytecode generation improves the loading time of rulesets that are built with the decision engine. In Rule Designer and Decision Center, the bytecode generation option is selected by default.

**Parent topic:** [Building and running rules](#)

**Related concepts:**  
[Optimizing the decision engine](#)

## Optimizing execution

You can improve performance and scalability of your application through the way Decision Server integrates with Java™, and settings for execution modes, algorithms, and various associated configuration files.

### Choosing an execution mode

You can change the execution mode for any task of a rule. Your choice depends on various criteria, and affects the connection between rules and application data.

### Exception handling in rules

Exceptions in rule conditions prevent rules from being fired. This default behavior stops rule processing so that you can interpret the cause and severity of the exception. You can alternatively handle exceptions by allowing the automatic processing of some exceptions, or by customizing a response to a specific exception.

### BAL building blocks

The **in** and **from** building blocks help you optimize performance.

### Optimizing the decision engine

You can optimize the performance of the decision engine by adjusting your rule conditions, and by selecting the most efficient execution mode for your rules.

**Parent topic:** [Building and running rules](#)

## Choosing an execution mode

You can change the execution mode for any task of a rule. Your choice depends on various criteria, and affects the connection between rules and application data.

### Engine execution modes

Decision Server supports the RetePlus, sequential, and Fastpath execution modes. The execution mode affects which rules are executed and in which order.

### Deciding on an execution mode

You can choose a different execution mode than the default one and provide selection criteria.

### Changing the execution mode

You can use the `algorithm` property to change the execution mode for a rule task.

### Data accessibility

The execution mode you choose affects the connection between rules and data.

**Parent topic:** [Optimizing execution](#)

## Engine execution modes

Decision Server supports the RetePlus, sequential, and Fastpath execution modes. The execution mode affects which rules are executed and in which order.

### RetePlus mode

Use RetePlus optimization techniques to improve performance through reduction of the number of rules and conditions, computation of the rules to run, and prioritization of the rule order.

### Sequential mode

The sequential mode provides performance advantages by running all the eligible rules for a rule task in sequence.

### Fastpath mode

Fastpath is a sequential execution mode that also detects semantic relations between rule tests in the same way as RetePlus. However, unlike RetePlus, the Fastpath mode does not support inference.

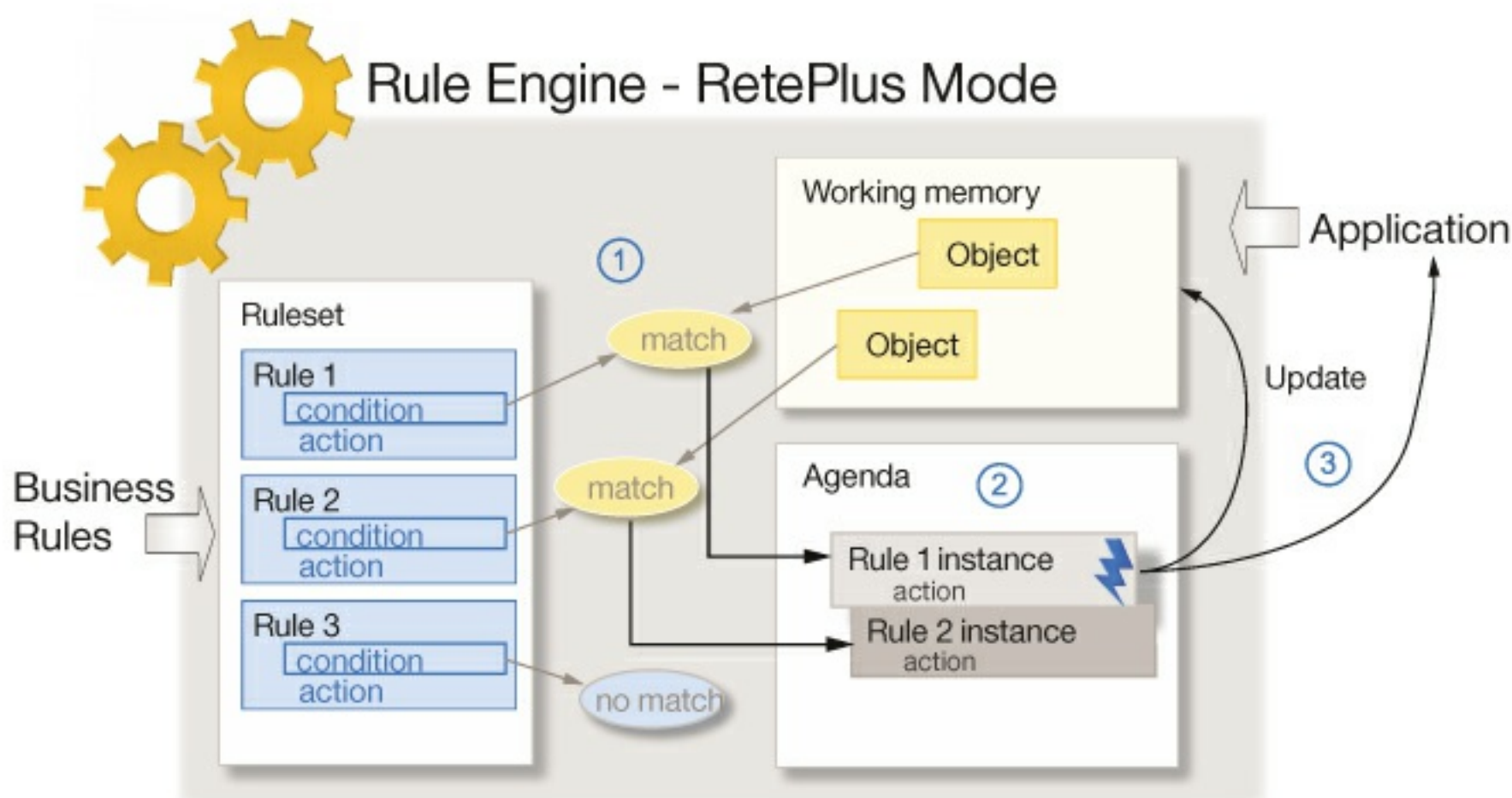
**Parent topic:** [Choosing an execution mode](#)

## RetePlus mode

Use RetePlus optimization techniques to improve performance through reduction of the number of rules and conditions, computation of the rules to run, and prioritization of the rule order.

In RetePlus mode, the rule engine minimizes the number of rules and conditions to be evaluated, computes which rules must be run, and identifies in which order these rules must be run. In RetePlus, the rule engine uses a *working memory* and an *agenda* for storing and manipulating application objects. The working memory contains references to the application objects. The agenda lists and orders the rule instances that are eligible to be run.

The following diagram shows how the RetePlus algorithm works.



The RetePlus algorithm operates as follows:

1. The rule engine matches the conditions of the rules in the ruleset against the objects in working memory. During the pattern matching process, RetePlus creates a network based on semantic relationships between rule condition tests.
2. For each match, a rule instance is created and put into the agenda. Then, based on some ordering principles, the agenda selects the rule instance to be run.
3. When the rule instance is executed, the rule action is executed.

This action modifies the working memory in the following way

- By adding an object to the working memory
- By removing an object from the working memory
- By modifying the attributes of an existing object.

4. The process carries on cyclically until no more rule instances are left in the agenda.

In RetePlus, whenever the working memory is modified, the rule engine repeats the pattern matching process. It reassesses matches after each rule instance is run and modifies the data. As a possible consequence, the list of rule instances in the agenda can change. Thus, RetePlus is incremental and data-driven. The RetePlus algorithm is based on an inference process that the sequential and FastPath algorithms do not support.

### Note:

As of V8.9.0, Reteplus for the decision engine considers the value instead of the reference to identify rule instances. For example, a list that includes several strings with the same value does not activate one rule fired for each list instance, as is the case with the classic rule engine.

**Parent topic:** [Engine execution modes](#)

### Related information:

[RetePlus algorithm](#)

[Improving the performance of the RetePlus execution mode](#)

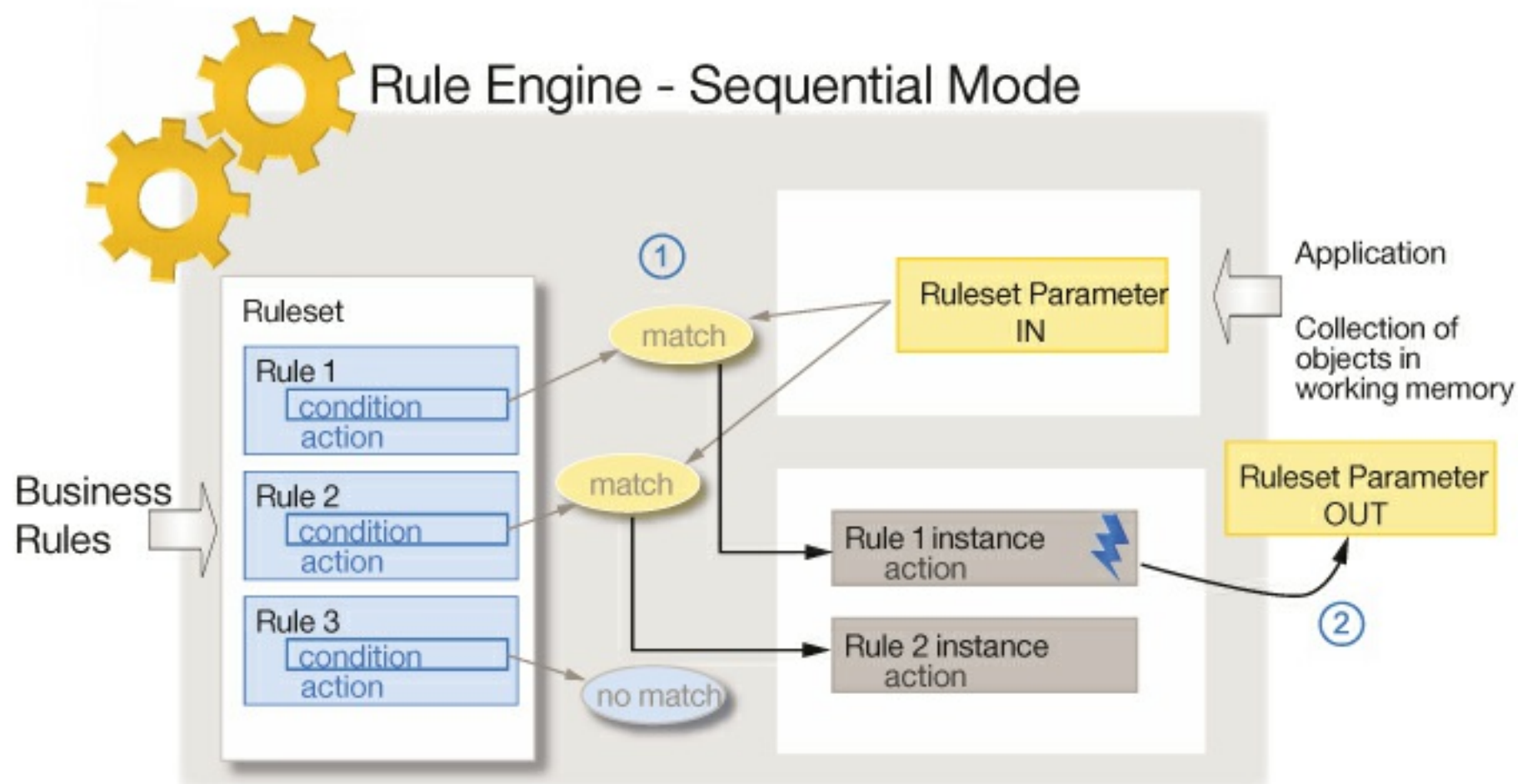




## Sequential mode

The sequential mode provides performance advantages by running all the eligible rules for a rule task in sequence.

The following diagram shows how the sequential algorithm works.



The sequential algorithm operates as follows:

1. The rule engine does pattern matching on input ruleset parameters and on the conditions defined on the collections of objects in working memory.
2. For each match, a rule instance is created and immediately run. When a rule instance is run, it sets the value of an attribute or an output ruleset parameter.

Rules that are run with the sequential algorithm are stateless. The sequential algorithm operates rather like an execution stack where pattern-matching rule instances are run once with no reevaluation of the rules. In rules that are run in sequential mode, you cannot use existence conditions, such as `there is at least one`, or the number `of`, in relation to objects in the working memory. However, you can make such conditions refer to a collection within an object in the working memory, see [in](#).

Because of its systematic nature, the sequential execution mode does well on validation and compliance types of applications.

Rules can be processed sequentially by using rule tasks within a ruleflow, see [Ruleflows](#).

Sequential processing is specified in the `algorithm` property of the rule task. You can select it explicitly in Rule Designer. See [Choosing an execution mode](#).

**Parent topic:** [Engine execution modes](#)

**Related information:**

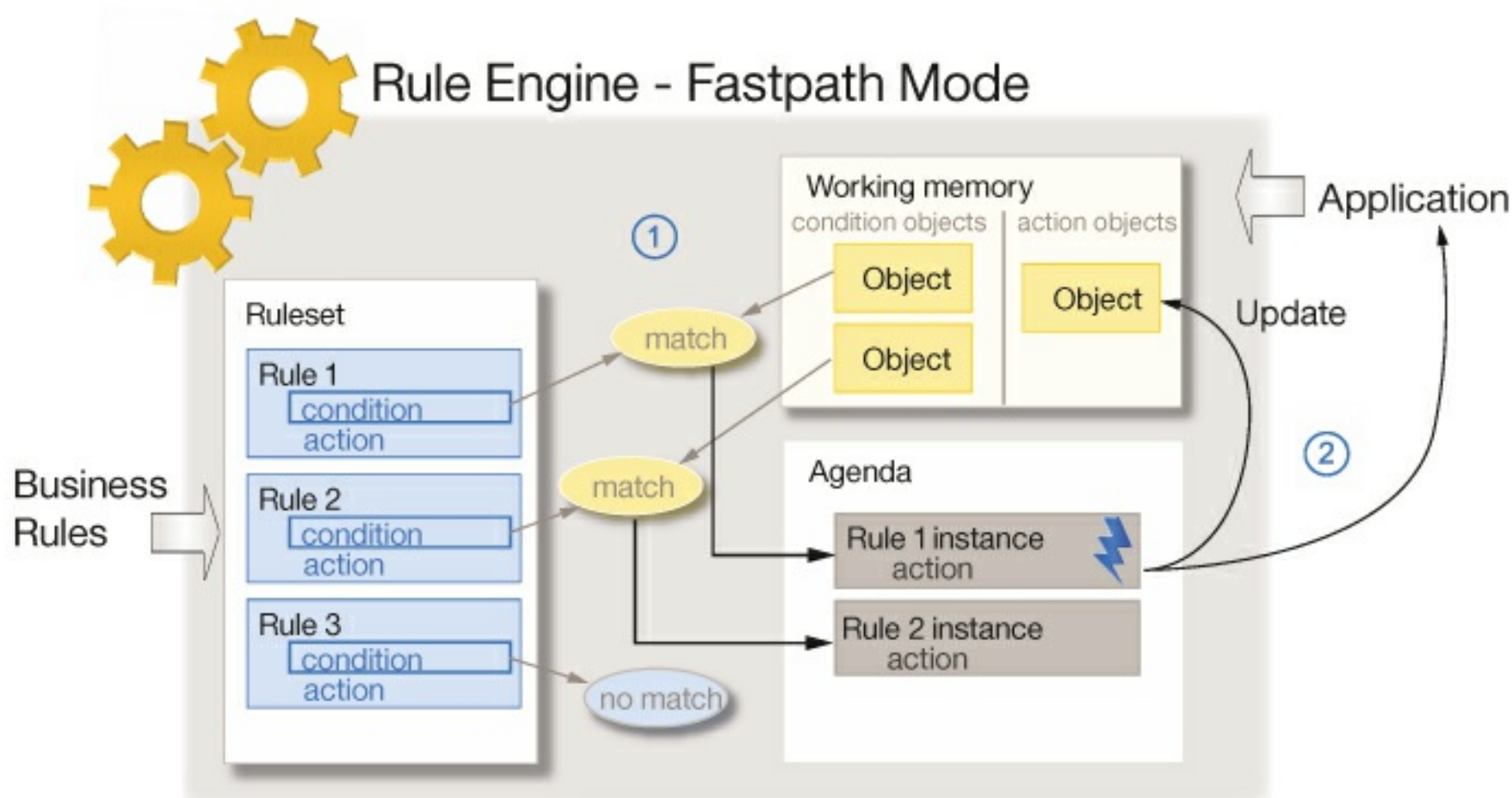
[Sequential algorithm](#)

## Fastpath mode

Fastpath is a sequential execution mode that also detects semantic relations between rule tests in the same way as RetePlus. However, unlike RetePlus, the Fastpath mode does not support inference.

The Fastpath execution mode improves the sequential compilation and execution of a rule project. Fastpath is a sequential mode of execution that also detects semantic relations between rule tests during the pattern matching process, like RetePlus.

The following diagram shows how the Fastpath algorithm works.



The Fastpath algorithm operates as follows:

1. The rule engine uses a working memory that references application objects or ruleset parameters. Fastpath does the pattern matching process, as in RetePlus, by creating a tree based on semantic relations between rule condition tests.
2. For each match, a rule instance is created and inserted in the agenda.
3. After the pattern matching process, the rule instances in the agenda are executed.
4. The rule engine stops after the rule instances have been executed. This behavior also depends on the exit criteria of the rule task. The pattern matching process is not repeated.

Fastpath combines features of the RetePlus mode for pattern matching and features of the sequential mode for rule execution. In this sense, it compares well for correlation types of applications as well as for validation and compliance applications.

Like the sequential mode, the Fastpath mode is stateless. As such, it is dedicated to matching objects against a very large number of rules that individually do simple discriminations or light join tests. It is best for very large number of rules to be executed in sequence directly without any inference support. In addition to its advantages as a variant of the sequential mode, the Fastpath execution mode is designed to further optimize the execution of the compliance and validation rules, which constitute a substantial part of business rules.

**Parent topic:** [Engine execution modes](#)

**Related information:**  
[Choosing an execution mode](#)

# Deciding on an execution mode

You can choose a different execution mode than the default one and provide selection criteria.

You can choose an execution mode for each rule task of a ruleflow. By default, a rule task is assigned the Fastpath execution mode when you create it. To achieve optimal performance, you might need to choose another execution mode that is better adapted for the rules in a particular rule task.

Here is a brief table that shows which execution mode to select in what cases.

Table 1. Quick view of execution mode selection

Choose this mode:	In this case:
RetePlus	<ul style="list-style-type: none"><li>• All included semantically.</li><li>• Stateful application.</li><li>• Rule chaining.</li><li>• Useful in the case of many objects and limited changes.</li></ul>
Sequential	<ul style="list-style-type: none"><li>• Covers most cases.</li><li>• Many rules, few objects. Has limitations. Use with homogeneous rules.</li><li>• Highly efficient in multi-threaded environment.</li></ul>
Fastpath (the default mode for ruleflow tasks)	<ul style="list-style-type: none"><li>• Rules implementing a decision structure, many objects.</li><li>• Highly efficient in multi-threaded environment.</li></ul>

To determine which execution mode to use on a rule task, you must analyze the structure of the rules in the rule task and what type of processing they do.

**Note:**

You cannot use the Fastpath or Sequential algorithm when the rules have actions that create, modify, or remove objects that are matched in the conditions of the rules.

To make the best choice, answer the following questions:

- [What type of application do your rules implement?](#)
- [What types of objects do your rules use?](#)
- [What is the effect of rule actions?](#)
- [What sort of tests do you find in rule conditions?](#)
- [What priorities have you set on your rules?](#)

## What type of application do your rules implement?

Depending on the purpose of the business logic that is defined in a rule task, you choose a different execution mode.

### Compliance and validation

Loosely interrelated rules that check a set of conditions to yield a go/no go or similar constrained result. Compliance business rule applications are often used in underwriting, fraud detection, data validation, and form validation. Business rules in such applications generally have a yes or no result and provide some explanation on the decision.

For compliance applications, preferably use the **sequential** or **Fastpath** execution modes.

### Computation

Strongly interrelated rules that compute metrics for a complex object model. Computation business rule applications are often used for scoring and rating, contracts, and allocation. Business rules in such applications carry out different calculations on an object that is responsible for providing a final value (or rating).

For such applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

### Correlation

Strongly interrelated rules that correlate information from a set of objects to compute some complex metrics. Correlation business rule applications are often used for billing. Business rules in such applications insert information.

For correlation applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

### Stateful session

Strongly interrelated rules that correlate events in a stateful engine session. Stateful applications are often used in alarm filtering and correlation, GUI customization, and web page navigation.

For such applications, preferably use the **RetePlus** execution mode without a ruleflow.

### What types of objects do your rules use?

Depending of the types of objects acted upon by your rules, you choose a different execution mode.

### Homogeneous or heterogeneous rules?

Bindings are heterogeneous when rules do not operate on the same classes. When bindings are heterogeneous, the rules might have condition parts of different heights. For example:

Rule	Condition
Rule1	... when{A();B()} ...
Rule2	... when{A()} ...
Rule3	... when{B()} ...

If your rules define heterogeneous bindings, use the **RetePlus** or **Fastpath** execution mode.

Bindings are homogeneous when all the rules operate on the same class (the same kind and number of objects), but introduce different tests, for example:

Rule	Condition
Rule1	... when{Person(age == 12);} ...
Rule2	... when{Person(age > 20);} ...

If your rules define homogeneous bindings, use the **sequential** execution mode.

### What is the effect of rule actions?

Depending of the types of effects generated by your rules on execution, you choose a different execution mode.

### Modifications to the working memory

If rule actions manipulate working memory objects and use of the IRL keywords insert, retract, or update, use the **RetePlus** execution mode. Because these keywords entail modifications to the working memory, the rule engine reevaluates subsequent rules. If you use another execution mode, the rule engine does not reevaluate subsequent rules after the modifications.

### Rule chaining

When rule actions cause modifications in the working memory or in the parameters, and when they do pattern matching on objects that have no relationship, like people and hobbies, there is chaining between your rules. This process is also known as inference.

For example:

SILVER LEVEL CUSTOMER, GREATER THAN \$5000 purchase promote to GOLD LEVEL GOLD LEVEL CUSTOMER, GREATER THAN \$5000 purchase apply 25% discount
---------------------------------------------------------------------------------------------------------------------------------------------------------

You can see there is chaining between these two rules because they do pattern matching on two different objects, customer and purchase, and that the second one modifies the level attribute of the customer object.

Basically, if you know your rule actions cause the execution of other rules, use the **RetePlus** execution mode.

### What sort of tests do you find in rule conditions?

Depending on the type of conditions used in your rules, you choose a different execution mode.

### Tests that require a working memory

If rule conditions test the existence of an object or gather items of a collection directly in working memory,



with the IRL keywords exists or collect, without in or from constructs, use the **RetePlus** or **Fastpath** execution modes.

### Regular pattern for tests

If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, use the **Fastpath** execution mode.

If the order of tests in rule conditions is not regular, use the **RetePlus** or **sequential** execution modes.

### What priorities have you set on your rules?

Depending on the priorities you have set on the rules, you choose a different execution mode.

- If you have set static priorities, you can use any algorithm.
- If you set dynamic priorities, that is, priorities that are defined as an expression, you must use the **RetePlus** execution mode.

### Summary

You can use the following table as a reference to make your decision when you choose an execution mode for a rule task. The number of stars indicates the degree of performance.

Table 2. Choosing an execution mode

In your rule task:	RetePlus	Sequential	Fastpath
Compliance and validation application	★	★★★★	★★★★
Computation application with inference	★★★★	⊖	⊖
Computation application without inference	★	★	★★★★
Correlation application with inference	★★★★	⊖	⊖
Correlation application without inference	★	★	★★★★
Working memory objects	★★★★	★★★★	★★★★
Rule chaining	★★★★	⊖	⊖
Tests on existence or collection items directly in working memory	★★★★	⊖	★★★★
Shared test patterns	★★	★	★★★★
Heterogeneous bindings	★★★★	⊖	★★★★
Dynamic priorities	★★★★	⊖	⊖
Runtime rule selection that selects a few rules among many	★★★★	★★★★	Decision engine: ★ ★
Numerous rules	★	★★★★	★★★★

**Parent topic:** [Choosing an execution mode](#)

## Changing the execution mode

You can use the algorithm property to change the execution mode for a rule task.

### About this task

You can specify the default execution mode by using the Preferences dialog.

### Procedure

To select an execution mode on the currently selected task:

1. Click **Rule Task** in the Properties view.
2. Select one of the algorithms: RetePlus, Sequential, or Fastpath.

**Note:**

Fastpath is the default mode for all newly created rule tasks. You do not have to choose Fastpath unless you are changing the execution mode of a rule task back to Fastpath. For rule tasks from a previous release, where RetePlus was the default mode, consider switching the tasks to Fastpath if you do not need inference.

**Parent topic:** [Choosing an execution mode](#)

**Related information:**

[Ruleflows](#)

[Engine execution modes](#)

[Working with ruleflows](#)

# Data accessibility

The execution mode you choose affects the connection between rules and data.

In general:

- In sequential mode, preferably use direct data connection with `from` or `in` keywords plus ruleset parameters or variables.
- In RetePlus and Fastpath modes, favor evaluation in the working memory.

The following table summarizes the effect on data accessibility of changing from one execution mode to another.

Table 1. Changing execution modes

From Mode1 to Mode2	Effect on data accessibility
RetePlus -> Sequential	<p>Sequential mode has limitations compared to RetePlus (see <a href="#">Sequential algorithm</a>).</p> <p>When the rules are connected to the data via working memory, executing traces is different in RetePlus/Fastpath and sequential.</p> <ul style="list-style-type: none"><li>• RetePlus: foreach rule by priority, all the tuples.</li><li>• Sequential: foreach tuple, all the rules by static priority.</li></ul> <p>When the rules are connected to the data via <code>in</code> and <code>from</code> keywords, the data is typically in ruleset parameters and variables. Only the executing trace should differ, as described above.</p>
RetePlus -> Fastpath	<p>Fastpath is a sequential-type algorithm, with static priorities and no support for update. However, the change to Fastpath should not affect the executing trace.</p>
Fastpath -> Sequential	<p>Fastpath has fewer rule condition limitations than the sequential mode. The executing trace of Fastpath is the same as RetePlus.</p> <p>When rules are connected to the data via working memory, there is the following difference:</p> <ul style="list-style-type: none"><li>• Fastpath: foreach rule by priority, all the tuples.</li><li>• Sequential: foreach tuple, all the rules by static priority.</li></ul>
Fastpath -> RetePlus	No effect.
Sequential -> RetePlus	Only the executing trace differs, as described above for RetePlus to Sequential.
Sequential -> Fastpath	Only the executing trace differs, as described above for RetePlus to Sequential.

Parent topic: [Choosing an execution mode](#)



## Exception handling in rules

Exceptions in rule conditions prevent rules from being fired. This default behavior stops rule processing so that you can interpret the cause and severity of the exception. You can alternatively handle exceptions by allowing the automatic processing of some exceptions, or by customizing a response to a specific exception.

Automatic exception handling in conditions and custom exception handler are two distinct features. If both features are activated together:

- Exceptions in conditions that are managed by automatic exception handling are caught and not seen by the `CustomExceptionHandler`
- Exceptions in actions are not handled by the automatic exception handling feature (only in the case of a rule variable defined in the condition part, and reused in the action part), so these exceptions trigger the execution of `handleActionException`

### Automatic exception handling in conditions

You can automate exception handling at the decision service project level so that the rule engine continues to process rule conditions that generate Java exceptions. When you activate automatic exception handling in conditions, the engine handles four Java exception types by default, but you can customize the list of Java exception types that the rule engine handles.

### Examples of automatic exception handling

When you enable automatic exception handling in conditions, the rule engine handles specific subclasses so that rule processing can continue after exceptions are thrown. You can understand the logic behind automatic exception handling by reviewing examples.

### Setting a custom exception handler

You can set an exception handler to customize exception handling in rule conditions and actions. When you implement the `CustomExceptionHandler` API, exceptions can either be ignored or rethrown.

**Parent topic:** [Optimizing execution](#)

# Automatic exception handling in conditions

You can automate exception handling at the decision service project level so that the rule engine continues to process rule conditions that generate Java exceptions. When you activate automatic exception handling in conditions, the engine handles four Java exception types by default, but you can customize the list of Java exception types that the rule engine handles.

The rule engine ignores unknown values during the evaluation of the rule condition when automatic exception handling is enabled. You can enable and disable this feature in Rule Designer in the Rule Engine panel of the rule project properties window. Automatic exception handling is disabled by default.

When you enable automatic exception handling, The following subclasses are handled by the rule engine when automatic exception handling is enabled:

- ArithmeticException
- NumberFormatException
- NullPointerException
- IndexOutOfBoundsException

**Important:** Exceptions that occur in the transition condition of a ruleflow are not automatically handled by the rule engine.

With automatic exception handling in conditions enabled, expressions in conditions that result in exceptions during their evaluation are treated as unknown values and are handled by the rule engine. The rule engine evaluates the overall rule conditions by using the following three-valued logic:

NOT(A)		AND(A,B)					OR(A,B)				
				B					B		
A	¬A	A ∧ B		F	U	T	A ∨ B		F	U	T
F	T		F	F	F	F		F	F	U	T
U	U	A	U	F	U	U	A	U	U	U	T
T	F		T	F	U	T		T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

If the rule engine evaluates the rule condition as true or false, despite unknown values in the condition expressions, the rule action logic is applied. If the rule engine evaluates the rule condition as unknown, because of unknown values in the condition expressions, the rule is not fired.

For example, in the following rule, when the value of the name of the customer is null, and the age of the customer is 20, the first condition cannot be resolved, and the second condition is met:

```
if
 the name of the customer is 'Paul'
 or the age of the customer is 20
then
 reject the loan;
```

Because the rule uses the logical or operator, the action to reject the loan is taken. If the age of the customer is not 20, the first condition cannot be resolved, and the second condition is not met. In this case, the rule is not fired, and the loan is not rejected.

See [Examples of automatic exception handling](#) for more examples.

## Conditions that use collections

When the rule engine assembles [collections](#) of objects, the objects for which the where clause is unknown or false are ignored. Consequently, the test on the number of objects in a collection reflects this behavior: Only objects that have a where clause evaluated true are counted.

The following business language expressions use tests on the number of objects:

- If the number of ...
- If there is no ...
- If there is at least one ...
- If there are at least <number> ...
- If there are more than <number> ...
- If there are less than <number> ...
- If there are <number> ...

The following example shows a condition that uses collections and tests the number of objects. In the example, the collection (the `borrowers`) contains only borrowers for whom the attribute values are known. So, when the count operation is performed (is less than 3), any borrower for whom an attribute is unknown is not included in the count. In other words, if 3 borrowers are evaluated and the name of one borrower is unknown while the name of the two other borrowers is the name of the `'borrower'`, then the condition is `True` and the then action is taken.

```
if
 the number of borrowers where the name of each borrower is the name of 'the
 borrower',
 is less than 3,
then
 add "less than 3 similar borrowers found" to the messages of 'the loan' ;
else
 add "duplicate detected" to the messages of 'the loan' ;
 reject 'the loan' ;
```

In the following example, if the where clause is true for two borrowers and unknown for two borrowers, then the rule engine considers that the rule condition is false. The `else` action is taken:

```
If there are more than 3 borrowers in the past borrowers of 'the loan' where
the name of each borrower is the name of 'the borrower',
then
 reject 'the loan';
 add "more than 3 similar borrowers found" to the messages of 'the loan' ;
else
 add "3 or fewer similar borrowers found " to the messages of 'the loan' ;
```

For more examples, see [Expressions that use collections](#).

## Adding exception types

You can customize the list of exception types automatically handled by creating a BOM with the corresponding classes and by adding specific properties, `de.autoCatchExceptionInConditions` and `de.doNotAutoCatchExceptionInConditions`, to each class where appropriate. When automatic exception handling is enabled in Rule Designer, the behavior is to handle automatically exceptions that occur in the condition parts of rules for exceptions that belong to the following subclasses:

- `ArithmeticException`
- `NumberFormatException`
- `NullPointerException`
- `IndexOutOfBoundsException`

You can customize the default behavior to add more exception types to be automatically handled by setting a property named `de.autoCatchExceptionInConditions` to the corresponding class using the BOM editor:

```
package java.lang;
 public class ClassCastException
 extends java.lang.RuntimeException
 property "de.autoCatchExceptionInConditions" "true"
 {
 public ClassCastException();
 public ClassCastException(string arg);
 }
```

## Excluding exception types

You can exclude exception types from automatic exception handling in one of the handled classes by overriding the definition of the class in the BOM with the property named `de.autoCatchExceptionInConditions`. For example, you can remove arithmetic exceptions from automatic exception handling by removing the `de.autoCatchExceptionInConditions` property from the `ArithmeticException` class:

```
public class ArithmeticException
 extends java.lang.RuntimeException
{
 public ArithmeticException();
 public ArithmeticException(string arg);
}
```

You can exclude a subclass of the four default classes from automatic exception handling by setting a property named `de.doNotAutoCatchExceptionInConditions` (in bold):

```
public class StringIndexOutOfBoundsException
 extends java.lang.IndexOutOfBoundsException
 property "de.doNotAutoCatchExceptionInConditions" "true"
{
 public StringIndexOutOfBoundsException();
 public StringIndexOutOfBoundsException(string arg);
 public StringIndexOutOfBoundsException(int arg);
}
```

## Logging

Automatic exception handling uses the standard Java™ logger logging service, so that you can log messages based on message type and level and control how these messages are formatted and stored at runtime.

In Rule Designer, to capture activity traces that occur in automatic exception handling, you must first define a `logging.properties` file and make it available to the JVM that is used in the rule execution run or debug configuration. Then, in the Run Configurations window, under the Parameters & Arguments tab of your decision operation, enter one of the following VM arguments: -

`Djava.util.logging.config.file=file_path/logging.properties` or -

`Djava.util.logging.config.file="${workspace_loc}/your_project/logging.properties"`.

In the following example, the log level for the logger name `com.ibm.rules.engine.aeh` is set to FINE.

```
handlers = java.util.logging.ConsoleHandler
#####
Handler specific properties.
Describes specific configuration info for Handlers.
#####
Limit the messages that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
com.ibm.rules.engine.aeh.level = FINE
```

To enable automatic exception handling logs in a server environment, define a `logging.properties` file and make it available to the JVM by configuring your application server, the log level for the Logger name `com.ibm.rules.engine.aeh` should be set to FINE.

For an introduction to the Java Logging API, see [Java Logging Overview](#). For more information about the `java.util.logging` Java logger, see [java.util.logging](#) in the Oracle documentation.

**Parent topic:** [Exception handling in rules](#)

# Examples of automatic exception handling

When you enable automatic exception handling in conditions, the rule engine handles specific subclasses so that rule processing can continue after exceptions are thrown. You can understand the logic behind automatic exception handling by reviewing examples.

With automatic exception handling in conditions, unknown values for Boolean expressions are resolved by using the following three-valued logic that applies for all combinations of Boolean expressions that can result in an exception:

NOT(A)		AND(A,B)				OR(A,B)			
		A $\wedge$ B		B		A $\vee$ B		B	
A	$\neg A$		F	U	T		F	U	T
F	T		F	F	F		F	F	U
U	U	A	U	F	U	A	U	U	U
T	F		T	F	U		T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

## Simple rule condition

In the following example, if the status of the borrower is unknown, no rule action is executed.

```
if
 the status of 'the borrower' starts with "duplicate"
then
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
else
 add "no duplicate found" to the messages of 'the loan' ;
```

## Rule condition that uses the OR operator

In the following example, if the status of the borrower is unknown and the comment of the loan contains the word duplicate, then the loan is rejected and a duplicate detected message is sent.

```
if
 the status of 'the borrower' starts with "Duplicate" or the comment of 'the loan' contains "duplicate"
then
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
else
 add "no duplicate found" to the messages of 'the loan' ;
```

However, if the status of the borrower is unknown and the comment is unknown, then no rule action is taken. The loan is not rejected and the message no duplicate found is not added to the loan.

## Rule condition that uses the AND operator

In the following example, if the status of the borrower is unknown and the comment in the loan contains the word duplicate, neither the then nor the else actions are taken. In other words, the loan is not rejected and the message no duplicate found is not added to the message of the loan.

```
if
 the status of 'the borrower' starts with "duplicate" and the comment of 'the loan' contains "duplicate"
then
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
else
 add "no duplicate found" to the messages of 'the loan' ;
```



For the loan to be rejected, both expressions must be true.

**If there is no ... then ...**

If a collection of 4 past borrowers contains 3 borrowers for whom the where clause is false and 1 borrower for whom it is unknown, then the rule action is taken.

```
If there is no borrower in the past borrowers of 'the loan' where the name of
this borrower is the name of 'the borrower',
then
add "no duplicate found" to the message of 'the loan'
```

**If there is at least one ... then ...**

If a collection of past borrowers contains 3 borrowers for whom the where clause is true in 1 case, false in 1 case, and unknown in 1 case, then the rule action is taken.

```
If there is at least one borrower in the past borrowers of 'the loan' where
the name of this borrower is the name of 'the borrower',
then
 reject 'the loan' ;
 add "duplicate detected" to the message of 'the loan'
```

**Rule conditions that use the NOT operator**

In the following example, if the status is unknown and the comment does not contain duplicate, the then action is taken and no duplicate found is added to the message of the loan.

```
if it is not true that
 (the status of 'the borrower' starts with "duplicate" and the comment of
'the loan' contains "duplicate")
then
 add "no duplicate found" to the messages of 'the loan' ;
else
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
```

If the status is unknown and the comment contains duplicate, no rule action is executed.

**Decision Table**

In the following example, the otherwise row to a condition column is selected when all the other conditions in the column are false. Alternatively, if one of the other conditions is unknown, then the otherwise row is not selected:

	Customer status contains...	duplicate detected		no duplicate detected		reject 'the loan'
1	Similar	duplicate detected	⊗	(nothing detected)		-
2	Duplicate	duplicate detected	⊗	(nothing detected)		-
3	Otherwise	⊗ (nothing detected)		no duplicate detected	⊗	-

**Rule variable**

If the rule variable 'statusIsNew' is unknown, the rule engine stops the evaluation of the expressions. The condition is considered unknown and no rule action is taken.

```
definitions
 set 'statusIsNew' to the status of 'the borrower' starts with "New";

if it is not true that 'statusIsNew' and the comment of 'the loan' contains
"similar"
then
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
else
 add "no duplicate found" to the messages of 'the loan' ;
```

**Collection of objects**

If the result of the test that the object must fulfill is unknown, then this object is not added to the collection

because only the objects tested true are added:

```
definitions
set 'duplicateBorrowers' to all borrowers where the status of each borrower
starts with "duplicate" ;
```

### Expressions that use collections

In the following example, if the where clause is true for three borrowers and unknown for one borrower, then the rule engine considers that the condition is true because the borrower for whom the where clause is unknown is not counted. The then action is taken:

```
if
 there are less than 3 borrowers where the name of each borrower is the
name of 'the borrower',
then
 add "3 or less similar borrowers found" to the messages of 'the loan' ;
else
 reject 'the loan';
 add "more than 3 similar borrowers found" to the messages of 'the loan' ;
```

In the following example, if the where clause is true for one borrower and unknown for two borrowers, then the rule engine considers that the rule condition is false because the two borrowers for whom the where clause is unknown are not counted. The else action is taken:

```
if
 there are more than 3 borrowers where the name of each borrower is the
name of 'the borrower',
then
 reject 'the loan';
 add "3 or more similar borrowers found" to the messages of 'the loan' ;
else
 add "less than 3 similar borrowers found" to the messages of 'the loan' ;
```

### Expressions that uses contain

The following example expression uses contain, which is an alternative way to test a collection. If the collection ('duplicateBorrowers') contains the object 'the borrower', then the condition is true.

```
definitions
 set 'duplicateBorrowers' to all borrowers where the customer status of
each borrower starts with "duplicate" ;
if
 'duplicateBorrowers' contain 'the borrower'
then
 add "duplicate detected" to the messages of 'the loan' ;
 reject 'the loan' ;
```

### Rule Variables

All rule variables should be resolved different from unknown to allow the evaluation of the condition part. If the rule variable statusIsDuplicate is unknown, the rule engine stops the evaluation of the expressions. The condition is considered unknown and no rule action is taken, even if the comment of 'the loan' contains "similar":

```
set 'statusIsDuplicate' to the customer status of 'the borrower' starts with
"Duplicate";

if
 'statusIsDuplicate' or the comment of 'the loan' contains "similar"
then
 reject 'the loan' ;
 add "duplicate detected" to the messages of 'the loan' ;
else
 add "no duplicate found" to the messages of 'the loan' ;
```

**Parent topic:** [Exception handling in rules](#)



## Setting a custom exception handler

You can set an exception handler to customize exception handling in rule conditions and actions. When you implement the CustomExceptionHandler API, exceptions can either be ignored or rethrown.

### About this task

The default engine behavior is to stop when an exception is thrown. When you implement the CustomExceptionHandler API, you can choose to enforce the default engine behavior or to ignore the exception so that the engine can continue to process rules.

To activate the [CustomExceptionHandler](#), you must implement the interface in the XOM and then specify the implementation class name in Rule Designer.

### Procedure

1. Implement the CustomExceptionHandler interface in the XOM. In the following example, the custom exception handler rethrows exceptions for NumberFormatException:

```
package com.acme;
import ...

public class MyCustomExceptionHandler implements CustomExceptionHandler {

 static Logger logger =
 Logger.getLogger(MyCustomExceptionHandler.class.getName());

 /**
 * The custom exception handler must contain a default constructor.
 */
 public MyCustomExceptionHandler() {
 }

 @Override
 public void handleConditionException(Exception e) throws Exception {
 if (e instanceof NumberFormatException) {
 // ignore number format exception
 } else {
 // rethrow the exception, it will stop the engine execution
 throw e;
 }
 }

 @Override
 public void handleActionException(Exception e, RuleInstance
ruleInstance) throws Exception {
 if (e instanceof NumberFormatException) {
 // ignore number format exception and log
 if (logger.isLoggable(Level.INFO)) {
 logger.log(Level.INFO,
 "Exception while executing action of rule " +
ruleInstance.getRuleName(), e);
 }
 } else {
 // rethrow the exception, it will stop the engine execution
 throw e;
 }
 }
}
```

2. Open Rule Designer.
3. In the Rule Engine panel of the rule project properties window, enter the custom exception handler name. For example, enter com.acme.MyCustomExceptionHandler in the Custom exception handler field.
4. Click **OK**.

### Results

The custom exception handler is created when the engine starts, and removed when the engine is reset.

**Parent topic:** [Exception handling in rules](#)

## BAL building blocks

The **in** and **from** building blocks help you optimize performance.

The rule engine runs on Java EE and integrates Enterprise JavaBeans components. You can optimize performance by using BAL building blocks, and execution modes appropriately.

Rules can also be internally rewritten by the engine to enable automatic optimization of the rules. By using the **from** and **in** building blocks, you reduce the number of objects on which pattern matching is applied, and hence the number of evaluations carried out. The rule engine automatically rewrites rules to use these building blocks whenever possible. For details see [Using objects that are not in memory in a condition](#).

**Parent topic:** [Optimizing execution](#)

# Optimizing the decision engine

You can optimize the performance of the decision engine by adjusting your rule conditions, and by selecting the most efficient execution mode for your rules.

## Using the same definition for several decision engines

To reduce memory consumption, you can create several engines for concurrent use from the same engine definition.

## Adjusting conditions

Most of the execution time is taken up by the pattern-matching process. Consequently, the most efficient way of improving performance in rulesets consists in modifying the condition part of rules.

## Improving the performance of the RetePlus execution mode

You can improve performance by avoiding certain operations in the RetePlus execution mode.

## Improving the performance of the sequential execution mode

To use the sequential algorithm, make sure that your rules are homogeneous and that the rules are not chained. In the sequential algorithm, you can define the number of rules to execute per tuple and the order of execution, by using control properties.

## Improving the performance of the Fastpath execution mode

You improve the performance of the Fastpath execution mode by modifying the condition part of rules.

**Parent topic:** [Optimizing execution](#)

**Related concepts:**

[Executing rulesets in the decision engine](#)

## Using the same definition for several decision engines

To reduce memory consumption, you can create several engines for concurrent use from the same engine definition.

### About this task

Creating several engines from one engine definition, instead of loading an engine definition to create each engine, reduces memory consumption. It does not degrade performance.

Rule Execution Server directly supports concurrent executions.

If you do not want to use Rule Execution Server, use an engine, as an [Engine](#) instance, in only one thread.

Do not use the same objects in different engines in different threads.

### Procedure

To create several engines from the same definition, use the following code:

```
EngineDefinition definition = ...
Engine engine1 = definition.createEngine();
Engine engine2 = definition.createEngine();
```

**Parent topic:** [Optimizing the decision engine](#)

## Adjusting conditions

Most of the execution time is taken up by the pattern-matching process. Consequently, the most efficient way of improving performance in rulesets consists in modifying the condition part of rules.

### Ordering tests

The order in which tests are carried out in rule conditions affects the performance of the pattern-matching process. To get the best performance for a specific condition, place the most discriminant tests at the start. This position accelerates the selection of objects in the discrimination tree for tests that involve constants. Also, when you place the most discriminant tests at the start, you know sooner when an object does not satisfy the tests between conditions in the joins.

### Sharing conditions

If the first N conditions of two rules are identical, these rules share the same portion of the network. As a consequence, searches for objects that satisfy these first N conditions are done only once for all the rules that share these conditions. It is therefore a good idea to modify the order of rule conditions so that they share the network as much as possible.

#### Note:

You cannot share conditions in the sequential execution mode.

### Ordering conditions in rules

Place the most selective conditions, or groups of conditions, at the beginning of tests. In this way, the objects that prevent a rule from being executed are removed sooner in the process. Tokens spend less time in the RetePlus network, joins require less memory, and fewer tests are necessary.

#### Example of the effect of condition order on RetePlus performance

Here is a rule with three object classes.

```
rule filter {
 when {
 A (a1==3; ?x:a2);
 B (b1==2; ?y:b2; b3==?x);
 C (c1==?y);
 }
 then {
 System.out.println("filter");
 }
}
```

Let's work on this example.

1. Imagine different contents for the working memory:
  - There are five objects of class A and their a2 attribute is 10.
  - There is an object of class B whose b2 attribute is 7 and b3 attribute is 10.
  - There is an object of class B whose b2 attribute is 4 and b3 attribute is 10.
  - There are 28 objects of class B. Their b3 attribute is 10 and their b2 attribute is neither 4 nor 7. In other words, there are thirty class B objects in all.
  - There is one object of class C and its c1 attribute is 4.
2. If you were to draw the corresponding RetePlus network, the contents of the first join node would be quite different.
3. It now contains 155 pairs of objects, which can be described like this:
  - Five pairs are formed by the objects of class A and the object of class B whose b2 is 7 and b3 is 10.
  - 150 pairs (that is,  $(5 * (28 + 1 + 1))$ ) are formed by the objects of class A and the objects of class B whose b3 attribute is 10.
4. If you now add an object of class C whose c1 attribute is 7, the work of the second join node consists of doing 155 tests, between the B object of each of the 155 pairs of the first join and the C object that you

have just added, to verify that attributes b2 and c1 have the same value.

5. Now, suppose you create a filter2 rule by changing the order of the conditions of the filter rule so that it is shown as follows:

```
rule filter2 {
 when {
 C (?y:c1);
 B (b1==2; b2==?y; ?x:b3);
 A (a1==3; a2==?x);
 }
 then {
 System.out.println("filter2");
 }
}
```

For the same initial working memory as before, the first join node, corresponding to the first two conditions, now contains 30 (instead of 155) pairs, formed by the class C object and the 30 class B objects whose b2 attribute is 4 and b3 attribute is 10.

6. If you now add an object of the C class whose c1 attribute is 7, the work of the first join node consists of doing 30 tests between the newly added object and the 30 class B objects in memory. This causes us to add a new pair, formed by the C object that you have just added and the B object whose b2 attribute is 7.
7. After all this is done, the second join node is activated. It does 30 tests, between the B object of each pair from the previous join and the A object in memory, to verify that b3 and a2 have the same value.

These two rules, which produce the same result, have quite different computing costs.

- The filter rule uses 155 pairs and a triple to represent the initial state, whereas the filter2 rule uses only 30 pairs and a triple.
- Adding a C object costs 155 tests and the construction of a triple in the filter rule, whereas the same adding costs only 60 tests, a triple, and a pair in the filter2 rule.

## Execution order of conditions and definitions

If a condition depends on a variable, the condition executes as soon as the variable exists and it is the turn of the condition to be executed. In the case where several variables are created in the definition part of the rule, it means that a condition might execute before all the variables are created.

In the following example, the condition the age of 'the winning customer' is at least 18 depends on the variable the winning customer and does not depend on the email. Therefore, this first condition executes as soon as the winning customer exists and before the variable the email is created.

```
definitions
 set 'the winning customer' to a Customer ;
 set 'the email' to the email of 'the winning the customer' ;
if
 the age of 'the winning customer' is at least 18
 and 'the email' is defined
then
 print "Send coupon to customer via email.";
```

**Parent topic:** [Optimizing the decision engine](#)

# Improving the performance of the RetePlus execution mode

You can improve performance by avoiding certain operations in the RetePlus execution mode.

## RetePlus algorithm

RetePlus is a rule execution mode based on the Rete algorithm. It relies on a working memory and agenda, and supports negative patterns, object notification, and logical objects.

## Optimizing the object model

You can decrease the cost of pattern matching by changing the representation of the object model.

## Improving the performance of equality or comparison evaluation

Hashing expressions can improve performance at equality-test stage.

**Parent topic:** [Optimizing the decision engine](#)



## RetePlus algorithm

RetePlus is a rule execution mode based on the Rete algorithm. It relies on a working memory and agenda, and supports negative patterns, object notification, and logical objects.

RetePlus is the Decision Server extension based on the Rete algorithm. In RetePlus mode, rule execution uses an environment based on a working memory and agenda.

- [Working memory and the agenda](#)
- [Negative patterns](#)
- [Object notifications](#)
- [RetePlus network operation](#)

### Working memory and the agenda

Under the RetePlus execution mode, the rule engine functions with [Working memory](#) and an [Agenda](#), adding [Rule instances](#) for execution when conditions are met.

### Working memory

Under the RetePlus execution mode, a rule engine in Decision Server is paired with a *working memory*. The working memory contains all the objects contained in the associated Engine object. You can modify the working memory by adding a statement in the *action part of a rule* or by using the Application Programming Interface (API). Thus, the rule engine is aware of the objects that are in the working memory and any objects that are linked to them. The rule engine can use only objects that are accessible from the working memory.

### Agenda

The *agenda* is where Decision Server stores the rules whose patterns are all matched. Any rule that enters the agenda is said to be instantiated, it becomes a *rule instance*, see [Rule instances](#).

The agenda stores rule instances that are eligible to be executed. If the agenda is empty, execution has no effect. Rule instances placed in the agenda are said to be *eligible*. Often, in the agenda, several rules are eligible. Consequently, the rule engine has to have some way of deciding which particular rule in the agenda should be executed.

In the agenda, rule instances are ordered according to three criteria that determine which rule should be executed first. Additional execution control is offered with the implementation of more complex features, see [Object notifications](#).

### Refraction

A rule instance that has been executed cannot be reinserted into the agenda if no new fact has occurred, that is, if none of the objects matched by the rule are modified, or if no new object is matched by the rule.

The refraction principle enforces that only one rule instance is created for a given tuple. Since the data unit for an engine cycle is the working memory, the engine can easily record all the possible tuples and check that there is only one rule instance for each tuple.

For example consider the following ruleset:

```
class Person {
 Person(int age, boolean sick);
 boolean sick;
 int age;
}

rule incrementAge {
when {
 p: Person(!sick; age < 50);
} then {
 p.age += 1;
 update p;
}

rule cure {
when {
 p: Person(sick);
} then {
 p.sick = false;
```

```
 update p;
}
```

If the object `Person(18, true)` is inserted in the working memory, the refraction principle produces the following execution trace:

1. cure
2. incrementAge

You can add the repeatable property in the `incrementAge` rule:

```
rule incrementAge {
 property com.ibm.rules.engine.repeatable=true;
when {
 p: Person(!sick; age < 50);
} then {
 p.age += 1;
}
```

If the object `Person(18, true)` is inserted in the working memory, the repeatable property produces the following execution trace:

1. cure
2. incrementAge
3. incrementAge
4. incrementAge
5. ...
6. incrementAge

Until the condition `age < 50` is false.

### Repeatable rules

To break the refraction principle, you can use the Boolean rule property

**`com.ibm.rules.engine.repeatable`**. In the decision engine, you must use this property to mark the sets of rules that you want to be repeatable.

- When a rule is not repeatable, the refraction principle prevents a rule instance from being posted again to the agenda.
- When a rule is repeatable, it can be inserted again as a rule instance in the agenda if a modification occurs against its condition objects.

### Recency

If two rule instances have the same priority, the rule that matches the most recent object (the most recently inserted, modified, or retracted object) is executed first.

Priority and recency are used to resolve conflicts when several rule instances are candidates for execution at the same time. If, after using all specified conflict resolution methods, several rule instances remain candidates, then the engine uses internal metarules; this assures that the same sequence of rule executing is always followed, given the same conditions.

### Example of rule executing order

The following technical rule example shows you the order in which rules are executed:

```
rule init {
 when {
 angel();
 }
 then {
 insert clown();
 insert dascyllus();
 }
};
rule last {
 priority = -100;
 when {
 dascyllus();
 }
 then {
 System.out.println("last");
 }
};
```

```

rule first {
 priority = 100;
 when {
 angel();
 clown();
 dascyllus();
 }
 then {
 System.out.println("first");
 }
};
rule second {
 priority = 0;
 when {
 angel();
 dascyllus();
 clown();
 }
 then {
 System.out.println("second");
 }
};
rule third {
 priority = 0;
 when {
 angel();
 clown();
 dascyllus();
 }
 then {
 System.out.println("third");
 }
};
};

```

Suppose that the object `angel` has been inserted from the application using the API, and therefore has been inserted into the working memory, and that we execute all the rules that can be executed.

Here is the order in which the rules are executed:

1. `init` rule: Because the object `angel` was inserted from the application, the `init` rule is the only rule that can be executed. It inserts the objects `clown` and then the object `dascyllus` into the working memory. The most recent object is therefore `dascyllus`.
2. `first` rule: This rule is executed before the second rule. The most recent object matched by these two rules is the same. However, the first object has a higher priority than the second, which determines their execution order.
3. `second` rule: This rule is executed before the third rule. The second and the third rules have the same priority, but the object `dascyllus` is more recent than the object `clown`.
4. `third` rule: This rule is executed before the last rule. Although the object `dascyllus` is more recent than `clown`, the third rule has a higher priority than the last rule.
5. `last` rule.

## Rule instances

A *rule instance* is added to the agenda when objects in the working memory satisfy the condition part of that rule.

Let us consider the following technical rule:

```

rule rule1 {
 when {
 Fish(color == green; type == shark);
 Fish(type == trigger);
 }
 then {...}
};

```

The following rule instances will be put into the agenda:

Rule	Rule	Rule	Rule
rule1(A,C)	rule1(A,D)	rule1(B,C)	rule1(B,D)

In this example, you can see that a rule instance is a dynamic concept: the rule instance is produced by any combination of objects in the working memory that match the patterns specified in the rule. In contrast, a rule is a static concept.

Negative patterns

Standard rule conditions might be described as existential, in the sense that they could be matched by one or several objects that actually happen to exist. In the RetePlus execution mode there is a complementary concept known as *negative patterns*. To express the nonexistence of a particular object in the working memory, we place the not keyword before the condition of the object, as shown here:

```
not Fish(type==eel);
```

This negative pattern gives rise to a successful match because there is no object that matches the corresponding positive pattern:

```
Fish(type==eel);
```

Conversely, this negative pattern:

```
not Fish(type==angel);
```

does *not* give rise to a successful match for the simple reason that there is an object that matches the corresponding positive pattern:

```
Fish(type==angel);
```

Object notifications

With the RetePlus execution mode, you have statements to control individual operations:

- [Object insertion](#)
- [Object removal](#)
- [Object update](#)
- [Attribute modification](#)

Object insertion

The insert statement lets you create a new object and insert it into the working memory. This keyword is followed by the name of the class and the values of the attributes of the object to be created.

Here is an example that involves the creation of a new Course object whose department and number attributes take the values History and 324, respectively:

```
insert Course(History,324);
```

There are two possibilities for the way that an insert operation is processed:

- The object is already in the working memory. In this case, the insert operation is equivalent to an update.
- The object is not in the working memory. In this case, the object is inserted into the working memory and the insert daemon of the corresponding class and its superclasses, if any, is called, and the agenda is updated.

Object removal

The retract statement lets you remove from the working memory an object that is bound to a rule variable.

In the following example, the retract primitive removes the object bound to the ?c variable:

```
rule RemoveCourse {
 when {
 ?c: Course(department == History; number == 254);
 }
 then {
```

```
 retract ?c;
 }
};
```

The retract daemon of the corresponding class, if any, is called, and the agenda is updated accordingly. Retracting an object from the working memory might cause a logical object to lose one or several of its justifications.

## Object update

When an object is modified by Java™, the modifications are not seen by Decision Server. This means that data maintained by Decision Server will not be consistent with the new contents of the modified object.

To avoid inconsistencies, the update statement should be called for such an object. This statement allows Decision Server to update its internal structures according to the new contents of the modified object.

In the following rule, the first action modifies the number attribute. The update primitive is then called to inform Decision Server that the object has been modified.

```
rule NumberingUpdate {
 when {
 ?c: Course(department equals "history"; number > 300);
 }
 then {
 ?c.updateNumbering(); // This call modifies ?c
 update ?c;
 }
};
```

### Important:

If you do not use modify, you must use the update primitive to inform Decision Server that objects have been modified. Forgetting to notify Decision Server of modifications leads to inconsistent states.

## Attribute modification

The modify statement lets you modify the values of the attributes of objects in the working memory.

Here is an example:

```
rule ModifyLecturer {
 when {
 ?c: Course(department equals "History"; lecturer equals "Tanner");
 }
 then {
 modify ?c {lecturer = "Chen"};
 }
};
```

## RetePlus network operation

The RetePlus network indexes rules so as to minimize the number of rules and conditions that need to be evaluated whenever the working memory is changed. The network minimizes the number of evaluations by sharing tests between rules and propagating changes incrementally. When all the tests have been completed, the network designates a rule.

In most cases, executing a rule modifies the working memory. Objects in the working memory referenced by these rules can be inserted, removed, and modified. The network processes the changes inferred by the execution of the rule and produces a new set of rules to be executed.

When a change occurs in the working memory, two things can happen, depending on the nature of the change:

- If there is an insertion of an object into the working memory (positive), the rule can be executed.
- If there is a removal of an object from the working memory (negative), the rule must be removed from the agenda. The agenda is the place where the current set of rules to be executed is maintained.

This process continues cyclically until there are no further rule instances left in the agenda.

### Note:

A not condition returns true for a simple condition if the working memory does not contain any object that can match the condition.

## Example of a RetePlus network

To illustrate our description of a RetePlus network, we shall make use of a trivial filter rule, written in the rule language that refers to classes named A, B, and C:

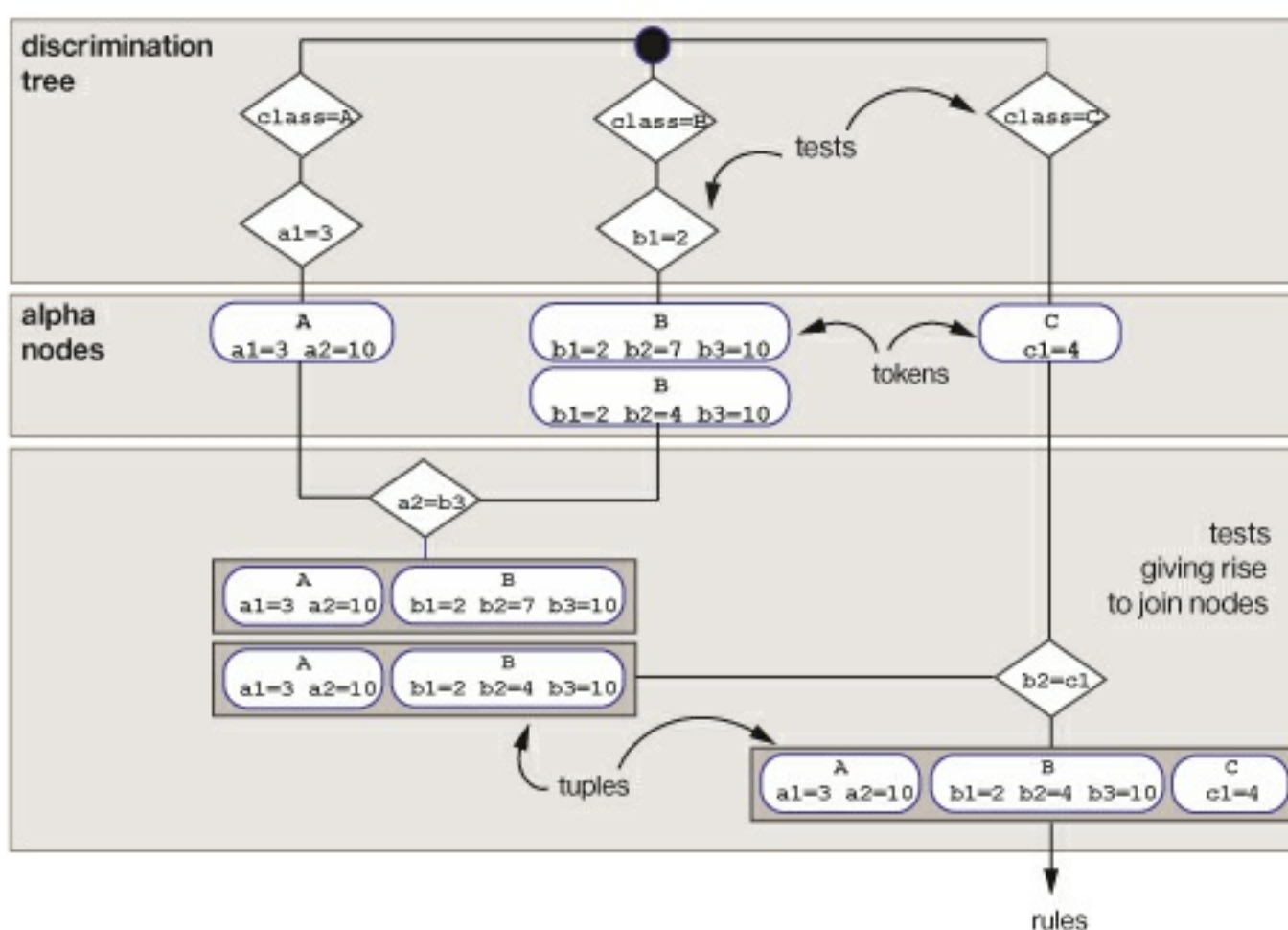
```
rule filter {
 when {
 A (a1==3; ?x:a2);
 B (b1==2; ?y:b2; b3==?x);
 C (c1==?y);
 }
 then {
 System.out.println("filter");
 }
}
```

The class attributes are the following:

- The A class has two attributes, named a1 and a2.
- The B class has three attributes, named b1, b2, and b3.
- The C class has a single attribute, named c1.

Assume that the working memory contains the following objects:

```
A(a1=3 a2=10)
B(b1=2 b2=4 b3=10)
B(b1=2 b2=7 b3=10)
C(c1=4)
```



## RetePlus network structure

A RetePlus network can be represented as a graph composed of three zones:

### Discrimination tree

A *discrimination tree* is a pattern-matching process that performs tests. These tests are represented by diamond shapes at the top of the network. The tests concern the classes of objects and the values of their attributes. Input to this tree consists of *tokens* representing each of the current objects in the working memory.

When the pattern deals with only one object and its attributes, it is said to be a discrimination test. When it is

a combination, it is called a join; these appear in the lower part of the graph.

## Alpha nodes

An *alpha node* is formed at the next level of the network, for each token that passes the tests of the discrimination tree. Each node is composed of one or several tokens, represented by round-cornered rectangles (there are three alpha nodes in the figure). One alpha node contains two B-class tokens. The other two nodes contain only one class token each—A-class and C-class tokens, respectively.

## Tests and tuples

The third zone of the network matches tokens of several classes of objects. The resulting nodes are known as *tuples*, which will be made up of several tokens. The equality test between attributes a2 and b3 gives rise to a node composed of two pairs of tokens, and the test between attributes b2 and c1 then filters out a triplet of tokens. In a RetePlus network, we often refer to tuples of this kind as *join nodes*.

## Object addition

To demonstrate further this idea of the RetePlus network, suppose an extra object is inserted into the working memory. The added object is of the C class, and the value of its c1 attribute is 7.

```
C(c1=7)
```

At this stage, when all the tests of the nodes are carried out, only the third test C (c1==?y); (class of object = C) is satisfied. The tests of the child nodes continue as the token descends through the network.

Here, there is no child node in the discrimination tree. For that reason, the object is stored directly in the alpha node.

When the token reaches the second join node B (b1==2; ?y:b2; b3==?x), the test is carried out between each B object of the pairs stored next to the memory of the join node and the C object contained in the token that just arrived at the join node. The test is satisfied for the B object of the second pair. We can therefore proceed and create a triplet formed by the second pair and the object in the token.

By forming this triplet we have, in fact, descended all the way through the network, and a new instance of the filter rule can be executed.

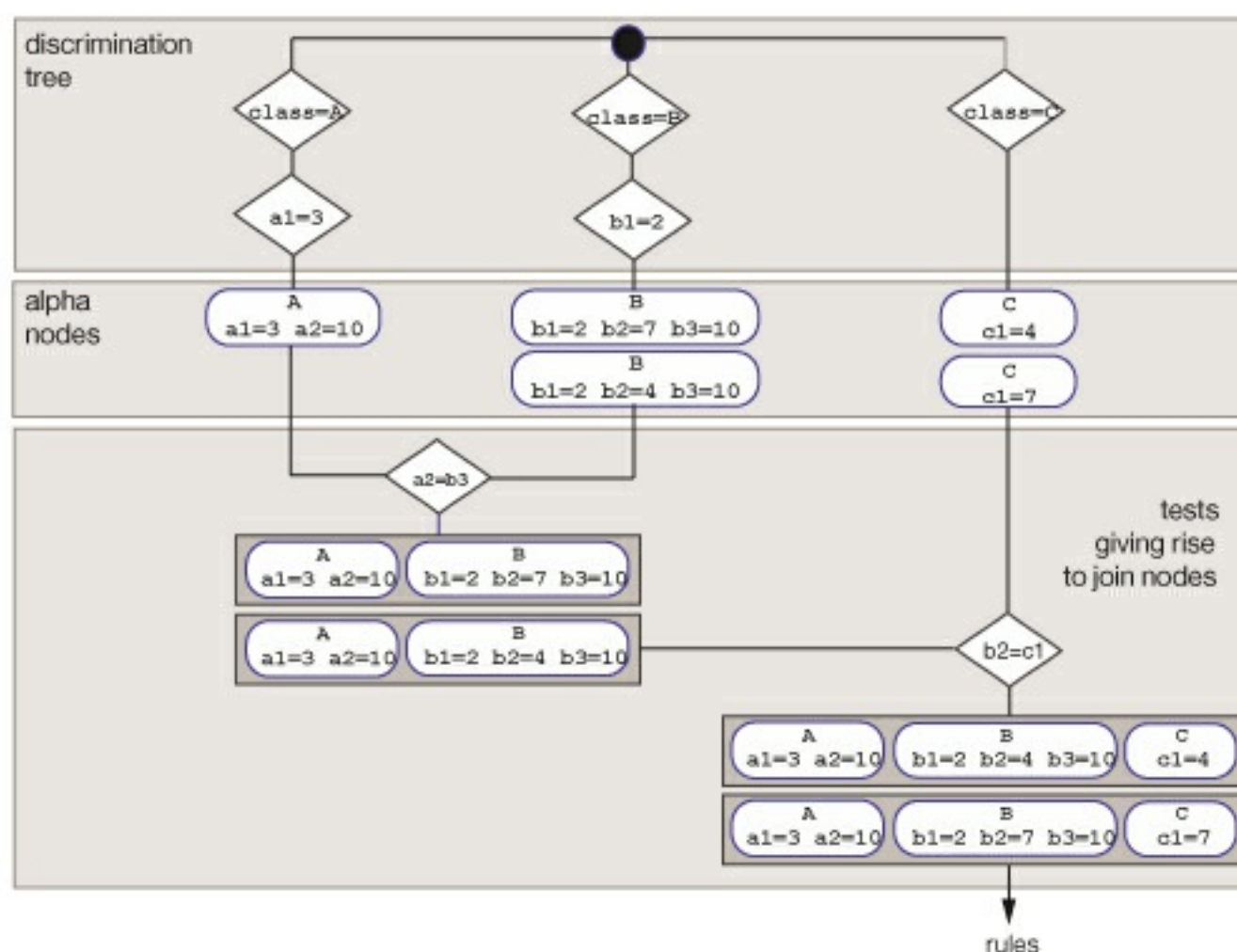
## Object removal

Suppose that we now remove the object that we just added to the working memory.

The progress of the token through the discrimination tree is the same as in the case of object addition, but the arrival of the negative token in the alpha node causes the removal of the object contained in the token.

When the token arrives at the second join node, all the triplets that contain the object in the token are removed and the token continues its progression through the network.

As in the object addition example, the rule node is reached. The instance of the filter rule whose objects satisfy the conditions corresponding to the objects in the token are removed from the agenda.







## Optimizing the object model

You can decrease the cost of pattern matching by changing the representation of the object model.

Suppose that you are filtering a set of ordered objects in which each object is identified by a rank. If you want to retrieve an object next to another just by the use of the ranks, you can write a complex rule such as the following:

### Low performing rule

The next rule expresses the following behavior: If you look for the object of rank ?n and if you find an object whose rank is greater than ?n and there is no object whose rank is between their ranks, then the next object has successfully been found.

```
rule next {
 when {
 element(?n:rank);
 ?next : element(?n2:rank & > ?n);
 not element(rank > ?n & < ?n2);
 }
 then {
 do something with ?next
 }
};
```

In such a rule, the object model and the reasoning process are not efficient. Performance is low.

### More efficient rule

To improve performance, each object keeps a pointer to the object next to it. You can easily use this representation for the pattern matching process, as in the following example.

```
rule next {
 when {
 element(?n:rank; ?next: next);
 }
 then {
 do something with ?next
 }
};
```

**Parent topic:** [Improving the performance of the RetePlus execution mode](#)

## Improving the performance of equality or comparison evaluation

Hashing expressions can improve performance at equality-test stage.

Test evaluations that involve variables from two different rule conditions can be time consuming. In the following example, the condition on Account references the variables ?p and ?b bound in the previous conditions.

### Example: Bank accounts

```
when {
 ?p:Person(age > 20);
 ?b:Bank();
 ?a:Account(name equals ?p.name; bank == ?b;
 balance < ?p.minBalance);
 ...
}
```

Without optimization, if there are 1,000 instances of Person with age > 20 and 5 instances of Bank, the three tests in the Account condition are executed 5,000 times every time a new instance of Account is added. This is because this new instance has to be tested against all the possible combinations of the ?p and ?b variables.

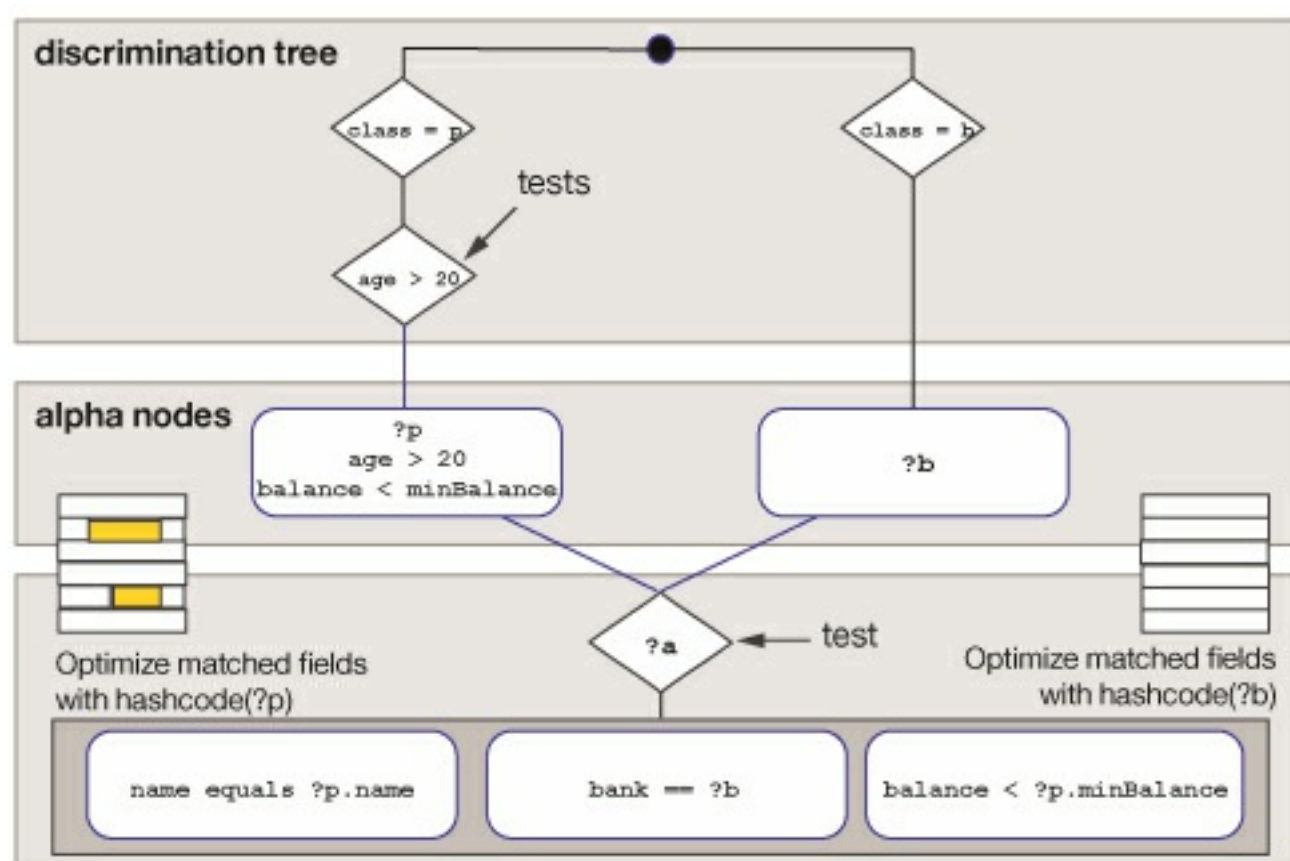
Autohashing optimization minimizes the execution of the tests based on the equality operators by relying on a hash table. A hash table is conceptually a contiguous section of memory with a number of addressable elements in which data can be quickly added, deleted, and found. Hash tables increase memory consumption for the purpose of gaining speed. They are certainly not the most memory-efficient means of storing data, but they provide very fast look-up times.

To successfully store and retrieve data from a hash table, a link is generated. That link can be mapped to one of the addressable elements of the hash table. Java™ classes implement the hashCode method to generate those links. Test execution can be reduced by a factor of 20, which is the dimension of the table that holds the links. So at best, there are 20 different entries for a value. In the Bank accounts example above, this saving would relate to 250 tests instead of 5,000 every time a new instance of Account is added.

#### Note:

Hashers always return a superset of the matching objects and the equality conditions still require a verification of the objects.

The following figure illustrates autohashing in the RetePlus network. It shows how the Bank accounts example would be handled in Decision Server.



The RetePlus network uses hashing to do operations on object equality tests such that a link is generated from that data. The link identifies which memory element of the hash table contains the satisfied equality test. In other words, the hash code is the link to the object matching the side of the test. The test is replaced by a hash code value and the corresponding object is stored in the hash table with that value. The hashing mechanism then uses the equality operator (==) to test whether two objects are equal according to the equals(Object) method. When the hashCode method is called on each of the two objects, it must produce the same integer result.

Join nodes are generated from these links inside the RetePlus network. A join node uses a single hash table. A hash table can process multiple equality tests.

To use autohashing, you need to create a configuration resource file so that the autohash mode is set to `true`. By default the autohash mode is turned off.

## Using hashing expressions

Within a ruleset header, hashing expressions can be specified on a class. Hashing expressions and autohashers are not mutually exclusive, if both are defined a class hasher always takes precedence over an autohasher.

The following ruleset header provides an example of when to use a class hasher. The class `Customer` is assigned a category, and the possible categories are defined as `int` values.

### Note:

Only methods that return an `int` can be used as a hasher.

```
ruleset CustomerRuleset
{
 hasher(Customer c) = c.getCategory();
 ...
};
```

The keyword `hasher` introduces a hasher definition. In this example, a hasher for the class `Customer` is defined. The definition says that for an object of the class `Customer` named `c`, a hasher is the `Customer`'s `Category`.

A class might have several hashers. For example, you could add this hasher to the previous header:

```
hasher(Customer c) = c.getName().length();
```

This `Customer` hasher is defined as the `length` of its `Customer.Name`.

The rule engine exploits the hashers to optimize its internal matching algorithm. Whenever the expression provided by the hasher is used in a `==` (equality operator) test, the hasher can be applied. Here are some rule conditions on which a hasher can be chosen and applied:

```
// Query customers by category
CustomerQuery(?c: getCategory());
?cus: Customer(?cus.getCategory() == c);
```

or:

```
// Query customers by the length of their names
CustomerQuery(?n: getName());
?cus: Customer(?n.length() == ?cus.getName().length());
```

For these two condition groups, the rule engine recognizes that one of the expressions on either side of the equals `==` sign matches one of the provided hashers, and it applies that hasher for optimization.

Internally, the engine classifies the instances of the class using the `int` value computed by the hashing expression.

### Note:

- There could be many tests in the condition part, and a hasher is applied when a side of a condition equals `==` test matches the hashing expression.
- Hashers cannot be used with an `equals` method.

Class hashers are more flexible than the autohashing mechanism provided through the configuration file, because you can choose the most appropriate hashing expressions with the ILOG® Rule Language (IRL).

## Using hasher properties

The ruleset hasher definition is extended with hasher properties, shown in the following property definition block:

```
hasher (Customer c) = c.age
{
 property accurate = true;
 property ordered = true;
 property final = true;
```

```
}
```

## The accurate property

The accurate property activates the internal use of more efficient hashing tables. However, this might induce greater memory consumption. Use the accurate property when the domain of the hashing expression is not too large. For example, you can expect that a `c.age` attribute has a small domain included. At the opposite end, it is likely that a population attribute of a `City` class has a large domain and should not be related to accurate tables.

## The ordered property

The ordered property indicates that the hasher is compatible with inequality test expressions. It requires that a domain be defined on the hashing expression in the XOM. The hash processing is then done on either equality or comparison expressions. Using a hashing comparison expression, you can expect to speed a test evaluation up to twice as much.

As an example, consider the activation of an ordered hasher on the `age > a.value` test in the rule `filterCustomer`:

```
rule filterCustomer
{
 when {
 a: Account();
 c: Customer (age > a.value);
 }
 then {
 ...
 }
}
```

Note that the `Customer.age` attribute must have been related to a domain in the XOM, or else the ordered hasher would not be activated on the comparison test.

## The final property

Declaring a hasher as `final` can speed up hash processing when the object set is constant or almost constant during ruleset execution. The Decision Server rule engine takes advantage of this constancy to further improve performance.

Declaring a changing hashing set as `final` does not cause execution errors or alter the behavior, but it might slow down the execution.

As an example, consider the following declaration:

```
hasher (Town t) = c.population
{
 property final = true;
}
```

The set of every town in the working memory is then supposed to be nearly constant. While any changes applied on the population of a town or insertions of new towns are possible, these operations could be more costly in execution time than if the `final` property had not been applied.

**Parent topic:** [Improving the performance of the RetePlus execution mode](#)

## Improving the performance of the sequential execution mode

To use the sequential algorithm, make sure that your rules are homogeneous and that the rules are not chained. In the sequential algorithm, you can define the number of rules to execute per tuple and the order of execution, by using control properties.

### Sequential algorithm

To guarantee the same execution with either algorithm, both the following conditions are required:

- The rules must be homogeneous. Homogeneous rules mean that the conditions of these rules must be set on the same kind and number of objects. If you try to apply the sequential algorithm to heterogeneous rules, it typically creates more tuple objects, which can result in more rules being executed than in the RetePlus mode. The common tuple signature can also cause heterogeneous rules not to fire: When there are no corresponding instances in working memory, no tuples can be created.
- The rules must not be chained, that is, the modification done in the action part of a rule must not lead to execution of another rule.

If you specify the sequential execution mode for rules that are not compliant with it, the engine raises an error.

The use of sequential execution mode is described further in the sections:

- [Tuples](#)
- [Control properties in sequential mode](#)
- [Known limitations of the sequential algorithm](#)

### Tuples

A tuple is a list of objects that complies with a particular structure. A *tuple structure* is simply a sequence of class descriptors. Each location in a tuple structure is called a *slot*. The same class descriptor can appear more than once at different slots.

Take a tuple structure:

```
(Customer,Product)
```

Here is a tuple that complies with this structure:

```
(new Customer("Henry"),new DVD("Mickey"))
```

Note that the subclassing is taken into account, since a DVD is a Product. However, some tuples that do not comply with the structure:

For example, the following tuple is too large:

```
(new Customer("Henry"),new CD("Madona"),new DVD("Mickey"))
```

This tuple is not correctly ordered:

```
(new DVD("Lord of the rings"),new Customer("Henry"))
```

And this tuple is too small:

```
(new Customer("Henry"))
```

In Java™, a tuple is easily implemented as an array of objects:

```
Java Tuple = Object[]
```

The Java code to create a tuple and its objects is:

```
new Object[] {new Customer("Henry"),new DVD("Mickey")}
```

At run time, the tuples are built automatically from the content of the working memory and then passed to the rule engine. The tuples are passed one after the other to the tuple-matching method.

### Control properties in sequential mode

Control properties affect the sequential mode execution and the tuple structure. See [Control properties for rule tasks in execution modes](#) for more general information.

The number of rules to be executed per tuple and the order in which they execute can be specified with control properties: **ordering**, **firing**, and **firinglimit**.

**The ordering property**

The ordering property is mandatory for sequential processing. It describes the order by which the rules of the rule task will be executed.

There are two values available:

- `literal`: The rules are kept in the order in which they are set into the task body (using up/down arrows).
- `sorted`: The rules are sorted according to their static priorities.

Note that another value, `dynamic`, is available only for rule tasks using the RetePlus execution mode.

**The firing property**

The firing property specifies whether, for each tuple, all the rules or only the first rule that applies should be executed. This property is optional.

It has two possible values:

- `allrules`  
This value means that all the rules that apply should be executed before skipping to the next tuple. This is the default value.
- `rule`  
This value means that only the first rule that applies should be executed on the first tuple.

**The firinglimit property**

The `firinglimit` property gives another level of control when the `firing` property is set to `allrules`. It is used to specify a number that represents the maximum number of rules to be executed before skipping to the next tuple. This number should be greater than zero.

**Known limitations of the sequential algorithm**

The following table outlines the limitations of sequential processing, in particular its use combined with certain ILOG® Rule Language (IRL) constructs.

Li mi ta ti on	Description
IL O G Rul e La ng ua ge (IR L) li mi tat io ns	<p>Not all the rule patterns that have been designed for the stateful Rete matching are available. Compile time errors occur when a rule is not compliant with the current tuple-matching implementation.</p> <p>Therefore, when sequential processing is used, the IRL does not support the following features:</p> <ul style="list-style-type: none"><li>• <code>dynamic</code> priorities</li></ul> <p>Because there is no agenda in sequential processing, only the rules with static priorities are allowed.</p> <ul style="list-style-type: none"><li>• <code>not</code>, <code>exists</code>, <code>collect</code> conditions without an enumerator</li></ul> <p>The <code>not</code>, <code>exists</code>, and <code>collect</code> conditions with an enumerator (<code>from</code> or <code>in</code>) are translated to Java bytecode. When an enumerator is specified, the set of objects is available as a value tied to the condition. (Note that this is not a limitation for RetePlus or Fastpath.)</p>
Co nd iti on co ns tru ct	<p>The following condition constructs are available in sequential processing:</p> <ul style="list-style-type: none"><li>• <code>simple condition</code>—bindings and tests are available</li><li>• <code>from</code></li><li>• <code>in</code></li></ul>

s	
Exception handling	<p>Exception handling is not supported in rule condition parts in sequential and Fastpath modes.</p>
Refraction	<p>The data unit for an engine cycle is the tuple. The engine does not record the tuples as they are matched, it rather forgets them between cycles. Therefore, there is no built-in support for refraction.</p> <p>Using the sequential execution mode, the above rule task becomes:</p> <pre>ruletask main {   algorithm = sequential;   ordering = literal;   body = {Person, PersonProduct} }</pre> <p>The execution trace shows that, using the sequential execution mode, the same rule are executed twice for the same parts of two different tuples.</p> <p>Although there is no built-in support for refraction, the sequential execution mode uses a particular rule application strategy when it generates the body of the tuple matching method. The strategy, however, tries to limit the use of refraction.</p>
IL OG Rule Language (IRL) facilities	<p>In contrast, practically all the script-level expressions and statements of the IRL are available in sequential processing. They serve principally to maintain the consistency of the other rule tasks that are not set to the sequential algorithm.</p> <p>Here is a list of these features:</p> <p><b>Functions</b></p> <p>Function definitions and calls are fully available.</p> <p><b>Script-level expressions</b></p> <p>All the IRL expressions are available. (Note that ?instance works in any execution mode.)</p> <p><b>Script-level statements</b></p> <p>All the IRL statements are available.</p> <p><b>Update</b></p> <p>The update construct is available since spurious updates might exist in rule tasks set to the sequential algorithm. A spurious update is an update on an object that cannot be matched by any condition part of rule tasks set to the sequential algorithm, but might be significant for other rule tasks using the RetePlus algorithm. All rule tasks are specified in the ruleset and should be kept consistent with the current engine. However, it is important to note that the usual update is not handled at all by the stateless sequential processing algorithm.</p> <p><b>Repeatable rules</b></p> <p>The repeatable rules are not handled by the sequential processing algorithm, but they might be relevant for other rule tasks that use the RetePlus algorithm.</p> <p><b>Insert</b></p> <p>Insertion of objects that could match condition parts of rule tasks set to the sequential algorithm might cause inconsistencies when the tuples of objects are extracted from working memory. This is because the objects are inserted in the first position, and the iterator building the tuples are not notified. The iterator does not, therefore, give the tuples involving the new object to the sequential processing engine. However, the insert is still be relevant for other rule tasks using the RetePlus algorithm.</p>

Parent topic: [Optimizing the decision engine](#)

## Improving the performance of the Fastpath execution mode

You improve the performance of the Fastpath execution mode by modifying the condition part of rules.

To improve the performance of the Fastpath execution algorithm, you can change the order of tests, or the order of conditions that are shared between several rules.

**Parent topic:** [Optimizing the decision engine](#)



# Publishing decision services to Decision Center

For business users and developers to work in collaboration, you publish and synchronize decision services from Rule Designer to Decision Center.

## About this task

To work within the governance framework, changes to the decision service in Decision Center take place in the change activities of a release. Both releases and change activities are branches of the decision service.

In Rule Designer, you can have only one branch of the decision service in your workspace at any given time. To synchronize with the different branches in use in Decision Center, connect, disconnect, and reconnect with each branch individually. When you disconnect, keep the connection entries for that branch. These entries are stored as part of the branch, allowing you to reconnect with that branch without introducing conflicts.

You must be familiar with section [Branches and releases in synchronization](#) to avoid misuse.

**Note:** For publishing to work, you must run Rule Designer in the same locale as the locale that is used to persist rules in Decision Center.

To publish a decision service:

## Procedure

1. Right-click the main project of the decision service in Rule Designer, and then click **Decision Center > Connect**.
2. Enter the URL and data source information that corresponds to your Decision Center session.  
  
In the URL, you can use the /teamserver extension, or /decisioncenter extension indifferently. If no data source is entered, the wizard uses jdbc/ilogDataSource.
3. Click **Connect** to establish the connection with Decision Center.
4. Proceed with the authentication by entering your Operational Decision Manager on Cloud credentials.
5. Optional: To limit the project elements that are synchronized, select **Use the specified query to filter rule elements for synchronization** and a query. The list of queries is defined in the decision service and its dependencies.
6. Click **Next**. In **Synchronization settings**, select **Use Decision Governance Framework**.
7. In the section **Enforce project security**, select the check box to publish your project with enforced security in Decision Center. For more information, see [Security](#).
8. Click **Finish** to publish the project.

## Results

If an error occurs during publishing to Decision Center, the **Problems** view displays a message, but the publishing completes. Publishing to Decision Center only copies the information. It does not notify Decision Center users of the changes.

### Related concepts:

[Synchronization commands in Rule Designer](#)  
[Branches and releases in synchronization](#)

### Related tasks:

[Creating projects from Decision Center](#)

### Related information:

[Decision services](#)  
[Change management](#)  
[Synchronization architecture](#)

## Adding a dependent project to a decision service

To add a dependent project to a decision service, you must add it in Rule Designer and synchronize the decision service with Decision Center.

### Procedure

1. From Rule Designer, connect to the branch or release of the decision service you want to modify.
2. Add the dependent project in Rule Designer:
  - a. If the project exists in Decision Center, import this project into your Rule Designer workspace (see [Creating projects from Decision Center](#)).
  - b. If this project does not exist in Decision Center, create a new project in Rule Designer and publish it to Decision Center (see [Publishing decision services to Decision Center](#)).
3. Add the dependency to this new project in the properties of the decision service.
4. Synchronize the decision service. You do not need to disconnect because you are already connected to it.
5. Publish the change to Decision Center. The `.ruleproject` file should be identified as changed.

### Results

If an error occurs during publishing to Decision Center, the **Problems** view displays a message, but the publishing completes. Publishing to Decision Center only copies the information. It does not notify Decision Center users of the changes.

### Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

## Managing decision services

You collaborate on the development of decision services in Decision Center.

### Governing rules with the Business console

You can govern and manage business rules with the Decision Center Business console.

### Managing decisions with the Enterprise console

You can manage business rules in the Enterprise console, which is a web user interface for Decision Center.

### Synchronizing and storing rules

You can maintain synchronized versions of rules between the business users in Decision Center and the developers in Rule Designer.

### Setting up notifications from Decision Center with webhooks

You can use webhooks to receive notifications from Decision Center to the application of your choice.





# Authoring and managing rules with Decision Center






Decision Center provides two web consoles that let you author, edit, and manage rules: the Business Console and the Enterprise Console.


The Business Console is the preferred environment for business users who manage and govern lifecycle of decisions. It allows business users to take advantage of change management and deployment of decision services.

The Enterprise Console allows IT users to take advantage of some advanced administrative features.

The following table summarizes which features are available in each console. If a feature exists in both consoles, it is recommended to use the Business Console.

Feature		Enterprise Console	Business Console
A u t h o r i n g	Creating rule projects	You can view and edit classic rule projects and decision services.	You can view and edit decision services, and create decision model services.
	Authoring rules	You can either use the guided editor or the Intellirule editor to build rules. <a href="#">Learn more</a>	Only the Intellirule editor is available to build rules. <a href="#">Learn more</a>
	Modeling decisions		You can create decision model services from scratch, or import them from IBM® Decision Composer. <a href="#">Learn more</a>
	Working with ruleflows	You can preview ruleflows but cannot create or edit them. <a href="#">Learn more</a>	You can create ruleflows and add different types of elements to control the execution of rules. <a href="#">Learn more</a>
	Working with technical rules	You can create technical rules if you have permission to do so. <a href="#">Learn more</a>	You can see and update technical rules. <a href="#">Learn more</a>
	Working with functions	You can create functions if you have permission to do so. <a href="#">Learn more</a>	You can see and update functions. <a href="#">Learn more</a>
V a l i d a t i n g	Running tests and simulations		You can validate your rules using test suites and simulations. The classic rule engine and the decision engine both support testing and simulation. <a href="#">Learn more</a>
D e p l o y i n g	Creating deployment configurations		As a permission manager, you can create and edit deployment configurations. <a href="#">Learn more</a>
	Managing servers		As a permission manager, you can manage the list of servers available to Decision Center and who has access to them. <a href="#">Learn more</a>
M	Managing		

a n a g i n g	changes using the decision governance framework		<p>You can use the decision governance framework, a predefined release workflow that is based on change and validation activities.</p> <p><a href="#">Learn more</a></p>
	Managing custom permissions		<p>The Business console proposes simplified permission profiles that you can combine with the decision governance framework.</p> <p><a href="#">Learn more</a></p>
	Managing dependencies between projects	<p>As a permission manager, you can create a project dependency to establish a reference from your project to another project.</p> <p><a href="#">Learn more</a></p>	<p>As an administrator or configuration manager, you can set up project dependencies to make a project available in different decision services or branches.</p> <p><a href="#">Learn more</a></p> <p>A decision service can contain several projects, which are linked together by default.</p> <p><a href="#">Learn more</a></p>
	Managing branches	<p>You can create and merge branches to manage the evolution of projects over time.</p> <p><a href="#">Learn more</a></p>	<p>You can create and merge branches to manage the evolution of projects over time.</p> <p><a href="#">Learn more</a></p>
	Creating decision operations		<p>You can create decision operations. Decision operations include all the settings that are needed to define the contents of a ruleset and its parameters.</p> <p><a href="#">Learn more</a></p>
	Working with dynamic domains	<p>You can update dynamic domains and update your project in the Enterprise console.</p> <p><a href="#">Learn more</a></p>	<p>You can update dynamic domains and update your project in the Business console.</p> <p><a href="#">Learn more</a></p>
	Taking snapshots	<p>You can create baselines to capture the state of a project or a branch at a specific moment in time.</p> <p><a href="#">Learn more</a></p>	<p>You can take snapshots to capture the state of a branch at a specific moment in time.</p> <p><a href="#">Learn more</a></p>
	Following streams and posting comments		<p>You can follow streams and post comments that other users can reply to.</p> <p><a href="#">Learn more</a></p>
	Using smart folders	<p>You can use smart folders to navigate through your projects according to criteria that you select.</p> <p><a href="#">Learn more</a></p>	<p></p> <p>Use the <b>Decision Artifacts</b> tab to navigate through your decision service. You can use the search bar or queries to find project elements according to specific criteria, and display the properties you want by selecting the columns in your user profile.</p> <p><a href="#">Learn more</a></p>

	Generati ng reports	You can generate project reports for classic rule projects and decision services.  <a href="#">Learn more</a>	You can generate project reports for decision services using the Decision Center REST API.  <a href="#">Learn more</a>
	Creating and running queries	You can use queries to search through classic rule projects.  <a href="#">Learn more</a>	You can use queries to search through elements of your projects.  <a href="#">Learn more</a>
<b>C o n f i g u r i n g</b>	Setting configura tion paramet ers		You can set some configuration settings from the <b>Settings</b> tab.  <a href="#">Learn more</a>
	Using build options	You can set build options.  <a href="#">Learn more</a>	You can set build options for decision services.  <a href="#">Learn more</a>
	Running diagnosti cs	As a permission manager, you can run diagnostics on different parts of Decision Center to check the state of your system.  <a href="#">Learn more</a>	As a permission manager, you can run diagnostics on different parts of Decision Center to check the state of your system.  <a href="#">Learn more</a>

**[Working with the Business console](#)**

You can govern and manage business rules with the Decision Center Business console.

**[Working with the Enterprise console](#)**

You manage decisions with Decision Center, which you can access in a web user interface, the Enterprise Console.

# Working with the Business console

You can govern and manage business rules with the Decision Center Business console.

## [Introducing the Business console](#)

To adapt to evolving business needs in their company, business users need to be involved throughout the development of a decision service. The Business console offers a collaborative and secure environment for business users and rule developers to author, manage, test, and deploy rules.

## [Managing decision services](#)

The Business console comes with features for managing decision services.

## [Decision artifacts](#)

In the **Decision Artifacts** tab, you create and edit rule artifacts, such as action rules, decision tables, or ruleflows, which you can organize in folders. You can define decision operations, ruleset variables, and handle resources for your projects.

## [Modeling decisions in the Business console](#)

Generally, IT creates the decision model on which rules are authored. For decision model services, you create both the model and the logic directly in the Business console.

## [Using queries](#)

In the Business console, you use queries to search for elements of your projects. You can apply actions to the results of the query.

## [Testing sets of rules in the Business console](#)

Test the rules that you create or edit to achieve the results that you expect.

## [Simulating business application results](#)

Run rules on representative data to generate results that you can use to improve the application.

## [Deploying from the Business console](#)

In a decision service, you can deploy a set of rules to a production environment or to a non-production environment for testing or quality assessment.

## [Administering projects from the Business console](#)

A user who has the permission manager role can access the **Administration** tab in the Business console to administer security, manage servers, or run diagnostics.

## Introducing the Business console

To adapt to evolving business needs in their company, business users need to be involved throughout the development of a decision service. The Business console offers a collaborative and secure environment for business users and rule developers to author, manage, test, and deploy rules.

In the Business console, you access decision services that contain all the elements necessary to author and manage business rules. These elements are contained in a rule repository, which is a database that is connected to Decision Center.

Changes to business rules are not reflected back automatically to the client applications that call them. They must first be deployed. In the Business console, users with the appropriate rights can deploy rules to a production environment. Other users can deploy rules to nonproduction servers to validate their changes.

The rules that you work on are organized as part of a greater set of rules, starting with a decision service, and then in a branching mechanism that handles changes over time. See [Identifying a set of rules](#) for more information.

### [Getting familiar with the Business console](#)

The Business console provides a collaborative environment for authoring and managing rules.

### [Identifying a set of rules](#)

The primary function of the Business console is to let you author business rules. A rule that you author does not exist by itself, however, and is organized as part of a greater set of rules. Identifying what set of rules you are ultimately working on is key in understanding how to manage rules.

### [Searching for rules and folders](#)

You can do a text search to find action rules, decision tables, or folders. The search can look across linked projects, and for rules that support different locales.

### [Following streams and posting comments](#)

In the Business console you can see what events are happening on releases or activities that you want to follow, and post comments that other users can reply to.

### [Accessibility](#)

You can use keyboard shortcuts and accessibility services with the Business console.

**Parent topic:** [Working with the Business console](#)



## Getting familiar with the Business console

The Business console provides a collaborative environment for authoring and managing rules.

If you are an authorized user, logging in gives you access to the **Home** page. Contact the administrator if you are unable to log in.

After logging in, the Business console presents three tabs:

- **Home:** Find useful links about the basic features of Decision Center in **Get started**, updates about your projects and followed items in **Recent activities**, **Followed Rules**, and **Rules Recently Worked On**.
- **Library:** See the available decision services.
- **Work:** See a list of all the ongoing activities in which you are a participant, with shortcuts to rename, cancel, or change the status of activities.

When entering your credentials, if you check the option **Keep me logged in**, you can close your browser window without logging out, and automatically log in as the same user the next time you access the Business console.

### Note:

Available features in the Business console may depend on your user profile. This arrangement helps ensure the full review of rules before they are deployed to a production environment.

## User profile

You can edit your user profile and select options according to your preferences. Your user options apply only to your account in the Business console. They do not affect any other users. To edit your profile, click **Profile** in the drop-down list next to the user name in the top banner of the Business console.

You can set the following options:

- Add a user photo.
- **Display:** Change your display name and select your preferred language. By default, the language is determined by a parameter in the URL that you use to access the Business console, or by the language of your browser if the URL has no language parameter. For information about how to set the browser language, refer to the help documentation for your browser.

You can change the language of the Business console to one in the list of languages, only if your development team translates the Business console environment and provides you with translated projects.

- **Grid columns:** Select and order the columns used to display the properties of rules, decision tables, ruleflows, and variable sets in:
  - The **Decision Artifacts** tab.
  - The query results in the **Queries** tab.
- **Automatically follow:** Specify what activities to follow.

## Direct links

Any URL in the Business console is accessible externally. You can bookmark these URLs in your browser, or reference them from an external source, and navigate directly to a specified branch, release, activity, or project element.

**Parent topic:** [Introducing the Business console](#)

### Related concepts:

[Accessibility](#)

[Following streams and posting comments](#)

[Managing changes with the decision governance framework](#)

[Governance principles](#)

[Rule properties](#)

[Snapshots](#)

## Identifying a set of rules

The primary function of the Business console is to let you author business rules. A rule that you author does not exist by itself, however, and is organized as part of a greater set of rules. Identifying what set of rules you are ultimately working on is key in understanding how to manage rules.

The first level of identification is the **decision service**. Rules are stored within **rule projects** contained in a decision service.

### Branches

The second level of identification is through **branches**. Starting from the rules contained in a rule project, Decision Center uses branches to manage rules over time. A branch of a rule project starts with the same rules as the parent, but then allows for a separate evolution of the same rules. Branches can be as follows:

- Releases and change activities of a decision service. Releases and change activities are **governed** branches because they are used within the decision governance framework, and they have their own characteristics (see [Managing changes with the decision governance framework](#)).
- A regular branch of a decision service, stemming from the main branch. This allows for work on a decision service without the decision governance framework.
- **Snapshots** of any branches can also be considered branches because they represent a read-only state of the rules of a branch at a past moment in time.

### Decision operations

Finally, the third level of identification is the **decision operation**. The decision operation further identifies which rules from a given branch are deployed or used for testing. Not all the rules contained in the branch that you are working on are necessarily destined to be validated or deployed. For example, you may want to test or deploy only those rules that are **Ready to be tested** or whose status is **deployable**. The decision operation defines which rules are included in the operation.

In the Business console, you choose which decision operation to use when creating a test suite, simulation, or deployment configuration. Information relating to a decision operation is visible when you select it when creating the test suite, simulation, or deployment configuration.

**Note:** For decision model services, the decision operation is automatically generated and contains all the rules of the model.

Creating decision operations is done in Rule Designer

**Parent topic:** [Introducing the Business console](#)

# Searching for rules and folders

You can do a text search to find action rules, decision tables, or folders. The search can look across linked projects, and for rules that support different locales.

## Searching for rules

The search makes a list of matching elements, and you use display filters to control the content of the list. The order of the list depends on how closely the elements match your text.

Type the text you want to find in the search bar from a release, change activity, or ungoverned branch. The search function takes into account the language of your profile, either chosen by you or the language of your browser. If the language is not supported by the searched project, the search function uses the default language of Decision Center.

If you want to find only specific elements, you can filter the search results by:

- **Type:** Filter by action rules, decision tables, or folders. Clear the **All** check box to select individual types.
- **Status:** Filter by the status property of a rule. Clear the **All** check box to select a specific status.
- **Found in:** Filter by content or properties. The search function looks for search strings in the content of rules, or only in the properties of project elements.
- **Projects:** Filter by the projects that are linked to the current one. All the projects of a decision service are linked.
- **Last edited:** Indicate a time period for the results. The filter includes calendars for selecting the start and stop dates of the period.

The search automatically updates the list of results. The elements are listed by the number of times your search string occurs in them. For example, the search lists a rule with 10 occurrences before a rule with fewer occurrences.

The search page shows the total number of project elements that match your search string. As you scroll to the bottom of the page, the search retrieves the succeeding results and appends them to the list. The page lists up to 25 entries at a time.

## Search strings and syntax

You can use words, numbers, or combinations of both. For best results, use uniquely identifiable search strings. The search ignores articles (a, an, the...), prepositions and conjunctions (and, or, of...), and words that are common to most rules (if, then, else...).

### Multiple search strings

If you enter more than one string of characters, the search looks for elements that contain all the strings, and the strings individually. For example, if you enter the word loan, the search looks for only those rules that contain the word loan. If you enter loan and duration, the search looks for all the rules that contain both words, and the words individually.

### Names

You can search for elements by file name or author. When you search by author, you can use the user name or display name. The user name is the name that is used to sign in to Decision Center, while the display name is the name displayed with messages and elements.

### Types of text and syntax

The search lists the results that are based on how closely they match your text. It lists the project elements with all the strings before the projects with individual instances.

You can type a string that consists of letters, numbers, or both into the **Search** field. You can search for a whole string or just part of a string. You can also use special syntax to prioritize or exclude text, or to search for an exact match.

The following table lists the types of text and search syntax that you can use in search strings.

Table 1. Types of text and search syntax

Text or search syntax	Description
Letters	Individual letters or groups of letters that do not form words. For example, WYSIWYG.
Words	A complete word or words. For example, loan, miniloan, or user name.
Uppercase and lowercase	The search ignores case. You can use uppercase and lowercase letters together. For example, New Loan.

Num bers	One or more numbers. For example, 1, 2 30, or 100.
Alpha nume ric comb inatio ns	Combinations of letters and numbers. For example, Y2K or Name123.
Priorit izing text with the plus (+) modif ier	You can type a plus sign (+) in front of a word to have the search look only for elements with this word. The search treats any other word in the search field as optional. For example, if you type loan +duration, the search lists all the elements that contain the word duration, and any element that contains both loan and duration. It does not list elements that contain loan but do not contain duration.
Exclu ding text with the minu s (-) modif ier	You can type a minus sign (-) in front of a word to have the search ignore all the elements that contain the word. When the search finds the word, it does not list the element with the word on the search page. For example, if you type loan - duration, the search lists only the elements with the word loan, and excludes any element that contains both loan and duration.
Exact matc hing with quota tion mark s (" ")	By placing quotation marks (" ") around a search string, you have the search match the string exactly. For example, a search for "mini loan rule 3" lists elements with only the quoted text.

**Parent topic:** [Introducing the Business console](#)

## Following streams and posting comments

In the Business console you can see what events are happening on releases or activities that you want to follow, and post comments that other users can reply to.

### Streams

The Business console provides a social stream feature that you use to follow work and interact with others.

Events relating to a release or activity are listed in the **Stream** view, available next to the **Properties** tab when you display this release or activity.

An event is when someone, including you, does tasks such as:

- Create, edit, or delete a rule or folder.
- Create or change the state of a release, change activity, or validation activity.
- Create or restore a snapshot.
- Post a comment.

Another **Stream** view is available on the **Home** page. This stream includes only the events that you subscribe to follow.

### Following events

The stream on the **Home** page displays events that you subscribe to follow, and any general posts. A golden star next to the name of an item (release, change activity, folder, or individual rule) means that you are subscribed to it. An empty star means that you are not.

Depending on the type of item you follow, you are notified of the following information:

#### Release

Important changes to the release, such as status changes, approval, cancellation, or rejection, and the main changes to the activities of the release.

#### Change or validation activity

Main changes to the activity, such as status changes and rule changes in the activity.

#### Folder

Changes to all the rules that are currently contained in the folder.

**Note:** To follow the entire folder, you must select all the rules of the folder, and click the star.

#### Individual rule

Follows events on the rule across all decision services, releases, or activities in which that rule is contained.

To reduce the number of manual steps that are required to subscribe to events, options are available to automatically follow certain events, such as rules you edit or releases or activities that you are an approver of. You can enable or disable these and more options by clicking **Profile** in the drop-down list next to your user name in the top banner.

### Posting comments

You can post comments in all **Stream** views:

- **On the Home page:**

Comments posted here are visible to all Business console users by default, and can include web links and file attachments. You can select **Private Posting** to restrict who can see the post. When you click **Post** with this option enabled, you can select the release or change activity. Only users that have access to these releases or change activities see your post.

- **When displaying or editing a release, activity, or rule:**

Comments posted here are visible to users that have access to the release, activity, or rule.

You can reply to any post, from either one of these views, by adding a comment to the post.

**Parent topic:** [Introducing the Business console](#)

**Related concepts:**

[Getting familiar with the Business console](#)

# Accessibility

You can use keyboard shortcuts and accessibility services with the Business console.

You use the Business console in a web browser. Mozilla Firefox, Microsoft Internet Explorer, and other browsers provide keyboard and mouse shortcuts for navigating web pages. They also provide features for the visually impaired, such as screen magnification, and color and font control. You can use these browser controls with the Business console.

You can also use the console with screen readers, which read the text aloud and can magnify it for easier viewing. Screen readers also provide navigation shortcuts.

## Special keyboard shortcuts

The following tables list actions and their keyboard shortcuts in the Business console.

Table 1. Rule editor

Action	Keyboard shortcut
Open the completion menu.	Ctrl+Spacebar
Open the completion menu in a tree view.	Ctrl+Shift+Spacebar
Open the next placeholder.	Alt+Right Arrow key
Open the previous placeholder.	Alt+Left Arrow key
Open a placeholder under a caret.	Alt+Down Arrow key
Open the menu of completion options in the toolbar.	Alt+O
Insert a tab space.	Tab
Format the text.	Ctrl+Shift+F
Undo	Ctrl+Z
Redo	Ctrl+Y
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Move the focus forward from the editor to the error grid.	Ctrl+Shift+Down Arrow key  When you edit a rule, press Tab to insert a tab space.
Move the focus backward from the editor to the toolbar.	Ctrl+Shift+Up Arrow key  Shift+Tab also moves the focus from the text editor to the editing toolbar.
Filter the contents of the completion menu.	As you type, the contents of the completion menu are filtered by their prefixes.

Table 2. Rule editor completion controls

Action	Keyboard shortcut
Close the completion menu.	Esc
Toggle the filter mode.	Ctrl+F
Scroll through the completion menu.	Up and Down Arrow keys
Page through completion menu.	Page Up and Page Down keys
Select the next or previous entry.	Home or End keys
Collapse or expand a tree node.	Left or Right Arrow keys
Insert the selected content.	Enter or Tab

Table 3. Decision table navigation

Action	Keyboard shortcut
Move the selected cell up, down, left or right among the table cells.	Up, Down, Left or Right Arrow key
Move the selected row up or down among the table rows.	Up or Down Arrow key
Move the selected column left or right among the table columns.	Left or Right Arrow key
Extend the selected cell left or right from the currently selected cell.	Shift+Left or Right Arrow key
Extend the selected row up or down from the currently selected row.	Shift+Up or Down Arrow key

Extend the selected cell to all the cells from the currently selected cell.	Ctrl+A
Extend the selected row to all the rows from the currently selected row.	Ctrl+A
Reset a decision table to its original layout.	Ctrl+Alt+P <div><b>Note:</b> Only available in automatic row ordering mode.</div>
Close the pop-up menu of the selected cell, column, or row if still open	Esc
View the definition and any errors of a selected cell, row or column.	F9
Select the column of the currently selected cell.	Ctrl+Spacebar
Select the row of the currently selected cell.	Shift+Spacebar
Toggle the column sort.	Spacebar  The column must first be selected by using the Ctrl+Spacebar. <div><b>Note:</b> Sorting is not available for action columns in manual row ordering mode.</div>
Make the selected column narrower by 5 pixels.	Ctrl+Shift+Left Arrow key
Make the selected column wider by 5 pixels.	Ctrl+Shift+Right Arrow key

Table 4. Decision table editing

Action	Keyboard shortcut
Edit the selected cell.	Enter
Copy a selected item.	Ctrl+C or Cmd+C
Cut a selected item.	Ctrl+X or Cmd+X
Paste a copied or cut item to another location in a table.	Ctrl+V or Cmd+V
Insert copied or cut cells.	Ctrl+Shift+V or Cmd+Shift+V
Delete the content of the selection.	Delete
Undo	Ctrl+Z or Cmd+Z
Redo	Ctrl+Y or Cmd+Shift+Z
Show the menu for a row header, column header, or cell.	Shift+F10  The table element must first be selected.
Edit the column header title.	Enter  The column must first be selected.
Edit the column definition.	Ctrl+Alt+A  A column must be selected.
Delete the selected column or selected row or rows.	Ctrl+Delete
Group rows.	Ctrl+Alt+P
Enable or disable an action cell or cells.	Ctrl+Q
Split	Ctrl+L  Only available for condition cells in manual row ordering mode.
Merge	Ctrl+G  Only available for condition cells in manual row ordering mode.

Table 5. Rule detail dialog

Action	Keyboard shortcut
--------	-------------------

Move through the active buttons.	Tab and Shift+Tab  Use the Down Arrow key to activate fields and edit them.
Activate a button.	Down Arrow key  The JAWS virtual PC cursor can read only the active buttons. To select a button, navigate to it by using Tab and Shift+Tab.

Table 6. Timeline navigator

Action	Keyboard shortcut
Move in the timeline navigator.	Tab and Shift+Tab
Set the focus on a timeline item in the timeline view.	Enter  Sets the focus on the first item in the month.

Table 7. Timeline view

Action	Keyboard shortcut
Move through the active item links.	Tab and Shift+Tab
Jump through timeline items.	Left and Right Arrow keys
Scroll through a timeline.	Up and Down Arrow keys
Open the snapshot dialog from a selected item.	Shift+Left Arrow key
Move the focus back to the matching month.	Esc
Open a rule or snapshot in a project timeline.	Press Tab to the hotlink inside the box to open the rule or snapshot.

Table 8. Project view of rules by folder

Action	Keyboard shortcut
Move through the folders.	Up and Down Arrow keys
Move through the rules.	Up and Down Arrow keys
Collapse a folder.	Left Arrow key
Expand a folder.	Right Arrow key
Select a folder or rule.	Click the first character of the folder or rule name.
Move through folder actions.	Left and Right Arrow keys  Press Enter to execute an action, or Esc to exit the drop-down menu.

Table 9. Release or Activity Properties view.

Action	Keyboard shortcut
Move into the Properties area from the top Properties tab.	Tab
Move through the editable property fields and buttons.	Left and Right Arrow keys
Move through the list sections.	Tab and Shift+Tab
Open and collapse a list section.	Spacebar
Move through the users in a list section.	Up and Down Arrow keys
Move through the editable user fields and buttons.	Left and Right Arrow keys
Open an editor in an editable field.	Enter
Leave an editor.	Enter* or Esc  *With some editors, you must press Enter twice.

Table 10. Validation Activity Test Plans view

Action	Keyboard shortcut
Move into the header section.	Tab
Move through the column headers.	Right and Left Arrow keys
Sort a column.	Enter or Spacebar
Move into the grid data section.	Tab



Move through the test plans.	Up and Down Arrow keys
Edit highlighted test plan.	Enter
Delete highlighted test plan.	Delete

**IBM and accessibility**

For more information about the commitment that IBM® has to accessibility, see [IBM Accessibility](#).

**Parent topic:** [Introducing the Business console](#)

**Related concepts:**  
[Getting familiar with the Business console](#)

# Managing decision services

The Business console comes with features for managing decision services.

## [Decision services](#)

A decision service contains one or more rule projects. Each rule project contains action rules and decision tables that you update directly. You can easily move among the projects to make changes, and deploy the decision service to a test or production environment.

## [Managing branches](#)

Decision Center enables branching so that you can manage the evolution of projects over time.

## [Using governance with decision services](#)

The decision governance framework is a methodology that provides you with tools to implement changes to your business rules in a structured, secure, and controlled way.

## [Linked projects](#)

When a decision service is published to Decision Center, it can contain linked (or dependent) projects.

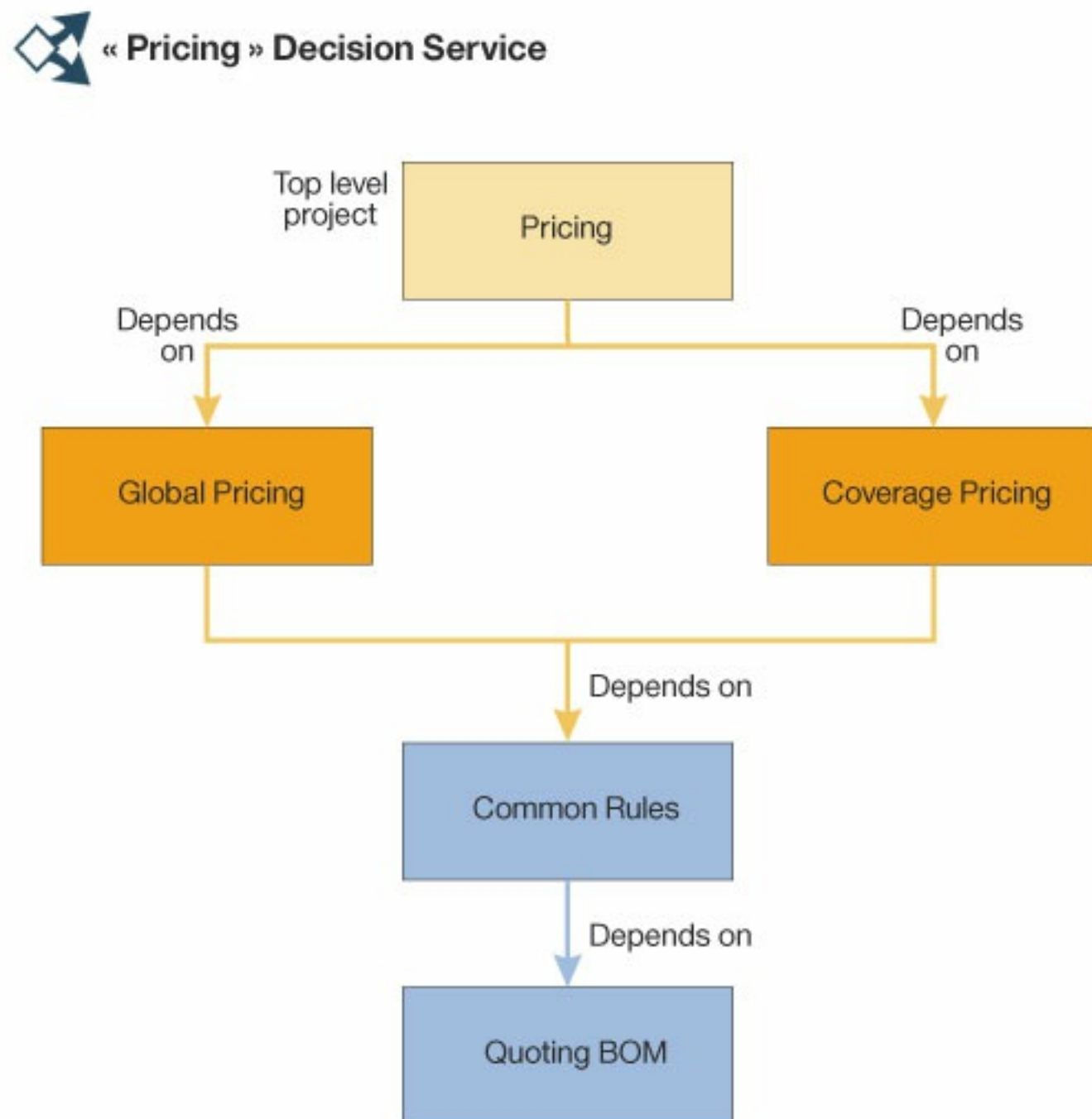
**Parent topic:** [Working with the Business console](#)

## Decision services

A decision service contains one or more rule projects. Each rule project contains action rules and decision tables that you update directly. You can easily move among the projects to make changes, and deploy the decision service to a test or production environment.

**Note:** A decision model service is also available, as described in [Modeling decisions in the Business console](#).

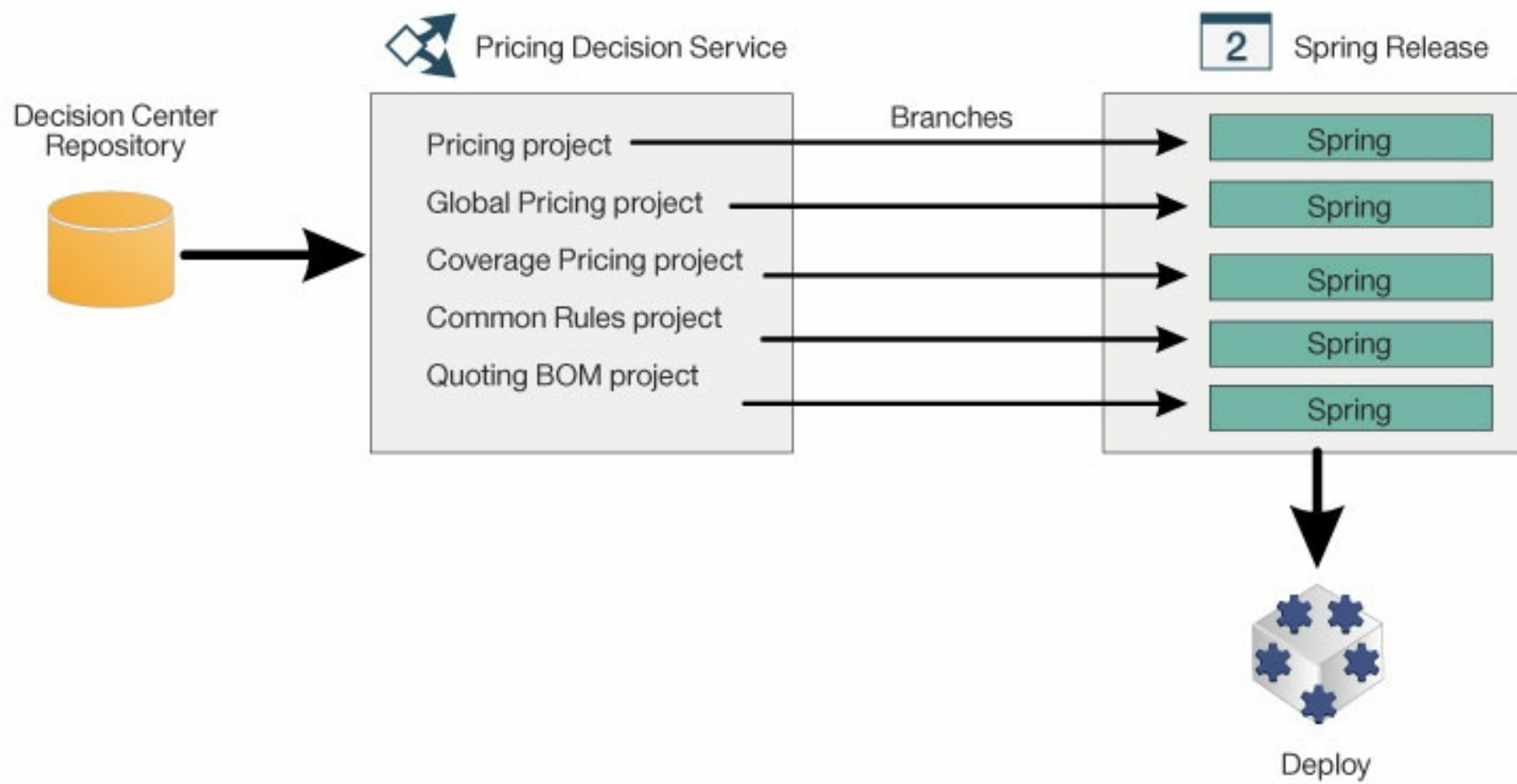
The following diagram shows an example of a decision service, with a top-level project and its dependent projects:



### Lifecycle of a decision service

A decision service is published to Decision Center, and appears in the Business console library. Each decision service contains an initial release, which cannot be changed, and subsequent releases. Changes to a decision service are managed in releases and activities.

Each release is made up of a branch of each project that is contained in the decision service. For example, the following diagram shows that the Spring release of the Pricing decision service contains the Spring branch of each project that is contained in the decision service:



When a release is fully developed and approved, it can be deployed to a production environment.

A permission manager can delete a decision service from the **Library > Decision services** page in the Business console. This action permanently removes all the projects, branches, releases, and associated entries of the decision service in the database, and it no longer appears in Decision Center.

**Parent topic:** [Managing decision services](#)

**Related concepts:**

[Deploying from the Business console](#)

[Governance principles](#)

# Managing branches

Decision Center enables branching so that you can manage the evolution of projects over time.

## What are branches

When a branch is created, it contains an exact replica of every project element that is contained in its parent branch. You can then work on the subbranch without affecting the contents of the parent.

## Snapshots

Snapshots capture the state of a branch at a specific moment in time.

## Merging branches

In the Decision Center Business console, you can merge the content of branches, releases, and activities in a decision service.

## Deleting and recovering elements in branches

When you delete project elements from the current state of a branch, they are put in the recycle bin of that branch. You can restore those elements to your branch.

## Generating a project report

You can generate reports in HTML format based on decision services, projects, or queries. These reports show the content and properties of deployed project elements.

**Parent topic:** [Managing decision services](#)

# What are branches

When a branch is created, it contains an exact replica of every project element that is contained in its parent branch. You can then work on the subbranch without affecting the contents of the parent.

There are several types of branches in Decision Center:

- The releases and activities of a decision service. Releases and activities are used with the decision governance framework, a prescriptive workflow to implement change management within Decision Center. They are **governed** branches, and have their own characteristics (see [Managing changes with the decision governance framework](#)).
- A regular branch of a decision service, stemming from the main branch. This branch allows for work on the decision service without the decision governance framework.
- **Snapshots** of any branches can also be considered branches because the snapshots represent a read-only state of the rules of a branch at a past moment in time.

When the initial version of a decision service is created, published, or imported, both a **main** branch and an **Initial Release** branch are created. The main branch is available if you want to work in a decision service but not use the governance framework. The Initial Release branch is published as the closed initial release of the decision service.

In the Business console, you can create new branches off the main branch, and rename them. If you use the governance framework, you can create new releases with any closed release as a starting point. The owner of a release can then create change and validation activities. You can also copy an open release and its content with any closed release as a starting point.

You can delete any branch, except for the main branch. To delete a release, you must have a Permission manager role. The release owner can also delete a release, if it is not in a *Complete* state.

**Note:** You might want to delete releases you do not use anymore, but be aware that all child releases, including the ones in progress, are also deleted. To keep your releases in progress, you can copy them from the **Releases** tab when you access the decision service. The copies are all created under the initial release, as child releases. That does not impact their content.

Merging of branches is done automatically within the context of the governance framework, but you might also be required to merge release and change activity branches, if changes to one need to be pushed to the other, or if the automatic merging fails due to conflicts. For these cases, you must consider the following actions:

- You can merge a change activity into any other release in the decision service when the change activity is completed and the release is not already completed.
- You can merge a release into a change activity in the decision service when the change activity is not already completed.
- You cannot merge a release with another release in that decision service.
- You can merge a change activity with another one in the decision service when the target change activity is not already completed.

**Parent topic:** [Managing branches](#)

**Related tasks:**  
[Managing subbranches and baselines](#)

# Snapshots

Snapshots capture the state of a branch at a specific moment in time.

You can create snapshots that are based on the current or past state of a branch, including releases and change activities.

There are different ways to take snapshots:

- You can take a snapshot of the current state of a branch by clicking **Take Snapshot** within the branch.
- Decision Center automatically creates a snapshot of a release or change activity when you create it, or of a release when a change activity merges with the release.
- You can take a snapshot that is based on what the state of a branch was at a previous moment in time. To do so, follow this procedure:
  1. Go to the branch and click **Timeline**.
  2. Make sure that you are showing **All** events and that no filters are applied.
  3. In the timeline, scroll to the time that you want to capture in the snapshot.
  4. Hover along the vertical axis of the timeline, which separates the timeline events.
  5. Click the snapshot icon that appears on the vertical axis. Enter a name and a description and click **Create**.

You can consult snapshots, and compare them with other snapshots of this branch, or with the current state of the branch. If you have the appropriate permissions, you can rename or delete existing snapshots from the **Snapshots** tab. When you consult a snapshot, you cannot edit its contents.

You can restore a snapshot so that it becomes the current state of a branch. To restore a snapshot of a release, you must be the owner of the release or an administrator. The release must be in progress, and all the activities of the release must be complete or canceled.

In addition to the snapshots that you create, Decision Center automatically creates some snapshots. For example, when you create a new change activity, Decision Center makes a snapshot of the initial state of the activity. Also, when a change activity is completed, Decision Center automatically takes a snapshot of the parent release before merging the contents of the change activity.

Finally, deployment snapshots are a special type of snapshot, which can be automatically taken at the moment of deployment, that capture the state of the rules at the moment of deployment. Deployment snapshots appear in the list of snapshots, and can be used to redeploy.

**Parent topic:** [Managing branches](#)

**Related concepts:**

[Getting familiar with the Business console](#)

**Related tasks:**

[Creating a snapshot](#)

[Redeploying](#)

[Viewing the timeline of a release or activity](#)

# Merging branches

In the Decision Center Business console, you can merge the content of branches, releases, and activities in a decision service.

From one of the branches you want to merge, click **Merge Branches** in the toolbar, and select the branch you want to merge with. After you specify the branches to be merged, Decision Center displays a table with the projects that were modified. You can expand each project to see the name and folder of project elements that are different between the branches, and their state in both branches. This table also contains an **Actions** column, containing a proposed action to take. You can click inside the cells of this column and decide what to do with each project element that is different between the two branches:

- Update the working branch with the modifications made in the branch you selected in the **Choose Branch to Merge With** dialog
- Update the branch you selected in the **Choose Branch to Merge With** dialog with the modifications made in the working branch
- Take no action at all

The elements taken into account during a merge operation are:

- Rule Artifacts (BAL rules, technical rules, decision tables, decision trees, functions)
- Simulation artifacts (metrics, KPIs, simulation models, input data, simulations)
- Testing artifacts (test suites, test cases)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- BOM
- Vocabulary
- B2X
- Resources
- Folders

**Note:**

The following changes cannot be merged:

- For classic rule projects: Changes to ruleset parameters, and extractors. You must change these manually in the other branch.
- For decision services: Changes to decision service properties, categories, queries referenced in decision operations, dynamic XOM (schemas), configuration file for the classic rule engine, and the engine mode. You must change them manually in the release branch.

For both classic rule projects and decision services, project dependencies cannot be merged either.

To help Decision Center propose the correct action to take, you can specify your preferred direction for merging from the following options:

**Bidirectionally**

This is the default option. The proposed action is based on the assumption that you want to reflect changes made in either branch as follows:

- If a new rule exists in one branch, you want to add it to the other branch.
- If a rule has been deleted in one branch, you want to delete it from the other branch.
- If a rule has been modified in one branch but not in the other, you want to update the unmodified one.
- If a rule has been modified in both branches, no action is proposed, and you must specify what action to take.

**Only to the branch you merge with**

When you want all changes you made in the working branch to be pushed to the branch you are merging with, but none of the changes made in that branch to be reflected in your working branch.

**Only to the working branch**

When you want all changes made in the branch you are merging with to be reflected in your working branch, but none of the changes made in your working branch to be pushed to the other branch.

You can compare the different versions of a project element by hovering your cursor over it, and clicking **Show compare view**.

After you click **Apply Merge** in the upper right corner, you can add a comment, and select the option to



create a snapshot of the branches before you merge them. When the merge operations are completed, a report is displayed where you can review the changes made. If you created a snapshot of the branches before merging, you can also review these changes later, by comparing the snapshot with the current version of the branch.

**Parent topic:** [Managing branches](#)

## Deleting and recovering elements in branches

When you delete project elements from the current state of a branch, they are put in the recycle bin of that branch. You can restore those elements to your branch.

You cannot edit an element in the recycle bin, but if you have sufficient permissions to edit in a change activity, validation activity, or ungoverned branch, you can restore it so that it returns to the current state of the branch. The recycle bin of a release is in read-only mode for all users, you must go in a change activity to restore elements.

The recycle bin also displays deleted folders. If you restore a deleted folder, you do not also restore the rules it contains. You must restore them explicitly.

Some artifact views in the recycle bin might display missing information or errors. This happens when a deleted object has a reference to an object that is not deleted. For example:

- When a deleted deployment configuration references an operation that is not deleted, the operation is not found in the recycle bin and cannot be displayed.
- When a deleted operation references a variables in a variable set that is not deleted.
- When the BOM is not found in the recycle bin, a deleted variable shows errors.

**Note:**

- You cannot restore elements in a closed activity.
- You can only restore elements in the recycle bin, you cannot permanently delete them.

**Parent topic:** [Managing branches](#)

## Generating a project report

You can generate reports in HTML format based on decision services, projects, or queries. These reports show the content and properties of deployed project elements.

To generate a project report in HTML format:

1. Select your decision service, and the branch or release/change activity for which you want to view the content.
2. Click **Reports** in the top toolbar to view the **Reports** tab. You can also select **Queries > Reports**.
3. From this view, you can generate a full report of your decision service, or a report of a project in your decision service.
4. To generate a report based on a query or a project, first open the **Queries** tab and select the query or the project you want. Then go back to the **Reports** tab, where you can now generate a report based on this query or project.

If you generate a report based on a query, this query must have been saved previously. If you are editing it and want to create your report from this new version, save the query before generating the report.

The report is generated in another browser window, and you can download it. It contains the details of the following deployed artifacts:

- Action rules
- Decision tables
- Ruleflows
- Technical rules
- Functions
- Variable sets
- Operations

The report does not show resources.

On Mac OS, you need to set the Java™ VM argument `-Djava.awt.headless=false` to generate ruleflow diagrams. See the limitation "In the Enterprise console on the Mac OS, the ruleflow viewer triggers an exception" in [Operational Decision Manager Known Limitations](#).

**Note:** The maximum size of the report that you can generate is 100 Mb by default. If you reach this size limit, the report will be incomplete. To be able to view all your content, you can generate several reports based on your projects or queries.

**Parent topic:** [Managing branches](#)

## Using governance with decision services

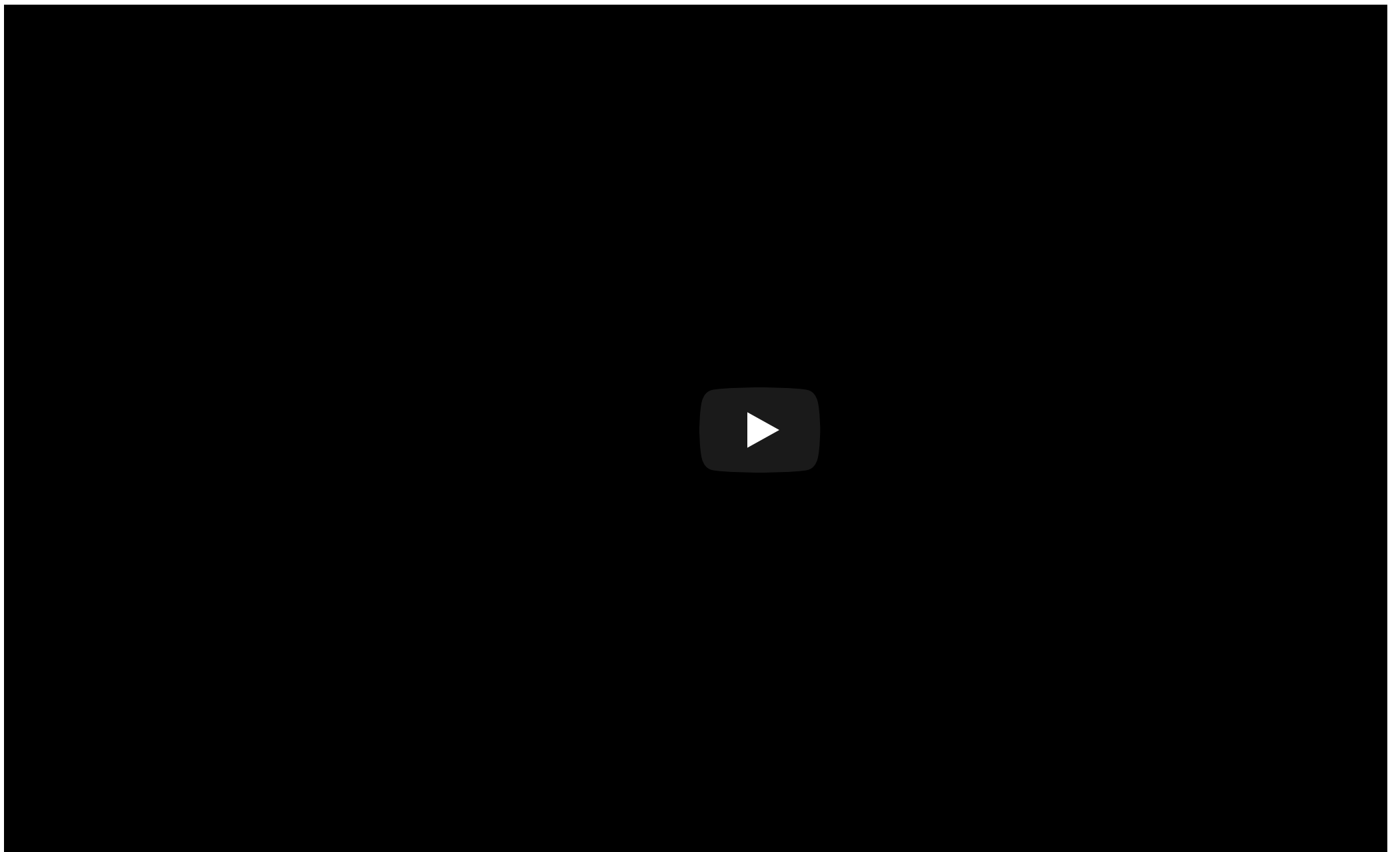
The decision governance framework is a methodology that provides you with tools to implement changes to your business rules in a structured, secure, and controlled way.

It is based on the management of releases and activities, and the distribution of tasks between users with specific governance roles. To understand the main aspects of governance, see [Governance principles](#).

Typically, the purpose of a release is to implement one or more changes to the business policy. The release is composed of change activities, and each change activity usually contains changes to a single business policy. Releases can also contain validation activities, which are used for testing and validating the change activities within the release. To know more about how to manage changes through releases and activities, see [Managing changes with the decision governance framework](#).

This structure for managing changes is enforced by a review cycle. When changes are made to an activity or release, they need to be reviewed and approved, before the activity or release can be marked as complete and deployed. This ensures that only users with the proper responsibilities can approve a release and deploy it.

Decision governance can be helpful, especially for larger teams, because it facilitates the decision-making process and makes changes easier to trace and review. You can have a look at what this feature looks like and how it is used in the following demo video.



To delve deeper into how decision governance works with Decision Center, see the [IBM® Governance Redbook: Governing Operational Decisions in an Enterprise Scalable Way](#).

### [Governance principles](#)

The governance aspects of Decision Center are based on the states of releases and activities, and on the governance roles of participants who work on these releases and activities.

### [Managing changes with the decision governance framework](#)

In Decision Center, you can manage changes to rules over time through the creation and management of branches. The recommended way of managing changes is by using the decision governance framework. With this approach, you work through releases and activities within a decision service.

**Parent topic:** [Managing decision services](#)

# Governance principles

The governance aspects of Decision Center are based on the states of releases and activities, and on the governance roles of participants who work on these releases and activities.

## State

The state of a release or activity can be one of *In Progress*, *Ready for Approval*, *Complete*, or *Canceled*. The state of a release can also be *Rejected*. The state of a release or activity dictates what work can be done and by whom.

## Governance Role

All releases and activities have an *owner* and an *approver* role. Change activities also have an *author* role, and validation activities have a *tester* role. Users with administrator privileges can carry out the governance operations of all roles, and the owner of a release or activity can carry out some of the operations on behalf of participants with other roles.

When you create a release or a change activity, Decision Center takes a snapshot to record the starting state of the release or activity. A transition from one state to another can generate automatic snapshots or merges. Most notably, when all the approvers approve a change activity, Decision Center takes a snapshot to record the state of the rules, and then merges the changes back into the release.

## Release governance

The user who creates a release:

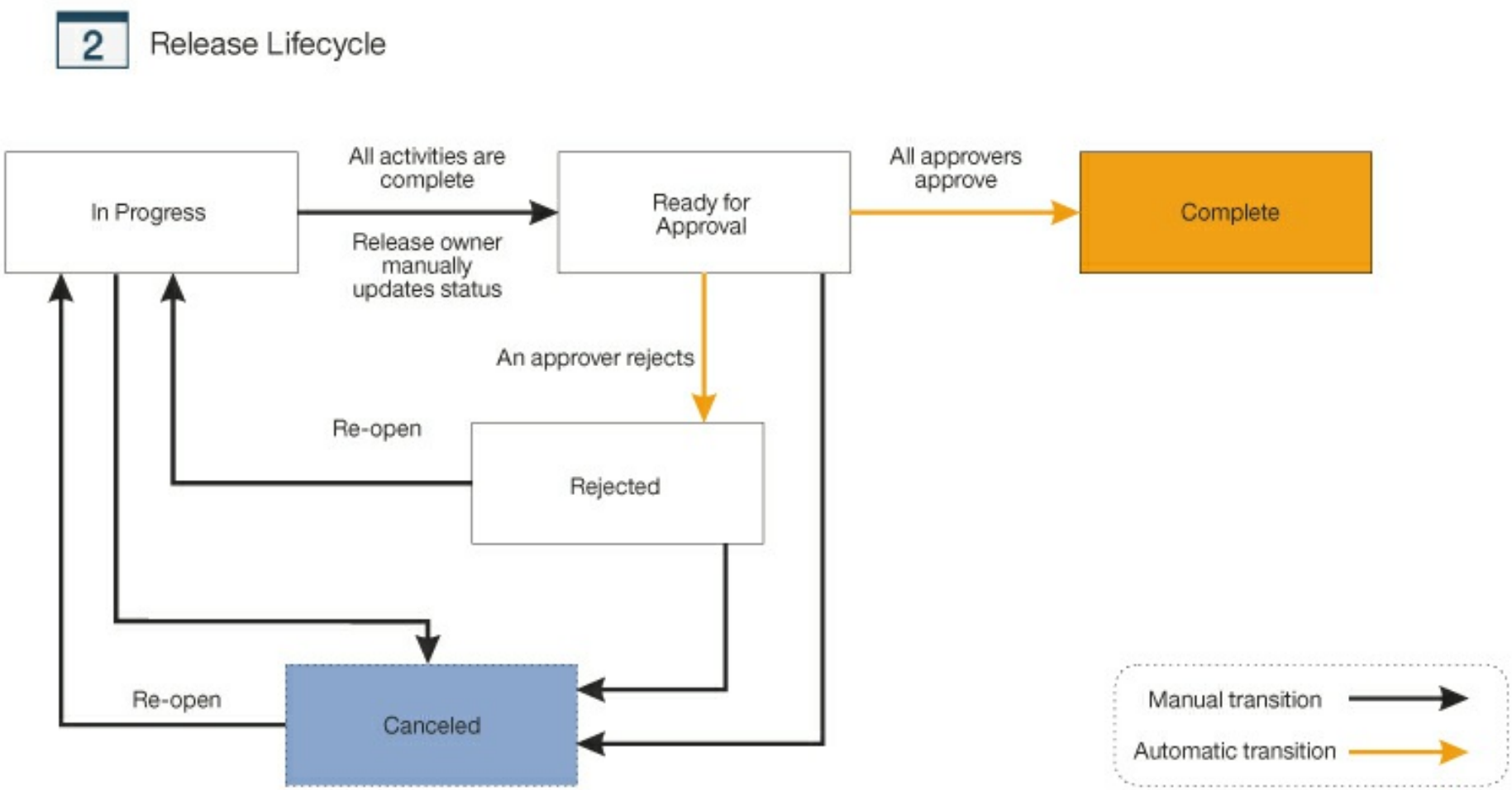
- Sets the owner of the release.
- Sets the goals of the release.
- Sets the date when the release must be completed.
- Assigns one or more participants as approver of the release.

As long as the release is in the *In Progress* state, the owner can perform the following actions:

- Change the owner of the release.
- Change the goals of the release.
- Change the due date of the release.
- Create change and validation activities.
- Delete the release

When all the activities of the release are complete, the release owner changes the state of the release to *Ready for Approval*. The approvers can then approve or reject the changes to the release.

The following diagram shows the lifecycle of a release, with manual transitions being done by the owner of the release:



The following table shows the most frequent user operations on a release, the role that is required to do the operation, and the state of the release afterward:

User operation	Role	Release state after operation
Create release	-	In Progress
Cancel	Owner	Canceled

Cancel release		Canceled
Reopen release	Owner	In Progress
Proceed to approval	Owner	Ready for Approval
Reject changes	Approver (or owner on behalf of approver)	Rejected
Approve changes	Approver (or owner on behalf of approver)	Complete
Delete a release	Owner or Permission manager <div> <p><b>Note:</b> The owner can delete only releases that are not <i>Complete</i>.</p> <p>The Permission manager can delete releases in any state, except for the initial release.</p> </div>	-

### Change activity governance

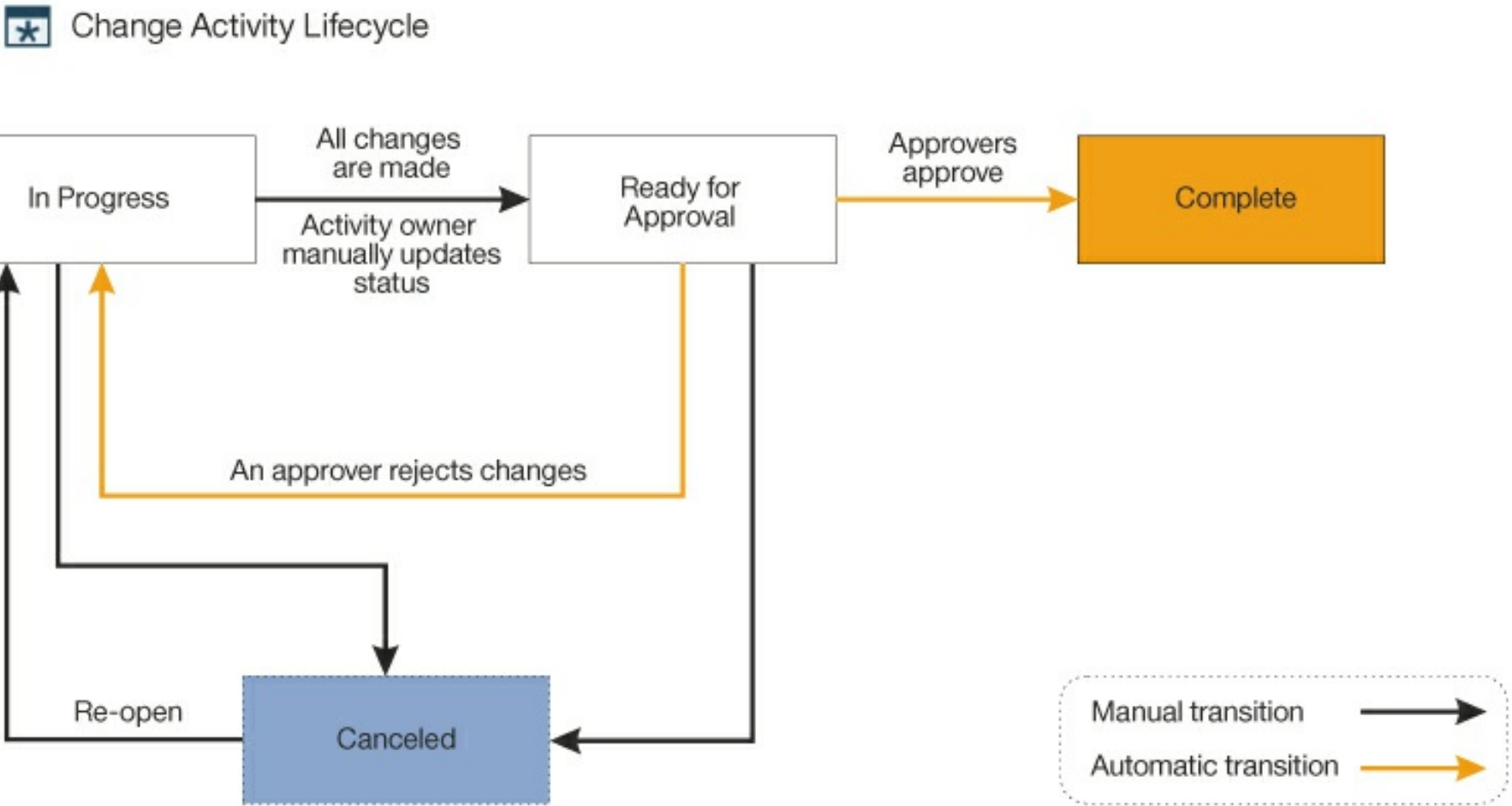
Then, as long as the change activity is not *Complete*, the owner of the activity can do the following actions:

- Change the owner of the activity.
- Change the due date of the activity.
- Change the goals of the activity.
- Assign approvers and authors to the activity.

Authors edit rules in a change activity. When they finish their work, authors change their status to **Finished**. When all the authors finish their work, the owner of the change activity acknowledges that the work is done by setting the state of the change activity to *Ready for Approval*. Then, the approvers approve or reject the changes. If there are no approvers, the change activity is automatically approved.

When all the approvers approve the change activity, Decision Center merges the changes back into the release. The change activity is then *Complete*. When a change activity is rejected, it returns to its state of *In Progress*.

The following diagram shows the lifecycle of a change activity, with manual transitions being done by the owner of the activity:



The following table shows the most frequent user operations on a change activity. It shows the role that is required to do each operation, the precondition to this operation, and the state of the change activity afterward:

User operation	Role	Precondition to the operation	State of activity after operation
Create activity	-	-	In Progress
Cancel activity	Owner	-	Canceled
Reopen activity	Owner	Activity is in Canceled state	In Progress

activity			
Proceed to approval	Owner	Authors finish work	Ready for Approval
Finish working	Author (or owner on behalf of author)	Author working	In Progress
Resume working	Author (or owner on behalf of author)	Author finishes work	In Progress
Approve changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	Complete
Reject changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	In Progress
Delete activity	Owner of the activity or the release.	Can be done only on activities that are not <i>Complete</i> or <i>Canceled</i>	-

Validation activity governance

Then, as long as the validation activity is not *Complete*, the owner of the activity can do the following actions:

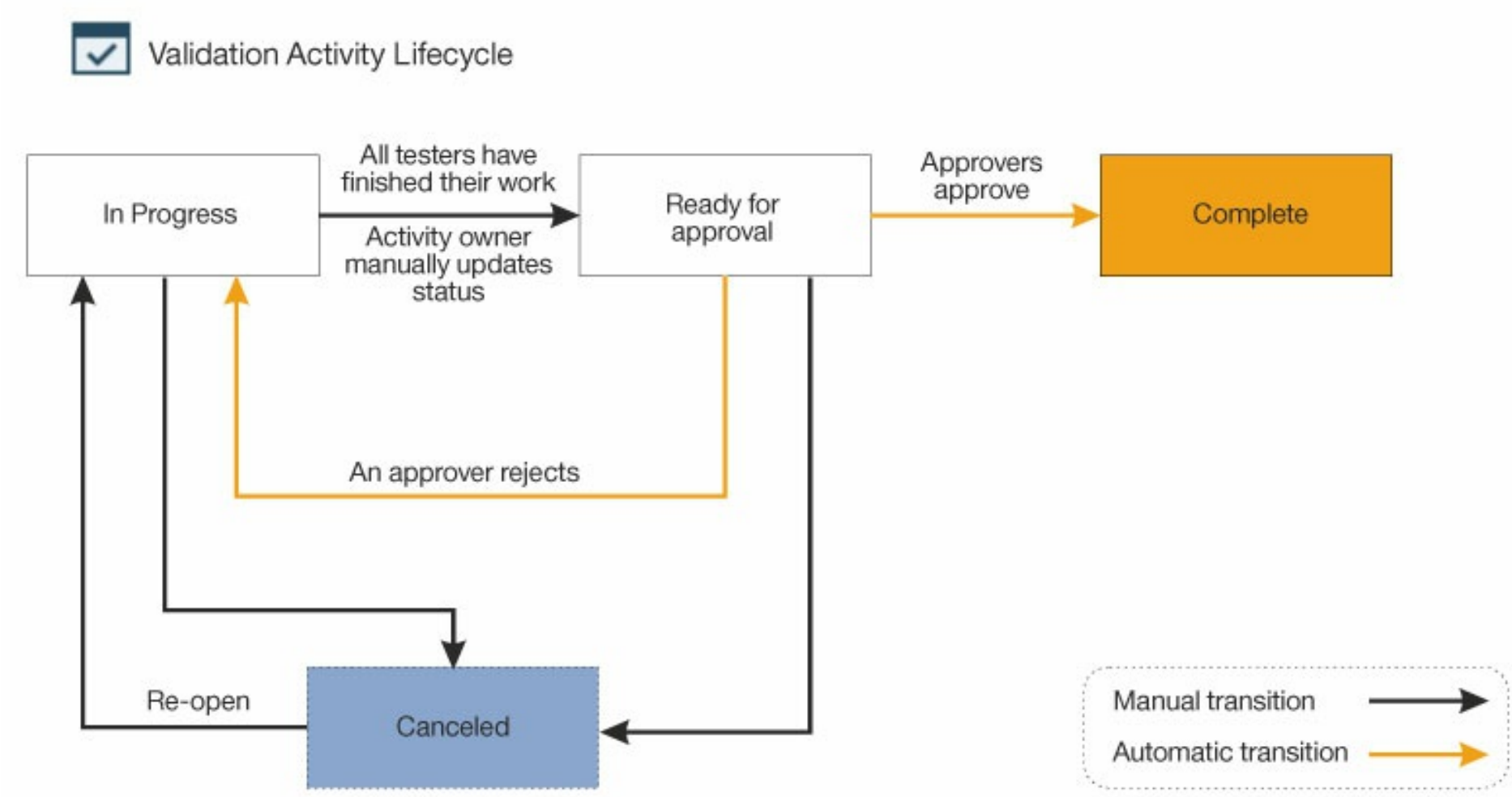
- Change the owner of the activity.
- Change the due date of the activity.
- Change the goals of the activity.
- Assign approvers and testers to the activity.

Testers run different tests that are aimed at validating a release, and note the results in the test plan. When they finish their work, and all the change activities of the release are *Complete*, testers change their status to *Finished*. When all the testers finish their work, the owner of the validation activity sets its state to *Ready for Approval*, at which point the approvers approve or reject the activity.

When all the approvers approve the validation activity, Decision Center sets the state of the validation activity to *Complete*.

**Note:** If a user creates a change activity in a release that contains a *Complete* validation activity, this validation activity is reopened.

The following diagram shows the lifecycle of a validation activity, with manual transitions being done by the owner of the activity:



The following table shows the most frequent user operations on a validation activity. It shows the role that is required to do each operation, the precondition to this operation, and the state of the validation activity afterward:

User operation	Role	Precondition to the operation	State of activity after operation
Create activity	-	-	In Progress
Cancel activity	Owner	-	Canceled



activity			
Reopen activity	Owner	Activity is in Canceled state	In Progress
Proceed to approval	Owner	Testers finishes work	Ready for Approval
Finish working	Tester (or owner on behalf of tester)	Tester working	In Progress
Resume working	Tester (or owner on behalf of tester)	Tester finishes work	In Progress
Approve changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	Complete
Reject changes	Approver (or owner on behalf of approver)	Activity is in Ready for Approval state	In Progress
Delete activity	Owner of the activity or the release.	Can be done only on activities that are not <i>Complete</i> or <i>Canceled</i>	-

**Parent topic:** [Using governance with decision services](#)

**Related concepts:**

[Getting familiar with the Business console](#)

[Decision services](#)

[Deploying from the Business console](#)

[Managing changes with the decision governance framework](#)

[Decision Center REST API](#)



## Managing changes with the decision governance framework

In Decision Center, you can manage changes to rules over time through the creation and management of branches. The recommended way of managing changes is by using the decision governance framework. With this approach, you work through releases and activities within a decision service.

Releases and activities behave as branches and subbranches of your decision services, but with some special considerations.

To understand how changes are managed under this approach in the Business console, consider the following sequence, and then read the sections that follow:

1. You complete a *release*, signifying the end of that release, and then deploy its content as a *decision service*.
2. You start a new release that is based on an existing, completed release.
3. You change rules in a release by using *change activities*, in which one or more participants work.
4. When all the change activities that are related to the release are complete, you validate the release, and then complete it.

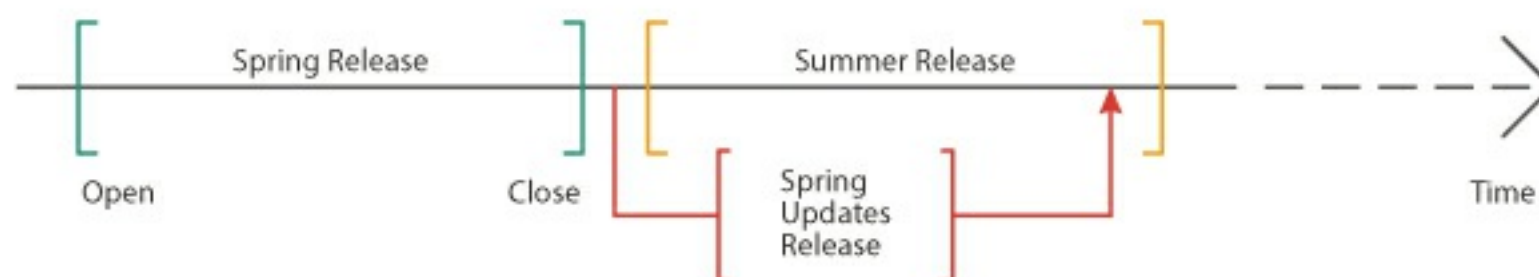
### Releases

Because rules change over time, Decision Center uses *releases* to capture and trace all the changes that are related to a purpose and period in time.

The purpose is a set of business-driven goals, and the time is a beginning date and an end date. A release tracks and manages the changes that are made by its participant users and the validation of these changes (see [Governance principles](#).)

Each new release stems from an existing release. Work occurs on the release while it is open, and ends when you complete, approve, and deploy the release.

The following image shows how releases evolve over time, and that the content of a closed release can be merged into an open one:



You cannot edit a release directly. You must create and work in change activities of the release. You can make as many change activities as are needed to carry out your objectives.

### Change activities

Decision Center uses *change activities* to manage the work of participants who are collaborating toward a goal, in the larger context of a release.

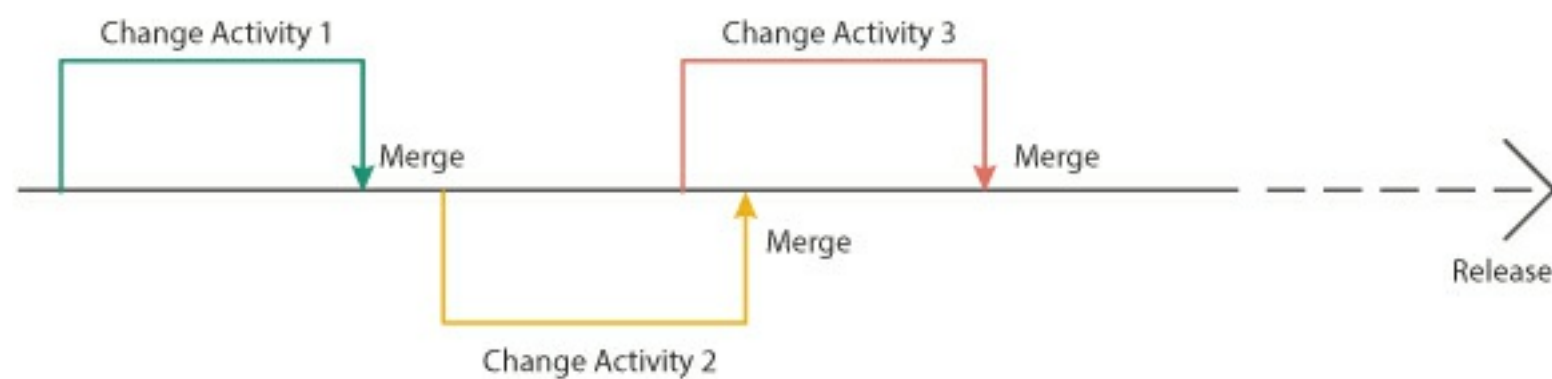
In a change activity, participants create, modify, or delete rules and get these changes approved (see [Governance principles](#).)

Each change activity stems from a release. The change activity contains the version of the rules that are found in the release at the moment when the change activity is created. When a change activity is complete, Decision Center merges it back into the release.

You can create many change activities for a release. When all the change activities are approved, a validation activity can occur on the release.

If you create a change activity before an existing one is completed and merged, the same rule might be subject to change in both activities. To avoid such conflicts, a rule that is being edited in a change activity is locked until the activity is completed and merged.

The following image shows how different change activities merge back into the release when they are completed:



## Validation activities

You create and manage *validation activities* in the Business console, and use them to track and manage a validation of the content of the release. This approach can be manual tests that are entered in a test plan, automatic tests through test suites, or simulations.

It is not possible to run a test suite or simulation directly from the release. You must use a validation activity to run a test or simulation on the version of the rules that are contained in the release.

When all the validation activities are completed, the release can be approved and completed, at which point deployment can occur.

**Parent topic:** [Using governance with decision services](#)

### Related concepts:

[Getting familiar with the Business console](#)

[Governance principles](#)

[Decision Center REST API](#)

# Linked projects


When a decision service is published to Decision Center, it can contain linked (or dependent) projects.


Dependent projects are typically created when the contents of a project become too large for one project. The project is split into one or more smaller projects that are linked together. Usually, the vocabulary that is used to create rules is kept in one project, and the other projects depend on the project with the vocabulary. This arrangement facilitates maintenance of the vocabulary and the business terms that are found in the rule editors.

In the Business console, all the linked projects are displayed under the decision service.

Also, when you open a decision service, its linked projects are listed in the release properties:

ReleaseStream

 Created by Paul  
Nov 4, 2018

▼ Goals 

Adjustments for Scoring and Insurance

▼ Linked Projects

**Loan Validation Service**  
Depends on: *Loan Validation Scoring, Loan Validation Determination, Loan Validation Check*

**Loan Validation Check**  
Depends on: *Loan Validation Base*  
Dependent on this project: *Loan Validation Service*

**Loan Validation Determination**  
Depends on: *Loan Validation Base*  
Dependent on this project: *Loan Validation Service*

**Loan Validation Scoring**  
Depends on: *Loan Validation Base*  
Dependent on this project: *Loan Validation Service*

**Loan Validation Base**  
Dependent on this project: *Loan Validation Check, Loan Validation Determination, Loan Validation Scoring*

The concept of dependency among projects implies that one project depends on another and not vice versa. However, in the Business console, linked projects allow for navigation and search between dependent projects regardless of direction.

**Parent topic:** [Managing decision services](#)

# Decision artifacts

In the **Decision Artifacts** tab, you create and edit rule artifacts, such as action rules, decision tables, or ruleflows, which you can organize in folders. You can define decision operations, ruleset variables, and handle resources for your projects.

You create business rules in two different formats: action rules and decision tables. You can create ruleflows to control the execution of rules, variable sets that you can use in the business rules of a ruleset, or decision operations to define which rules are included in a ruleset. You can also store any type of file within the Decision Center repository by uploading resources to the Business console. You find resources in the **Resources** folder, where you can create or download resources. You view and edit them offline, then refresh the existing file by re-uploading the newest version in the **Resources** folder.

**Note:** Some artifacts are not visible by default, and you need to activate them in the **All types** window, below the tab name.

You edit the content of a decision artifact in the editor (see [Building rules using the Intellirule editor](#) or [Editing decision tables](#)), and the properties by clicking **Details**. When you edit a business rule, the rule is locked and other users cannot edit the rule. You can copy or move any project element or folders within a project, if it is not locked by someone else and you have permission to create or modify those types of project elements. If you have permission to delete or rename project elements, you can delete or rename any project element that is not locked by someone else. To restore a deleted project, or a decision artifact, by selecting it in a timeline or snapshot, and clicking **Restore**.

## Action rules

You express business policies in action rules.

## Decision tables

In the Business console, you can view and edit decision tables that express sets of rules with similar conditions and actions.

## Ruleflows

With ruleflows, you can manage the flow of rule execution within your ruleset. In Decision Center Business console, you create ruleflows and add different types of elements to control the execution of rules.

## Decision operations

The decision operation defines which rules from a given branch are part of the ruleset. You choose which decision operation to use when creating a test suite, simulation, or deployment configuration.

## Ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project, and as input and output parameters in decision operations.

## Dynamic domains

A domain defines a set of values in your business terms. For example, a domain can define that the category of a customer can have one of the following values: silver, bronze, gold. You can modify these values and update your project in the Business console.

## Versions and history

Decision Center creates versions of elements that you can view in a timeline and restore.

## Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping. Technical rules are written using the ILOG® Rule Language (IRL). IRL is a Java™-like rule language that can be executed directly by the rule engine.

## Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

## Rule verification

You can perform different levels of verification on your rules.

## Rule properties

Properties associate additional information with a project element, including the development status, storage location, and user group.

## Tags for rule artifacts

You use tags to store data with your rule artifacts (action rules, decision tables, ruleflows).

## Rule overriding

You use rule overriding to give rules precedence over other rules.

**Parent topic:** [Working with the Business console](#)

## Action rules

You express business policies in action rules.

### [How action rules work](#)

Action rules use a sentence-like structure to facilitate the creation business rules.

### [Building rules using the Intellirule editor](#)

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

**Parent topic:** [Decision artifacts](#)

# How action rules work

Action rules use a sentence-like structure to facilitate the creation business rules.

## **Action rules**

You express business policies in rules that match actions to conditions.

## **Rule variables**

You create a rule variable to define the scope of a rule.

## **Types of rule variables**

You can assign different types of values to your rule variables.

## **Rule conditions**

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

## **Conditions that compare business terms and values**

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

## **Conditions that test for existence**

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

## **Conditions that test set membership**

You can use conditions to test whether a business term belongs to a set.

## **Combinations of conditions**

You can apply conditions to groups, and test nested groups.

## **Condition negation**

You can set a rule to perform an action when a condition is not true.

## **Rule actions**

Rule actions define what to do when the if part of the rule is true or false.

## **Rule actions for lists of business terms**

You can define actions that a rule executes for each item in a list.

## **Dependency of rule actions on rule variables**

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

**Parent topic:** [Action rules](#)

## Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
 the credit score of 'the borrower' is less than 200
then
 in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

### Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

### Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

### Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

### Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

**Parent topic:** [How action rules work](#)

### Related concepts:

[Rule variables](#)

[Rule conditions](#)

[Rule actions](#)

[Overview: Intellirule rule editor](#)



# Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

## Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
 the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
 set 'the cart' to the shopping cart of customer;
if
 the value of 'the cart' is less than $100
then...
```

## One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
 set Smith to a customer;
if
 the category of Smith is Gold
then
 apply 10 % discount to the shopping cart of Smith;
```

## When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

## Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
( )	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

## Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation



marks:

<b>R u l e d i t o r</b>	
	<b>Description</b>
In t e l l i r u l e e d i t o r	If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.
G u i d e d e d i t o r	When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.

**Parent topic:** [How action rules work](#)

**Related concepts:**

- [Action rules](#)
- [Rule conditions](#)
- [Rule actions](#)
- [Types of rule variables](#)
- [Dependency of rule actions on rule variables](#)

## Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, `customer`). Once you set a variable, you can use it in any part of the rule that declares the variable.

### Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and `then` parts of the rule use the same value:

```
definitions
 set maxAmount to 1000000;
if
 the amount of 'the loan' is at least maxAmount
then
 in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

### Restrictions on rule variables

You can further restrict a variable in the `definitions` part of a rule by using the operator `where`.

#### Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
 set 'loyal customer' to a customer
 where the category of this customer is Gold;
if
 the value of the shopping cart of 'loyal customer' is more than $200
then
 apply the super discount;
```

#### Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
 set 'senior Gold customer' to a customer
 where all the following conditions are true:
 - the category of this customer is Gold
 - the age of this customer is at least 65;
```

### Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the `definitions` part of the rule, for example:

```
definitions
 set applicant to a customer;
 set 'loyal customer' to a customer;
if
 all of the following conditions are true:
 - applicant is married to 'loyal customer'
```

```
 - 'loyal customer' is insured
then
 upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
 'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
 set 'customer 1' to a customer;
 set 'customer 2' to a customer;
if
 'customer 1' is married to 'customer 2'
 and 'customer 2' is insured
then
 upgrade 'customer 1''s rating;
```

**Note:** When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

### Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
 set 'gold customers' to all customers
 where the category of this customer is gold;
 set 'junior gold customer' to a customer in 'gold customers'
 where the age of this customer is at most 15;
 set 'senior gold customer' to a customer in 'gold customers'
 where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

**Parent topic:** [How action rules work](#)

#### Related concepts:

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

## Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
 the customer category is Gold
then
 redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

**Parent topic:** [How action rules work](#)

### Related concepts:

[Action rules](#)

[Rule variables](#)

[Rule actions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

## Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

### starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
 the customer's maintenance number starts with "TX"
then
 redirect the call to call center A;
```

### is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
 the purchase value of the shopping cart is more than $100
then
 apply a 10% discount to the value of the shopping cart
```

### after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
 the return date of 'rented car' is after 'pickup date' and before
 'scheduled return date'
then...
```

**Parent topic:** [How action rules work](#)

#### Related concepts:

[Rule conditions](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

## Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

### there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
 there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
 there are 10 customers where the category of each customer is Gold,
then...
```

### there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
 there are at most 3 customers
 where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
 set 'silver customers' to all customers
 where the category of each customer is Silver;
 set 'silver count' to the number of 'silver customers'
if
 there are at least ('silver count' + 1) customers
 where the category of each customer is Gold,
then...
```

### there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:

```
if
 there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
 set 'gold customers' to all customers
 where the category of each customer is Gold;
if
 there is at least one customer in 'gold customers'
 where the age of this customer is at least 65,
then...
```

**Parent topic:** [How action rules work](#)

**Related concepts:**

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

[Condition negation](#)

## Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
 the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
 set 'luxury car groups' to all car groups where the daily rate of each
 car group is at least 50;
if
 'luxury car groups' contain the car group of 'the current rental
 agreement'
```

**Parent topic:** [How action rules work](#)

**Related concepts:**

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Combinations of conditions](#)

[Condition negation](#)



## Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting
longer than 5 minutes
```

### Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if
 the customer is older than 60
 or the customer is younger than 21
 and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

### Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if
 the customer is older than 60
 or (the customer is younger than 21 and the category of the customer
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if
 (the customer is older than 60 or the customer is younger than 21)
 and the category of the customer is Student
```

### Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true`: This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true`: This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if
 all of the following conditions are true:
 - the category of the customer is Gold
 - a member of the Gold team is available
then
 redirect the call to a member of the Gold team;
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
 any of the following conditions is true:
 - the category of the customer is Gold
 - the customer has been waiting longer than 5 minutes
then
 redirect the call to a member of the Gold team;
```

**Parent topic:** [How action rules work](#)

**Related concepts:**

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Condition negation](#)

## Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

### **it is not true that**

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
 the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
 it is not true that the category of the customer is Gold
then...
```

### **none of the following conditions are true**

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
 all the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
 none of the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

**Parent topic:** [How action rules work](#)

### **Related concepts:**

[Rule conditions](#)

[Conditions that compare business terms and values](#)

[Conditions that test for existence](#)

[Conditions that test set membership](#)

[Combinations of conditions](#)

## Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

### Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
 the value of the shopping cart is more than $100
then
 apply a 15% discount on the shopping cart;
```

### Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
 'the loan report' is approved
 and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
 in 'the loan report', accept the loan with the message
 "Congratulations! Your loan has been approved";
else
 in 'the loan report', refuse the loan with the message "We are sorry.
 Your loan has not been approved";
```

**Parent topic:** [How action rules work](#)

### Related concepts:

[Action rules](#)

[Rule variables](#)

[Rule conditions](#)

[Rule actions for lists of business terms](#)

[Dependency of rule actions on rule variables](#)

## Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
 for each item in expensive items:
 - apply 5% discount to this item;
 - display the message: "A 5% discount has been applied";
```

**Parent topic:** [How action rules work](#)

**Related concepts:**

[Rule actions](#)

[Dependency of rule actions on rule variables](#)

## Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
 all of the following conditions are true:
 the value of the customer's shopping cart is more than $100
 the category of the customer is Gold
then
 apply a 15% discount
else
 apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

**Parent topic:** [How action rules work](#)

**Related concepts:**

[Rule variables](#)

[Types of rule variables](#)

[Rule actions](#)

[Rule actions for lists of business terms](#)

# Building rules using the Intellirule editor

Intellirule provides guided editing for selecting phrases and filling placeholders to create rules.

## Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

## Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

## Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

## Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

## Activating and deactivating the keyword filter

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

## Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

**Parent topic:** [Action rules](#)

## Overview: Intellirule rule editor

You build rules with the assistance of a completion menu.

When you create or open an action rule, the Intellirule Editor opens by default. If you are currently using the guided editor in a Designer, you can switch to the Intellirule editor: close the guided editor, right-click the action rule you want to edit in the Rule Explorer view, and then click **Open With > Intellirule Editor**.

You can add terms and phrases to a rule by typing or pasting in text, or by selecting them from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, value, or other phrase.

### Note:

See the [Rule editor flash demo](#) on the IBM® Customer Support site for an English-only tour of the rule-editing features.

## Terms and phrases

The rule editor provides an editing area for building rules. You can add terms and phrases in the following ways:

- Type directly in the editing area.
- Copy text from another editor or application, and paste it into the editing area.
- Select predefined terms and phrases from a completion menu.

## Completion menu

The business vocabulary of your company defines the terms and phrases in the completion menu, and the ways you can combine them.

## Placeholders

Placeholders indicate places in a term or phrase that you must complete. You must insert the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain more placeholders that you must also complete to construct a valid expression.

**Parent topic:** [Building rules using the Intellirule editor](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

[Highlighting and correcting errors](#)



## Creating rule parts

You can display the completion menu as you build your rule. The menu lists terms and phrases that are valid in the current context of the rule.

### About this task

An action rule can comprise some or all of the following parts. The parts must be defined in the following order: definitions, if, then, and else.

### Procedure

1. Click anywhere in the editing area and press Ctrl+Spacebar. The completion menu displays a list of available insertion options that are based on your current position in the rule.
2. Select an insertion option from the completion menu.
3. Press Esc or click anywhere in the editing area to hide the completion menu.

**Parent topic:** [Building rules using the Intellirule editor](#)

### Related concepts:

[Overview: Intellirule rule editor](#)

### Related tasks:

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

[Highlighting and correcting errors](#)


## Using punctuation to clarify rules

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.

Ambiguities occur when part of a rule has at least two interpretations that are semantically correct, or when there are two identical terms or phrases in the vocabulary.

For example, the following statement is ambiguous:

```
if
all of the following conditions are true :
 - the category of the customer is Gold
 - the age of the customer is at most 15
and
 the salary of the customer is more than 100
```

Rule Designer raises an ambiguity error (which appears in green ) because it cannot tell whether the salary of the customer is more than 100 is associated with the age of the customer is at most 15, or whether it is the third condition of the rule.

To remove the ambiguity, you can add a comma at the end of the the age of the customer is at most 15 condition statement:

```
if
all of the following conditions are true :
 - the category of the customer is Gold
 - the age of the customer is at most 15,
and
 the salary of the customer is more than 100
```

An alternative way to remove ambiguity is to use parentheses to group expressions.

In cases where there are two identical terms or phrases in the vocabulary, change one of the terms or phrases to remove the ambiguity.

**Parent topic:** [Building rules using the Intellirule editor](#)

# Setting completion menu options

You can set options for the way the rule editor presents terms and phrases in the completion menu, and parses rules.

## Procedure

- 1. In the toolbar above the editing area, click **Completion Menu Options**. A window opens.
- 2. Select the check boxes for the options you want to set.

Table 1. Completion menu options

Option	Description
Enable on Spacebar	<p>Select to open the completion menu whenever you press Spacebar while you are typing in the editing area.</p> <p>Clear this check box if you want the completion menu to open only when you press Ctrl+Spacebar.</p>
Enable on double-click	<p>Select to open the completion menu by double-clicking a word in the editing area.</p> <p>Clear this check box if you want double-clicking to select the current word instead.</p>
Enable auto-restart	<p>Select to automatically reopen the completion menu when a term or phrase is selected.</p> <p>Clear this check box if you want to open the completion menu by pressing Ctrl+Spacebar.</p>
Enable smart mode	<p>Select to filter the completion menu entries in a smart way to reduce the number of entries.</p>
Enable template mode	<p>Select if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable this mode if you want to insert longer phrases, and then substitute placeholders.</p> <p>Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate this mode if you prefer to build a complete rule in the same way that you might compose an email message.</p>
Use Hierarchical View	<p>Select if you want the completion menu to group terms and phrases by type, for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.</p> <p>Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.</p>
Filter unreachable phrases	<p>Select if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable.</p>
Display toolbar	<p>Select if you want the completion menu toolbar to be displayed above the list of available terms and phrases.</p>

	Clear this check box if you want to hide the completion menu toolbar.
<b>Display documentation</b>	<p>Select if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.</p> <p>Clear this check box if you want to hide the hover help.</p>

3. Click outside the pop-up window to save your settings.

**Parent topic:** [Building rules using the Intellirule editor](#)

**Related concepts:**  
[Overview: Intellirule rule editor](#)

**Related tasks:**  
[Creating rule parts](#)  
[Activating and deactivating the keyword filter](#)  
[Highlighting and correcting errors](#)

# Activating and deactivating the keyword filter

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

## About this task

The completion menu automatically filters the list of terms and phrases as you type text to build a rule. The list becomes more restricted as you type. How the automatic filtering works depends on whether you use the keyword filter.

The keyword filter is deactivated by default. When you type with the completion menu open, the list of completion menu options shows only the terms and phrases that start with the entered text. You continue until you complete a term, or select a term or phrase from the menu.

When you activate the keyword filter, the completion menu shows a filter field above the list of options. When you type with the menu open, the filter field displays your text, and the menu displays only the terms and phrases that contain your text. The list includes all the items that contain the keyword, and not just the items that start with the keyword.

You can activate and deactivate the keyword filter as you build your rule.

## Procedure

1. Click the **Toggle Keyword Filtering** button at the top of the completion menu.

The **Filter** field opens above the list of completion menu options.

2. Click the button again.

The **Filter** field closes.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Related concepts:

[Overview: Intellirule rule editor](#)

## Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Highlighting and correcting errors](#)

# Highlighting and correcting errors

The rule editor highlights errors in red in the editing area and lists them in the Problems view below the editing area.

## About this task

Correct errors manually in rule and business model files. You can use the Problems view to view the error messages and navigate to the errors in your files.

## Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
  - Hover the mouse over a highlighted error in the rule editor to display an error tip icon.
  - Double-click the text that is displayed in the error description in the Problems view to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Related concepts:

[Overview: Intellirule rule editor](#)

## Related tasks:

[Creating rule parts](#)

[Setting completion menu options](#)

[Activating and deactivating the keyword filter](#)

# Decision tables

In the Business console, you can view and edit decision tables that express sets of rules with similar conditions and actions.

## [How decision tables work](#)

Learn the basics for using decision tables to apply rules that have the same rule statement but different values.

## [Previewing decision tables](#)

You can display and sort information about a decision table in the preview window.

## [Decision table editor](#)

You use the decision table editor to edit rules in decision tables.

**Parent topic:** [Decision artifacts](#)

# How decision tables work

Learn the basics for using decision tables to apply rules that have the same rule statement but different values.

## Decision tables

A decision table groups rules that share a rule statement but have different conditions and actions.

## Columns

Each column in a decision table represents a condition or an action.

## Rows and cells

Each row in a decision table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

## Preconditions

You can pretest data before using it with a decision table.

**Parent topic:** [Decision tables](#)



# Decision tables

A decision table groups rules that share a rule statement but have different conditions and actions.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	0.001
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.005

To create a rule, you add a row to the decision table and enter values in the new cells. When an application calls the table, it runs the rules. If the conditions in a row are met, the rule that is formed by the row does the actions in the row. You can also define preconditions that apply to all the rules in a table, and warning tags during development show overlaps, gaps, and other errors in your rules.

In the Business console, you can use the keyboard shortcut Ctrl+F to open a search field within the decision table. The search is performed on all values of the decision table.

Parent topic: [How decision tables work](#)

# Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. As shown in the following table, the top cell of each column identifies the object of a condition, or the target of an action.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	0.001
2	A	100,000	300,000	true	0.001

Each row in the table forms a rule. The rule does the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of the loan is between 100000 and 300000
then
 set the Insurance required to true
 set the Insurance rate to 0.001
```

## Condition operators

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [How decision tables work](#)

# Rows and cells

Each row in a decision table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:


Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

## Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

**Note:** If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In manual row ordering mode, you manage the grouping of conditions as you edit the table and add rows. In automatic row ordering mode, rows are automatically grouped when they share a condition value when you save and when you click **Optimize Row Order** . For more information, see [Row ordering](#).

The following table shows the rows before they are organized into partitions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	B	< 100,000		false	
3	A	100,000	300,000	true	0.001
4	B	300,000	100,000	true	0.0025

The following table shows the rows after they are organized into partitions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	
4	B	300,000	100,000	true	0.0025

In the following table, the Grade A cells of rows 1 and 2 are grouped. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1
  - if
    - all of the following conditions are true:
      - the loan grade is A
      - the amount of loan is between 100000 and 300000
  - then
    - set the insurance required to true
    - set the loan rate to 0.001
- Rule 2

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row at the top of the table with the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003

The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.005
5	B	< 100,000		false	
6	B	100,000	300,000	true	0.0025
7	B	300,000	600,000	true	0.005
8	B	600,000	800,000		
9	B	≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003

3	A			1 2	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	A	300,000	600,000	true	0.003
4	A	≥ 600,000		true	0.004

### Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	A	Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	A	Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```

if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
then
 - set the insurance required to true
 - set the loan rate to 0.001

```

- Rule 2

```

if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then

```

- set the insurance that is required to true
- set the loan rate to 0.002

- Rule 3

```
if
 all of the following conditions are true:
 - the loan grade is A
 - it is not true that the amount of loan is between 300000 and
600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200,000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

**Parent topic:** [How decision tables work](#)

## Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
 set 'wealthy customer' to a customer
 where the average monthly balance of this customer
 is more than $1 000 000
if
 the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

**Parent topic:** [How decision tables work](#)



# Previewing decision tables

You can display and sort information about a decision table in the preview window.

## About this task

You can preview a decision table and sort its contents in the Business console. You can also display information about selected items.

	Grade	Amount of loan	
		Min	Max
1	A	< 100,000	
2	A	100,000	300,000
3	A	300,000	600,000
4	B	300,000	600,000

**if**

all of the following conditions are true :

- ( the loan grade in 'the loan report' is "A" )
- ( the amount of 'the loan' is at least 100000 and less than 300000 ) ,

**then**

set insurance required in 'the loan report' to true ;

set the insurance rate in 'the loan report' to 0.001 ;

## Procedure

The following steps take you through preview features.

1. Click your decision table in **Decision Artifacts**. A preview of the table opens. To open a different artifact, click the file in the column of decision artifacts. You can also filter the artifacts by using the **All Projects** and **All Types** buttons.
2. Select your decision table in **Decision Artifacts**, and click **Open Decision Table View**.
3. To display information about an item in a decision table, such as a column, row, or error tag:
  - a. Point at the item in the decision table. A preview window opens.
  - b. Check the preview window for information about the selected item. In the image above, the pointer is at the beginning of a row, and the preview window shows the rule that is formed by the cells in the row.
4. To view the preconditions:
  - a. Click **Preconditions** to display the preconditions. The preview shows the rule statement for the preconditions.
  - b. Click **Preconditions** again to hide the preconditions.
5. To sort or filter the contents of a column in a decision table:
  - a. Click the arrow next to the condition column header in your decision table. A window opens with sorting and filtering options.
  - b. You can select the sorting of values in ascending, or descending order.

The behavior of the sorting depends on the row ordering mode. When you preview a table for which no row ordering mode is explicitly set, the default view is in **Manual** mode, and sorting restrictions apply. For example, you cannot sort an action column. If you edit the table in **Automatic** mode and save it, then you can view it in **Automatic** mode.

- c. To filter values in a column, apply your filter in the section **Filter**. You can filter by value from an enumeration, by text, or use comparison operators and intervals for columns with numbers and dates. Filtering is semantic, which means that it takes into account all cells whose expressions include the value of your filtering. For example, if you filter with a value of > 100, results would include a cell that displays an interval from 0 to 200.
- d. You can clear the filter for each column in the **Sorting/Filter** window, or press **Backspace** to return to the original layout of the table.

Parent topic: [Decision tables](#)

Related concepts:  
[Decision table editor](#)



# Decision table editor

You use the decision table editor to edit rules in decision tables.

In the Business console, you can define conditions and actions, and enter values directly into table cells to form business rules. You can also use preconditions to test incoming data before the table runs a rule, and define variables for the table.

The decision table editor supports both manual and automatic row ordering. You can set the order of the rows in your decision tables, or have the decision table editor set the order for you and automatically group identical conditions.

If you want to protect your decision tables against editing by others, you can apply different types of locking mechanisms in Rule Designer (see [Using decision table locking facilities](#)). When you publish your project to Decision Center, the locks are available in the Business console.

**Note:** You can only create and edit decision tables locks in Rule Designer.

## [Columns](#)

In the Business console, you can change the columns that define the conditions and actions for business rules in decision tables.

## [Rows](#)

In the Business console, you can change the rows that form the rules in decision tables.

## [Preconditions](#)

In the Business console, you can use preconditions to check incoming data, and define variables for decision tables.

## [Table cells](#)

In the Business console, you can update table cells by entering changes directly or by using a rule editor.

## [Errors and warnings](#)

In the Business console, you can use automatic cell checking to highlight problems in decision tables.

## [Row ordering](#)

In the Business console, you can select either automatic or manual row ordering to organize the rows in decision tables.

## [Transferring data to and from Excel](#)

You can export decision tables as Excel files to work offline on their content, and import these Excel files back to the Business console with your changes. You can also manually transfer data such as rows, columns, and cells between Excel files and decision tables.

**Parent topic:** [Decision tables](#)

**Related tasks:**

[Previewing decision tables](#)

# Columns

In the Business console, you can change the columns that define the conditions and actions for business rules in decision tables.

A decision table contains a group of similar rules that use different conditions and actions. Each row in a table forms a rule, and each column in a decision table represents a condition or an action. With the decision table editor, you can add or remove columns through the column menu, and change the constraints that are applied by each column. When you add or remove a column, you change the rule statement that is used by all the rules in the table.

## Defining a column

To define the condition or action of a column, select the column by clicking the header of the column, or selecting a cell in the column and pressing Ctrl+Space. To edit the condition or action, open the contextual menu by right-clicking the column.

You can change the condition or action expression of a column whose associated cells are already filled. The editor will try to apply the existing values in the cell to the new expression. When the number of parameters differ, the editor retains only the values that can be applied. For example, if you change the expression `score is between <min and <max> to score equals <number>`, the `<min>` value becomes the equal value. If the number of parameters increases, the editor keeps the values, but you have to complete each cell so that a complete condition or action is specified.

## Editing column titles and subtitles

To edit a column title or subtitle, click the part of the title that you want to change to open an input field. You can also open the input field by selecting a column and pressing Enter. The text that you enter in the title or subtitle does not change the rule statement of the column.

## Sorting and filtering columns

When you click the arrow next to the column title, a window opens with sorting and filtering options.

You can sort the rows of a column in ascending or descending order. The row sorting also depends on the row ordering mode selected for the decision table (see [Row ordering](#)). When you preview a table for which no row ordering mode is explicitly set, the default view is in **Manual** mode, and sorting restrictions apply. For example, you cannot sort an action column. If you edit the table in **Automatic** mode and save it, then you can view it in **Automatic** mode.

You can apply filtering to reduce the number of rows displayed in the table. You can filter by value from an enumeration, by text, or use comparison operators and intervals for columns with numbers and dates. Filtering is semantic, which means that it takes into account all cells whose expressions include the value of your filtering. For example, if you filter with a value of `> 100`, results would include a cell that displays an interval from 0 to 200.

In the decision table editor, you can use the **Undo** and **Redo** buttons to go quickly from a view of your filtered subset of data to your previous, unfiltered table, and vice versa.

## Resizing automatically

You can set the display of a decision table to resize automatically when you change the size of your browser window. In **Details**, select **Auto-resize columns**.

## Column menu

To open the column menu, right-click the top cell of the column that you want to change.

Table 1. Table of column menu commands

Co mm and	Description
Def ine Col um n	Opens a rule editor to change the condition of the column. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at bottom of the editing area.
For mat col um n	Opens an editor to set the formatting options for displaying values in a column (dates, numbers, and so on). You can select the format from the drop-down list options, and optionally edit the # symbols to personalize the format.
Che ck Ga n	Looks for gaps between values in the cells of condition columns.

<b>r</b>	
<b>Check Overlap</b>	Looks for overlapping values in the cells of condition columns.
<b>Cut</b>	Cuts a selected column. You can paste the column to another location.
<b>Copy</b>	Copies a selected column. You can paste the copy of the column to another location.
<b>Paste</b>	Pastes a copied or cut column to a selected location.
<b>Insert Column</b>	Inserts a condition or action column, depending on the current selection.
<b>Delete</b>	Removes the selected column and its part of the rule statement in the decision table. Deleting a column can disable a decision table.
<b>Clear</b>	Deletes the contents of the cells in the column.

Parent topic: [Decision table editor](#)

# Rows


In the Business console, you can change the rows that form the rules in decision tables.

The rows contain the values that complete the rule statement that is defined by the condition and action columns in a table. Each row constitutes a complete rule.

## Editing a row

To view the rule that is formed by a row, hover your mouse over the first cell in the row, and a window displays the rule. To edit a row value, type your changes in the cell or right-click the cell to open a menu of options.

## Optimizing

When you use automatic row ordering, click **Optimize Row Order**  to group the rows that have the same conditions. The button is activated by any change to a condition in the decision table.

## Row menu

To use the row menu, right-click the first cell in the row that you want to change.

Table 1. Table of row menu commands

Command	Description
Cut	Cuts a selected row. You can paste the row to another location.
Copy	Copies a selected row. You can paste the copy of the row to a selected location.  This command does not work with manual row ordering.
Paste	Pastes a cut or copied row to a selected location.  This command does not work with manual row ordering.
Insert copied rows	Inserts a group of copied rows. The command inserts the rows above the selected row.
Insert row	Inserts a new row above or below the selected row. Alternatively, hover your mouse over the border between the selected row and the row above or below, and click +.
Delete	Deletes the selected row.
Clear	Deletes the contents of the cells in the selected row.

Parent topic: [Decision table editor](#)

## Preconditions

In the Business console, you can use preconditions to check incoming data, and define variables for decision tables.

You use preconditions to check incoming data to determine whether a decision table can process the information. You can also use the preconditions to define one or more variables for the decision table. The variables are used in only the decision table.

### Setting preconditions

To change the preconditions, click **Preconditions**. One of the following dialogs opens:

- **Define Preconditions:** Opens if the table is new, or the table does not have a precondition.
- **Update Preconditions:** Opens if the table has a precondition.

Use the rule editor to create or modify the preconditions, and click **Define** or **Update** to apply your changes.

**Parent topic:** [Decision table editor](#)

# Table cells

In the Business console, you can update table cells by entering changes directly or by using a rule editor.

The cells in the rows that form the rules in a decision table contain the values that complete the conditions and actions. The columns define the conditions and actions, which determine what you can enter in the cells.

## Editing a cell

To edit a cell in a decision table, type your changes directly in the cell. You can use common keyboard commands such as Ctrl+C for copy.

## Replacing Boolean values with check boxes

You can replace the Boolean values with check boxes, which can be selected or cleared to indicate actions. In **Details**, select **Use check boxes for Boolean values**.

## Cell menu

To open the cell menu, right-click the cell that you want to edit. The cell menu provides the following commands, which vary between the condition and action cells.

Table 1. Table of cell editing commands

Com man d	Description
Cut	Copies and deletes the contents of the cell. The contents can then be pasted to another cell.
Copy	Copies the contents of a cell.
Past e	Pastes the copied contents of one cell to another cell.
Inser t copie d cells	Inserts copied cells and their values.
Inser t row	Inserts a row above or below the row of the selected cell.
Clear	Deletes the contents of the cell.
Chan ge oper ator	Inserts an operator to define the range of values. Condition columns only.
Disa ble/E nabl e	Disables or enables an action cell. A rule ignores the action of a disabled cell. Action columns only.
Set to other wise	Sets the value of the cell to 0therwise. Condition columns only.
Edit custo m value	Displays the contents of the cell in a rule editor. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at the bottom of the editing area.

## Operators

When you edit a cell, the console can provide a list of operators based on the contents of the cell. To open the list of operators, right-click the cell and select **Change operator**. Operators are provided for numbers, strings, dates, and domains.

Table 2. Number operators

Operat or	Description
=	The input data equals the number in the cell.
≠	The input data does not equal the number in the cell.
in	The input data is in the numbers in the cell.
!in	The input data is not in the numbers in the cell.
<	The input data is less than the number in the cell.
<=	The input data is less than or equal to the number in the cell

-	The input data is less than or equal to the number in the cell.
>	The input data is more than the number in the cell.
≥	The input data is more than or equal to the number in the cell.
[..]	The input data is between the two numbers, or it is equal to one of the numbers.
]..]	The input data is more than the first number and less than or equal to the second number.
[..[	The input data is more than or equal to the first number and less than the second number.
]..[	The input data is more than the first number and less than the second number.

Table 3. String operators

Operator	Description
=	The input data is the same as the string in the cell.
≠	The input data is not the same as the string in the cell.
in	The input data is in the strings in the cell.
!in	The input data is not in the strings in the cell.
is empty	The input data is an empty string.
is not empty	The input data is not an empty string.
contains	The input data contains the string in the cell.
!contains	The input data does not contain the string in the cell.
starts with	The input data starts with the string in the cell.
!starts with	The input data does not start with the string in the cell.
ends with	The input data ends with the string in the cell.
!ends with	The input data does not end with the string in the cell.

Table 4. Date operators

Operator	Description
=	The input data is the same as the date in the cell.
≠	The input data is not the same as the date in the cell.
in	The input data is in the dates in the cell.
!in	The input data is not in the dates in the cell.
<	The input data is before the date in the cell.
≤	The input data is before or the same as the date in the cell.
>	The input data is after the date in the cell.
≥	The input data is after or the same as the date in the cell.
[..]	The input data is between the two dates, or it is same as one of the dates.
]..]	The input data is after the first date and before or the same as the second date.
[..[	The input data is after or the same as the first date, and before the second date.
]..[	The input data is after the first date and before the second date.

Table 5. Domain operators

Operator	Description
=	The input data is the same as the domain value in the cell.
≠	The input data is not the same as the domain value in the cell.
in	The input data is in the domain values in the cell.
!in	The input data is not in the domain values in the cell.

Parent topic: [Decision table editor](#)

## Errors and warnings

In the Business console, you can use automatic cell checking to highlight problems in decision tables.

The decision table editor tags inconsistent content, overlaps, and gaps with triangles. The console automatically checks the contents of a decision table when you open the table. It displays error and warning tags in preview and the decision table editor.

To view the cause of an error or warning, hover your mouse over the tagged cell or the header of the column that contains the cell. A message box opens with a description of the problem. When you hover over the column header, the message box shows the rule statement for the column along with a list of any problems in the column.

**Tip:** You can open the error list by selecting a tagged cell or header, and pressing F9.

You can turn off the cell checking. In the decision table editor, open **Details** and clear the check box for the type of checking that you want to stop. You can completely stop the checking by clearing the boxes for both **Check gaps** and **Check overlaps**.

### Errors

Errors prevent a decision table from working. The following list provides common examples of errors:

#### Unexpected value

Entering the wrong type of value, for example, a word where a number is expected.

#### Mixed characters

Entering an unexpected character, for example, adding a letter to a number.

#### Incorrect value

Entering a word or number that does not match any of the values that are defined for a parameter.

#### Wrong operator

Entering an operator that places a value out of scope for the rule statement.

When an error occurs, the console tags it as shown in the following illustration:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 10000A		false	
2	A	<div>Error The word 'I' is missing.</div>		true	0.001
3	A			true	0.003

### Warnings for overlaps and gaps

The console warns you when overlaps or gaps occur between rows with shared conditions. These anomalies do not stop a decision table from working. However, they might produce incorrect results. Typically, ranges in tables are contiguous, and do not overlap.

Warnings are given for the following reasons:

#### Overlaps

When two cells in a column contain ranges that include the same values. In the example, the ranges [200,000; 300,000] and [200,000; 600,000] both include the numbers 200,000 to 300,000.

#### Gaps

When two cells in a column contain ranges that do not include values between the ranges. In the example, the ranges < 100,000 and [200,000; 600,000] do not cover the numbers between 100,000 and 200,000.

When overlaps and gaps occur in a table, the console tags them as shown in the following illustration:



	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<div> <div> <div></div> <div>the amount of 'the loan' is at least <u>&lt;min&gt;</u> and less than <u>&lt;max&gt;</u></div> </div> <div> <div>Errors</div> <div>Lines 5 to 8 have gaps</div> <div>Line 6 overlaps with line(s) 7</div> <div>Line 7 overlaps with line(s) 6</div> </div> </div>			
2	A				
3	A				
4	A				
5	B	< 100,000		false	⊗
6	B	200,000	300,000	true	0.002
7	B	200,000	600,000	true	0.005
8	B	≥ 600,000		true	0.008

You can select overlap and gap checking for an individual column or an entire table.

For an individual column:

- 1. Right-click the header of the column.
- 2. Select or clear **Check Gap** or **Check Overlap**.


For an entire table:

- 1. Click **Details**.
- 2. Select or clear **Check Gap** or **Check Overlap**.

Updating warnings in a decision table

The decision table rule editor provides manual and automatic row ordering. How you update the warnings varies with the different row ordering modes.

Automatic row ordering

Click **Optimize Row Order**  to update the gap and overlap warnings. The warnings are displayed when the row order and partitions are updated. Any change in a partition resets the errors, for example, a change in the value of a condition cell.

Manual row ordering

All the editing commands update the errors. For example, if you change the value of a cell and press Enter, the editor checks the table for errors and updates the warnings.

Displaying errors and warnings together

If a table contains errors and warnings, both are tagged in the table. However, because errors are more severe than warnings, the table displays an error marker in the header of any column that contains both errors and warnings.

The following table shows a column with errors and warnings:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<div> <div> <div></div> <div>the amount of 'the loan' is at least <u>&lt;min&gt;</u> and less than <u>&lt;max&gt;</u></div> </div> <div> <div>Error</div> <div>Line 5 : The word '/' is missing.</div> <div>Line 6 overlaps with line(s) 7</div> <div>Line 7 overlaps with line(s) 6</div> </div> </div>			
2	A				
3	A				
4	A				
5	B	< 10000A		false	⊗
6	B	200,000	300,000	true	0.002
7	B	200,000	600,000	true	0.005

Parent topic: [Decision table editor](#)

# Row ordering

In the Business console, you can select either automatic or manual row ordering to organize the rows in decision tables.

Automatic row ordering groups rows that have the same condition values. You can sort any column, and you are not constrained by the partition structure. You can edit your decision table as a flat spreadsheet, and let the decision table editor combine conditions for you.

Automatic row ordering is not appropriate for a table that must have its rows in a specific order. For this type of table, you must organize the rows manually.

When you open a decision table for the first time, you are asked to select the type of row ordering. Automatic row ordering is selected by default because it is appropriate for most use cases. Manual row ordering is used in advanced cases where the order of the rows must be controlled.

You can also set the type of row ordering through **Details** in the decision table editor. You must save and reopen the decision table to implement the property change.

## Automatic row ordering

Automatic row ordering organizes a decision table by grouping rows that share condition values.

Table before optimization:


	Grade	Amount of loan		Insurance	Insurance
		Min	Max		
1	A	< 100,000		false	⊗
2	B	< 100,000		false	⊗
3	A	100,000	300,000	true	0.001
4	B	100,000	300,000	true	0,0025

Table after optimization:

	Grade	Amount of loan		Insurance	Insurance
		Min	Max		
1	A	< 100,000		false	⊗
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	⊗
4	B	100,000	300,000	true	0,0025

You can add, move, and remove rows when a table is set to automatic row ordering. You can apply automatic row ordering in two ways:

### Optimize Row Order button

**Optimize Row Order**  reorganizes the rows during an editing session. The button is activated by changes to the contents of the table.

### Save

The editor automatically reorganizes the rows into groups when you save the table.

Automatic row ordering lists the groups alphabetically or numerically depending on the shared values. You can use the sort buttons to change the order of the rows. When you sort a column that does not contain shared conditions, all the rows in the table are reorganized by the values in the sorted column.

## Manual row ordering

Manual row ordering organizes rows into partitioned groups. As the following example shows, a partitioned group has a shared condition value:

	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2	A	100,000	300,000	true	0.001
3	B	< 100,000		false	
4	B	100,000	300,000	true	0.002

Rows 1 and 2 contain a partitioned group that shares Grade A, and rows 3 and 4 contain a partitioned group that shares Grade B. With manual row ordering, you work with rows in groups. You manually add rows to groups, and set the order of the groups. If you change the value of a cell in a group, you update all the other cells in the group. You cannot copy and paste rows, but you can copy and paste cells.

With manual row ordering, you can group or split rows by selecting cells, right-clicking them, and selecting **Group** or **Split**.

You can use the sort buttons to sort condition columns when you use manual row ordering, but you cannot sort action columns. When you sort a column, the grouped rows are sorted within their groups. However, independent rows can be redistributed throughout a table. Sorting does not keep split rows together. You can regroup split rows by sorting the column that contains their shared condition value.

The following images show the sorting patterns of the different row ordering modes.

Before sorting:

Amount of loan	
Min	Max
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	
< 100,000	
100,000	300,000
300,000	600,000
≥ 600,000	

Sorting with automatic row ordering:

Amount of loan	
Min	Max
< 100,000	
< 100,000	
< 100,000	
100,000	300,000
100,000	300,000
100,000	300,000
300,000	600,000
300,000	600,000
300,000	600,000
≥ 600,000	
≥ 600,000	
≥ 600,000	

Sorting with manual row ordering:

Amount of loan	
Min	Max
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	
≥ 600,000	
300,000	600,000
100,000	300,000
< 100,000	

Parent topic: [Decision table editor](#)

# Transferring data to and from Excel

You can export decision tables as Excel files to work offline on their content, and import these Excel files back to the Business console with your changes. You can also manually transfer data such as rows, columns, and cells between Excel files and decision tables.

## Export to Excel

To export an entire decision table from the Business console to Excel:

1. Select your decision table in the **Decision artifact** tab.
2. Click **Download this decision table as an Excel file**.

You can also click **Download as Excel** in the **Decision Table** view.

You can then share your decision table, as an Excel spreadsheet, for reviewing or reporting purposes.

If you have several decision tables in a project, you can download them as a single Excel spreadsheet. In the **Decision artifact** tab, select the check boxes next to your decision tables, then click **Download the selected decision tables...** in the toolbar. Your tables are exported in one file, with a table per sheet.

When you generate the Excel file, condition cells are merged. If you primarily want to generate the file to edit it and then paste the modifications into the Business console, you should deactivate condition cell merging. Set the `decisioncenter.web.dt.excelExport.mergeConditionCells` preference to **false**.

## Import from Excel

To import your Excel file back in the Business console:

1. Open the decision table editor.
2. Click **Import Excel File**.
3. Select your Excel spreadsheet.
4. Click **Import**.

You cannot change the structure of the decision table before you import it in the Business console. For example, you cannot add columns, which would introduce a new condition or action, or edit definitions comments in column headers. You can modify only values, or add and remove rows. Importing a table with an altered structure might cause errors or unexpected behavior. You can always undo unwanted changes with the **Undo** button.

## Formatting in Excel

If you are making modifications in Excel to re-import the file back into the Business console, you must write operators manually and without Excel formatting, for example:

- Operators on numbers and dates, such as:

```
[10 | 20]
>50
[01/01/2019 | 02/05/2019[
```

- Operators on strings, such as contains, starts with, ends with
- Operators on sets, such as in X, Y, !in X, Y
- Operators without parameters, such as is null, is not null, is empty, is not empty

Dates and number formatting is not supported, only values can be exported to Excel and imported into the Business console. For example, if your date value is MM/DD/YYYY and is formatted to display as MM/DD/YY, when you export or import your decision table, it displays the original value MM/DD/YYYY.

Calls to methods defined in your BOM file are supported when you export and import the decision table. The generated Excel file displays the full expression, which is underlined. When you import the file back into the Business console, underlined text is interpreted as a call to a BOM method.

## Transfer data manually

You can also transfer data between Excel and decision tables by copying and pasting the elements. When copying from Excel to the Business console, the content that you select must match the content of the decision table. If the contents of a transfer does not match the contents of the decision table, the decision table editor can block the transfer or show an error.

Moving information between the two environments can alter data. You can transfer only values and not data type information. For example, the Boolean values `true` and `false` are lowercase in decision tables. However, Excel automatically displays Boolean values in uppercase: `TRUE` and `FALSE`. In this case, you must set Excel to treat the Boolean values as text to retain their lowercase format for their transfer back to the decision table.

How you transfer information depends on its role and formatting. The contents of cells can be transferred between cells, while a complete row or column must be transferred as a group of cells.

**Note:**

- The menu commands do not work in all browsers. You must use the keyboard shortcuts in some browsers. Also, when pasting from an external source, specifically from Excel, make sure that there is no region in the table that is marked as the paste source, which is indicated with an orange rectangle. The paste operation always selects an available source within the table first, and then checks for data in the clipboard.
- If you copy and paste content from a decision table in the Business console to Excel, you might see hyphens in cells, which represent grouped cells in the decision table. In this context, a hyphen in a cell means the cells above it must be grouped in the decision table. If you need to insert an actual hyphen, use quotation marks around it: " - "

**Parent topic:** [Decision table editor](#)



# Ruleflows

With ruleflows, you can manage the flow of rule execution within your ruleset. In Decision Center Business console, you create ruleflows and add different types of elements to control the execution of rules.

Rules apply decisions, but they do not have a defined sequence or succession. A ruleflow organizes rules into a sequence of decisions by assembling the rules into a group of rule tasks that uses a set execution pattern.

Each rule task is evaluated to produce a result, or decision. All these results and decisions are combined to produce a single business decision, which is represented by a ruleflow. Ruleflows also specify the transitions between rule tasks. The transitions determine how, when, and under what conditions to use each rule task.

The following diagram shows the levels of refinement for selecting the rules.



The evaluation during ruleset execution selects rules in the following manner:

## 1. Ruleflow scope selection

Each rule task in a ruleflow has a scope that is defined by a list of rule packages and individual rules. At run time, ruleflow scope selection generates a list of the rule packages and rules that you defined for each task. When a task is run, the rule engine considers only the rules within this scope.

## 2. Runtime rule selection

The runtime rule selection process further determines which rules the rule engine must consider. Filtering is typically based on the value of rule properties and execution parameters.

## 3. Rule overriding

After the other selection mechanisms are executed, certain rules that you defined beforehand override other rules. The overridden rules are filtered out of the selection.

The rule engine evaluates the conditions of the selected rules, and runs their rule actions on the matching objects.

### Overview: Ruleflows

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

### Creating a ruleflow

In the Business console, you can create, and add elements to a ruleflow. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

### Configuring ruleflow properties

To display the properties of ruleflow elements, select each one, and a window with properties opens. When you add a node or a task to your ruleflow, you must set their properties.

**Parent topic:** [Decision artifacts](#)

## Overview: Ruleflows

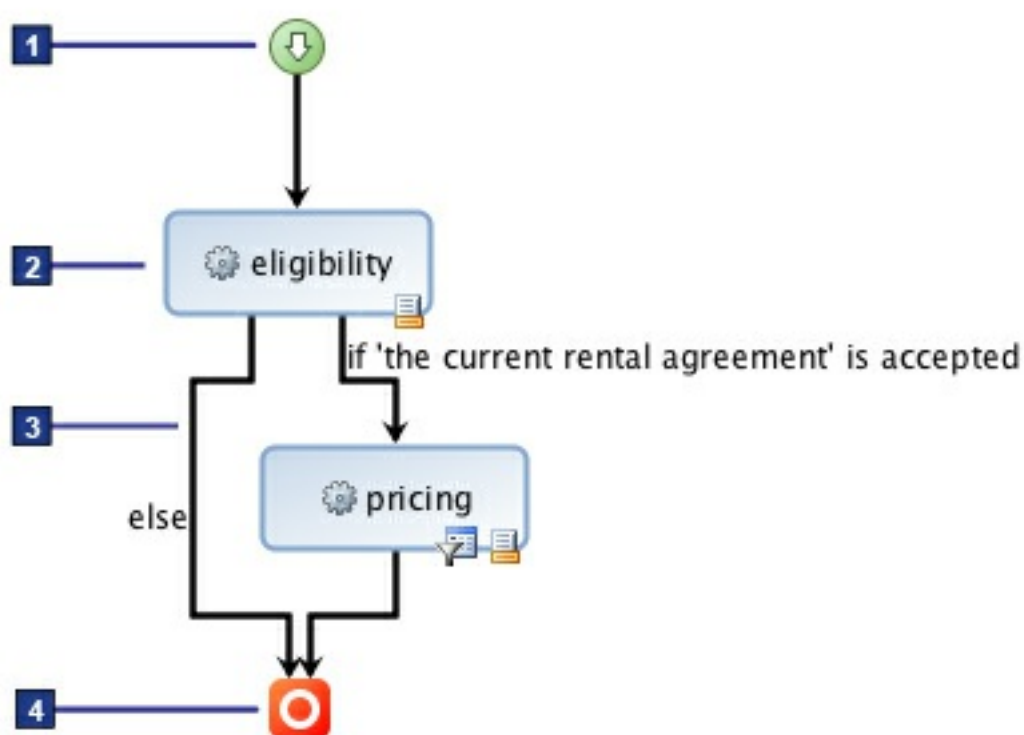
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



**Note:** The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

### Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

### Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

#### Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter




statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

## Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

## Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

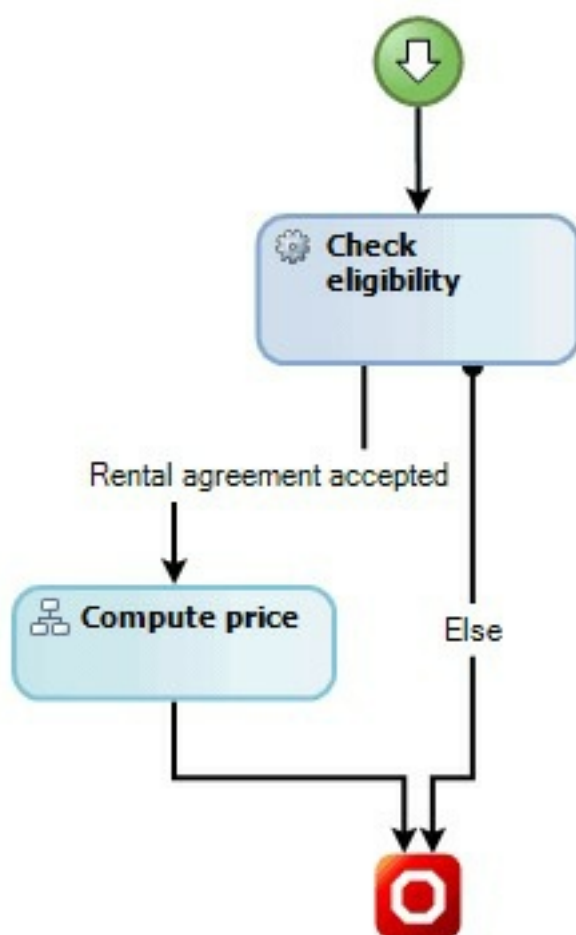
You can specify initial and final actions on subflow tasks.

## Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.




A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.


## Branches

A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch.

Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

## Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

## Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

**Parent topic:** [Ruleflows](#)

## Creating a ruleflow

In the Business console, you can create, and add elements to a ruleflow. You use these elements to determine a sequence in which the rules are selected, and define which methods the rule engine uses to evaluate those rules.

### About this task

You can add a ruleflow to a project, or any of its packages. After adding a ruleflow to a rule project or package, you define the structure by adding the elements that you need in the ruleflow editor. To understand the use of each element, see [Ruleflows](#).

### Procedure

1. In the **Decision Artifacts** tab, click **Create an artifact > New Ruleflow** to add a ruleflow in the project of your choice.

In the ruleflow editor, a start node and an end node are added by default.

2. Add elements to your ruleflow with the **+** button along the transitions to add a node between the elements, or under an element to create a node connected to this element. If you drag the new node to an existing node, a transition is created between them.

**Note:** You are notified of errors, or elements that are incorrectly positioned, by an icon displayed inside the node.

3. Create rule tasks, and add rules to be executed at this point in the ruleflow.
  - a. Add a rule task element in the editor, and click the rule task to open a window where you set its properties.
  - b. Next to **Uses**, click **Edit** to add rules and packages to a rule task. You can use the arrow icons to order the rules and packages. Depending on the rule execution properties of the task, this order might impact the output of ruleflow execution.
  - c. Optional: For advanced users, configure the execution of rule tasks at run time:
    - Select the rule engine algorithm, ordering, and exit criteria. For more information, see [Execution properties for rule tasks](#).
    - Use the option **Selects rules where** to select rules to be executed at run time, and **Rule selection** to define whether the runtime rule selection is static or dynamic. For more information, see [Runtime rule selection](#).

If you need to delete an element, select the node or transition that you want to delete, and press **Delete** or **Backspace** on your keyboard.

4. Optional: If you need to execute rule action statements, add action tasks in your ruleflow. Then, set the action statement, initial action, and final action.
5. Configure the properties of transitions between the tasks.
  - a. Select the transition, and provide a name for the condition in the **Label** field.
  - b. Next to **Conditions**, click **Edit**.
  - c. Select **BAL**, or **IRL**, and type a condition statement. For example, in BAL:

**'the current rental agreement' is accepted**

If you write the condition using IRL, make sure the text fields contain a valid Boolean expression.

In transition conditions, the variable scope is restricted to ruleset parameters and variables. It does not include access to the working memory.

6. Optional: You can create multiple, parallel paths in your ruleflow, if you need to execute rules simultaneously. For example, if you are checking the eligibility of a customer for a loan, you might want to check whether the customer meets the criteria for the loan, and also if the amount requested is valid. To do so, you use forks and joins in your ruleflow.
  - a. Add a fork node where you want your ruleflow to execute several rules in parallel. You can then add your rule tasks in the ruleflow.
  - b. Add a join where you want to combine the transitions created from the fork.

The transitions from a fork node to a join node must not have conditions, because the ruleflow follows all paths in parallel between the fork and the join.

7. Optional: You can add branches to the ruleflow to organize conditional transitions, in the same way that you could start several conditional transitions from a task.

### Important:

When multiple transitions originating from a branch or a task define overlapping conditions, the path

taken to execute the ruleflow is unpredictable. Make sure the conditions you define for multiple transitions do not overlap.

- a. Add a branch node where you want your ruleflow to organize the different conditions.
  - b. To name the branch, click the branch node, and enter the name in the **Label** field.
  - c. Add transition conditions for each transition from the branch. One of the transitions must be an Else transition.
8. Optional: If you need to execute another ruleflow at some point in your main ruleflow, add a subflow task, and click it to open the properties window. Select the ruleflow you want to execute in the **Subflow** drop-down list.
  9. To align the ruleflow automatically, click **Arrange all** in the toolbar.
  10. Save the ruleflow.

**Parent topic:** [Ruleflows](#)

# Configuring ruleflow properties

To display the properties of ruleflow elements, select each one, and a window with properties opens. When you add a node or a task to your ruleflow, you must set their properties.

Some properties are common to several ruleflow elements, and are described in the section [Common properties](#). Properties that are specific to other ruleflow elements are detailed in the following sections:

- [Rule tasks properties](#)
- [Action tasks properties](#)
- [Subflow tasks properties](#)
- [Transitions properties](#)

## Common properties

Some properties are common to several ruleflow elements.

Table 1. Properties common to several ruleflow elements

Property	Pertains to	Purpose
Label	Transitions, rule tasks, action tasks, subflow tasks, branch nodes.	You can specify a label for the ruleflow element, that is displayed in the ruleflow editor.
Initial action	Start nodes, rule tasks, action tasks, subflow tasks.	The action that launches before the task starts.  Select <b>BAL</b> or <b>IRL</b> to define the language to use to write the action.
Final action	End nodes, rule tasks, action tasks, subflow tasks.	The action that launches after the task ends.  Select <b>BAL</b> or <b>IRL</b> to define the language to use to write the action.
Document action	All ruleflow elements.	This property displays optional information about the node, or transition. Use this property to provide notes, comments, and other useful information.

## Rule task properties

Table 2. Rule task properties

Property	Purpose
Uses	List the rules and rule packages that are considered when the rule task is executed. To edit this list and reorder it, use the arrow icons.
Selects rules where	Use this property to specify runtime rule selection using a statement in BAL, or IRL code. Define a rule filter in IRL with "body = . . . ."

Table 3. Advanced rule task properties

Property	Purpose
Rule selection	After you entered a code fragment for the property <b>Select rules where</b> , you can specify if your runtime rule selection is dynamic, or static: <ul style="list-style-type: none"><li>• <b>Dynamic</b>: The property runs each time you call the task.</li><li>• <b>Static</b>: The property runs the first time you call the task.</li></ul> For more information, see <a href="#">Runtime rule selection</a> .

<b>Algorithm</b>	<p>Use this section to specify the processing algorithm for the rule task:</p> <ul style="list-style-type: none"><li>• <b>RetePlus</b></li><li>• <b>Sequential</b></li><li>• <b>Fastpath</b></li></ul> <p>For more information, see <a href="#">Engine execution modes</a>.</p>
<b>Exit Criteria</b>	<p>Use this section to specify if all rules, one rule, or one rule instance execute:</p> <ul style="list-style-type: none"><li>• <b>None</b>: The default setting that all rules are executed until conditions terminate execution. The rules are executed in a particular order determined by the selected ordering.</li><li>• <b>Rule</b>: Execution terminates after the chosen rule is executed, according to the selected algorithm.</li><li>• <b>RuleInstance</b>: A single instance of one rule is executed. This rule is determined by the selected ordering.</li></ul>
<b>Ordering</b>	<p>Use this section to specify the order of rule execution in a rule task as follows:</p> <ul style="list-style-type: none"><li>• <b>Default</b>: The algorithm determines the execution order.</li><li>• <b>Literal</b>: The order of the rules in the rule task determines the order of rule execution</li><li>• <b>Priority</b>: the rules are executed following a static priority in decreasing order.</li></ul>

**Note:** The properties **Algorithm**, **Ordering**, and **Exit criteria** are for advanced users. For more information, see [Execution properties for rule tasks](#). If you do not know about these properties, you can leave the default settings.

Action task properties

Table 4. Action Task properties

Property	Purpose
<b>Action code</b>	Set the rule action statements to be executed, in BAL or IRL.

Subflow properties

Table 5. Subflow properties

Property	Purpose
<b>Subflow</b>	Select a ruleflow to be executed at this point in the ruleflow that you are editing.

Transition properties

Table 6. Transition properties

Property	Purpose
<b>Conditions</b>	<p>In the <b>Transition Conditions Editor</b>, you must first select the rule language you use to write the transition: <b>BAL</b> or <b>IRL</b>. In the rule language you chose, you write an expression whose return value must be true or false, for example:</p> <ul style="list-style-type: none"><li>• <b>BAL:</b></li></ul> <div>'the loan' is approved</div> <ul style="list-style-type: none"><li>• <b>IRL:</b></li></ul> <div>(bicycle.speed &gt; 10) &amp;&amp; (bicycle.speed &lt;= 30)</div> <p>If you have several transitions originating from a task, there must be one (and only one) transition with no conditions, which is considered as the default transition, also called else transition. By default, if you do not specify a label, it is displayed as "else".</p>

With several transitions, you must make sure that transitions do not overlap, which means that no more than one transition condition must return "true". Typically, you might want to use two transitions, to choose between one ruleflow path with a specific condition and one default else transition for the remaining cases.

If there is only one transition that links two tasks, you must not specify any conditions.

**Parent topic:** [Ruleflows](#)

## Decision operations

The decision operation defines which rules from a given branch are part of the ruleset. You choose which decision operation to use when creating a test suite, simulation, or deployment configuration.

A ruleset includes rule artifacts and other elements. A decision operation includes all the settings needed to define the contents of a ruleset and its parameters. The ruleset content and parameters allow the client application to exchange information with the ruleset.

### Ruleset content

The ruleset content is defined by specifying the following information:

- The **source rule project**. All the rules and variables contained in this project and any of its dependent projects become eligible to be included in the ruleset.
- Any **ruleflow** to include in the ruleset.
- A **query** or **validator** to filter the rules, so that only a subset of the rules of the source rule project and its dependent projects are included in the ruleset. Typically, this is used to include rules from specific folders of a project or according to a rule property such as status. You write queries in the **Queries** tab of the Business console. A validator can be written in Rule Designer to select the required rules. If you specify both, the query is processed first, then the validation is applied.

### Ruleset signature

Client applications interact with a ruleset by using input and output parameters. These parameters are defined in the signature of a decision operation.

The parameters of a ruleset can have three directions:

- **INPUT**: The parameter value is provided as input to the ruleset on execution.
- **OUTPUT**: The parameter value is set by the execution of the ruleset and provided as output from the ruleset at execution completion.
- **INPUT\_OUTPUT**: The parameter value is provided as input to the ruleset on execution and its value can be modified by the ruleset and provided as output at execution completion.

You create the parameters from any of the ruleset variables that are available in the projects that are part of the scope of the decision operation. Ruleset variables are internal to a ruleset and provide a way to exchange data between rules, functions, and tasks.

**Parent topic:** [Decision artifacts](#)

**Related concepts:**

[Using queries](#)

**Related tasks:**

[Viewing the history of a rule](#)

[Restoring a version of a rule](#)



## Ruleset variables

A ruleset variable defines a value of a specified type and verbalization. You can use ruleset variables in all the business rules you add to a rule project, and as input and output parameters in decision operations.

### About this task

To define a ruleset variable, you first need to create a variable set in which to group your ruleset variables. You then define one or more variables in the variable set. Ruleset variables are variables that you can use in all the business rules of the ruleset. You can also use ruleset variables to pass data between tasks in a ruleflow. Ruleset variables are accessible from business rules only if you verbalize them.

### Procedure

1. In your Rules or BOM project, click the **create an artifact** button and select **New variable set**.
2. Enter a name for the variable set and click **Create**.
3. In the variable set editor, click **Add variable** to add a variable to this variable set.
4. Enter a name and a verbalization, and select a type for the variable.

Name: the name of the variable must be a valid Java™ identifier. A Java identifier can contain uppercase, lowercase, and modifier Unicode letters, Unicode digits, currency symbols, and the ASCII underscore (\_). The first character of the name must be a letter, a currency symbol, or an underscore.

Type: the type must be one of the available types in the drop-down menu, or a technical Java type.

Verbalization: specifying a verbalization is optional, but it is necessary to make the variable visible from business rules.

5. Click **Save**.

### Results

The ruleset variable is now defined and you can start using it in business rules or as input and output parameter in decision operations.

**Parent topic:** [Decision artifacts](#)

## Dynamic domains

A domain defines a set of values in your business terms. For example, a domain can define that the category of a customer can have one of the following values: silver, bronze, gold. You can modify these values and update your project in the Business console.

You must first define the domain in Rule Designer before you can access it from Decision Center.

Domains are available to Decision Center through a domain provider that you manage in an Excel file. You must upload any changes to the domain to Decision Center.

### Updating domains from an Excel file

You can modify the values of dynamic domains that are stored in an Excel file and update the BOM of your project with the new values.

**Parent topic:** [Decision artifacts](#)

# Updating domains from an Excel file

You can modify the values of dynamic domains that are stored in an Excel file and update the BOM of your project with the new values.

## Before you begin

You must use a new or existing change activity to update domains in projects that are using the governance framework in Decision Center.

## About this task

When a project contains a dynamic domain defined in an Excel file, the Excel file is stored as a resource in the project. You can modify the Excel file and then upload the changes to Decision Center.

The Excel file has a specific structure with columns that map properties in the BOM:

- A column for the domain values
- A column for the label of each value
- A column that defines the BOM-to-XOM mapping for each value

Other optional columns for the documentation or labels in other locales might be defined in the Excel file.

You can modify the values in each column, but you must not change the columns or column headings.

## Procedure

To edit the Excel file:

1. In the **Decision artifacts** tab, in the **Resources** default folder, browse to the resource that corresponds to the Excel file. If the **Resources** folder is not in the folders list, edit the artifacts filters to enable the **Resources** type.
2. Click the Excel file in the list to open the preview.
3. To download the file, click its name.
4. Open the Excel file, modify the values, and save the changes.

You can now upload the modified Excel file to Decision Center.

5. In the preview of your file in the Business console, click the edit button, then click **Choose**.
6. Select the new version of the file and click the save icon.
7. Open the **Model** tab.

This tab lists the domains of the project. You can expand a domain to see its current values and the updates that have been made in the Excel file.

8. To update the BOM with the new values, select the domains that you want to update and click **Apply changes**. This process can take several minutes to complete if the domain contains many values.

**Parent topic:** [Dynamic domains](#)

## Versions and history

Decision Center creates versions of elements that you can view in a timeline and restore.

### [Versioning in decision services](#)

Decision Center creates archived versions of elements each time you modify them, including moving or renaming them.

### [Viewing the timeline of a release or activity](#)

You can view the complete history of a release or activity.

### [Viewing the history of a rule](#)

Decision Center keeps a history of all the changes that you make to your rules.

### [Comparing versions of a rule](#)

You can compare two versions of an action rule or decision table.

### [Restoring a version of a rule](#)

You can restore a previous version of a rule to the current state.

**Parent topic:** [Decision artifacts](#)

## Versioning in decision services

Decision Center creates archived versions of elements each time you modify them, including moving or renaming them.

You can consult the history of modifications and restore a previous version. However, you can edit or delete only the current version of an element.

The version number of an element is in the form  $x.y$ , where  $x$  is the major number that corresponds to a release or activity in a decision service and  $y$  is the minor number that is increased with each change:

### Major number

Decision Center gives every branch of a decision service a major version number. The version number 1.0 is given to the main branch created during the initial publication from Rule Designer. Branch 2 is always the Initial Release, which is also created automatically during the initial publication. Branch 3 is the first release that you create. Then, branch 4 is the first change activity and so on.

### Minor number

The initial version of an element has the minor version number 0. This minor version number is increased by 1 each time you edit the element. In a release or change activity, each element initially has the major and minor version number of its parent branch. When you edit the element in an activity for the first time, this element takes on the version numbers of the activity. For example, an element with the version number 3.0 keeps this number in the activity until you edit the element for the first time, when it becomes, for example 4.0. Subsequent changes that you make to this rule in the change activity gives it version number 4.1, 4.2, and so on.

When you complete the change activity, it is merged with the release. The major version number of any rule modification done in a change activity adopts the major version of the release. The new version is an exact copy of the version in the activity, and takes the next minor version number in the parent release.

When you restore a snapshot, Decision Center creates a version of all the elements that are found in the snapshot, and copies them to the current state of the release or activity.

When you restore a version, either from a snapshot or from the timeline of a change activity, the new version is a copy of the restored version, and takes the next version available number.

#### Note:

You cannot clear the history of an element.

**Parent topic:** [Versions and history](#)

#### Related tasks:

[Viewing the history of a rule](#)

[Restoring a version of a rule](#)

[Viewing the timeline of a release or activity](#)

# Viewing the timeline of a release or activity

You can view the complete history of a release or activity.

## About this task

A timeline shows the events that occur during the life of a decision service release or change activity, such as:

- Creation or deletion of a release or activity, or changes to its name or description.
- Creation, modification, restoration, or deletion of a rule.
- Creation, restoration, or deletion of a snapshot, or changes to the name or description of the snapshot
- Changes to the status a release or activity.

**Note:** Each event in the timeline contains a **Comments** field for you to enter comments to add to the stream.

## Procedure

1. In the **Project** view, click **Timeline**.
2. Select an entry in the **Show** field to reduce the number of visible events to rules, activities, or snapshots.
3. Hover over the **Comments** section of an event and click the arrow to display events that relate only to that project element.
4. Click **Exit Timeline** to exit the timeline.

**Parent topic:** [Versions and history](#)

### Related concepts:

[Versioning in decision services](#)  
[Snapshots](#)

### Related tasks:

[Viewing the history of a rule](#)  
[Restoring a version of a rule](#)  
[Creating a snapshot](#)  
[Redeploying](#)

# Viewing the history of a rule

Decision Center keeps a history of all the changes that you make to your rules.

## About this task

The history of changes to rules is kept in a timeline, with the most recent version at the top. Versions are shown alternating on each side of the timeline down the page.

You cannot edit previous versions of a rule, but you can restore them to the current state.

<b>Note:</b> You cannot delete previous versions.
---------------------------------------------------

## Procedure

1. Click the name of the rule to access its **Details** view.
2. Click **Timeline**.
3. In the timeline, click an event to display its contents in read-only mode, or enter a comment to publish it to the activity stream.
4. Click **Exit Timeline** to return to the **Details** view.

**Parent topic:** [Versions and history](#)

### Related concepts:

[Versioning in decision services](#)

[Decision operations](#)

### Related tasks:

[Restoring a version of a rule](#)

[Viewing the timeline of a release or activity](#)

## Comparing versions of a rule

You can compare two versions of an action rule or decision table.

### About this task

When you compare two versions of a rule, Decision Center shows a side-by-side comparison of the differences in the contents or the properties.

### Procedure

1. Start in one of the following ways:
  - In the **Project** view, hover on the rule and click **Compare** in the drop-down menu.
  - Click **Compare** in the **Details** view for the rule.
  - Hover on the rule in the timeline or in a snapshot and click **Compare**.
2. Click two versions of the rule, and then click **Compare**. You can click a selected version to clear it.
3. Decision Center displays the total number of differences between the two versions. Select **Content** to see which rules or rows were added or deleted, or **Properties** to compare the properties.
4. Click **X** under the top banner to return to the current version of the rule.

**Parent topic:** [Versions and history](#)



# Restoring a version of a rule

You can restore a previous version of a rule to the current state.

## About this task

Each rule has a timeline that contains versions of the rule. You can reinstate a previous version of the rule from the timeline.

### Note:

In a decision service, you cannot restore a previous version of a rule directly to a release. Changes to a release are made in change activities, and when the changes are approved, they are merged back into the release.

## Procedure

1. In the timeline, click the version that you want to restore.
2. In the toolbar, click **Restore**.
3. Enter a comment and click **Restore Version**.

## Results

Decision Center displays the new version in its **Details** view, and the history of the rule is updated.

**Parent topic:** [Versions and history](#)

### Related concepts:

[Versioning in decision services](#)

[Decision operations](#)

### Related tasks:

[Viewing the history of a rule](#)

[Viewing the timeline of a release or activity](#)

## Technical rules

Technical rules are a useful alternative to business rules. They support constructs unavailable in BAL such as loops, and features unavailable in BAL such as explicit IRL mapping. Technical rules are written using the ILOG® Rule Language (IRL). IRL is a Java™-like rule language that can be executed directly by the rule engine.

In the Business console, technical rules are shown when you activate the type **Technical Rules** in the **Decision Artifacts** tab.

Technical rules are an advanced subject, useful for developers. See the [Rule Designer](#) documentation for more information.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
  - `evaluate`
  - `exists`
  - `from`
  - `in`
  - `instanceof`
  - `not`
  - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

**Parent topic:** [Decision artifacts](#)

## Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG® Rule Language (IRL) and its code is evaluated when the ruleset runs.

Functions are an advanced subject, useful for developers. See the [Rule Designer](#) documentation for more information.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)
{
 code
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the `return` keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of `return` that does not return a value:

```
return;
```

**Note:** A function is not required to declare in its `throws` statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

**Parent topic:** [Decision artifacts](#)

## Rule verification

You can perform different levels of verification on your rules.

### Basic syntax checks

Decision Center verifies the syntax of business rules.

### Decision table checks

You can check decision tables for overlaps, gaps, and symmetry.

**Parent topic:** [Decision artifacts](#)

## Basic syntax checks

Decision Center verifies the syntax of business rules.

When you save a rule, Decision Center displays any errors or warnings that result from checking the syntax. If a rule contains a syntax error, the administrator excludes it from the ruleset that is deployed to the production system.

Decision Center checks for the following syntax errors:

- Unrecognizable or invalid rule statement
- Ambiguous rule statement with more than one correct interpretation
- Repeated or wrong variable in the definitions part
- Incomplete rule with empty placeholders
- No if part in a rule with an else part

For a full list, see the **Rule Designer** online help.

**Parent topic:** [Rule verification](#)

## Decision table checks

You can check decision tables for overlaps, gaps, and symmetry.

### Overlap checks

You can set Decision Center to check that your decision tables do not contain duplicated values.

For example, in the following condition statements, customers who are 29 or 30 years old satisfy both rules:

- Age is between 17 and 30
- Age is between 29 and 40

If these two conditions are in the same column or within a partitioned groups of cells in a decision table, Decision Center signals an overlap error.

### Gap checks

You can set Decision Center to check that the values you enter in your decision tables consider all possible cases. This ensures that there are no gaps in your conditions.

For example, the following condition statements do not include customers who are 31 years old:

- Age is between 17 and 30
- Age is between 32 and 40

Decision Center reports a gap error.

### Symmetry checks

You can set Decision Center to check that all the partitions in a decision table use the same set of items. You can verify the symmetry of the entire table or specified columns. This analysis helps ensure that you have covered all the possible choices in the decision table.

By default, symmetry analysis is turned off so that you can create nonsymmetrical tables.

**Parent topic:** [Rule verification](#)

# Rule properties

Properties associate additional information with a project element, including the development status, storage location, and user group.

The properties are displayed in the **Properties** tab next to the project element.

To edit the properties, click **Details** in the edit view of the project element. If you have the correct permissions, you can change the properties.

Before you change the properties in a project, make sure that your development team implements the use of properties. Otherwise, your changes might not be used.

**Note:**

You might have project properties that are specific to your business domain.

The following table shows the default properties:

Prop erty	Description
Folder	Set the location of the element within the project.
Statu s	Set the status of a rule, for example: <ul style="list-style-type: none"><li>• New</li><li>• Defined</li><li>• Validated</li><li>• Rejected</li><li>• Deployable</li><li>• Inactive</li></ul>
Categ ories	Select a category for the project element. The default category is Any. Categories can be used to filter rules in queries.
Group	Associate the rule with a specific group. The <b>Group</b> property is used by the administrator to establish permissions to view, edit, create, and delete project elements.
Priorit y	Set the priority of the rule to define the order in which rules are executed.
Effecti ve Date	Specify the date on which the rule goes into effect.
Expira tion Date	Specify the date on which the rule expires.
Tags	Add tags to your action rule, decision table, or ruleflow.

**Parent topic:** [Decision artifacts](#)

**Related concepts:**  
[Getting familiar with the Business console](#)

## Tags for rule artifacts

You use tags to store data with your rule artifacts (action rules, decision tables, ruleflows).

For example, you can store data related to the geography of the rule or its development phase. You can retrieve this information in a query or in a report.

Tags are visible in the properties of your rule artifact, which you can access by selecting the artifact in the **Decision Artifacts** tab, then clicking **Show details**. To add or remove tags, edit your rule artifact, click **Details**, and select the **Tags** tab. When you add a tag, you declare a name and give it a value, for example:

Name	Value
Geography	New York
Phase	1

For developers, tags can be useful when you have implemented a rule model extension in Rule Designer and you synchronize with Decision Center. In this case, extended properties are converted into tags so that the synchronization remains smooth. Later, if you implement the rule model extension in both Rule Designer and Decision Center, these tags are converted back to properties.

**Note:** If you defined tags on rule packages in Rule Designer, you cannot synchronize them with Decision Center, because Decision Center does not support tags on rule packages.

**Parent topic:** [Decision artifacts](#)



## Rule overriding

You use rule overriding to give rules precedence over other rules.

For example, you can set an entire decision table to override another decision table.

A rule that overrides one or more other rules is executed in place of the rules that are overridden. Normally, rule overriding operates within a hierarchy of rules, with a local or more specific rule overriding a more general rule. For example, you can set a minimum customer age for a particular state to take precedence over a minimum customer age for the country.

If an overridden rule is selected in the same rule task of a ruleflow, then this rule is filtered out and not executed.

Overridden rules are visible in the properties of your rule artifact, which you can access by selecting the artifact in the **Decision Artifacts** tab, then clicking **Show details**. To set rule overriding, edit your rule artifact, click **Details**, and select the **Overridden Rules** tab. You can then add the rules that you want the current rule to override.

**Parent topic:** [Decision artifacts](#)

## Modeling decisions in the Business console

Generally, IT creates the decision model on which rules are authored. For decision model services, you create both the model and the logic directly in the Business console.

### [Creating or importing a decision model service](#)

You can create a decision model service from scratch, or import an existing project that was created in IBM Decision Composer.

### [Decision modeling basics](#)

In Decision Center, you the business expert can create and deploy a decision service that gets called by client applications when they require a specific business decision.

### [More on data modeling: Custom types, default values, lists](#)

In addition to creating the data nodes, data modeling requires you to assign the appropriate type to all the nodes.

### [More on decision authoring: Coverage and rule interactions](#)

Authoring the decision logic requires you to create the business rules, but also to consider coverage and rule interactions.

### [Deploying a decision model service](#)

You deploy a decision model service to an execution environment using a deployment configuration, the same way as you deploy a regular decision service.

### [Validating and testing decision model services](#)

With a decision model service, you can validate your decision model during development. You can also create more complete usage scenarios that you use in test suites and simulations, just like regular decision services.

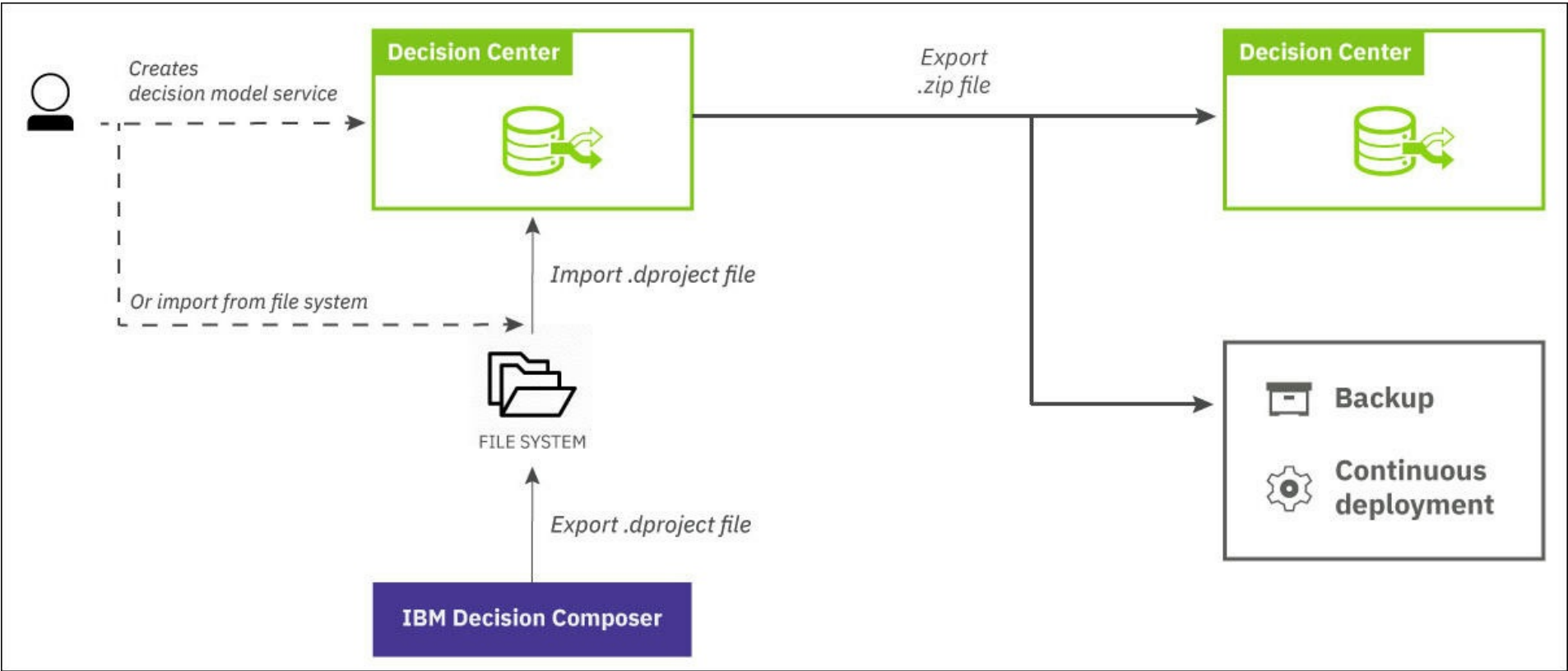
**Parent topic:** [Working with the Business console](#)

# Creating or importing a decision model service

You can create a decision model service from scratch, or import an existing project that was created in IBM Decision Composer.


Decision projects that are exported from IBM Decision Composer as .dproject files can be imported to Decision Center, where they become integrated as decision model services. You can then export a decision model service as a .zip file to other Decision Center repositories, or to a file system for:

- Regular backup of the project.
- Use within a continuous deployment pipeline.



A decision model service makes use of a diagram in which you link decision and data nodes, and then author the decision logic (see [Decision modeling basics](#)). With a decision model service, there is no interaction with Rule Designer because the model on which you write the decision logic is part of the diagram.

A decision model service behaves the same way as a regular decision service, with the following differences:

Features in decision model services	Description
A decision model service contains only one artifact, which is the decision model.	You create rules and decision tables directly in the decision nodes.
There is no ruleflow to determine rule execution.	<p>A decision model service makes use of a diagram to link decision and data nodes, which all contribute to the final decision.</p> <p>Decision Center generates an empty ruleflow for integration, but the ruleflow is not editable in the Business console.</p>
The decision model works as one versionable element.	Changes to individual business rules, decision tables, and the nodes can be consulted through the version of the decision model itself.
Cannot query or text search individual rules.	The <b>Queries</b> tab and the search field are not visible.
Viewing and comparing versions of a decision model.	To view, compare, and restore versions of a decision model, open the <b>Decision model view</b>  and use the <b>Compare</b> tool.
Cannot edit the decision operation.	In a decision model service, the decision operation includes the entire decision model. The decision operation is automatically regenerated every time that you save (see <a href="#">Deploying a decision model service</a> .)
Using the decision governance framework.	<p>Because a decision model works as one unit, you cannot create separate change activities to work simultaneously on different parts of the decision model.</p> <p>For more information, see <a href="#">Governance principles</a>.</p>

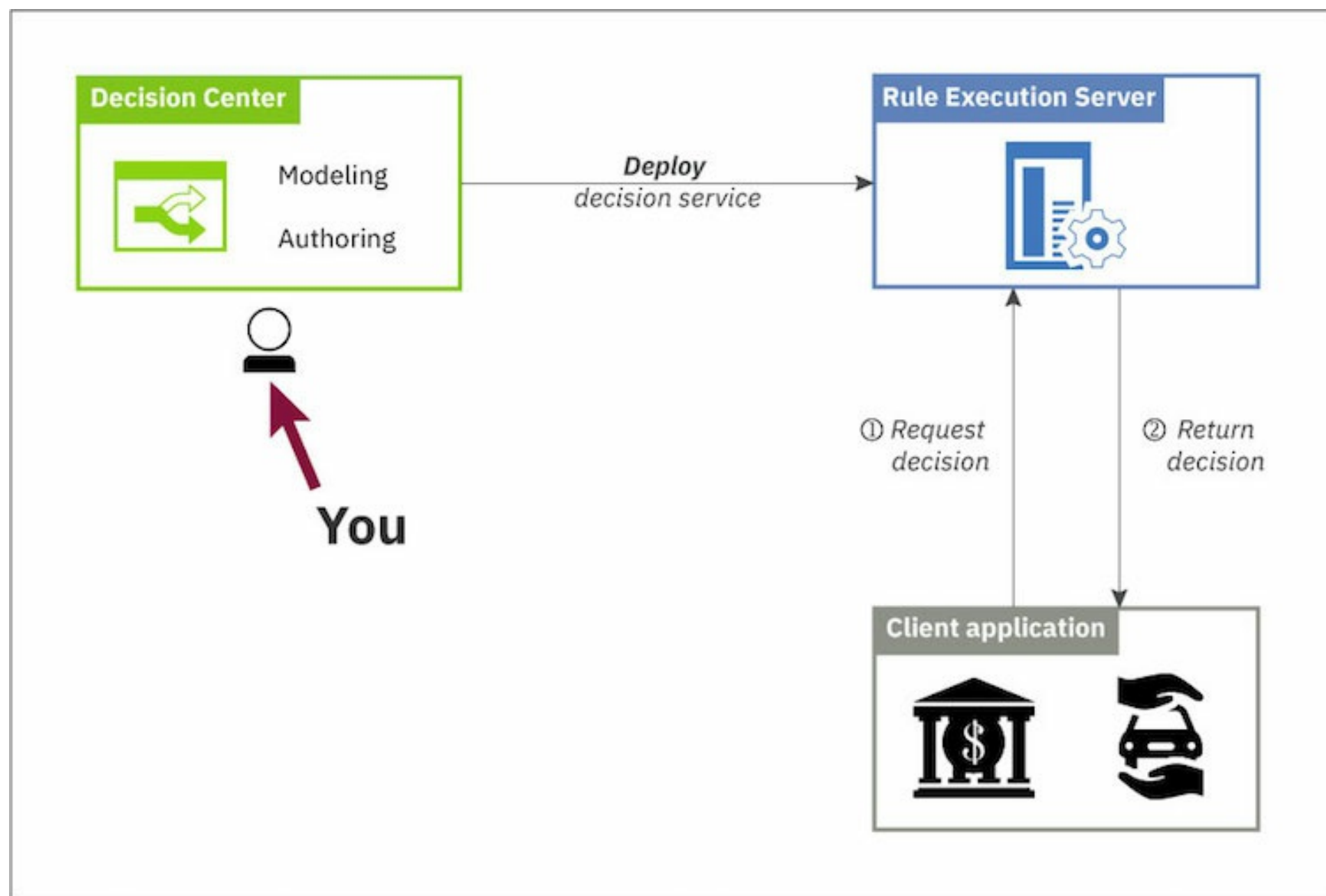
Parent topic: [Modeling decisions in the Business console](#)



## Decision modeling basics

In Decision Center, you the business expert can create and deploy a decision service that gets called by client applications when they require a specific business decision.

You create this decision service in Decision Center and then deploy it to an execution environment. Client applications requesting this business decision call your decision service in the execution environment, as shown here:



In Decision Center, you can create decision services in one of two different, distinct ways:

1. Your development team creates the underlying model, and you, the business expert, author the decision logic based on this model.
2. You, the business expert, create both the model and the logic of the decision service. The term **decision model service** distinguishes this second case from the first one.

This section describes a decision model service by presenting two separate but intertwined notions:

- Modeling the node structure
- Authoring the decision logic

**Note:** For simplicity, the term decision service is used throughout, except when there is the need to distinguish the two approaches or when referring specifically to the modeling part of the decision service.

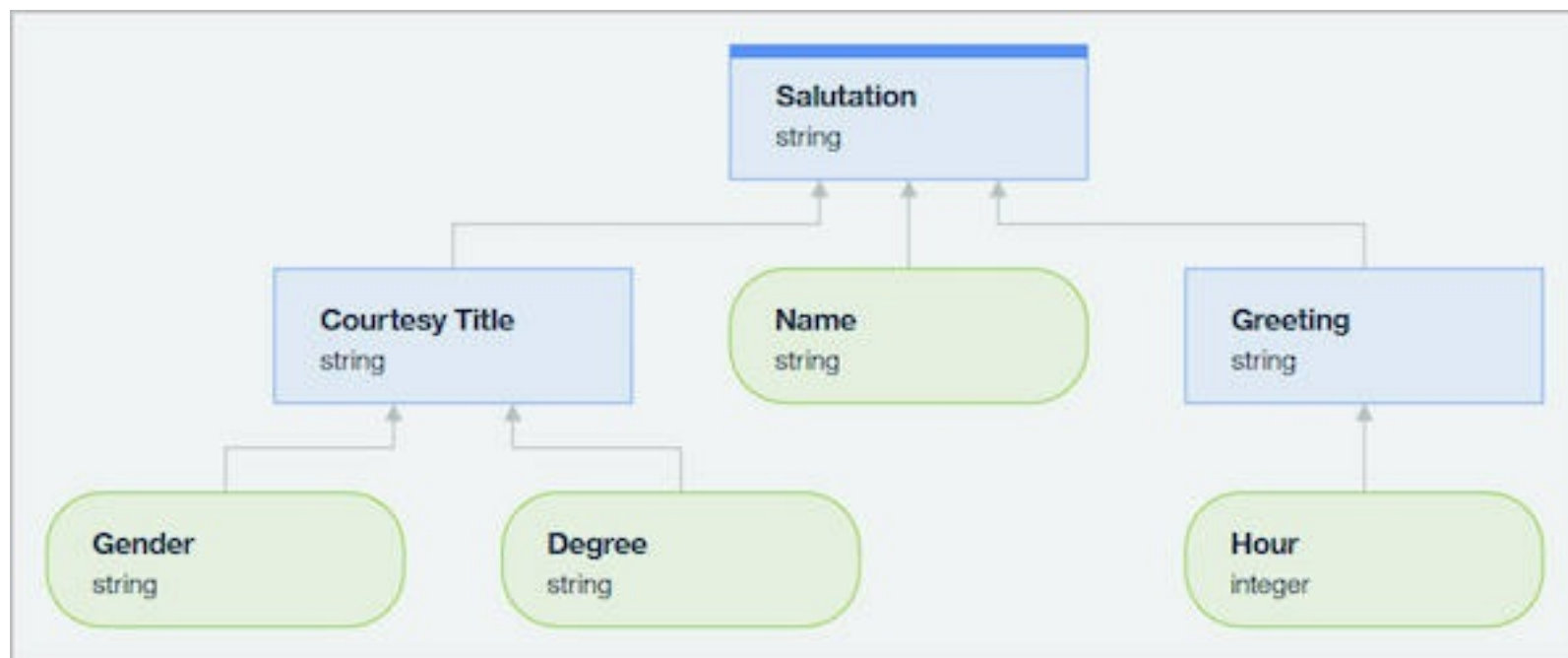
### Modeling the node structure

Consider a simple decision service that greets a customer with a salutation. An example of the salutation is "Good evening Mrs. Jones".

The information required from the client application for your decision service to generate the appropriate salutation is:

- The name of the customer
- The gender and any degree (Dr, Professor, ...) that the customer has
- The hour of the day

The model of this decision service could look like this:



From a modeling perspective, you the business expert create and define the relationship between:

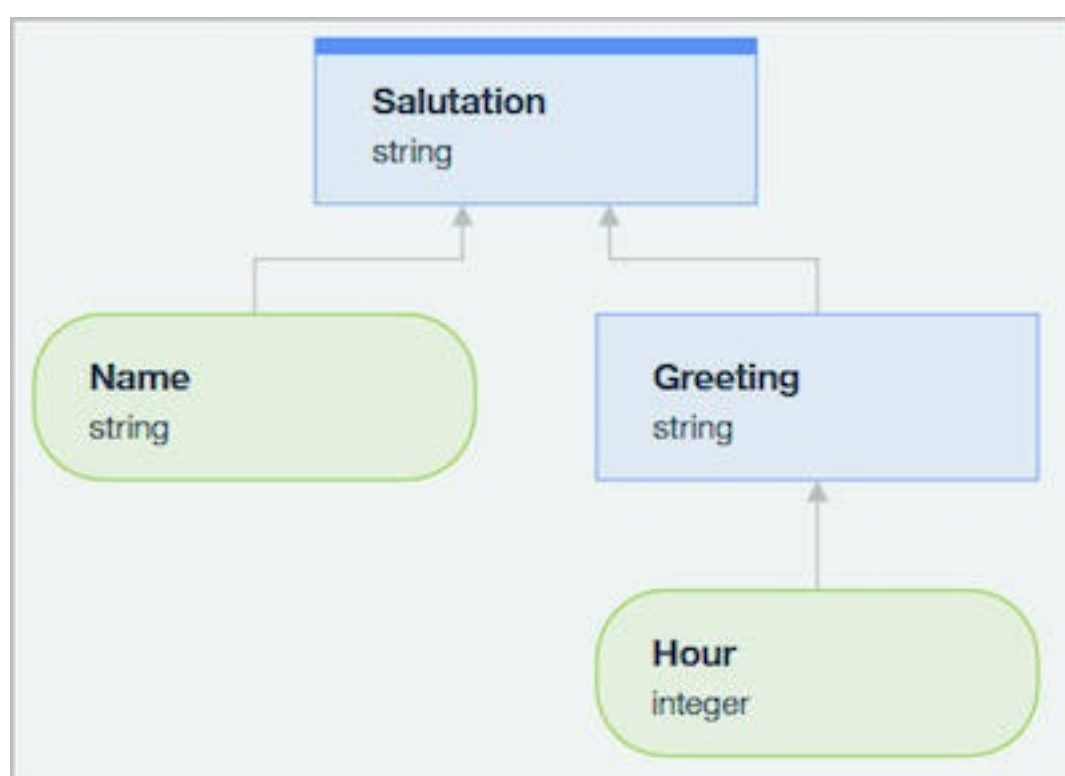
- Visual representations of the decisions required.
- Visual representations of the information (data) that must be provided to make each decision.

In the diagram, these visual representations are called **nodes**. The following shows a decision node 'Salutation' fed by a data node 'Name':



At this point, the salutation returned by your decision service when given 'Jones' for the name would be: "Hello Jones."

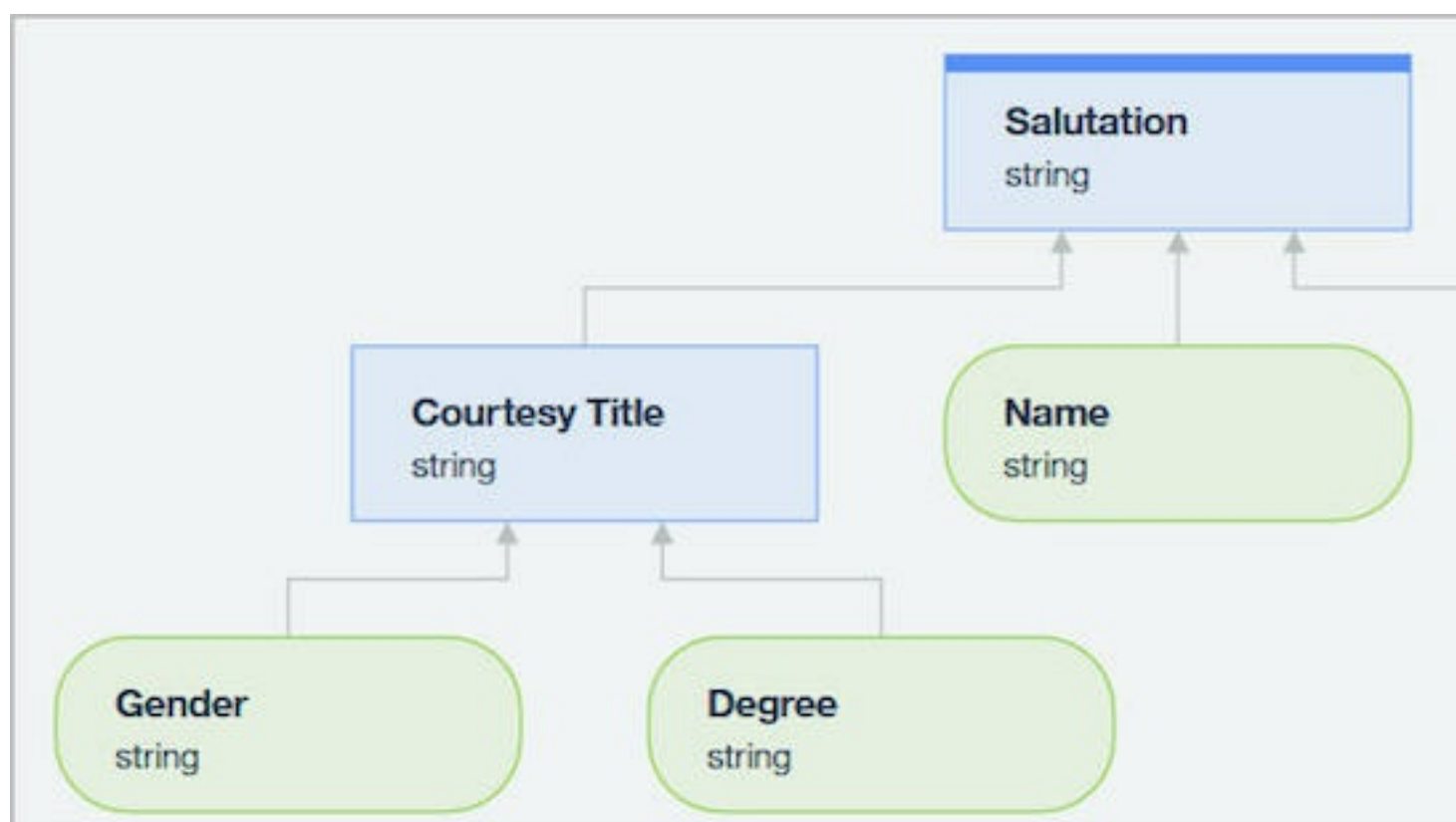
To make the diagram of your decision service easier to understand and maintain, you can make a decision node dependent on other decision nodes that handle a specific aspect. For example, to refine your decision service so that it returns a salutation based on the time of day, you create another decision node called 'Greeting' that uses 'Hour' as input, and link this decision node so that its result can be used by the 'Salutation' node:



At this point the salutation returned by your decision service when given the name 'Jones' at 8 p.m. would be: "Good evening Jones."

Then, to add a courtesy title (Mr, Mrs, Dr, and so on) to personalize your decision service further, you create a decision node 'Courtesy Title' fed by two data nodes, one for gender and one for degree. This node feeds the final decision node, just like the 'Greeting' node:





The salutation returned by your decision service when given 'Jones' at 8 p.m. with the gender Mrs. and no degree would be: "Good evening Mrs. Jones."

When a client application provides the information, your decision service processes this data and returns the decision. To summarize the modeling part:

- You create and link together decision and data nodes.
- You design each decision node to accept input data and output the result to the node above it or to the client application. How the result is obtained is the subject of [Authoring the decision logic](#) below.

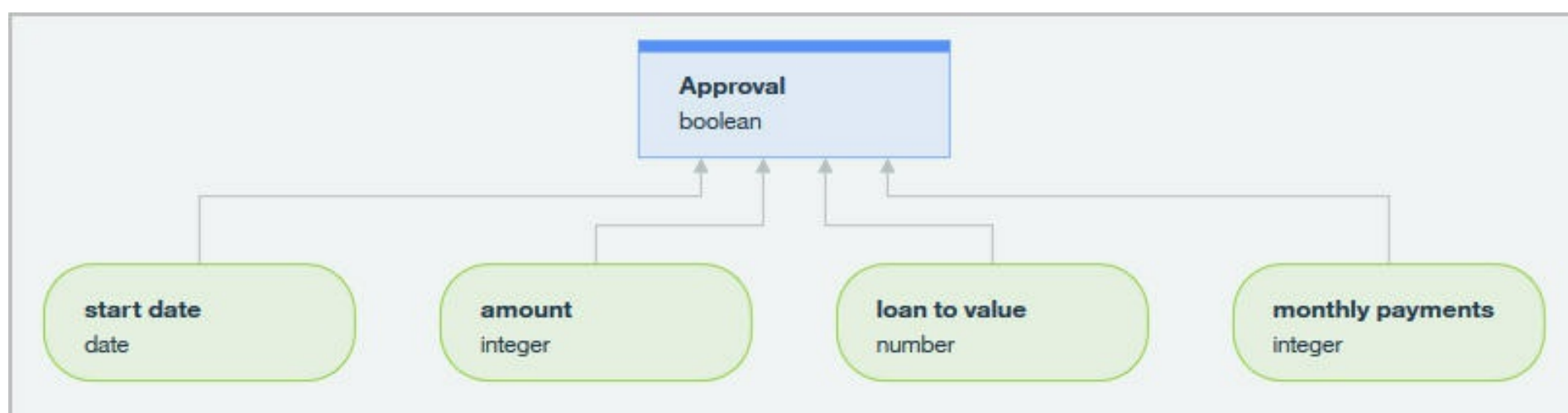
## Data model

The term **data model** can be used to distinguish specifically the data aspects of the decision model.

In addition to creating the data nodes, data modeling requires you to assign the appropriate **type** to all the nodes. Assigning a type to a node specifies:

1. What information the node can accept and send, for example, a number, date, and so on.
2. The possibilities when authoring the decision logic.

In the following example, four different data nodes, of various types, feed the decision node above it:



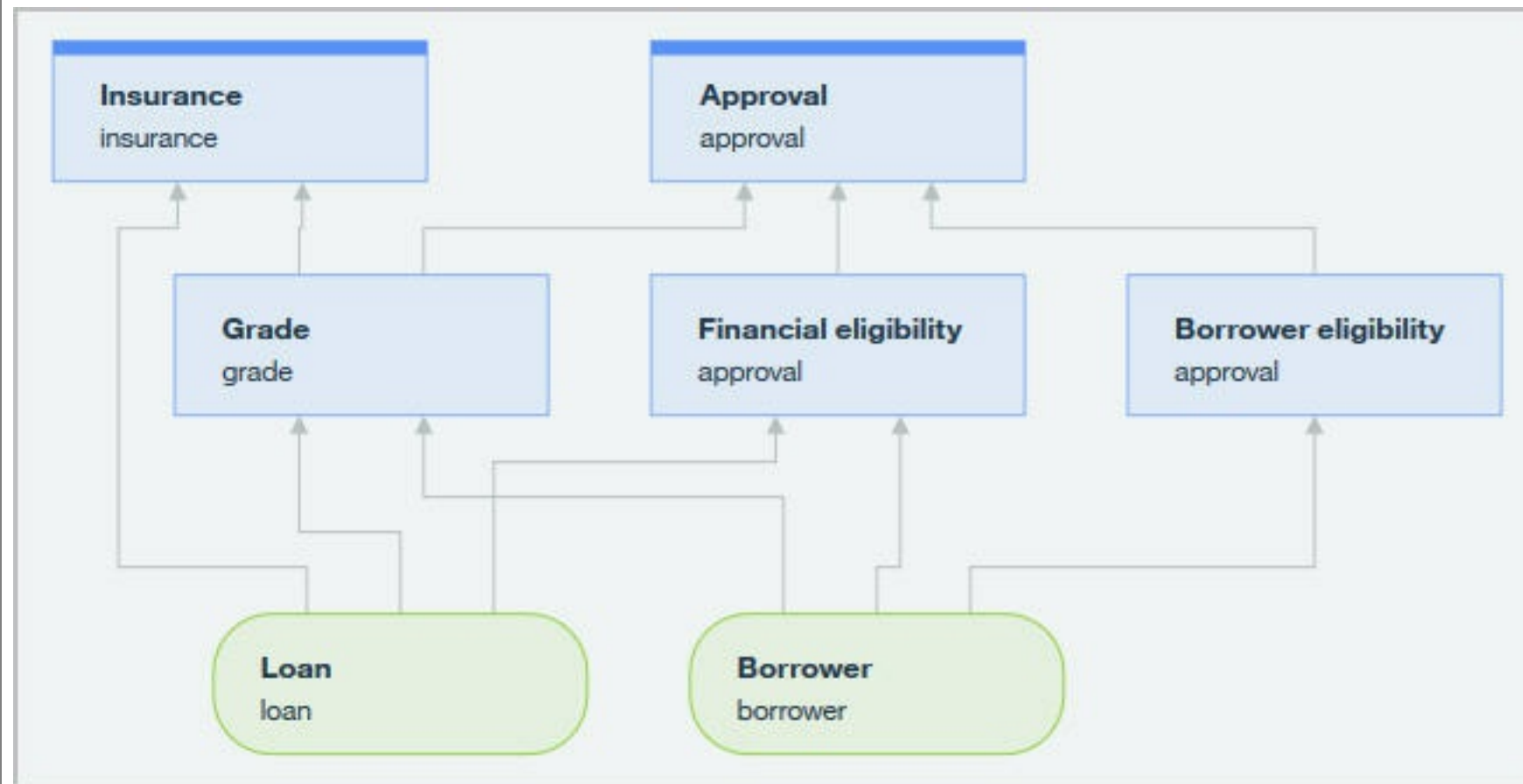
In this diagram, the four data nodes are laid-out flat. It is often more convenient for you to create **custom types**, which regroup related data under one entity. The diagram below contains the same input data as the diagram above, but all the data nodes relating to the loan have been regrouped under one node that handles data of a custom type 'loan'. This type regroups all the attributes of the loan:



Data modeling is described in more detail in [More on data modeling: Custom types, default values, lists](#), which also explains default values that you can set on the data nodes.

**Note:** The example used so far is simple. A more complex decision service can have more than one top-level

decision node, that is, it can return more than one decision to the client application. Also, a data node may feed more than one decision node. The following example shows both these cases:



## Authoring the decision logic

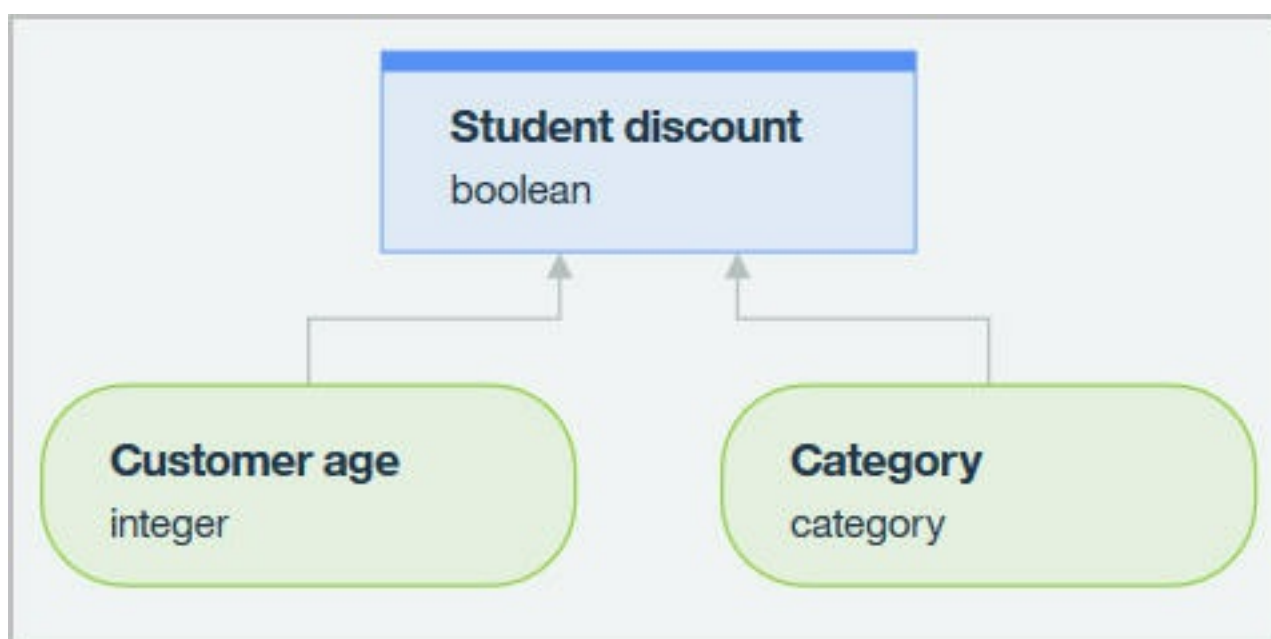
You author the logic of each decision node by creating one or more *business rules* that:

- Evaluate the input data that each decision node receives against conditions that you establish.
- Take actions to set or modify the output of the decision node, that is, the **decision**.

These business rules are simply your business policies expressed in the following form, understandable by a computer:

```
if
 <conditions>
then
 <actions>
```

Consider a decision service that decides whether a customer is eligible for a student discount, based on the customer being under 18 **and** a student. The decision model has two data nodes 'Customer age' and 'Category' that feed the decision node 'Student discount':



You implement the decision logic of the 'Student discount' node by creating the following business rule:

```
if
 'the customer age' is less than 18
 and 'the category' is Student
then
 set 'the student discount' to true,
 print "You benefit from the 20% young student discount!" ;
```



In this example, the content of the business rule is color-coded to demonstrate the different building blocks at your disposal when creating or modifying a business rule, as follows:

- The yellow parts represent the input or output data available to the decision node.
- The grey parts represent values of the data that you identify as pertinent to authoring the decision logic.
- The rest corresponds to the modeling language itself, which is used to compare, evaluate, and modify the different types of data (see [Decision Center modeling language reference](#)).

In other words, the data handled by a decision node, that is, its output and the data from the nodes that feed it directly, can be used immediately when authoring your business rules, in the form 'the <data>'. In the example, *Student discount* is usable in the rules as 'the student discount', *Customer age* is usable as 'the customer age', and *Category* is usable as 'the category'.

To further illustrate this important point, each time that you create a new rule, Decision Center evaluates the data available as input to the decision node in which you are creating the rule. Then, it proposes, as a starting point for your new rule:

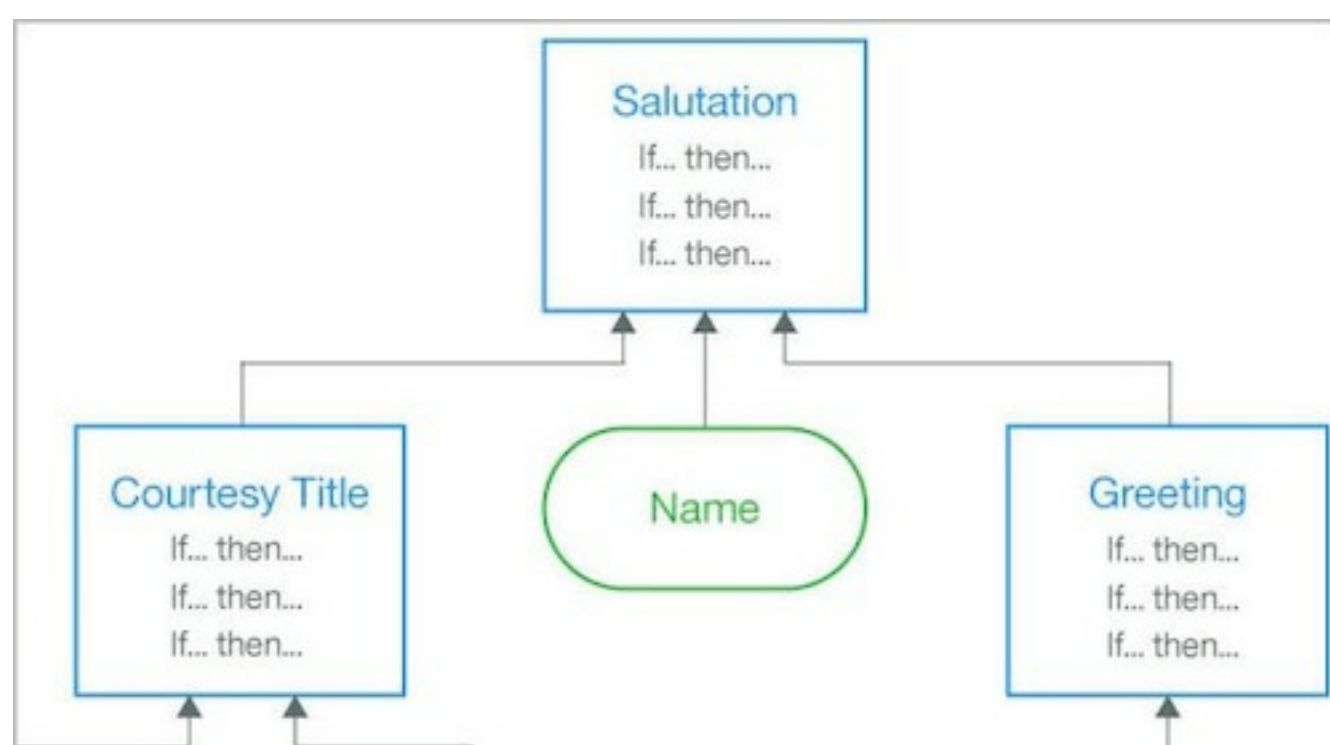
1. A typical condition statement (in the **if** part) comparing the input data according to its type. It proposes a condition statement for all the data available to the node, but you can clear them, since not all rules act on all the input data at the same time.
2. An action statement (in the **then** part) setting the value of the output of the node according to its type.

```
if
 'the customer age' is at least <min> and less than <max>
 and 'the category' is <a category>
then
 set decision to <a boolean>;
```

**Note:** The modeling language contains aspects that make your business rules more readable. For example, the keyword **decision** can be used interchangeably with the name of the node, which helps identify visually this important aspect of the rule, say in a report.

## Coverage

When you author a decision node, you create the business rules required to reach a decision for the possible cases that the client application might provide as input data. This is referred to as coverage. Typically a decision requires several business rules to cover as many cases as possible:



To handle situations where there are many possible conditions for the input data provided, you can use decision tables, which regroup business rules that have a common structure but different values for the conditions and actions. For example, to set a salary score based on income, many business rules are required to cover different ranges of income. The following decision table handles this in a simpler way:

	Yearly income		Salary Score
	min	max	
1	< 10,000		21
2	10,000	20,000	50
3	20,000	30,000	80
4	30,000	50,000	120
5	50,000	80,000	150
6	80,000	120,000	200
7	120,000	200,000	250
8	≥ 200,000		300

Each row in a decision table corresponds to a business rule. The columns on the left represent the conditions (the **if** part of the rule) and the columns on the right specifies the decision returned for that case.

The decision logic can require one or more decision tables and some business rules to get complete coverage. Note that in these cases, the order in which the rules appear in your node is important (see [More on decision authoring: Coverage and rule interactions](#)).

**Parent topic:** [Modeling decisions in the Business console](#)

## More on data modeling: Custom types, default values, lists

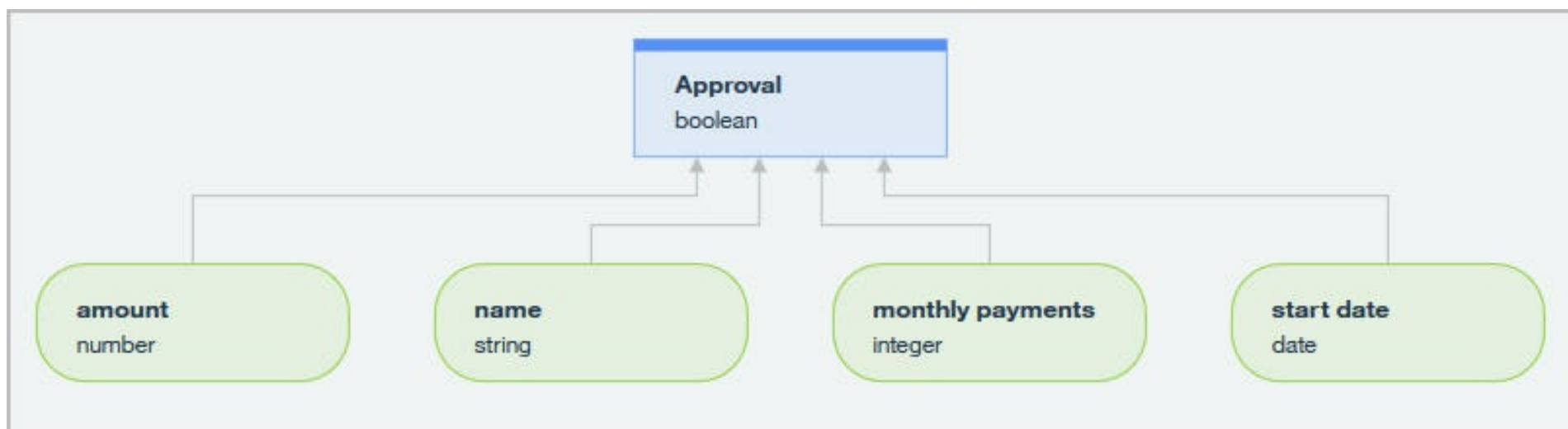
In addition to creating the data nodes, data modeling requires you to assign the appropriate type to all the nodes.

**Note:** This topic builds further on the information presented in [Decision modeling basics](#), and assumes that you are familiar with decision nodes, data nodes, and decision logic.

Assigning a type to a node determines:

1. What information the node can accept and send.
2. The possibilities offered in the rule and decision table editors when authoring the decision logic.

The following example shows a decision service that decides if a loan is approved. The diagram has four different data nodes feeding the decision node above it:



All the data nodes in this example handle a simple, standard **system** data type:

- **amount** handles numbers
- **name** handles text (strings)
- **monthly payments** handles integers
- **start date** handles dates

The **Approval** decision node is of type `boolean` (true/false). The node accepts data of various types, and then returns a decision of type true/false to the client application that makes a request.

Any business rule or decision table that you create in the **Approval** node will be made up of:

1. Condition statements based on the type of the data nodes that feed it directly.
2. A decision that sets the value of the output according to its type.

Each time you create a new business rule or decision table, Decision Center proposes, for your convenience, condition and action statements based on the most common formulation for the types of data handled by the decision node. In the **Approval** decision node, a business rule involving all the data nodes initially appears as follows:

```
if
 'the amount' is at least <min> and less than <max>
 and 'the monthly payments' is at least <min> and less than <max>
 and 'the name' is <a string>
 and 'the start date' is on or after <a date> and before <a date>
then
 set decision to <a boolean>;
```

From this starting point, you fill in the placeholders, for example `<min>` and `<max>`, with values pertinent to your decision logic, or change the statements with a more appropriate one from the modeling language. The modeling language contains extensive constructs to compare, evaluate, and modify data and decisions based on their type (see [Decision Center modeling language reference](#)).

### Custom types

An important consideration when you create the data model is to create **custom types**, which regroup related data under one entity.

In our example, the four data nodes that feed the **Approval** decision node are laid-out flat. Because all the data nodes are not themselves the result of a previous decision, and they all feed the same decision node, these data nodes can be replaced with a single data node without any loss of information.

This makes the diagram easier to read. A second benefit is that it is not necessary to change the diagram when you add new attributes to the custom type.

To obtain a node that regroups all the data relating to the loan, you create the custom type **loan**, which sets as attributes all the data that was previously laid-out flat:

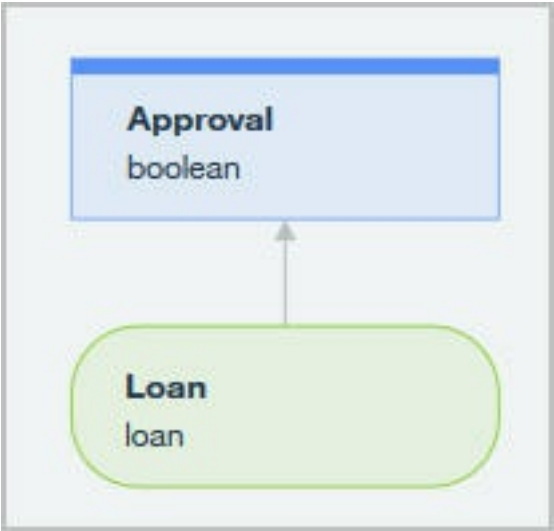
Name:

loan

Attributes:

Name	Type
amount	number ▼
name	string ▼
monthly payments	integer ▼
start date	date ▼

Then you create a single data node, here called **Loan**, and give it your new custom type loan:



The business rules and decision tables that you write now use the formulation the attribute of 'the node name', for example the amount of 'the loan'. This makes for more explicit condition statements, which is convenient on more complex business rules. The starting point proposed by the editors now looks like this:

```
if
 the amount of 'the loan' is at least <min> and less than <max>
 and the monthly payments of 'the loan' is at least <min> and less than
<max>
 and the name of 'the loan' is <a string>
 and the start date of 'the loan' is on or after <a date> and before <a
date>
then
 set decision to <a boolean>;
```

**Enumerations**

Another possible custom data type is an **enumeration**. An enumeration is a data type in which you establish all the possible values beforehand. In the following example, you see an enumeration called **category** with its possible values:

Name:

category

Values:

student

employed

retired

You should use enumerations over text entries for the following reasons:

- The business rule and decision table editors propose the values from the ones in your enumeration. This improves the rule editing experience, and it also helps prevent data entry errors.
- The correctness of your rules is verified immediately by comparing the rule statements with the values of your enumeration. If you use text entries, errors in the text will only be revealed during execution.
- Gaps in your decision tables are immediately revealed by the decision table editor.

## Default values on data

You can set default values on decision nodes or data nodes, to provide a fall-back value if no value is available. For decision nodes, setting a default value is an important aspect of coverage, described in [More on decision authoring: Coverage and rule interactions](#).

For data nodes, you can set default values for the following purposes:

1. The data handled by the node is optional, and might not always be provided by the client application. For example, if the customer's country is USA for almost all transactions, the client application might only provide this information if the country is different.
2. For convenience, the validation feature uses the default value if no value is provided in the input data set. This can reduce your data entry in the data sets that you create.

The following examples show how to set the default value on a data node:

- On a system data type:

```
set 'the gender' to "Female";
```

- On a custom data type:

```
set 'the loan' to a new loan where
 the amount is 100000,
 the monthly repayment is 688;
```

## Lists

You use lists when your decision acts on more than one item of something, and you need to make a distinction between the items. For example, when you check-in baggage on a plane:

- You use a list if your decision is about checking-in oversized baggage, since more than one of the baggages may be oversized.
- You do not use a list if your decision is about the fee charged, since there is only one fee for the sum of the baggages checked-in.

In the diagram, you specify whether a data node, decision node, or custom type is a list using a check box.

When checked, constructs to handle lists are then available in your rules, such as:

- there are at least <number><items> in <list>
- for each <item> in a <list>
- ...

See [Decision Center modeling language reference](#) for the available constructs.

**Parent topic:** [Modeling decisions in the Business console](#)



# More on decision authoring: Coverage and rule interactions

Authoring the decision logic requires you to create the business rules, but also to consider coverage and rule interactions.

**Note:** This topic builds further on the information presented in [Decision modeling basics](#), and assumes that you are now familiar with decision nodes, data nodes, decision logic, and how these work together.

You author the logic of each decision node in your diagram by creating one or more *business rules* that:

- Evaluate the input data that each decision node receives against conditions that you establish.
- Take actions to set or modify the output of the decision node, that is, the **decision**.

Creating rules and decision tables is essentially the same for a decision model service as it is for a regular decision service:

- For rules, see [Action rules](#).
- For decision tables, see [Decision tables](#).

See [Decision Center modeling language reference](#) for the available constructs.

When creating rules or decision tables in a decision model service, the only information available for making a decision are the values of the direct predecessors of the corresponding decision node. You cannot use a rule to change these variables or those from another dependency.

When a rule tries to modify an input value, Decision Center creates a copy of the data so that the input value is never modified.

## Coverage

When you author a decision node, you create the rules required to reach a decision for the possible cases that the client application might provide as input data. In practice, it is usually not possible to consider all possible cases, even for a simple decision.

An easy way to make the decision logic complete consists in specifying a default value for the decision. The default value applies if no other rule has made a decision, and is always the last rule to execute.

This default value is just a fall-back measure and should not prevent you from completing the decision logic in an informed and deliberate way.

You specify the default value as an action of the form:

```
set decision to <value>
```

If the node uses multiple values enabled by lists, use the form:

```
add <value1> to decision
add <value2> to decision
...
```

## Rule interactions

For each decision node, you select a rule interaction policy that applies to all the rules and decision tables in this decision node. Interaction policies define how rules interact with each other by governing their order of execution. It is up to you to decide which of these rule interaction policies should be applied.

Int era cti on pol icy	Description
Seq uen tial	The default policy, in which rules execute in the order in which they are listed in a decision node. A rule can modify or overwrite decisions that were previously made by other rules.
Firs t rule app lies	Rules execute in the order in which they are listed in the decision node, but as soon as a rule makes a decision, this decision is final.  Do not use this policy if the decision is a list that results from multiple instances of one or more rules.
Sm alle st and gre	The value of the decision is set to the minimum value available among the values that are obtained by the applicable rules. If you use the greatest value policy, it is set to the maximum value available. If no rule applies, the value of the decision is null. You can use these policies for decisions of type number and integer, and that use the set decision to construct.

ate st val ue	
Coll ect	The values of all applicable rules are collected and returned as a list. The execution order of the rules does not influence which values are collected, but it impacts the order in which the collected values appear in the list. If no rule applies, an empty list is returned. You can use the collect policy for decisions that are multi-valued, and that use the add . . . to decision construct.
Su m	The values of all applicable rules are summed up. If no rule applies, the value of the decision is 0. You can use the sum policy for decisions of type number and integer, and that use the add . . . to decision construct.

**Parent topic:** [Modeling decisions in the Business console](#)

## Deploying a decision model service

You deploy a decision model service to an execution environment using a deployment configuration, the same way as you deploy a regular decision service.

Deploying from Decision Center is described in [Deploying from the Business console](#). You can also automate rule deployment by using the [Decision Center REST API](#).

When you deploy a decision model service, note the following:

- For your convenience, Decision Center creates a deployment configuration when you create a decision model service.

This deployment configuration deploys to a non-production server, and does not give any groups the right to deploy. Users with release manager or permission manager rights can edit the deployment configuration, for example, to enable specific groups of users to deploy with the deployment configuration.

- Decision Center creates a **decision operation** and automatically refreshes its content each time you save. You cannot edit the decision operation or create another one, because a decision model service references all the rules of the decision model, and uses the input/output parameters of the diagram.

**Note:** You can review the decision operation details in the deployment configuration read-only view: select the **Operations** tab, and hover your mouse over the operation name.

**Parent topic:** [Modeling decisions in the Business console](#)



## Validating and testing decision model services

With a decision model service, you can validate your decision model during development. You can also create more complete usage scenarios that you use in test suites and simulations, just like regular decision services.

Each time that you save a new version of your decision model, Decision Center compiles the content of your decision model and displays any errors or warnings that you must fix.

Decision Center also provides a **Validate** feature for decision model services, to rapidly try out your decision model when you are defining it. At each step, you provide data to ensure that your model returns the results you expect. Decision Center runs your data on a local embedded engine. You provide the data in a friendly form or in its underlying JSON format.

### Test suites and simulations

Decision model services behave the same way as regular decision services for test suites and simulations.

- [Testing sets of rules in the Business console](#) describes how to setup test suites that run rules, on an execution environment, against usage scenarios. You setup test suites after your decision model is complete, for regression testing.
- [Simulating business application results](#) describes how to create simulations to see how changes to rules or data affect the results of your decision model service, based on key performance indicators.

There are some differences to consider, because the decision model is considered as one artifact:

- You run test suites or simulations on the entire decision model, because you cannot create a decision operation that defines a subset of rules.
- When you configure a test suite or generate a scenario file, the only execution detail that can be included in the report is the duration of execution.

**Parent topic:** [Modeling decisions in the Business console](#)

## Using queries

In the Business console, you use queries to search for elements of your projects. You can apply actions to the results of the query.

With queries, you can filter business rules or other project elements. You can perform actions to all the results of a query, or only to specific results. For example, if you have a modification to apply to a certain type of rule, you can create a query that searches for the rules corresponding to this type and modifies them. You can also create queries to find which rules have been affected by any modification you made.

You can also use queries to extract rules for generating a ruleset. To use queries as an extractor, you must define the query to use in a decision operation.

Queries are made up of two parts:

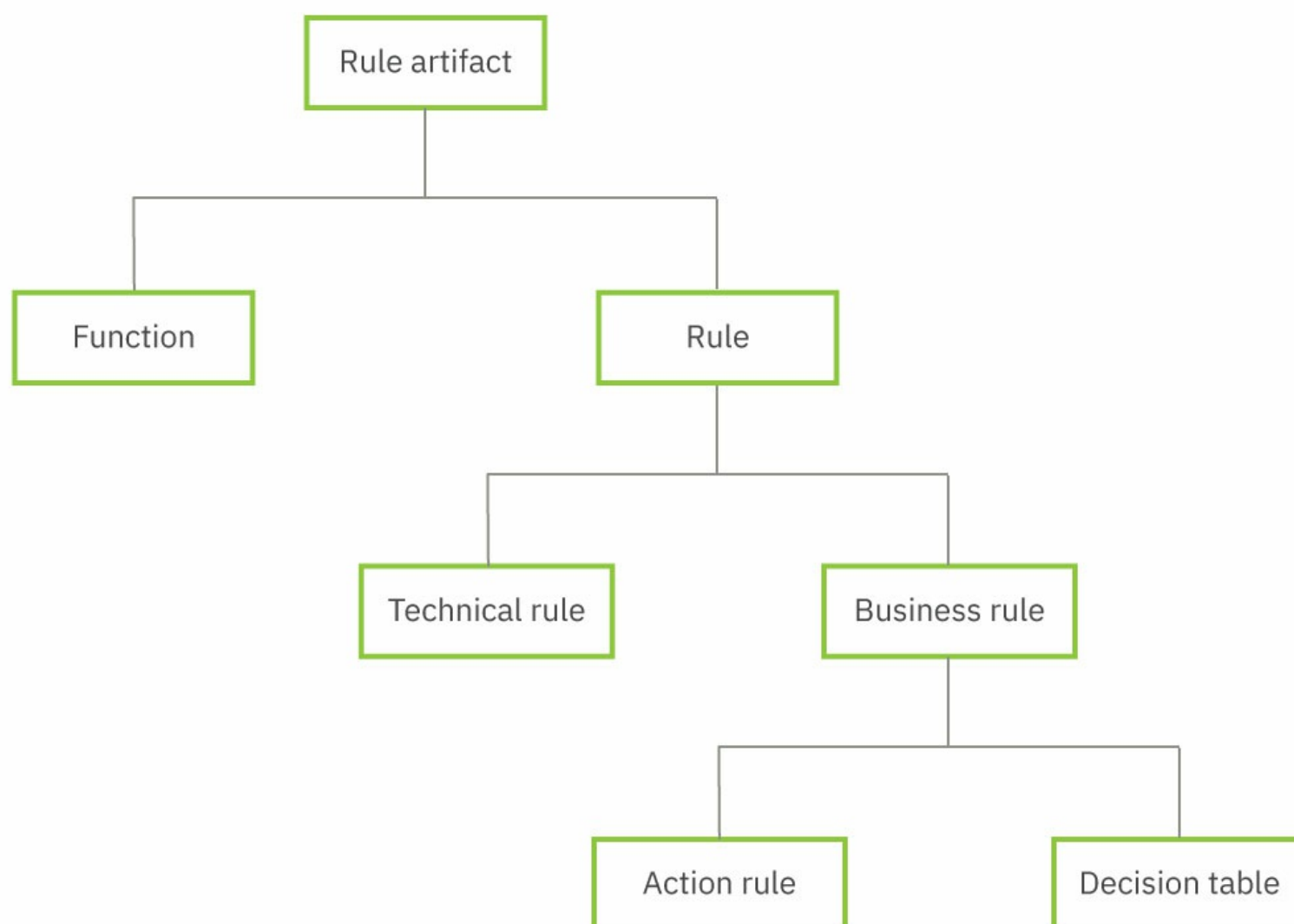
- The **query** part, for example, Find all business rules such that the creator of each business rule is me
- Optionally, an **action** part, for example Do set the status property to deployable

You construct queries in the same way that you construct rules, with a condition, and optionally an action. The default query action is to display the results, but you can also have the query carry out other actions such as moving the results of the query to another folder.

Queries are run against the current project, as well as any linked projects if you specify this option. Queries apply to the current branch, release, or change activity where you run the query.

To search for specific types of rule artifacts you must use the correct rule type in the query part, according to the following rule hierarchy:

Figure 1. Rule hierarchy



For example:

- Find all rules returns all action rules, decision tables, and technical rules.
- Find all business rules returns action rules and decision tables.

To run a query on functions, you must explicitly state it in the query part: Find all functions.

### Query conditions

Query conditions are the criteria that you define for the search.

### Query actions

The default action of a query is to display the query results, but you can add further actions by specifying

them under the **Do** part of the query.

#### **Creating and running a query**

Create and run a query on the **Queries** tab. You can also save your query, view its results, and apply actions to the query results.

**Parent topic:** [Working with the Business console](#)

**Related concepts:**  
[Decision operations](#)

## Query conditions

Query conditions are the criteria that you define for the search.

The conditions of your query appear under the Find part of the query.

By default, you run queries on the business rules but you can also query other types of project elements, such as folders (rule packages), ruleflows, variable sets, and so on.

You start by selecting the type of project element you want to query. You can then refine the query by adding a filter in the form of a `such that` statement, followed by condition statements.

### Example

```
Find all business rules
 such that the creator of each business rule is me
 and the creation date of each business rule is after 3/23/2016
```

You can refine queries using a `such that` statement to filter different project items:

- [Filter by properties](#)
- [Filter by business terms](#)
- [Filter by behavior of rules during execution](#)
- [Filter by impact of rules during execution](#)

#### Note:

You can identify which rules have changed in the current session (by querying on `after my login time`) or which rules you have changed (by querying on `creator is me`).

### Filter by properties

You can filter the query according to one of the properties of the project elements being queried (see [Rule properties](#)).

For example, the following query displays only business rules whose status is new:

```
Find all business rules
 such that the status of each business rule is new
```

### Filter by business terms

You can filter the query according to the business terms used in the conditions and actions of your rules. This type of filter is useful when trying to find rules affected by a policy change, for example:

```
Find all business rules
 such that each business rule modifies the value of the insurance rate
```

The following query constructs find rules that use or modify business terms:

#### uses the value of

Returns rules that use the values of certain business terms.

#### Example

The following query returns all business rules that use loan grade values:

```
Find all business rules
 such that each business rule uses the value of the loan grade in 'a
report'
```

#### uses the phrase

Returns rules that call a given business term.

#### Example

The following query returns all business rules that use 'add to the corporate score in the loan report':

```
Find all business rules
such that each business rule uses the phrase [add 'a number' to the
corporate
score in 'a report']
```

### uses the phrase ... where

Returns rules that call a given business term to which a constraint is applied

#### Example 1

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can equal 100:

```
Find all business rules
such that each business rule uses the phrase [set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

#### Example 2

The following query returns all business rules that call the term that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a
number' to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When the elements mentioned in the constraint do not appear in a rule, the rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query would return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10
to 'the borrower';
```

### modifies the value of

Returns rules that modify certain business terms.

#### Example

The following query returns all decision tables that modify the value of the insurance rate:

```
Find all decision tables
such that each decision table modifies the value of 'the insurance rate'
```

### Filter by behavior of rules during execution

You can filter the query according to the semantics of the rules, that is, their behavior during execution. This type of filter finds rules that could be applicable when certain conditions are true or become true. This filter is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can use the following semantic query constructs:

#### may apply when

Returns all rules whose condition part could meet the query condition, or where there is nothing in the rule condition that would contradict the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

**Example**

Table 1. List of example rules

Rule	Rule content
Rule 1	If the score of the borrower is at least 10 then...
Rule 2	If the age of the borrower is at least 21 then...
Rule 3	If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

**Query 1:**

*Find all business rules  
such that each business rule may apply when the score of the borrower is 20*

This query returns Rule 1 and Rule 2. It returns Rule 1 because if the score of the borrower is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the score of the borrower could be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

**Query 2:**

*Find all business rules  
such that each business rule may apply when the score of the borrower is 5*

This query returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to less than 10. In Rule 2, nothing specifically stops the rule from being applicable when the score is 5. In Rule 3 at least one of the conditions could apply, and there is nothing in the other condition that negates the fact that the score could be 5.

**may become applicable when**

A more specific query that returns only those rules whose condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition.

**Example**

Table 2. List of example rules

Rule	Rule content
Rule 1	If the category of the customer is Platinum then...
Rule 2	If the category of the customer is not Platinum then...
Rule 3	If the age of the customer is at most 65 then...
Rule 4	If the age of the customer is at most 65 and the category of the customer is not Platinum...

**Query 1:**

*Find all business rules  
such that each business rule may become applicable when the category of 'a customer' is Gold*

This query returns Rule 2 and Rule 4. It returns Rule 2 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 4 for the same reason. The additional condition relating to the age of the customer does not contradict the condition in which the category may be Gold.

The query does not return Rule 1 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 3 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer category is Gold.

**Query 2:**

*Find all business rules  
such that each business rule may become applicable when [the age of*

'a customer' is at least 21]

This query does not return Rule 3 because it searches for rules that could “become” applicable when the customer age is over 21. Rule 3 is applicable even if the customer age is under 21. Therefore Rule 3 does not “become” applicable, but it “remains” applicable even if the customer age changes from under 21 to over 21.

### may lead to a state where

Returns rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of a rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

#### Example

Table 3. List of example rules

Rule	Rule content
Rule 1	If the age of the borrower is at least 25 then set the credit score of the borrower to 60
Rule 2	If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

#### Query:

Find all business rules  
such that each business rule may lead to a state where the credit score  
of  
the borrower is more than 50

This query returns Rule 1 but not Rule 2 because, when the age of the borrower has been checked and the credit score set, only Rule 1 shows a result of over 50.

### Filter by impact of rules during execution

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule impacts the applicability of other rules, and how a rule is impacted by the execution of other rules.

### may select

Returns the ruleflows and rule tasks that may select a given rule.

#### Example

The following query returns the ruleflows and rule tasks that may select "Rule 1".

Find all ruleflows  
such that each ruleflow may select "Rule 1"

### may enable

Returns rules that make a given rule applicable.

#### Example

Table 4. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 25 then set the category of the customer to Bronze ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;

#### Query:

Find all business rules  
such that each business rule may enable "Rule 2"

This query returns Rule 1 because it sets the category of the customer to Bronze, and thus makes the condition of Rule 2 valid.

### may disable

Returns rules that make a given rule inapplicable.



**Example**

Table 5. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 18 then set the category of the customer to Copper ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;
Rule 3	if the age of the customer equals 25 and the category of the customer is Bronze then set the category of the customer to Diamond ;

**Query:**

*Find all business rules  
such that each business rule may disable "Rule 2"*

This query returns Rule 1 and Rule 3. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It returns Rule 3 because although one of the conditions is that the category of the customer is Bronze, the action is to set the category of the customer to Diamond. Therefore, if the category of the customer is Diamond, Rule 2 is not applicable.

**may be enabled by**

Returns rules that are made applicable by a given rule.

**Example**

Table 6. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 18 then set the category of the customer to Copper ;
Rule 2	if the category of the customer is Bronze then give Champagne to the shopping cart of the customer with message: "Congratulations" ;
Rule 3	if the age of the customer equals 25 and the category of the customer is Gold then set the category of the customer to Diamond ;

**Query:**

*Find all business rules  
such that each business rule may disable "Rule 2"*

This query returns Rule 1. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It does not return Rule 3 because the action is to set the category of the customer to Diamond, and it can only be executed from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

**may be disabled by**

Returns rules that are made inapplicable by a given rule.

**Example**

Table 7. List of example rules

Rule	Rule content
Rule 1	if the age of the customer is 25 then set the category of the customer to Bronze ;
Rule 2	if the age of the customer equals 25 and the category of the customer is Copper then set the category of the customer to Gold

**Query:**

*Find all business rules  
such that each business rule may be disabled by "Rule 1"*

This query returns Rule 2 because it is made inapplicable by Rule 1. In Rule 1 the category of a customer whose age is 25, is set to Bronze, therefore it cannot be Copper.

**Parent topic:** [Using queries](#)



## Query actions

The default action of a query is to display the query results, but you can add further actions by specifying them under the **Do** part of the query.

Query actions can make the following changes:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add or remove a category to the displayed elements
- Set any of the properties of the displayed elements to a value that you specify

### Example

```
Find all business rules
such that the status of each business rule is new
Do set the status of each business rule to validated
```

By default, the actions are carried out on all the displayed elements when you click **Apply Actions** in the **Query results** tab. You can select specific elements among the results, and apply actions only to these elements.

**Parent topic:** [Using queries](#)

## Creating and running a query

Create and run a query on the **Queries** tab. You can also save your query, view its results, and apply actions to the query results.

### Procedure

1. Open your decision service, and click the **Queries** tab.
2. In the **Queries list** subtab, select the project in which you want to run your query.
3. In the field **Query expression**, type your query. You can add information in the **Description** field.

If you want to save this query, click **Save**. The query is added to the list of queries under the project. You can run a query without saving it first. If you click another project or query in the queries list, you must confirm that you want to discard your current query.

4. Click **Run**. The query results display in a subtab **Query results**.

When your query completes, you can apply actions if the query has an action part. You can review the results before you decide to apply the actions.

5. In the toolbar above your query results, click the icon **Apply the query actions to the selected elements** to perform the actions from the query. By default, all the elements retrieved by the query are selected, so the action is performed on all of them. If you select certain project elements among the results, the action is performed only on the selected project elements.

**Parent topic:** [Using queries](#)

## Testing sets of rules in the Business console

Test the rules that you create or edit to achieve the results that you expect.

### Testing in the Business console

The Business console provides capabilities to test sets of rules against usage scenarios.

### Understanding the reports

When you run a test suite, it produces a large amount of information and places the relevant information in a report.

### Excel scenario files

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

**Parent topic:** [Working with the Business console](#)

# Testing in the Business console

The Business console provides capabilities to test sets of rules against usage scenarios.

During rule development, you can create and run test suites in change activities or ungoverned branches in a decision service. You can also run test suites in validation activities to test the content of a release before deployment.

To understand how testing works, consider what you can change in a decision service:

- The **set of rules** that you are testing. The set is identified by the **decision operation**, which defines a subset of rules within the release or change activity on which the testing is being done (see [Identifying a set of rules](#)).
- The **scenarios** that you are submitting. Each scenario contains all the information that is required to process a transaction. This information can be real or fictitious data. You store this scenario data in a scenario file, which you generate, complete with data, and then upload to Decision Center. You also use this scenario file to store the **expected results** for each scenario.
- The **test suite** that you create and run to provide feedback on the performance of a set of rules without changing the rules or the applications on which they act.

Running a test suite compares the results that you expect to the actual results that are obtained from applying rules to your scenarios. The test generates a report that shows the results for each scenario, and the success rate as the percentage of scenarios that generated their expected results.

**Note:** If you do not see the option to select an operation when you create a test suite, you can go back to the **Test** tab and select the operation from the drop-down list in the upper-right corner.

## Scenarios

Scenarios represent real or fictitious use cases that you use to validate the behavior of your rules. Each scenario contains all the information that is required for your rules to run properly.

For example, a loan application might require the following information, which must be completed for each scenario:

- Borrower: Sam Adams
- Credit Score: 600
- Yearly Income: 80000
- Duration: 24 (months)
- Amount: 100000
- Yearly Interest Rate: 5 (%)

You can generate scenarios in an Excel format. Each row represents a scenario, and the columns indicate where to put the data for each scenario. For example, the following Scenarios sheet contains four scenarios that validate loan risk from a very low risk loan to a loan where the amount is too high:

		the borrower					the loan		
Scenario ID	description	first name	last name	b	S	c	y	start date	nu amount
Very low risk	Very low risk loan accepted	Sam	Adams	4	95			8/1/2009	# 100000
Low risk	Low risk loan accepted	Bob	Schwartz	4	94			8/7/2010	# 500000
Average risk	Average risk loan accepted	John	Johnson	4	85			9/1/2010	# 900000
Amount too high	Loan rejected when amount too high	Mary	Doe	4	32			8/15/2010	# 1100000

Scenarios Expected Results HELP

## Test suites

You run test suites to verify that your rules are correctly designed and written. The test suites compare your expected results with the actual results that come from applying your rules against the scenarios that you defined.

You set up the expected results in a separate sheet alongside your Scenarios sheet.

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Scenarios Expected Results HELP

The example tests two different aspects of the loan request:

- Whether the loan is approved.
- Which message the loan request application generates.

The Expected Results sheet represents these two tests as columns. You specify the expected results for all the scenarios necessary to cover the validation of your rules.

The test suite returns a report that compares your expected results with the actual results of the execution. Each test in a test suite is successful if all the expected results match the actual results.

You can include an Expected Execution Details sheet for more technical tests, for example, to list the rules that are run or the duration of a run.

**Note:** When you import a test suite from the release in a validation activity, this test suite is considered as a reference, and is locked in all change activities and validation activities of this release.

**Parent topic:** [Testing sets of rules in the Business console](#)

## Understanding the reports

When you run a test suite, it produces a large amount of information and places the relevant information in a report.

All test reports contain the following information:

- **Summary:** Shows the scenario success rate as a percentage of scenarios that ran successfully. It also shows information about the test, including the scenario file, decision operation, and rule source. Links are provided to rule and data sources.
- **Results:** Shows the results from running the test. It lists the scenarios in the scenario file, and the status and result of each scenario. If you ran your test on collections, you can click **Details** to obtain more information on the observed and expected results.

The results for the scenarios use the following statuses:

- **Successful:** A test is successful when the expected results match the actual results.
- **Unsuccessful:** A test is unsuccessful when the expected results are different from the actual results.
- **Error:** An error is reported when the test cannot run the scenarios, for example, when an entry in the scenario file is not correctly formatted.

**Parent topic:** [Testing sets of rules in the Business console](#)



# Excel scenario files

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

An Excel scenario file can contain the following sheets:

- Scenarios: Contains the input data for scenarios. Both testing and simulation scenario files have this sheet.
- Data entry: Regroups information that is used in other sheets.
- Expected Results: Holds the results that you expect from tests.
- Expected Execution Details: Holds the execution details that you expect from tests.

**Important:**

Never modify the column structure of the sheets.

## Scenarios sheet

Each row in the Scenarios sheet represents one scenario. Each scenario must contain all the information needed to process a transaction.

		the borrower					the loan	
Scenario ID	description	first name	last name	b	S	c	y	z
Very low risk	Very low risk loan accepted	Sam	Adams	4	1	9	5	
Low risk	Low risk loan accepted	Bob	Schwartz	4	1	9	4	
Average risk	Average risk loan accepted	John	Johnson	4	1	8	5	
Amount too high	Loan rejected when amount too high	Mary	Doe	4	1	3	2	

The columns indicate the information that you must provide for each scenario:

- **Scenario ID:** The name identifies the scenario and must be unique.
- **description:** A free text cell to describe the scenario.
- **business terms:** Values for the scenarios. Bold columns or subcolumns are mandatory.

**Note:**

The reduced rows between the column headers and the first scenario contain information that might be useful to developers. Never edit these rows.

The following visual aids help you complete your scenarios:

- Red triangle: When shown in a column header, it indicates the presence of a comment that explains the type of values to be entered, such as text, numbers, dates, or true or false. Hover over a triangle to display a comment as follows:

the borrower				
first name	last name	credit score	yearly income	
John	Smith	600	80000	
John	Smith	600	80000	

- Dark green triangle: When shown in a cell, it suggests a better format for the cell. For example, if you enter a numerical value in a text cell, Excel suggests that you format the cell for numerical values. To make sure that the scenario runs correctly, ignore this suggestion.
- Arrow: When shown before a column name, it indicates that the values must correspond to entries that are specified in a separate **data entry** sheet.

## Data entry sheets

You create data entry sheets when you generate a scenario file. Data entry sheets regroup data to be used in other sheets.

The name of each data sheet serves as the type of data that is expected in the column of the other sheet that uses the data. For example, in the following figure, the address sheet **1** contains different addresses that are used in the addresses column **2** of the Scenarios sheet.

Entry ID	Street	City	State	Zip Code
billing address	2207 7th avenue	New York	NY	10027
shipping address	25 Elm Street	Dallas	TX	75201
personal address	110 Cactus Street	Phoenix	AZ	85003

1

	the borrower						
description	first name	last name	credit score	yearly income	amount	→ addresses	st
Loan rejecte	John	Smith	600	80000	500000	personal address	
Loan approve	John	Smith	600	80000	25000	billing address	

Enter the name of an object defined in the sheet address

### Note:

A column that uses values from a data entry sheet has an arrow in its column header.

You create an entry in a data entry sheet by completing a row that includes a unique name to identify the entry in the other sheets.

## Expected Results sheet

The Expected Results sheet contains the results that you expect to obtain when you run tests that use the scenarios.

You can test many aspects of a set of rules, but the Expected Results sheet contains only the tests that you specified when you generated the scenario file. For example:

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Each blue column in the Expected Results sheet corresponds to a test. If you do not enter a value in a column, the corresponding test is skipped.

### Tip:

You cannot create a new column in Excel. However, you can generate another empty scenario file template, and then copy and paste a column from it.

You link the sheets that contain the scenarios and their expected results by entering names in their respective Scenario ID columns. In addition to making sure that the names match between the two sheets, it is good practice to keep the scenarios and their expected results in the same order.

When you test for a list of values, you enter each item in the corresponding cell on a separate row. You do not have to duplicate the Scenario ID for each row.

## Expected Execution Details sheet

The Expected Execution Details sheet contains the execution details that you expect to obtain when you run tests that use the scenarios.

The following are examples of execution details:

- List of rules fired
- List of executed ruleflow tasks
- Duration of execution

You can test many aspects of a run, but the Expected Execution Details sheet contains only the tests that you specified when you generated the scenario file. Each green column in the Expected Execution Details sheet



corresponds to a test.

You link the sheets that contain the scenarios and expected execution details by entering names in their respective Scenario ID columns.

When you test for a list of values, you enter each item in the corresponding cell on a separate row.

4			
5		Scenario ID	the list of fired rules contains
9		Big Loan	eligibility.checkIncome
10			eligibility.approval
11			eligibility.checkCreditScore
12		Small Loan	
13			
▶▶ Scenarios Expected Execution Details HELP			

You do not have to duplicate the Scenario ID for each row because the last ID entered is used.

**Parent topic:** [Testing sets of rules in the Business console](#)

## Simulating business application results

Run rules on representative data to generate results that you can use to improve the application.

### **Overview: Simulations in the Business console**

You can determine how changes to rules or data affect the results of your business rule application before deployment.

### **Workflow to create and run simulations**

The Business console takes you through the process for assembling and running a simulation.

**Parent topic:** [Working with the Business console](#)

## Overview: Simulations in the Business console

You can determine how changes to rules or data affect the results of your business rule application before deployment.

You run a simulation on a decision service in a change or validation activity in a release, or an ungoverned branch. You can use fictitious data or real data from your organization, and display the results in a report defined by you. You can then use the results of the simulation to refine the behavior of your business rules.

The simulation brings together data, rules, and key performance indicators (KPIs). You select the data and rules for the simulation, and you define the KPIs by using metrics that take values from the rules.

The KPIs represent the results of the simulation. You define a simulation report by adding KPIs in a report template, and then defining their appearance. You can display KPIs as text, or in graphs to make their results easier to understand.

Versions of the simulation artifacts are saved, and the simulation runs the most recent versions of its artifacts. The Business console maintains a history of simulations, and you can rerun an old version of a simulation by restoring it.

You can change the elements in a simulation, and rerun the simulation to assess the results of the changes. You can also compare reports from different runs side by side in the console. Through this iterative approach, you can refine your rules to get the results that you want from your business rule application.

### Change and validation activities

You can run simulations in the change and validation activities of a release in a decision service. In a change activity, you can use simulations to iteratively refine rules, and in a validation activity, you can use simulations to check the rules that were approved for the release.

Both types of activities use the same artifacts, that is, metrics, KPIs, scenarios, report templates, and simulation configurations. The activities within a release also share resources such as servers and decision services.

You can define artifacts in both types of activities. You can also import a simulation from a completed change activity into the validation activity of the same release. When you do so, all the artifacts in the simulation are imported, giving you a ready-to-run simulation in the validation activity. Importing a simulation saves you time in setting up a simulation in the validation activity, and you can apply a simulation that was used in developing the rules. To use a simulation from a change activity, the changes that were made in the activity must be merged into the release before the validation activity can import the simulation.

The two types of activities handle reports differently. In a change activity, reports are listed in the **Reports** tab, and a new entry is added each time that you run a simulation. The validation activity does not have a **Reports** tab. When you run a simulation in the validation activity, the report is listed with the simulation in the **Simulations** tab, and if you run the simulation again, the last report is replaced with a new report.

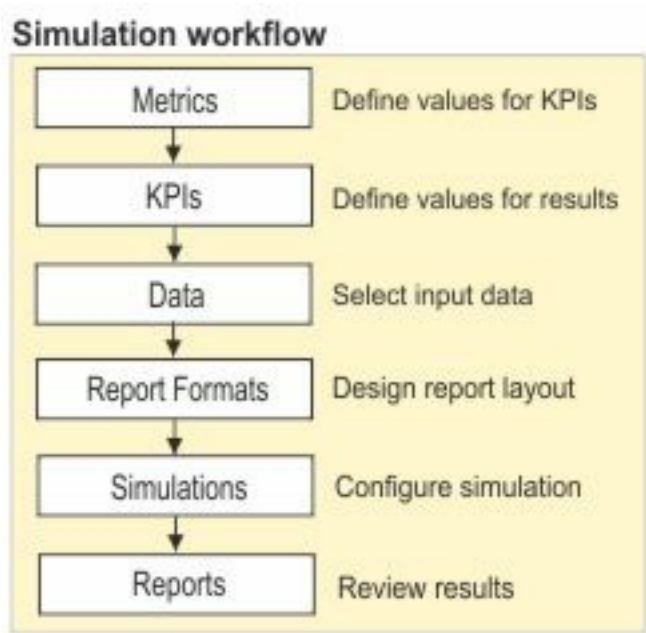
The validation activity lists the five most recent reports from different runs of a simulation in **Recent reports for this version**. You can compare simulation reports in a change activity or a validation activity, or you can compare a simulation report in a validation activity with a simulation report from a change activity.

**Parent topic:** [Simulating business application results](#)

# Workflow to create and run simulations

The Business console takes you through the process for assembling and running a simulation.

To run a simulation, you must assemble elements that include metrics, key performance indicators (KPIs), and a scenario file. The simulation interface is organized to take you through a workflow:



## Metrics

Metrics define the values that are used in the KPIs. You must create metrics before you can make KPIs.

You define a metric by using a business model language. For example, you might define a metric that is named `Loan amount` by making the metric expression the `amount` of `'the loan'`.

The simulator provides the following types of metrics:

- **Numeric:** Uses number values, for example, the `age` of `'the borrower'` or the `amount` of `'the loan'`.
- **DateTime:** Uses date and time values, for example, the `birth date` of `'the borrower'` or the `start date` of `'the loan'`.
- **Boolean:** Checks whether a value is true or false, or matches a specific group, for example, the `corporate score` of `'the loan'` is at least 10.
- **String:** Checks for text values, for example, the `zip code` of the address of `'the borrower'`.
- **Domain:** Uses predefined values in metric expressions, for example, for the expression the `category` of the customer, a domain metric might limit the category values to `Platinum`, `Gold`, and `Silver`. In this case, a simulation uses only the input data that matches the predefined values.

You must define conditions in metric expressions. You cannot create conditional KPIs. To add a condition, use `when` in your metric expression, for example, the `amount` of `'the loan'` when `'the loan'` is approved.

You can provide a default value for when a condition is not met. For example, you can set the default value of the metric `Age of Borrower` to 0. If a set of data does not meet the condition statement of the metric, a KPI that uses the metric can display 0 as its result in a simulation report.

If you do not specify a default value or select **Undefined** for Boolean types, the metric condition is not defined. Do not specify a default value when you want the KPI to determine the value. However, leaving the default value undefined is not equivalent to 0.

When you create an element for a simulation, you must select a decision operation. The **Select an operation** window displays a list of the available operations in the decision service. Each entry includes a description of the operation. The operations contain the rules that you can use in the simulation. You can select only one operation for a simulation.

**Important:** When you create a metric or a KPI, you cannot use a one-word verbalization such as `age`. You must use a phrase such as `age of the borrower`. A short, implicit verbalization generates an error.

## KPIs

KPIs show the results from running a simulation. To define a KPI, you pair a metric with a KPI value. For example, to create a KPI that shows the total amount of the loans that are handled by a loan application, you might use the KPI expression `sum of 'Loan amount'`. When the simulation runs, it adds up the amounts of the loans and displays the total in the simulation report.

You can create two types of KPIs:

- **Scalar:** Shows a single value that was taken from a group of values. For example, in a loan validation application, you might have a KPI value for the sum of all the loan amounts.
- **Grouped:** Shows a set of values. For example, a simulation might group the average loan amounts for a set of customer categories. In this case, the report can display the set of values in a graph such as a pie chart.

A KPI can use either one or two metrics. When you use two metrics, the second metric defines a distribution of

values.

The provided KPIs cover common values. Contact your system administrator to add more KPI values.

Data

Simulations use business information that is provided as input data. To run a simulation, you must use an Excel scenario file.

You can use either real or fictitious data. The Excel format displays data clearly, and can be changed easily.

You can generate and download an Excel data file from the Business console, add data to it locally, and then upload it back to the console.

Scenarios

Scenarios represent business use cases. They can contain real or fictitious data. You use scenarios to validate the behavior of your rules. When you use a group of scenarios in a simulation, each scenario must contain all the data that is needed to run the rules.

In an Excel file, you import the scenarios in a table:

	the loan						the borrower						
description	start date	numb	amount	Loan	mon	yearly	first na	last na	birth date	SSN	credit	yearly in	zip code
Loan approved	1/1/2014	240	100000	0.2		0.04	John	Doe	1/1/1980	1234 600	70000	75001	
Loan rejected	1/2/2014	120	200000	0.2		0.04	John	Doe	1/2/1980	1234 600	70000	75001	
Loan rejected	1/3/2014	240	100000	0.2		0.04	John	Doe	1/3/1980	1234 600	70000	75001	

Scenarios

HELP

+

Each row in the table represents a complete scenario. Typically, the more scenarios, the more accurate your results.

To use the scenario data file, you must import it into the Business console.

Important:

You can use scenario data files that are made to test applications (see [Testing sets of rules in the Business console](#)). However, you cannot test business applications with data files that are made specifically for simulations. The data files for testing contain expected results. Simulations do not use expected results and ignore them in data files for testing.

Report formats

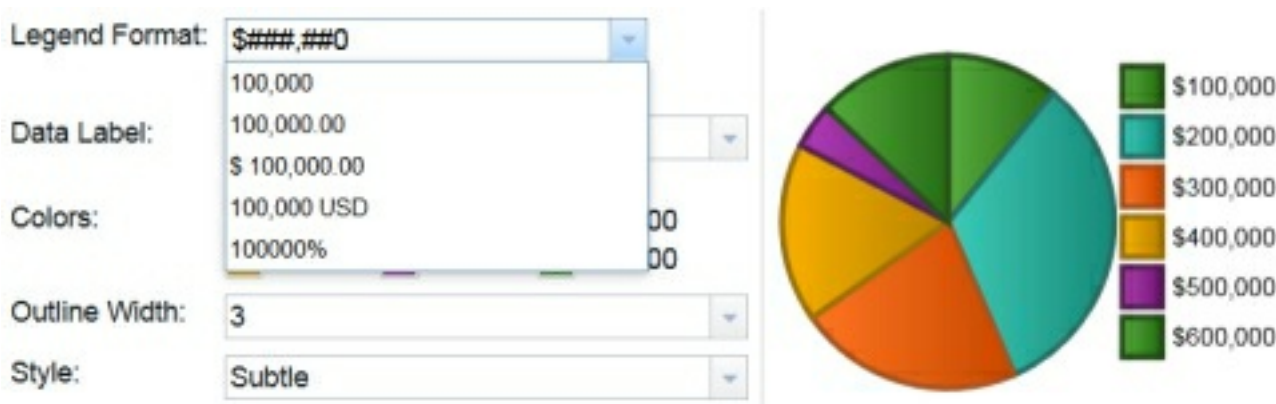
When you run a simulation, it displays its results in a report. You can define various reports for a simulation to generate different reports.

You lay out a report in a template by adding sections and KPIs. You can add as many elements as necessary.

A report can format KPIs as follows:

- Text: When a KPI produces a single value from a set of data, you display the value as text. You can configure the appearance of the text in the simulation report. The formatting options include fonts, font sizes, and colors.
- Graph: When a KPI produces a set of comparable values, you can display the values in a graph to highlight their differences. The simulation feature includes four types of graphs: bar, line, area, and pie chart. You can format a graph, including the title, colors, and axis labels.

When you format a report, you can select descriptors that match the values of the KPIs, for example:



Simulations

You configure a simulation to bring together input data, an application server, and a report format. When you run the simulation, it uses rules from the associated decision service and places the results in a report.

You can interrupt the generation of a report during a simulation. If you do, the report is still created, but it contains only partial results.

Note:

The simulations run on application servers that are defined by system administrators in the Decision Center Enterprise console.

## Reports

When you run a simulation, it generates a report that you use to evaluate the results.

You can compare two reports side by side in the Business console. You can also access different parts of a simulation from a report. For example, you can go directly to the rules, change data sources, and modify the report format. When you change aspects of a simulation from a report and run the simulation again from the **Simulations** tab, a new report is generated.

**Note:** In validation activities, the simulation reports are listed in the **Simulations** tab.

**Parent topic:** [Simulating business application results](#)



## Deploying from the Business console

In a decision service, you can deploy a set of rules to a production environment or to a non-production environment for testing or quality assessment.

You deploy from a decision service by using a deployment configuration. Any user who can access a decision service in the Business console can deploy its branches (release, change activity, regular branch) from any deployment configuration available to that user in that branch. However, the following conditions apply:

- In a change activity, you can deploy only with a deployment configuration whose type is non-production.
- In a release, you can deploy with a deployment configuration whose type is production only if the release is completed.

<b>Note:</b> For these two cases you can deploy to a RuleApp archive.
-----------------------------------------------------------------------

Only a user with configuration manager or administrator rights can create, edit, or delete deployment configurations.

Only a user with Permission Manager rights can create, edit, or delete deployment configurations.

This ensures a basic level of security on deployment. These users define deployment configurations with information that includes the target servers, the decision operations that define the business rules, and settings for versioning of rulesets and RuleApps. The following aspects of a deployment configuration relate to making sure that deployment is secure:

- The configuration type as production or non-production. This has the effects noted above.
- Which of the existing target servers this deployment configuration can deploy to. When deploying, the list of possible servers is then presented as a selectable option.
- Users belonging to which groups can deploy with the configuration.

When working within the governance framework, a deployment configuration can only be created or edited within a change activity.

The deployment configuration contains a setting to establish if a deployment snapshot is required when deploying. Deployment snapshots can be found in the list of snapshots. You can then redeploy from the deployment snapshot or from the report of the deployment.

### [Deployment configurations](#)

You use deployment configuration in a decision service branch (release, change activity, or regular branch) to deploy sets of rules.

### [Redeploying](#)

You can redeploy a decision service branch from a deployment snapshot or from the original report of the deployment.

### [Version policies](#)

Version policies determine how client applications identify deployed rulesets.

**Parent topic:** [Working with the Business console](#)

**Related concepts:**

[Decision services](#)

[Governance principles](#)

# Deployment configurations

You use deployment configuration in a decision service branch (release, change activity, or regular branch) to deploy sets of rules.

Each deployment configuration contains all the information required to deploy rulesets, including the target server. When working within the decision governance framework, creating and editing a deployment configuration must be done within an open change activity.

**Note:** You need Permission Manager rights to create and edit deployment configurations.

A deployment configuration is organized in five sections:

## General

Contains the name and description of the deployment configuration, and indicates the project in which it is stored. Also contains the name and the base version number of the deployed RuleApp. Also contains properties for the RuleApp.

You set the configuration type to *Nonproduction* or *Production*. Typically, a nonproduction configuration is used in developing and testing a decision service. A production configuration is used to deploy a finished release to a production environment.

## Operations

Lists the decision operations that are available in the decision service, and those selected for deployment.

With an initial deployment, you typically deploy the available decision operations. In updates, you might limit deployment to specific decision operations.

### Note:

When selecting decision operations, be careful to avoid conflicts during deployment. Make sure that every ruleset name in the deployment configuration is unique.

## Targets

Lists the Rule Execution Server instances that are available in Decision Center, and those selectable when deploying. The deployment process packages the rulesets in the decision service into a RuleApp archive that is sent to each selected server.

If no servers are available, you can still use the deployment configuration to create a RuleApp archive that can be saved locally.

## Ruleset Properties

Contains the base version number to be used in the ruleset path. You also set the version policy for calculating the version numbers that are used on successive deployments.

You use the Advanced properties section to configure individual rulesets. You select a ruleset and choose options for enabling, debugging, and tracing the ruleset. You can also select and define properties that control in detail how the ruleset executes in Rule Execution Server.

## Groups

You select which user groups can see this deployment configuration.

**Parent topic:** [Deploying from the Business console](#)

### Related concepts:

[Administering projects from the Business console](#)



# Redeploying

You can redeploy a decision service branch from a deployment snapshot or from the original report of the deployment.

## About this task

When you deploy a decision service, settings are available in the deployment configuration to create a deployment snapshot. Deployment snapshots are visible in the **Snapshots** tab. You can use the deployment snapshot to redeploy at a later time.

You can also redeploy from the original report of the deployment, available in the **Reports** section of the **Deployment** tab.

To repeat a decision service deployment, you use its deployment snapshot.

## Procedure

1. In the **Snapshots** tab of your release, change activity, or branch, locate the deployment snapshot.
2. In the drop-down list next to the name of the deployment snapshot, click the **Redeploy** icon. The deployment dialog opens.
3. Click **Deploy** to deploy the deployment snapshot.

**Note:** If you originally used the deployment to generate an archive file, you can use the deployment snapshot to generate only another archive file.

**Parent topic:** [Deploying from the Business console](#)

**Related concepts:**  
[Snapshots](#)

**Related tasks:**  
[Creating a snapshot](#)  
[Viewing the timeline of a release or activity](#)

## Version policies

Version policies determine how client applications identify deployed rulesets.

In a deployment configuration for a decision service, you establish the version policies to define the paths of the rulesets in the RuleApp deployment. The ruleset path determines how an application calls a ruleset through Rule Execution Server.

The ruleset path can have the following structure:

DeploymentConfiguration/RAVmajor.RAVminor/Operation/RSVmajor.RSVminor

DeploymentConfiguration identifies the decision service. RAVmajor.RAVminor identifies the version of the decision service interface that is being used. Client applications use the version number to ensure that they are compatible with the decision service interface. Operation is the name of the ruleset. The major version number of a ruleset (Vmajor) often represents a specific release of the ruleset, and the minor version number (Vminor) represents a deployment version of the ruleset. Typically, client applications call the latest enabled ruleset, but what they actually call depends on the ruleset path that is used in the client application.

The version numbers use base numbers that you define in the deployment configuration. You manually define a base number for the RuleApp and each ruleset. Deployment increments the minor ruleset version number, or replaces or creates a ruleset version number (Vmajor.Vminor).

**Note:** The version policies that are set in a Rule Designer deployment configuration are synchronized with Decision Center.

### Select a policy by the type of deployment

The version policy that you use depends on the type of deployment:

Type of deployment	Version policy
Successive deployments of a release to a specific server	Increment minor version numbers
Development of a decision service	Use the base version numbers
Test fix to correct a deployed solution	Use the base version numbers
Test fixes or updates to an earlier release	Define the version numbers

#### Increment minor version numbers

This policy serves as the default setting. It deploys the decision operations of the release, changing the minor version number of each ruleset. It looks for the latest version number to increment on the server to which it has access, and the same deployed version number is used.

A client application uses a ruleset based on the ruleset path that is defined in the client application., but previous deployments remain available on Rule Execution Server. You can still use a previous deployment by specifying its full ruleset path. You use this policy with successive deployments of a release to a specific server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/2.1</div>

#### Use the base version numbers

This policy deploys the decision operations and replaces the base version on Rule Execution Server. The replacement ensures that the deployment configuration path corresponds exactly to what is deployed on the server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0  /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0  /ruleset/2.0 (replaced)</div>

**Define the version numbers**

With this policy, you enter a specific ruleset version number at deployment time. You use this policy with test fixes or updates to an earlier release. Upon deployment, Rule Execution Server uses the specified ruleset version, and replaces the last version of the ruleset.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 (replaced) /ruleset/2.0</div>

**Parent topic:** [Deploying from the Business console](#)

# Administering projects from the Business console

A user who has the permission manager role can access the **Administration** tab in the Business console to administer security, manage servers, or run diagnostics.

## [Administering security](#)

You can manage groups of users, and set project security to control access to decision service branches.

## [Managing servers](#)

As a permission manager, you use the Business console to manage the list of servers available to Decision Center for deployment, testing, and simulations. You can also restrict access to specific groups.

## [Diagnostics](#)

To get information about your system, you can run diagnostics in the **Diagnostics** subtab of the **Administration** tab.

## [Configuration settings](#)

You can set some configuration options to customize Decision Center behavior, or display in the Business console. You can also set custom parameters.

## [Exporting or importing the current project](#)

As a permission manager, you can export the working branch of the current project as a .zip file, and import the project previously exported back into Decision Center.

## [Setting up project dependencies](#)

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in different decision services or branches.

**Parent topic:** [Working with the Business console](#)

## Administering security

You can manage groups of users, and set project security to control access to decision service branches.

### [Security](#)

Decision Center provides a security feature to control access to branches of decision services, and set permission profiles.

### [Managing groups of users](#)

From the **Administration** tab of the Business console, you can create, delete, and edit groups, and assign users to these groups.

### [Setting project security](#)

You can control access to decision service branches by enforcing security for projects in releases, activities, or ungoverned branches. You can also control access to these projects for specific groups of users.

**Parent topic:** [Administering projects from the Business console](#)

# Security

Decision Center provides a security feature to control access to branches of decision services, and set permission profiles.

You can enable security so that branches of decision services are only visible to certain groups of users. You can enable security in the Business console (see [Setting project security](#)), but security applies to both consoles.

When you have enabled security, you must specify, for each group that can access the branch, what permissions they have on artifacts: read only, full authoring, or none. You can set permission profiles from the Business console **Administration** tab, when you edit a group in the **Groups** subtab.

Groups are managed by a permission manager in Decision Center, and users are managed by the administrator in the Operational Decision Manager on Cloud portal. For more information, see [User roles](#).

## Branch security and decision governance

To use the decision governance framework (see [Governance principles](#)) with the Decision Center security feature, you must understand how both work and how they interact.

Releases and activities are types of branches, and as such are subject to the Decision Center security and permissions feature. You can enable security on individual branches to define which groups of users have access to them. When security is enabled on a branch, you specify the permissions of groups of users to read only, have full authoring on the artifacts that are contained in the branch, or have no access to these artifacts.

Branches in the decision governance framework inherit the security state of their parent branches. If you enable security on the initial release of a decision service, all the releases and activities that stem from the initial release have security that is enabled by default. Similarly, you can enforce security on a validation activity by setting security on its parent release.

Releases, change activities, and validation activities inherit artifacts from their parent branch, and are subject to permissions. You can, for example, give a group of users security access to a release but restrict their permission to view the change activities of the release.

You can also give or restrict the permissions of a group of users to update the following properties of releases and activities:

Property	Used to give or restrict permission to...
Owner	Change the owner
Due date	Change the due date
Goals	Change the goals
Status	Proceed to approval, cancel, and reopen
Approvers	Add or remove approvers and change their working status
Authors / testers	Add or remove authors in change activities or testers in validation activities, and change their working status

The decision governance framework provides governance aspects that are based on states and governance roles:

- The states of releases and activities (for example, *In Progress* and *Complete*).
- The current user's governance role as a participant in the release or activity (owner, author, tester, approver).

A user who has administrator privileges in Decision Center can do the operations of all the different governance roles. Similarly, users with these privileges have access to all the branches and have all the permissions on all artifacts. However, administrators cannot force state transitions that are not allowed by the decision governance framework.

When you use the decision governance framework, some conditions are imposed, and you must take into account any conditions that are imposed by the security feature. For example, you can rename a release only under the following conditions:

- The state of the release is not *Complete*.
- You are the owner of the release.
- You have the Update/Release/Name permission on the release, or security on the release is not enforced.

**Parent topic:** [Administering security](#)

## Managing groups of users

From the **Administration** tab of the Business console, you can create, delete, and edit groups, and assign users to these groups.

In the **Users** tab, you see a list of users who have access to the Business console. Users are created and managed by the administrator in the Operational Decision Manager on Cloud portal. From the **Users** tab of the Business console, you can assign users to one or more groups, which you define in the **Groups** tab.

Decision Center uses groups for security access to the different branches, and permissions on the artifacts (see [Security](#)). To make use of this security and permissions feature, you must create groups in the Business console, and assign all users to one or more of these groups. You create in Decision Center as many groups as you need to organize your users functionally.

**Note:** Permissions are computed at login time, and are kept during the entire session. If permissions for a group change, or if a permission manager adds a member in a group, users need to log out and log in again to view the artifacts with the appropriate permissions.

### Permission profiles

In Decision Center, security defines which groups have access to the different branches of a decision service. By default, no group is allowed to create, update, view, or delete anything until these permissions are explicitly granted, unless the group has administrator privileges.

In the Business console, you can assign one of the following permission profiles to each group:

#### None

Groups assigned this permission profile have access to the branch, but cannot see its content.

#### Read Only

Groups assigned this permission profile can view the contents of the branch, but cannot create, update, or delete content.

#### Full Authoring

Groups assigned this permission profile can view, create, update, or delete all content in the branch.

**Note:** Permissions that you define for a group apply to all Decision Center branches that enforce security.

**Parent topic:** [Administering security](#)

## Setting project security

You can control access to decision service branches by enforcing security for projects in releases, activities, or ungoverned branches. You can also control access to these projects for specific groups of users.

### About this task

When you enforce security within a decision service, you can control which groups of users have access to projects in the different branches. For example, you can enforce security for an entire decision service, and select groups that can access this decision service. You can then restrict access for a specific project, so that among these groups, only some groups can access this project.

You define which users are members of a group in the tab **Groups**.

### Procedure

1. Log in to the Business console as a permission manager.
2. In the **Administration** tab, click **Project security**. In this tab, you have a tree grid view of the decision services. You can expand each decision service and their branches, to see the projects in each branch.
3. Hover your cursor over the branch for which you want to enforce security, and click the icon to edit it.
4. In the window that opens, a status under **Security** indicates whether the security is enforced or not. Click this status to modify the state of the security.

**Note:** If the parent branch already has security enforced, the security settings are inherited from this branch. You can click the message **Security settings are inherited from *Branch*** to override the security settings.

5. For branches on which security is enforced, select the groups of users who can access the branch in the **Groups** section.
6. Optional: After you enforce security in a branch, you can restrict access to some projects in this branch for specific groups of users. Click the Edit icon for your project, and select the groups that can access this project.
7. Click **Done** to save your changes.

**Parent topic:** [Administering security](#)



# Managing servers

As a permission manager, you use the Business console to manage the list of servers available to Decision Center for deployment, testing, and simulations. You can also restrict access to specific groups.

## Procedure

To manage the available servers:

1. On the **Administration** tab, click the **Servers** subtab.

The release manager can access this subtab in read-only mode. Only the permission manager can manage servers.

2. Click **New Server** to add an external server. To edit it, click **Edit Server**.

**Note:** You cannot edit the servers that are accessible by default.

3. Enter the following server details:

- **Server name:** Name displayed when selecting the server from the list of servers.
- **Server URL:** Web address of the Rule Execution Server or Decision Runner server.

**Note:** A Rule Execution Server is used for deployment, and a Decision Runner server is used for testing and simulation purposes. By default, the URLs are `http://<hostname>:<port>/RES` and `http://<hostname>:<port>/DecisionRunner`.

You can add or manage HTTPS servers if the certificate that you use is a CA-signed certificate.

- **User name/Password:** Access credentials to the server. You can leave these credentials blank if you do not want them to be stored.

For servers used to deploy RuleApps, you are asked to enter them when deploying. For running test suites and simulations, you are asked to enter them when clicking the **Test** button. In both cases, your credentials are not stored in Decision Center.

- **Description:** Text to help you identify the server in the table of servers.

4. Specify how each server is used and who has access to it:

- **Usage:** Select whether you want to use the server for deploying RuleApps, or for running test suites and simulations.
- **Groups:** Control which Decision Center groups can access this server when configuring deployment or running test suites and simulations. By default, all groups can access a server. To reduce this access, clear **All groups** and select the required groups in the list.

5. Test that the connection is working correctly by clicking **Test**.

6. Click **OK**.

**Parent topic:** [Administering projects from the Business console](#)

## Diagnostics

To get information about your system, you can run diagnostics in the **Diagnostics** subtab of the **Administration** tab.

The permission manager can run diagnostics on different parts of Decision Center to check the state of the system. It shows results on versioning, data source, extensions, verbalizers, and database, which can be used to troubleshoot issues.

Diagnostics for artifact counts or projects give you information about how many artifacts you have, which type, and the organization of your decision services. This helps you get precise metrics about your projects and decision services.

You can also find the results of the diagnostics that you run in the Business console in your server logs, or download the results as a JSON file.

**Parent topic:** [Administering projects from the Business console](#)

# Configuration settings

You can set some configuration options to customize Decision Center behavior, or display in the Business console. You can also set custom parameters.

Table 1 shows the configuration options that you can set from the **Administration > Settings** tab.

**Note:** You must have the Permission manager role to edit configuration settings.

Table 1. Business console settings

Setting	Use
Show number of artifacts in Decision Artifacts tree (may impact performance)	Displays the number of artifacts in each project from the <b>Decision Artifacts</b> tab.
Default row ordering for decision tables	Select whether you want to order rows in decision tables with automatic row ordering, which organizes a decision table by grouping rows that share condition values, or manual row ordering, which organizes rows into partitioned groups.
Do not show customer survey	Activate this setting to disable the customer survey that pops up in the Business console and solicits feedback from users.

## Build options for a decision service

You can set build options for a decision service. Open the decision service, and set the options in the **Decision Service** panel.

### Choose when to cancel archive generation

Allow ruleset archive generation to be cancelled on warnings, or errors, or never cancelled.

### Build automatically

Decision Center generates the compiled version of a rule as soon as you save it. Otherwise, this generation occurs when generating the ruleset. You might find enabling this option useful to accelerate ruleset generation.

**Parent topic:** [Administering projects from the Business console](#)

## Exporting or importing the current project

As a permission manager, you can export the working branch of the current project as a .zip file, and import the project previously exported back into Decision Center.

### Exporting

You can export the current state of your working branch from the Business console. Select the branch, release, or activity that you want to export, and click **Export** in the toolbar. In the dialog that opens, generate your .zip file, then click the link to download it. Some artifacts are not included in the .zip file, or included with limitations (see the note at the end of this page).

### Importing

You can import decision services or branches from projects that you previously exported as .zip files. For example, you might want to import the .zip file in another Decision Center, or after recreating the database.

#### Importing a decision service

From the **Decision Services** page, click the **Import Decision Service** icon to import an entire decision service into Decision Center. You can select **Use Decision Governance Framework** if you want your decision service to use the decision governance framework. In this case, the decision service contains a main branch and an initial release. If you do not select this option, your imported decision service contains only a main branch.

If the decision service you are importing already exists in Decision Center, an error message indicates that you must import the elements from a branch.

#### Importing projects

To import decision artifacts into an existing decision service, select your release, change activity, or branch, then click **Import** in the toolbar. During the import, Decision Center synchronizes the existing project with the imported .zip file.

During this synchronization, Decision Center handles differences the following way:

- New elements found in the .zip file are added to the Decision Center version of the project.
- Deleted elements are not synchronized. If you deleted an element in Decision Center and not in the .zip version, the element is added to the Decision Center version. Similarly, if an element is deleted in the .zip version, it is not removed from the Decision Center version.
- Changes made to the elements in the .zip version override elements in Decision Center if you check the option **Replace existing elements in Decision Center**. If this option is not checked, the Decision Center version remains intact.

#### Note:

##### Limitations:

- The history of your branch is not exported.
- Test suites and simulations are exported, with the following limitations:
  - Reports for tests and simulations are not exported.
  - Test suites based on snapshots are not exported.
  - When you import a project into Rule Designer that was previously exported from Decision Center, test suites and simulation artifacts are converted into text files and put in a folder named .validation, hidden by default in Rule Designer. They are available for your information, but **should not be modified**. These files cannot be synchronized with Decision Center.

**Parent topic:** [Administering projects from the Business console](#)

## Setting up project dependencies

Administrators and configuration managers can set up project dependencies in the Business console to make a project available in different decision services or branches.

A project in a decision service can be referenced by other decision services, or in other branches of the same decision service. All items in the referenced project are then available to you for use in your current decision service or branch. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that you need to maintain only one BOM and one vocabulary, and changes made to the BOM and vocabulary are automatically propagated to all the projects that use them.


Before you create project dependencies, make sure that you have a clear idea of how your decision services and branches will be organized to share one or several projects. Take some time to assess how any modification to a rule artifact in a shared project can impact the different branches and decision services that share this project.

When you share a project between branches of a decision service, or in other decision services, you must create a project dependency. You create a project dependency in the Dependencies editor of the Business console where you specify which project and which branch of that project to reference.

**Note:** You cannot share a project if you use the Decision Governance Framework.

### Creating a project dependency

To create a project dependency:


1. Open the **main** branch of your decision service.
2. In the **Branch** pane, expand the **Linked Projects** section, and click **Edit dependencies**. The Dependencies editor opens.
3. Select your project in the **Project** list to see the list of its dependencies (if any). Click the **+** icon to add a new dependency.
4. Double-click the blank fields of the new dependency to select the decision service, project, and branch you want to reference in your current project.
5. Click **Save**.
6. Go back to your branch with the breadcrumbs. Your shared project is indicated by the  shared icon in the **Decision Artifacts** tab, and a tooltip when you hover over the project name. The tooltip is limited to five dependencies at most, but you can review the full list in the **Linked Projects** section.

**Note:** Some operations on versioned elements have no impact on the dependencies. For example, if you restore a snapshot that was taken before creating or editing a project dependency, or if you merge branches that share projects, the dependencies will not be modified.

### Working with branches

When you create a branch in a decision service, it is populated with a new version of each project from the parent branch. If there were shared projects available in the parent branch, those shared projects *are no longer shared* in the new branch. If you need to continue sharing these projects, you must manually re-establish the dependencies.

To re-establish the project dependencies in a new branch:

1. Open your new branch.
2. In the **Linked Projects** pane, click **Edit Dependencies**.
3. In the dependencies list, you see your decision service and project. Double-click the branch name and select **main**.
4. Click **Save**.
5. Go back to your branch with the breadcrumbs. You can see that your common project is shared again, as shown by the  shared icon, and the tooltip where you can see the decision services this project is dependent on.

**Note:** When you re-establish the dependency, the shared project *replaces* the copy of this project that was made when you created a branch. Make sure that you have not modified your business rules or BOM in this copy because in this case you might lose your changes.

### Setting project security for shared projects

Administrators can set permissions for groups of users to access a project from the **Administration > Project Security tab** (see [Setting project security](#)).

When you enforce security on a decision service, the security settings do not apply to shared projects in this decision service. You must set the permissions for these shared projects manually.

The security settings that you apply for a shared project are available to all decision services that references this project.

**Parent topic:** [Administering projects from the Business console](#)

# Working with the Enterprise console

You manage decisions with Decision Center, which you can access in a web user interface, the Enterprise console.

## [Introducing the Decision Center Enterprise console](#)

You use the Decision Center Enterprise console to author, edit, organize, and search for business rules.

## [Decision Center basics](#)

Find out about rule projects, and the basic components: smart folders, versions, branching, baselines, and the different types of project elements that you can use in Decision Center.

## [Explore: Navigate your projects](#)

The Explore tab provides a number of features that you can use to navigate within your rule projects.

## [Compose: Create project elements](#)

You use the Compose tab in the Enterprise console to create project elements.

## [Query: Search your projects](#)

You use queries to search through your rule projects to display the business rules or other project elements that correspond to criteria of your choice.

## [Analyze: Check your projects](#)

You can check your projects for consistency and completeness and generate rule project reports.

## [Project: Manage your project](#)

You use the Project tab to manage advanced features of your projects.

## [Working with the editors](#)

You can choose which editor to use when editing business rules.

## [Configure: Manage your project configuration](#)

You use the Configure tab to carry out some administrative tasks.

# Introducing the Decision Center Enterprise console

You use the Decision Center Enterprise console to author, edit, organize, and search for business rules.

## **Overview: Collaborative working**

Decision Center provides a designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules.

## **The Enterprise console environment**

Learn how to access all the features available in the Decision Center Enterprise console.

## **User options**

You can change user options to make them match your preferences or specific requirements. These options only apply when you use Decision Center, and do not affect any other users.

## **The online help**

You can obtain help when using the Enterprise console, by clicking **Help** in the top banner.

**Parent topic:** [Working with the Enterprise console](#)

## Overview: Collaborative working

Decision Center provides a designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules.

The corporate policies that govern your company have been extracted from its various computer applications and now exist as business rules within a decision management system. Business rules are policy statements expressed in a way that a computer system can interpret. Decision Center serves as a designated workspace where business users can work collaboratively within the decision management system.

The business rules that you author or modify within Decision Center are not immediately reflected back to the rule-based computer applications on which they act. You must deploy them. Deployment involves sending sets of rules to a Rule Execution Server that handles the execution of the rules and its interaction with the rule-based applications.

**Parent topic:** [Introducing the Decision Center Enterprise console](#)

**Related information:**

[The Enterprise console environment](#)

[Decision Center basics](#)



# The Enterprise console environment

Learn how to access all the features available in the Decision Center Enterprise console.

## Overview of the Enterprise console environment

The Enterprise console is designed with options in the top banner, tabs for various operations, navigation for selecting projects and branches, and explanations to help you work.

### The Home tab

You use the **Home** page to select the decision service to work on. You choose a branch or a release activity, and an action to do.

### The Explore tab

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

### The Compose tab

Use the **Compose** tab to create or edit project elements.

### The Query tab

Use the **Query** tab to create, edit, and run queries.

### The Analyze tab

Use the **Analyze** tab to generate reports on your projects and their project elements.

### The Project tab

Use the **Project** tab to carry out advanced project-related tasks.

**Parent topic:** [Introducing the Decision Center Enterprise console](#)

# Overview of the Enterprise console environment

The Enterprise console is designed with options in the top banner, tabs for various operations, navigation for selecting projects and branches, and explanations to help you work.

The Enterprise console provides a group of tabs for you to access information and features:

## Home

To select a rule project or decision service, a branch or release and activity, and an action.

## Explore

To move through the current project.

## Compose

To create or edit project elements.

## Query

To do specific searches on your projects.

## Analyze

To generate reports on your projects and project elements.

## Project

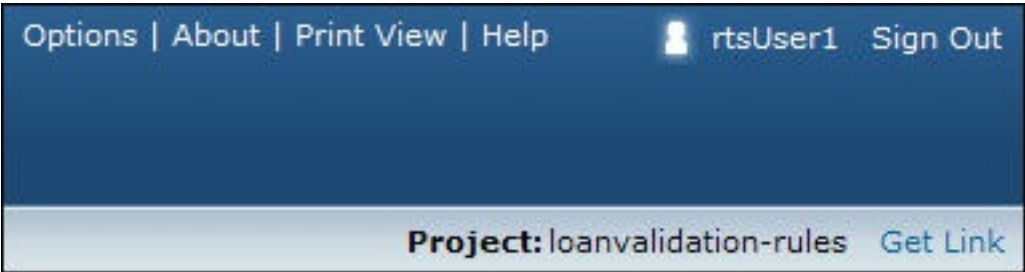
To carry out project-related tasks.

**Note:**

Some types of users have extra privileges that give access to other tabs used for specific tasks. For information about the extra privileges, refer to [Configure: Manage your project configuration](#).

## The top banner

The top banner is available regardless of which tab you select.



You use the top banner to sign out of the Enterprise console or to access the following buttons:

## Options

To set user options such as language, theme, columns to include in tables, order of smart folders, and choice of editor.

## Help

To display online help.

## Print View

To display the current view in a format for printing. Click **Normal View** to return to a view with the top banner.

The name of the current project and branch is visible under these buttons.

**Note:**

The Details or Version pages provide a **Get Link** hyperlink next to the project name or next to the name of project elements. Right-click this hyperlink to copy the exact URL leading to the project or project element.

## Interaction between tabs

When you do certain actions, you might move from one tab to another. For example, clicking **New** or **Edit** in the **Explore** tab takes you to the **Compose** tab.

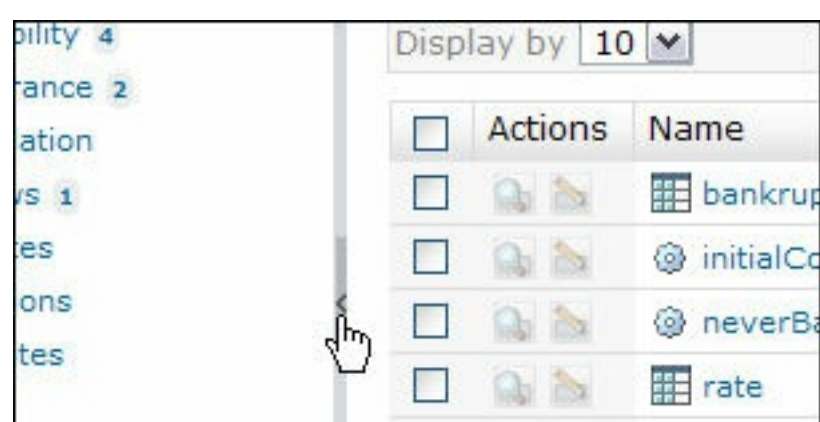
A tab can contain more than one level of pages. For example, you consult previous versions of a project element ([Exploring the history of an element](#)) on the **History** page of the **Explore** tab:

Home	Explore	Compose	Query	Analyze
Details > History				
<a href="#">Explore Version Details</a>   <a href="#">Compare 2 Versions</a>   <a href="#">Restore Version</a>   <a href="#">Copy</a>				
Display by 10 ▾				
<input type="checkbox"/>	Version	Changed By	Comment	
<input type="checkbox"/>	1.0	rtsAdmin		
<input type="checkbox"/>	1.1	rtsUser1		

The breadcrumbs, which are located just under the tab names, indicate which page of a tab you are currently on. You also use them to move back to previous pages. Use the breadcrumbs to move back to a page, not the **Back** button.



You can adjust the size of your work area in the **Explore**, **Compose**, and **Query** tabs by sliding the division area. The button on this division area hides or shows the area completely.



**Parent topic:** [The Enterprise console environment](#)

**Related information:**

[Overview: Collaborative working](#)  
[Decision Center basics](#)

## The Home tab

You use the **Home** page to select the decision service to work on. You choose a branch or a release activity, and an action to do.

When the Enterprise console opens, it displays the **Home** page.

The **Home** page contains the following sections:

- **Work on a decision service:** You work on the rules in a decision service without the decision governance framework.
- **Work on a decision service within the decision governance framework:** You use the decision governance framework that is used in the Business console. You work in change activities within releases.

### Note:

When you select a decision service, it is displayed in the top banner.

After you select a decision service and a release activity, you choose an action:

### Work on branch/release

This action is selected by default to carry out work on your decision service.

### View a baseline/deployment baseline

These actions are available when baselines exist for the selected decision service.

### View deleted items

This action activates the recycle bin. You can restore items from the bin.

### Create subbranch

This action creates a subbranch of the current branch. You can also create a subbranch from the **Project** tab (see [Managing subbranches](#)).

**Parent topic:** [The Enterprise console environment](#)

### Related concepts:

[Rule projects](#)

### Related tasks:

[Signing in to the Enterprise console](#)

[Baselines](#)

[Branches](#)

[Recycle bin](#)


# The Explore tab

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

In the **Explore** tab, you can select project elements and then edit, copy, delete, and lock them. For more information about what you can do on the **Explore** tab, refer to [Explore: Navigate your projects](#).

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed.

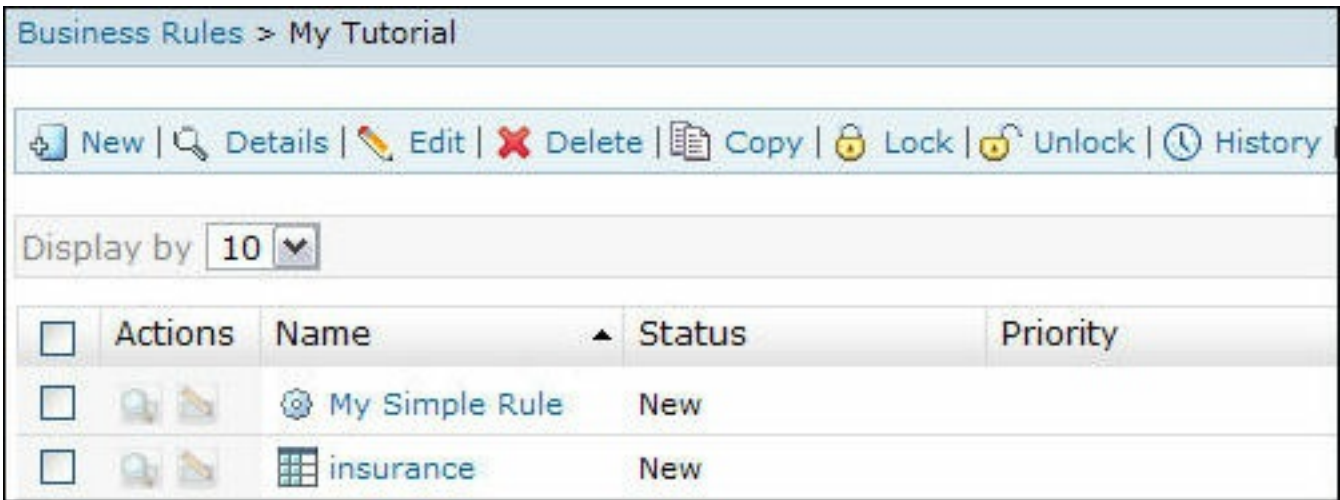
The default smart folder for a rule project is the **Business Rules** smart folder, which displays all business rules (action rules, decision tables, and decision trees) contained in the project, organized in the folders to which they belong.

You can add new smart folders or customize existing ones (see [Creating a smart folder](#)). Use the refresh icon  on the toolbar if you want your current session of Decision Center to display immediately changes done to the project by concurrent users.

The default smart folders display most of the project elements you need to work on. However, some more technical project elements, such as resources, variable sets, technical rules, and functions are not displayed in any of the default smart folders. You can create new smart folder for these types of project elements.

## Tables


Click a folder to display its contents in a table:



Tables consists of rows and columns. The first row displays the headings, and each subsequent row corresponds to an item in the folder or view. Click a heading name to sort the items alphabetically or by date, depending on the nature of that column.








**Note:**

You can choose what information the tables display by clicking **Options** in the top banner.

The first column contains check boxes. To work on one of the items in the table, select it by clicking its check box , and then use the toolbar to perform specific tasks on it.


The icon next to the name of the project element indicates the type of element represented in that row. The following table describes the different icons.

Table 1. Icons for types of elements


Icon	Type of element
	Action rule.
	Decision table.
	Ruleflow.
	Smart folder.
	Variable set.
	Function.
	Resource.


The **Locked** icon  indicates that an item is locked.


Use the following commands to edit an element:

- Click **Edit** on the toolbar to access the Compose wizard for that element.
- Click the **Edit this element** icon  to access the editor area under the table on the Explore tab.

Click the name of the project element to display its Details page.

Click the **Preview** icon  to display a quick preview of the selected project element just under the table:

 **Rule Preview**

 Edit

**Name**

My Simple Rule

**Status**

New

*if*

*it is not true that* the spouse of **'the borrower'** has filed a bankruptcy

*then*

set the credit score of **'the borrower'** to the credit score of **'the borrower'** + 20 ;

**Parent topic:** [The Enterprise console environment](#)

**Related concepts:**

[Smart folders](#)  
[Folders](#)

**Related tasks:**

[Locking project elements](#)  
[Displaying project element details](#)

**Related information:**

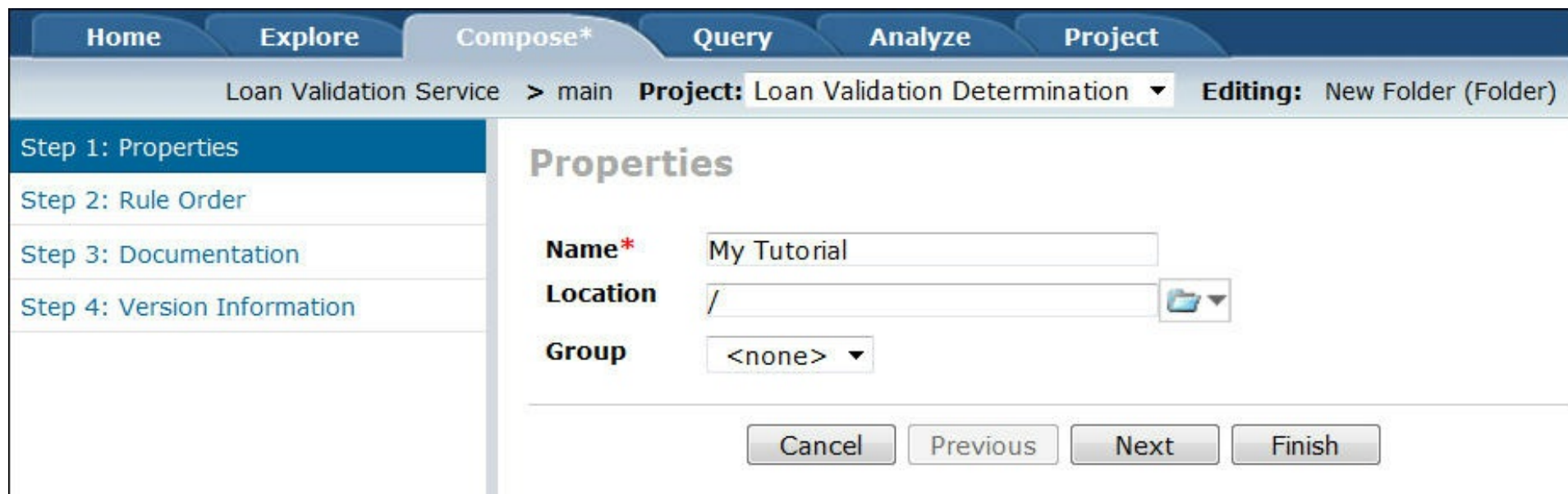
[Editing a project element](#)



## The Compose tab

Use the **Compose** tab to create or edit project elements.

To create a new element, select the type of element from the available types or templates, and then use the Compose wizard to work through the steps required to create or edit your element:



The screenshot shows the 'Compose\*' tab selected in a navigation bar with other tabs: Home, Explore, Query, Analyze, and Project. Below the navigation bar, a breadcrumb trail reads 'Loan Validation Service > main'. To the right, it says 'Project: Loan Validation Determination' with a dropdown arrow, and 'Editing: New Folder (Folder)'. On the left, a sidebar lists four steps: 'Step 1: Properties' (highlighted), 'Step 2: Rule Order', 'Step 3: Documentation', and 'Step 4: Version Information'. The main area is titled 'Properties' and contains three fields: 'Name\*' with the value 'My Tutorial', 'Location' with a slash '/' and a folder icon, and 'Group' with a dropdown menu showing '<none>'. At the bottom right of the main area are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

An asterisk (\*) next to the name of the **Compose** tab indicates that you are in the process of creating or editing a project element. This asterisk disappears when you click **Finish** or **Cancel**. If you move to another tab while editing a project element, the asterisk remains and serves as a reminder that you are currently editing a project element.

### Note:

The rules that you create or edit are verified at different levels before they are deployed (see [Rule verification](#)).

**Parent topic:** [The Enterprise console environment](#)

### Related information:

[Decision Center basics](#)

[Compose: Create project elements](#)

## The Query tab

Use the **Query** tab to create, edit, and run queries.

You use queries to search through your projects for business rules or other project elements that correspond to criteria of your choice. For example:

```
Find
 all business rules
 such that the status of each business rule is deployable
```

On the results of a query you can perform actions, generate a report, or carry out rule analysis.

**Parent topic:** [The Enterprise console environment](#)

**Related concepts:**  
[Introducing queries](#)

**Related information:**  
[Query: Search your projects](#)



## The Analyze tab

Use the **Analyze** tab to generate reports on your projects and their project elements.

You use the **Analyze** tab to perform the following actions:

### Check Project Consistency

Checks the consistency of the rules in your project.

### Generate Project Report

Provides detailed information on the ruleset parameters, project elements, and ruleflows in your project.

### Generate Task / Rule Report

Generates an Excel report with detailed information on the rule project elements and rule tasks in your project.

**Parent topic:** [The Enterprise console environment](#)

### Related information:

[Analyze: Check your projects](#)

[Decision Center basics](#)

## The Project tab

Use the **Project** tab to carry out advanced project-related tasks.

You use the **Project** tab to perform the following actions:

### Manage Subbranches and Baselines

To create and manage subbranches and baselines.

### Merge Branches

To merge the contents of different branches.

### Edit Project Options

To enable or disable rule analysis.

### Reload Dynamic Domains

To update the values of certain business terms.

**Parent topic:** [The Enterprise console environment](#)

### Related information:

[Project: Manage your project](#)

## User options

You can change user options to make them match your preferences or specific requirements. These options only apply when you use Decision Center, and do not affect any other users.

To change the following options, click **Options** in the top banner of the Enterprise console to open the **User Options** page.

### Change language to

To select the language in which the Enterprise console is displayed. The language in which you display the Enterprise console is determined by a parameter in the URL you use to access Decision Center, or by the language of your browser if the URL has no parameter. For information about how to set the browser language, refer to the documentation for your browser.

You can also change the language in which you display the Enterprise console to one in the list of languages in **User Options**. Regardless of whether you set the language through the URL, browser, or options, the Enterprise console displays the language only if your development team has translated the Enterprise console environment and provided you with translated projects.

#### Note:

If you select a language and the Enterprise console does not display it, contact your development team.

### Select a theme

To change the general appearance of the Enterprise console. The themes for the user interface include classic, gray, and blue.

### Select a calendar type

To select a calendar for the Enterprise console. Calendars include Gregorian, Hebrew, Islamic and Buddhist.

### Select the columns

To select the columns displayed in the rule tables.

### Number of characters for values

To set the number of characters displayed for values in tables. The minimum number of characters is 10, and the maximum number of characters is 100. This is useful when the names of your project elements are long and get truncated in the tables.

### Display order of smart folders

To set the order in which smart folders are displayed in the **Explore** tab.

### Show the stack

To show the stack when an exception occurs.

### Show a warning for display performance

To receive a warning when heavy smart folders might slow down the display of a page.

### Select the rule editor

To set the editor used by the Enterprise console:

- **Default rule editor**, the editor already associated with a project (see [Project options](#)).
- **Guided editor**
- **Intellirule editor**

### Mirror GUI

To flip the interface horizontally, from right to left. This supports bidirectional (bidi) languages such as Arabic and Hebrew.

### Input Text Orientation

To set the direction in which text is entered.

The options include:

- **Contextual**: matches the direction of the input text (right to left for bidi, and left to right for non-bidi).
- **UI Based**: matches the **Mirror GUI** option (user input language).

- **Right to Left:** enforces right-to-left text direction, usually to support languages such as Arabic and Hebrew.
- **Left to Right:** enforces left-to-right text direction, usually to support languages such as English and French.

### Use keyboard shortcuts

In the Enterprise console, you can navigate through the different tabs and features by pressing the **Tab** key until the feature is highlighted, and then pressing **Enter** to select it. You can also use the **Up** and **Down** arrows to navigate through the smart folders.

In addition, if you select the **Use keyboard shortcuts** option, you can press **Ctrl+Alt** with the following numbers or letters to access different features:

- 1–7: To select a tab, starting at 1 for the **Home** tab.
- 0: To select **OK** on the first page of the **Compose** wizard.
- N: To select **Next** in the steps of the **Compose** wizard.
- P: To select **Previous** in the steps of the **Compose** wizard.
- C: To select **Cancel** in the steps of the **Compose** wizard.
- F: To select **Finish** in the steps of the **Compose** wizard.

**Parent topic:** [Introducing the Decision Center Enterprise console](#)

#### Related tasks:

[Signing in to the Enterprise console](#)

#### Related information:

[The Enterprise console environment](#)  
[Decision Center basics](#)

## The online help

You can obtain help when using the Enterprise console, by clicking **Help** in the top banner.

You can navigate the online help from the table of contents, or using the search tab. In addition, clicking on the **Help** icons located throughout the Enterprise console opens the online help with the appropriate topic displayed.

If you are unable to find the information you are looking for, contact IBM® Customer Support.

**Parent topic:** [Introducing the Decision Center Enterprise console](#)

**Related information:**

[The Enterprise console environment](#)

[Decision Center basics](#)

## Decision Center basics

Find out about rule projects, and the basic components: smart folders, versions, branching, baselines, and the different types of project elements that you can use in Decision Center.

### [Rule projects](#)

A rule project corresponds to a specific organization of rules and other project elements.

### [Branches](#)

Branches facilitate work done on parallel releases of a project.

### [Smart folders](#)

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed in the Enterprise console.

### [Folders](#)

Folders in Decision Center help you organize your project elements.

### [Versioning](#)

Decision Center creates archived versions of elements contained in your projects each time you modify them.

### [Baselines](#)

Baselines capture the state of a project or branch at a specific moment in time.

### [Recycle bin](#)

The **recycle bin** contains project elements that have been deleted from the current state of a branch.

### [Action rules](#)

You express business policies in rules that match actions to conditions.

### [Decision tables](#)

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

### [\(Deprecated\) Decision tree overview](#)

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

### [\(Deprecated\) Templates](#)

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

### [Project element properties](#)

A property provides additional information about a project element.

### [Overview: Ruleflows](#)

You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

### [Rule verification](#)

You can perform different levels of verification on your rules before you deploy them to the rule-based computer applications on which they act.

### [Variable sets](#)

Variable sets contain variables that you can use across the different elements of a rule project. You can also use these variables to pass data between tasks in a ruleflow.

### [Functions](#)

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

### [Technical rules](#)

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

**Parent topic:** [Working with the Enterprise console](#)

## Rule projects

A rule project corresponds to a specific organization of rules and other project elements.

The Decision Center repository can contain many projects, but projects are displayed one at a time in the console.

A project is made of a main line, sometimes called the main branch. To support the development cycle of projects, you can create subbranches of the main line or of any branch, and then merge changes between branches.

You can also create baselines of a project or branch. Baselines are a snapshot of the state of a project or branch at a given point in time.

**Note:**

You can see what the current project, branch, and baseline is, either in the top banner or on the **Home** tab.

**Parent topic:** [Decision Center basics](#)

**Related tasks:**

[Baselines](#)

[Branches](#)

[Recycle bin](#)

[Erasing the current project](#)

**Related information:**

[Explore: Navigate your projects](#)

[Project: Manage your project](#)

# Branches

Branches facilitate work done on parallel releases of a project.

## About this task

You select the branch you want to work on in the **Home** tab. You can also create subbranches from the **Home** tab, and manage them from the **Project** tab.

Initially, a project contains only a main line in which you work. From the main, sometimes referred to as the main branch, you can create one or more subbranches. From any new branch you can create other subbranches, as many as you need to manage your releases.

### Note:

The name of the branch you are currently working on appears in the top banner next to the project name.

When you create a subbranch, it contains an exact replica of every project element contained in the parent branch. You can then work on the subbranch without affecting the content of its parent, and eventually merge different branches together ([Merging branches](#)).

### Note:

To understand how different versions of project elements are handled, see [Versioning](#).

## Procedure

To work on a branch:

1. Go to the **Home** tab.
2. On the **Home** tab, select the required branch from the drop-down list next to **Branch in use**.

**Parent topic:** [Decision Center basics](#)

### Related concepts:

[The Home tab](#)

### Related tasks:

[Managing subbranches](#)

[Versioning](#)

[Recycle bin](#)

### Related information:

[Merging branches](#)





## Smart folders

You use smart folders to specify which elements you want to access in a given rule project, and how they are displayed in the Enterprise console.

For example, you might want to display only the rules that are new to the project so that you can validate that they correctly implement company business policies.

The default smart folder for a rule project is the **Business Rules** smart folder, which displays all business rules contained in the project, organized in the folders to which they belong.

To access the smart folders for a given rule project, go to the **Explore** tab. Each smart folder has a **smart folder** icon  next to it. Click the expand  icon next to a smart folder to see its contents.

To change the order in which your smart folders are displayed on the **Explore** tab, click **Options** in the top banner. You cannot lock smart folders manually, but a view locks automatically when you edit it.

### Note:

If project security is enforced, you must at least have View permission on smart folders to see other project elements on the **Explore** tab.

**Parent topic:** [Decision Center basics](#)

### Related concepts:

[Rule projects](#)

[Folders](#)

### Related tasks:

[Creating a smart folder](#)

[Locking project elements](#)

# Folders

Folders in Decision Center help you organize your project elements.

Folders in Decision Center are similar to folders in a file system, but are closely linked to smart folders (see [Smart folders](#)) in the Enterprise console. Existing folders are only visible under smart folders that are set up to display their project elements by folders.

You assign a project element to a given folder through the project element's Folder property (see [Project element properties](#)). Changing the folder property of a project element is the same as moving it from one folder to another (see [Moving a folder or rule](#)).

**Note:**

A project element does not have to be assigned to a folder, but it is good working practice to reduce the number of project elements that are not assigned.

To ensure that folders meant to display certain types of project elements appear under the appropriate smart folders, the Enterprise console proposes different folder hierarchies:

Folder type:	Under smart folders that display:
Business rules	Business rules (action rules and decision tables), technical rules, ruleflows, functions, or variable sets.
Resource	Resources
Templates	Templates (Templates are deprecated)

When you create a project element, the Enterprise console only proposes folders of the correct type for that project element.

**Note:**

An empty folder does not synchronize with Rule Designer.

**Parent topic:** [Decision Center basics](#)

**Related concepts:**  
[Smart folders](#)

**Related tasks:**  
[Creating a folder](#)

**Related information:**  
[Explore: Navigate your projects](#)

# Versioning

Decision Center creates archived versions of elements contained in your projects each time you modify them.

## About this task

You can consult the history of modifications made over time, and restore a previous version if required. You can edit or delete only the current version of an element.

The version number of a project element is made of a major number and a minor number:

### Major number

Found before the decimal point, the major version number corresponds to the branch. Decision Center gives the main line the major version number 1, and attributes the next available number (2, 3, and so on) for each branch of a project that you create.

### Minor number

Found after the decimal point, the initial version of a project element has the minor version number 0. This minor version number increments by 1 each time you edit the project element.

When you create a subbranch, each project element initially keeps the version number of the parent branch. Then, if you edit the element in the subbranch, it takes on the versioning policy of the subbranch. For example, a project element with the version number 1.5 keeps this number in the subbranch until you edit it, at which point it becomes x.0.

When you merge a subbranch with its parent branch, Decision Center creates a new version of the project element in the parent branch. This new version is an exact copy of the version in the subbranch, and takes the next version number available in the parent branch.

Similarly, when you restore a baseline, Decision Center creates a new version of all the elements found in that baseline, and makes them the current state.

When you restore a version, either from a baseline or from the history of the project or branch, the new version is a copy of the version restored, with the next version number available. For example, if the current version is 1.4 and you restore it with version 1.2, both versions 1.4 and 1.2 remain in the history, but the new working version is 1.5, and 1.5 is an exact copy of version 1.2.

**Note:**

You cannot clear the history of a project element.

## Procedure

To restore a previous version:

1. Access the **History** page for the project element, as described in [Exploring the history of an element](#).
2. On the **History** page, select the version you want to restore by clicking the check box in the corresponding row.
3. Click **Restore Version** in the toolbar.

A new version of the element is created and added to the table.

**Note:**

To copy a previous version, click **Copy** instead and specify the location.

**Parent topic:** [Decision Center basics](#)

**Related concepts:**  
[Version information](#)

**Related tasks:**  
[Exploring the history of an element](#)  
[Branches](#)

**Related information:**  
[Editing a project element](#)

# Baselines

Baselines capture the state of a project or branch at a specific moment in time.

## About this task

For example, if you create monthly baselines of a project, you will be able to consult what the state of the project was in past months, and revert back to a previous baseline if required. See [Managing baselines](#) for information on creating baselines.

### Note:

Note also the existence of deployment baselines, which can be created when deploying RuleApps.

Each branch of a project can contain many baselines, all of which you can consult. When you consult a baseline you cannot edit any of the project elements; you must return to the current state of the project or branch to do so. However, when you consult a baseline you can access the history of the project elements, and restore individual elements to the current state.

### Note:

When you consult a baseline, the name of the baseline appears in the top banner.

## Procedure

To consult a baseline:

1. On the **Home** tab, from the **Current action** drop-down list, select **View a baseline** and the required baseline.
2. To return to the current state, select **Work upon branch** from the **Current action** drop-down list.

**Parent topic:** [Decision Center basics](#)

### Related concepts:

[Rule projects](#)

### Related tasks:

[Managing baselines](#)

[Managing deployment baselines](#)

[Recycle bin](#)

# Recycle bin

The **recycle bin** contains project elements that have been deleted from the current state of a branch.

## About this task

You cannot edit an element in the recycle bin but you can restore it so that it returns to the current state of the branch. You can only restore elements in the recycle bin. You cannot permanently delete them.

### Note:

The recycle bin does not display project elements that you restore to the current state.

The recycle bin contains the two standard smart folders (Business Rules and Ruleflows) to help you to organize the content, as well as any other smart folders that you have deleted.

The recycle bin also displays deleted folders. If you restore a deleted folder, you do not also restore the rules it contains. You must restore them explicitly.

## Procedure

To consult the recycle bin:

1. On the **Home** tab, select **View deleted items** from the drop-down list next to **Current action**.  
Decision Center displays all deleted project elements that have not been restored.
2. To return to the current state of the branch in the project, select **Work upon branch** from the drop-down list next to **Current action**.

**Parent topic:** [Decision Center basics](#)

## Action rules

You express business policies in rules that match actions to conditions.

An action rule defines the specific actions to take when certain conditions are met. A basic action rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what action to perform when a condition is true, for example:

```
if
 the credit score of 'the borrower' is less than 200
then
 in 'the loan report', reject the loan with the message "Credit score
too low.";
```

You write an action rule as a sentence in a natural language. The rule is composed of business terms, operators, and values. In the example, the credit score of the borrower is a business term, is less than is an arithmetic operator, and 200 is a value.

Business applications call the rules to execute them, and provide data values for the business terms. In the example, the action rule must access data for the business terms the credit score of the borrower and the loan report.

To provide a complete statement, an action rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

### Definitions (definitions)

Use the definitions part to define variables for the rule.

The definitions part is optional.

### Conditions (if)

Use the if part to specify the conditions for performing the actions in the then and else parts. In the example, the condition is the value of the applicant's shopping cart is more than \$100.

The if part is optional. Rules without conditions perform their actions under all circumstances.

### Actions (then)

Use the then part to define one or more actions to perform if the if part is true. The action in the example says to apply a 15% discount if the if part is true.

The then part is mandatory. The rule must have at least one action.

### Alternative actions (else)

Use the else part to define one or more actions to perform if the if part is false. The else part of the example says to apply a 5% discount if the if part is false.

The else part is optional. If the if part of a rule is false, and there is no else part, the rule does not execute an action.

#### Rule variables

You create a rule variable to define the scope of a rule.

#### Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

#### Rule actions

Rule actions define what to do when the if part of the rule is true or false.

**Parent topic:** [Decision Center basics](#)

# Rule variables

You create a rule variable to define the scope of a rule.

A rule variable is a name to which you assign a value. You define the variable in the definitions part of an action rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your action rules easier to build and understand by simplifying terms.

## Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term of to chain together business terms:

```
if
 the value of the shopping cart of a customer is less than $100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named the cart to represent the business terms shopping cart and customer:

```
definitions
 set 'the cart' to the shopping cart of customer;
if
 the value of 'the cart' is less than $100
then...
```

## One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the definitions part defines the variable Smith, which is then used in the if and then parts of the rule.

```
definitions
 set Smith to a customer;
if
 the category of Smith is Gold
then
 apply 10 % discount to the shopping cart of Smith;
```

## When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the rule. For example, you might define a variable named 'low risk borrowers' for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in the then and else parts of the rule. For more information, see [Dependency of rule actions on rule variables](#).

## Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
( )	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
'	Curly quotes

## Single quotation marks

Depending on the rule editor and the variable name, you might have to enclose a variable in single quotation

marks:

<div>R u l e d i t o r</div>	<div></div> <div>Description</div>
<div>In t e l l i r u l e e d i t o r</div>	<div>If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.</div>
<div>G u i d e d e d i t o r</div>	<div>When you define or use a rule variable, you do not enclose the name of the variable in single quotation marks. However, if you create a rule with the guided editor, and then use the Intellirule Editor to edit it, you must use single quotation marks to enclose any rule variable name that contains one or more spaces.</div>

**Types of rule variables**

You can assign different types of values to your rule variables.

**Parent topic:** [Action rules](#)

**Related concepts:**

[Types of rule variables](#)

[Dependency of rule actions on rule variables](#)



## Types of rule variables

You can assign different types of values to your rule variables.

A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, customer). Once you set a variable, you can use it in any part of the rule that declares the variable.

### Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable `maxAmount` makes the following rule easier to understand, and ensures that the `if` and then parts of the rule use the same value:

```
definitions
 set maxAmount to 1000000;
if
 the amount of 'the loan' is at least maxAmount
then
 in 'the loan report', reject the data with the message "The loan
cannot exceed " + maxAmount;
```

### Restrictions on rule variables

You can further restrict a variable in the definitions part of a rule by using the operator `where`.

#### Example

In the following rule, `where` restricts the `loyal customer` variable to customers in the Gold category:

```
definitions
 set 'loyal customer' to a customer
 where the category of this customer is Gold;
if
 the value of the shopping cart of 'loyal customer' is more than $200
then
 apply the super discount;
```

#### Example

The following rule declares the `senior Gold customer` variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
 set 'senior Gold customer' to a customer
 where all the following conditions are true:
 - the category of this customer is Gold
 - the age of this customer is at least 65;
```

### Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

When developers enable the automatic variable for a business term, the word `the` identifies the automatic variable in locales that use this type of article. For example, if you use the word "customer" as a business term in your rules, developers can define an automatic variable named `the customer`, which you can then refer to in rules.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the definitions part of the rule, for example:

```
definitions
 set applicant to a customer;
 set 'loyal customer' to a customer;
if
 all of the following conditions are true:
 - applicant is married to 'loyal customer'
```

```
 - 'loyal customer' is insured
then
 upgrade applicant's rating;
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
 'customer 1' is married to 'customer 2'
```

The condition identifies and names two different customers: customer 1 and customer 2. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
 set 'customer 1' to a customer;
 set 'customer 2' to a customer;
if
 'customer 1' is married to 'customer 2'
 and 'customer 2' is insured
then
 upgrade 'customer 1''s rating;
```

**Note:** When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

### Rule variables that retrieve all the occurrences of a business term

You can use the BAL operator `all <...>` to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
 set 'gold customers' to all customers
 where the category of this customer is gold;
 set 'junior gold customer' to a customer in 'gold customers'
 where the age of this customer is at most 15;
 set 'senior gold customer' to a customer in 'gold customers'
 where the age of this customer is at least 65;
```

The example creates three variables:

- gold customers: A list of the customers in the gold category.
- junior gold customer: A customer from the list of gold customers whose age is at most 15 years old.
- senior gold customer: A customer from the list of gold customers whose age is at least 65 years old.

**Parent topic:** [Rule variables](#)

#### Related concepts:

[Rule variables](#)

[Dependency of rule actions on rule variables](#)

## Rule conditions

You use rule conditions to state when to carry out the rule actions in the then and else parts of a rule.

You define conditions in the if part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the rule. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule redirects a call to a member of the Gold team only if the customer category is Gold:

```
if
 the customer category is Gold
then
 redirect the call to a member of the Gold team;
```

The if part in this rule contains one condition, which can be either true or false. The rule carries out the action statement in the then part only if the condition is true.

### Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

### Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

### Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

### Combinations of conditions

You can apply conditions to groups, and test nested groups.

### Condition negation

You can set a rule to perform an action when a condition is not true.

**Parent topic:** [Action rules](#)

**Related concepts:**

[Conditions that test for existence](#)

[Combinations of conditions](#)

## Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and boolean (true or false) values, dates, business terms, and text strings.

The following sections show how to use the business operators to compare data:

### starts with

You can use the text operator `starts with` to determine whether a text string starts with a specific sequence of characters:

```
if
 the customer's maintenance number starts with "TX"
then
 redirect the call to call center A;
```

### is more than

You can use the numerical operator `is more than` to compare two numerical values:

```
if
 the purchase value of the shopping cart is more than $100
then
 apply a 10% discount to the value of the shopping cart
```

### after <date> and before <date>

You can use the date and time operators `after <date>` and `before <date>` to compare two dates:

```
if
 the return date of 'rented car' is after 'pickup date' and before
 'scheduled return date'
then...
```

**Parent topic:** [Rule conditions](#)

## Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.

The following definitions cover operators that test for specified data.

### there are

The operator `there are` tests whether there is a specific number of business terms of a certain type in the data from the application that calls the rule.

The following condition checks for two customers:

```
if
 there are 2 customers
then...
```

You can use the operator `where` to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
 there are 10 customers where the category of each customer is Gold,
then...
```

### there are at least and there are at most

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- `there are at least`: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- `there are at most`: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
 there are at most 3 customers
 where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- `silver customers`: The list of customers in the Silver category.
- `silver count`: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
 set 'silver customers' to all customers
 where the category of each customer is Silver;
 set 'silver count' to the number of 'silver customers'
if
 there are at least ('silver count' + 1) customers
 where the category of each customer is Gold,
then...
```

### there is at least one

The BAL operator `there is at least one` checks that there is at least one instance of a business term in the data from the application that calls the rule. The data can contain more than one instance of the business term.

The following condition looks for at least one customer object:

```
if
 there is at least one customer
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
 set 'gold customers' to all customers
 where the category of each customer is Gold;
if
 there is at least one customer in 'gold customers'
 where the age of this customer is at least 65,
then...
```

**Parent topic:** [Rule conditions](#)

**Related concepts:**

[Rule conditions](#)

[Combinations of conditions](#)

## Conditions that test set membership

You can use conditions to test whether a business term belongs to a set.

The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
 the customer category is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable `luxury car groups` contains the car group that is specified in the current rental agreement. The rule first defines the variable `luxury car groups` as the set of all car groups with a daily rate above a certain value:

```
definitions
 set 'luxury car groups' to all car groups where the daily rate of each
 car group is at least 50;
if
 'luxury car groups' contain the car group of 'the current rental
 agreement'
```

**Parent topic:** [Rule conditions](#)

## Combinations of conditions

You can apply conditions to groups, and test nested groups.

You can use logical operators to combine conditions in the `if` part of a rule.

The following `if` statement contains one condition:

```
if the customer category is Gold
```

In the following `if` statement, the logical operator `and` links two conditions:

```
if the category of the customer is Gold and the customer has been waiting
longer than 5 minutes
```

### Precedence of `and` over `or`

When the logical operators `and` and `or` link conditions in the `if` part of a rule, the `and` operator takes precedence over the `or` operator.

Consider the following `if` statement:

```
if
 the customer is older than 60
 or the customer is younger than 21
 and the category of the customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

### Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following `if` statement, parentheses group an age limit with a category:

```
if
 the customer is older than 60
 or (the customer is younger than 21 and the category of the customer
is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following `if` statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if
 (the customer is older than 60 or the customer is younger than 21)
 and the category of the customer is Student
```

### Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- `all of the following conditions are true:` This business term links all the conditions in the group with the logical operator `and`.
- `any of the following conditions is true:` This business term links all the conditions in the group with the logical operator `or`.

In the following rule, the action is performed only when both conditions are true:

```
if
 all of the following conditions are true:
 - the category of the customer is Gold
 - a member of the Gold team is available
then
 redirect the call to a member of the Gold team;
```



In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
 any of the following conditions is true:
 - the category of the customer is Gold
 - the customer has been waiting longer than 5 minutes
then
 redirect the call to a member of the Gold team;
```

**Parent topic:** [Rule conditions](#)

**Related concepts:**

[Rule conditions](#)

[Conditions that test for existence](#)

## Condition negation

You can set a rule to perform an action when a condition is not true.

You can use these operators to test whether one or more conditions are not true.

### **it is not true that**

You use the operator `it is not true that` to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
 the category of the customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
 it is not true that the category of the customer is Gold
then...
```

### **none of the following conditions are true**

You can negate a group of conditions by using the operator `none of the following conditions are true`. The logical operator `and` links the conditions in the group:

```
if
 all the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

The following `if` statement is true only if all of the conditions are not true:

```
if
 none of the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

**Parent topic:** [Rule conditions](#)

## Rule actions

Rule actions define what to do when the `if` part of the rule is true or false.

You define actions in the `then` and `else` parts of a rule. A rule action consists of at least one action phrase that the rule executes when the `if` part of the rule is true. It can also contain action phrases to execute when the `if` part of the rule is false.

You define actions in the following parts of a rule:

- `then`: This part defines actions that the rule executes when the `if` part of the rule is true or when no `if` part exists.
- `else`: This optional part defines actions that are executed when the `if` part of the rule is false. You do not have to declare an `else` part for a rule.

The `then` part of a rule must define at least one action. If the `then` or `else` part contains multiple actions, the rule executes the actions in their listed order.

### Example of the `then` part

The following rule applies a discount of 15% only when the value of the shopping cart is more than \$100, otherwise the rule applies no action.

```
if
 the value of the shopping cart is more than $100
then
 apply a 15% discount on the shopping cart;
```

### Example of the `else` part

The following rule applies a change when the conditions are met (`then`), or a different change when the conditions are not met (`else`):

```
if
 'the loan report' is approved
 and the loan grade in 'the loan report' is one of { "A", "B", "C" }
then
 in 'the loan report', accept the loan with the message
 "Congratulations! Your loan has been approved";
else
 in 'the loan report', refuse the loan with the message "We are sorry.
 Your loan has not been approved";
```

### Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

### Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

**Parent topic:** [Action rules](#)

### **Related concepts:**

[Rule actions for lists of business terms](#)

[Dependency of rule actions on rule variables](#)

## Rule actions for lists of business terms

You can define actions that a rule executes for each item in a list.

You use the operator for each <item> in <list> to apply actions to items in a list.

The following group of action statements applies to every item in the list of expensive items.

```
then
 for each item in expensive items:
 - apply 5% discount to this item;
 - display the message: "A 5% discount has been applied";
```

**Parent topic:** [Rule actions](#)

**Related concepts:**

[Rule actions](#)

[Dependency of rule actions on rule variables](#)

## Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

You define rule variables in the definitions part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its else part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
 all of the following conditions are true:
 the value of the customer's shopping cart is more than $100
 the category of the customer is Gold
then
 apply a 15% discount
else
 apply a 5% discount
```

However, you can change the rule by using the definitions part to give a discount to only customers in the Gold category:

```
definitions
 set applicant to a customer
 where the category of this customer is Gold
if
 the value of the applicant's shopping cart is more than $100
then
 apply a 15% discount
else
 apply a 5% discount
```

**Parent topic:** [Rule actions](#)

**Related concepts:**

[Rule variables](#)

[Types of rule variables](#)

[Rule actions](#)

[Rule actions for lists of business terms](#)

# Decision tables

A decision table groups rules that have similar conditions and actions, and helps you spot problems such as overlaps and gaps among the rules.

A decision table contains rows and columns that work together to form rules. The decision engine executes the rules row by row, from the first row to the last row.

In the following table, each numbered row expresses a rule. The columns define the conditions (Grade and Amount of loan) and actions (Insurance required and Insurance rate) of the rules in the table. The cells at the top of the table are the column headers, which hold the names of the columns.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	<100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥600,000		true	0.005
5	B	<100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		≥600,000		true	0.0075

When an application calls a decision table, the action rules are executed. If the conditions in a row are met, the rule that is formed by the row performs the actions in the row.

You can add rows to the decision table and enter values in their cells to create new rules. You can also define preconditions that apply to all the rules in a table. Error markers help you find overlaps and gaps in your rules.

## Columns

Each column in a decision table represents a condition or an action.

## Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

## Preconditions

You can pretest data before using it with a decision table.

Parent topic: [Decision Center basics](#)

# Columns

Each column in a decision table represents a condition or an action.

The columns in a decision table hold the conditions and actions of the rules in the table. The top cell of each column identifies the object of a condition or the target of an action.

Row	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001

Each numbered row in the table forms a rule. The rule performs the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of the loan is between 100000 and 300000
then
 set the Insurance required to true
 set the Insurance rate to 0.001
```

## Condition operators

You can split a condition across columns in a decision table when a rule statement contains more than one value. For example, the following condition requires you to specify values for <min> and <max>:

```
if
 the age of the customer is between <min> and <max>
```

In condition columns, you set a default operator for all the cells. You can change this operator for certain cells if you need to. The number of parameters depends on the operator, so columns with different operators might show a different number of subcolumns. For example, the Age condition column might have been defined with the `is between` operator, which takes two parameters, to define different age ranges. In the third row, the operator `is more than` is used to cover the remaining values over 40, and this operator takes only one parameter.

Age	
Min	Max
18	25
26	40
> 40	

Parent topic: [Decision tables](#)

Related concepts:

- [Rows and cells](#)
- [Preconditions](#)

Related tasks:

- [Adding, removing, and editing columns](#)
- [Creating business rules](#)
- [Adding, removing, and editing rows](#)
- [Defining preconditions](#)

# Rows and cells

Each numbered row in a table forms a rule. The values in the cells of the row describe the conditions and actions of the rule.

You create a rule by adding a row to a table, and entering values in the cells for conditions and actions:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3	B	600,000	900,000	true	0.005

The decision engine executes the rules row by row, from the first row to the last row.

## Partitions

You can group a set of consecutive rows that have the same value for a given condition. In this case, the cells to the right of the grouped values are part of the same partition. All cells in the first column of the table belong to the same partition by default.

**Note:** If you are using the editor with right-to-left orientation, the partition consists in the cells to the left of the grouped values.

In Rule Designer, you group cells by merging them.

In the following table, the Grade A cells of rows 1 and 2 are grouped into one cell. As a consequence, the cells that correspond to the amount of the loan become part of the same partition.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003

You read the two rules as follows:

- Rule 1

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
then
 - set the insurance required to true
 - set the loan rate to 0.001
```

- Rule 2

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

A partition cannot be derived solely based on the fact that rows have the same a value. In the following table, we added a row that has the same value for the grade as the other rows. Because its value for the grade is not grouped with the others, it creates a new partition for the amount of the loan and the existing partition remains the same.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	50,000	100,000	false	0.001
2	A	100,000	300,000	true	0.001
3		300,000	600,000	true	0.003



The following example shows a bigger table with several partitions. Cells A and B in the Grade column each have a partition of cells in the Amount of loan column. It means that if a loan request has an A grade, one of rules 1 - 4 might apply depending on the amount of the loan. Here, for example, the possible values for the amount are partitioned into the following ranges: equal to or less than 100000, 100000 - 300000, 300000 - 600000, and more than 600000. These ranges are tested in the cells on the right of grade A.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.005
5	B	< 100,000		false	
6		100,000	300,000	true	0.0025
7		300,000	600,000	true	0.005
8		600,000	800,000		
9		≥ 600,000		true	0.0075

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column and there is at least one condition in the row, the condition that is associated to the empty cell is ignored. In this case, the next cell or partition on its right, if any, is evaluated. If a row contains only empty cells in the condition columns, the rule is not applicable and the entire row is ignored, even if there are actions in the action columns.

The following table illustrates different rows that contain empty cells:

- In row 3, the decision is based on the grade and on the duration of the loan. The amount of the loan is ignored.
- In row 4, the decision is based solely on the grade.
- Row 5 is ignored because it contains no condition.

Grade		Amount of loan		Duration of loan	Insurance required	Insurance rate
		Min	Max			
1	A	100,000	300,000	12	true	0.001
2	A	300,000	600,000	24	true	0.003
3	A			12	true	0.008
4	B				true	0.004
5					false	0.005

If an empty cell is in an action column, the action cell is ignored. In the following table, the first rule does not set an insurance rate.

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	< 100,000		false	
2		100,000	300,000	true	0.001
3		300,000	600,000	true	0.003
4		≥ 600,000		true	0.004

Otherwise cells

You can use an Otherwise cell to collect all the values that are not already covered in a partition.

The behavior of otherwise cells can change in the presence of empty cells. Using both otherwise and empty cells in the same partition is not recommended because it might create overlaps and make the table difficult to understand.

The meaning of otherwise cells follows these principles:

- If a partition contains an otherwise cell and cells with values, the otherwise cell applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell, empty cells, and cells with values, the otherwise cell ignores empty cells and applies to values that are not already covered by other cells in the partition.
- If a partition contains an otherwise cell and no other cell, the otherwise cell becomes irrelevant and is ignored.
- If a partition contains an otherwise cell and all its other cells are empty, the otherwise cell becomes irrelevant and is ignored.

In the following table, the Otherwise cell collects all the values before 100000 and after 600000:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2		300,000	600,000	true	0.003
3		Otherwise		true	0.004

If the grouping of the values for the grade was different, the table might look like this:

Grade		Amount of loan		Insurance required	Insurance rate
		Min	Max		
1	A	100,000	300,000	true	0.001
2	A	300,000	600,000	true	0.003
3		Otherwise		true	0.004

You read the three rules as follows:

- Rule 1

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 100000 and 300000
then
 - set the insurance required to true
 - set the loan rate to 0.001
```

- Rule 2

```
if
 all of the following conditions are true:
 - the loan grade is A
 - the amount of loan is between 300000 and 600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.002
```

- Rule 3

```
if
 all of the following conditions are true:
 - the loan grade is A
 - it is not true that the amount of loan is between 300000 and
600000
then
 - set the insurance that is required to true
 - set the loan rate to 0.003
```

If the grade is A and the amount of the loan is 200000, both rules 1 and 3 apply and the table displays a warning because there is an overlap. The last row that applies is the third row, so the third rule is executed and the rate of the insurance is set to 0.003.

Parent topic: [Decision tables](#)

**Related concepts:**[Columns](#)[Preconditions](#)**Related tasks:**[Adding, removing, and editing columns](#)[Creating business rules](#)[Adding, removing, and editing rows](#)[Defining preconditions](#)

# Preconditions

You can pretest data before using it with a decision table.

Preconditions give you a way to test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
 set 'wealthy customer' to a customer
 where the average monthly balance of this customer
 is more than $1 000 000
if
 the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable `wealthy customer`. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of \$1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

**Parent topic:** [Decision tables](#)

**Related concepts:**

[Columns](#)

[Rows and cells](#)

**Related tasks:**

[Adding, removing, and editing columns](#)

[Creating business rules](#)

[Adding, removing, and editing rows](#)

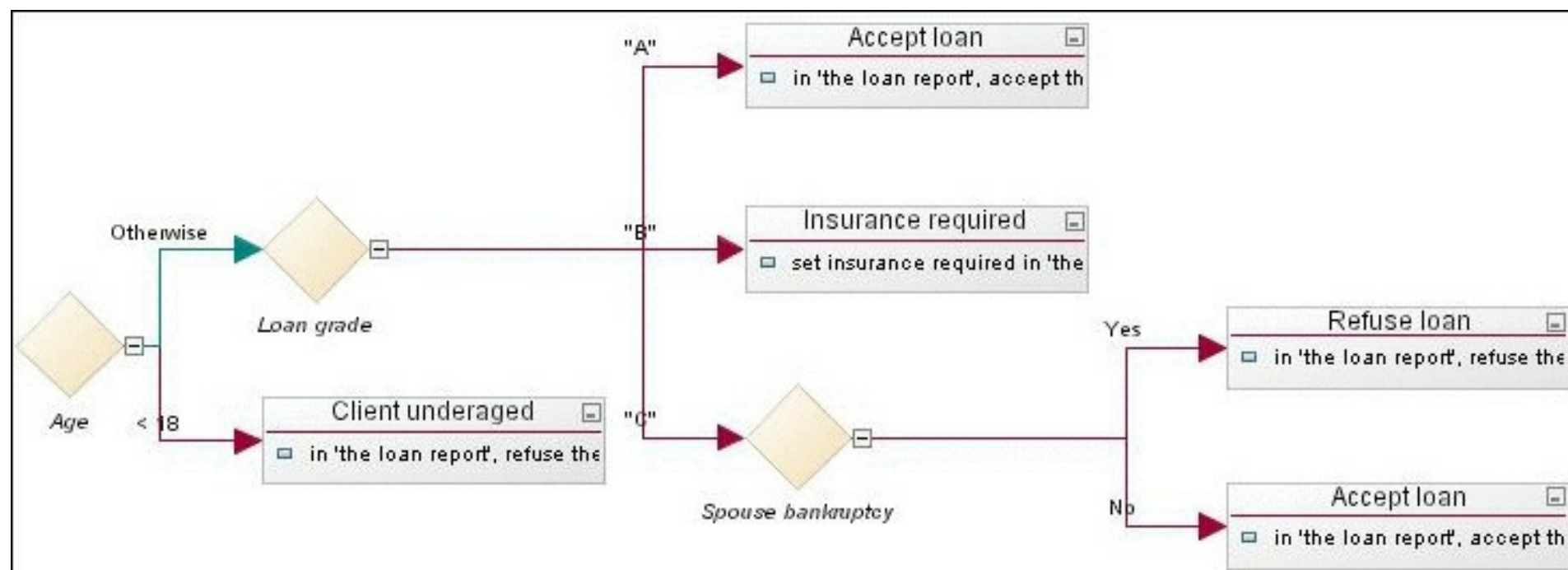
[Defining preconditions](#)

## (Deprecated) Decision tree overview

In a decision tree, conditions are shown as nodes, values as branch lines, and actions in boxes at the ends of branches.

Decision trees provide a way to view and manage large sets of business rules in diagrams.

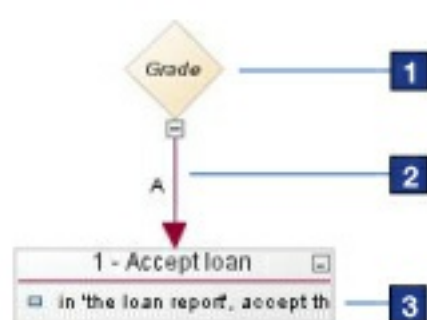
Decision trees make the interaction of nonsymmetrical rules easier to understand. The path from the first condition to the end of the actions along any branch represents one rule.



Looking at the following figure, you can see why decision trees are easy to understand:

- A condition is declared in its diamond-shaped node **1**.
- The possible values for the condition are represented by branches **2**.
- The actions are declared at the end of each branch **3**.

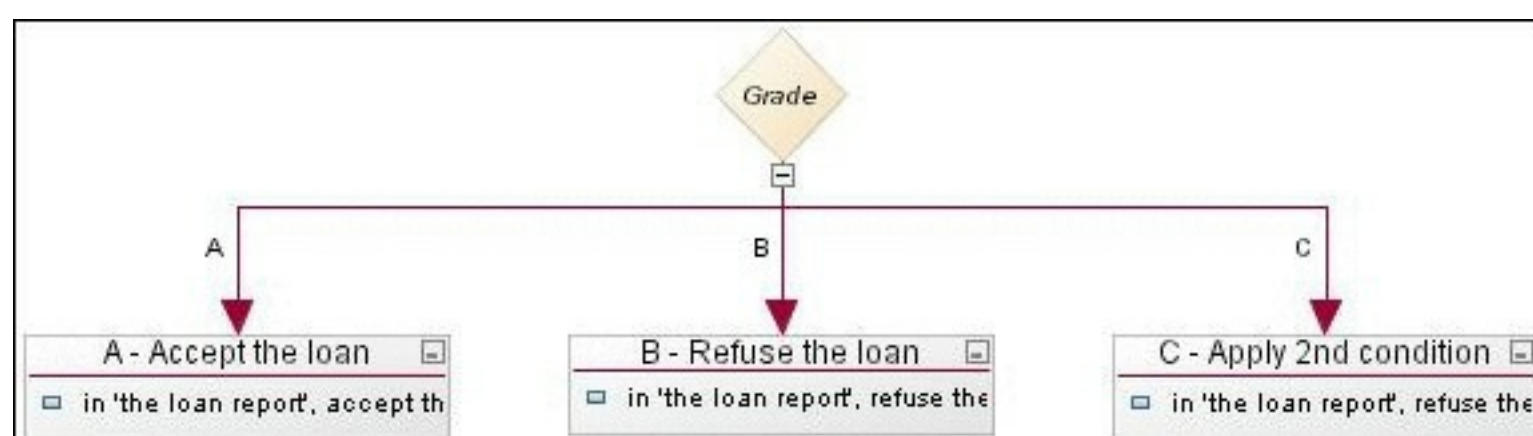
This simple decision tree corresponds to the following rule:



**if**  
the grade in the loan report is 'A'  
**then**  
in the loan report, accept the loan with the message "Loan accepted"

By adding branches, you add new rules that have different values for the condition.

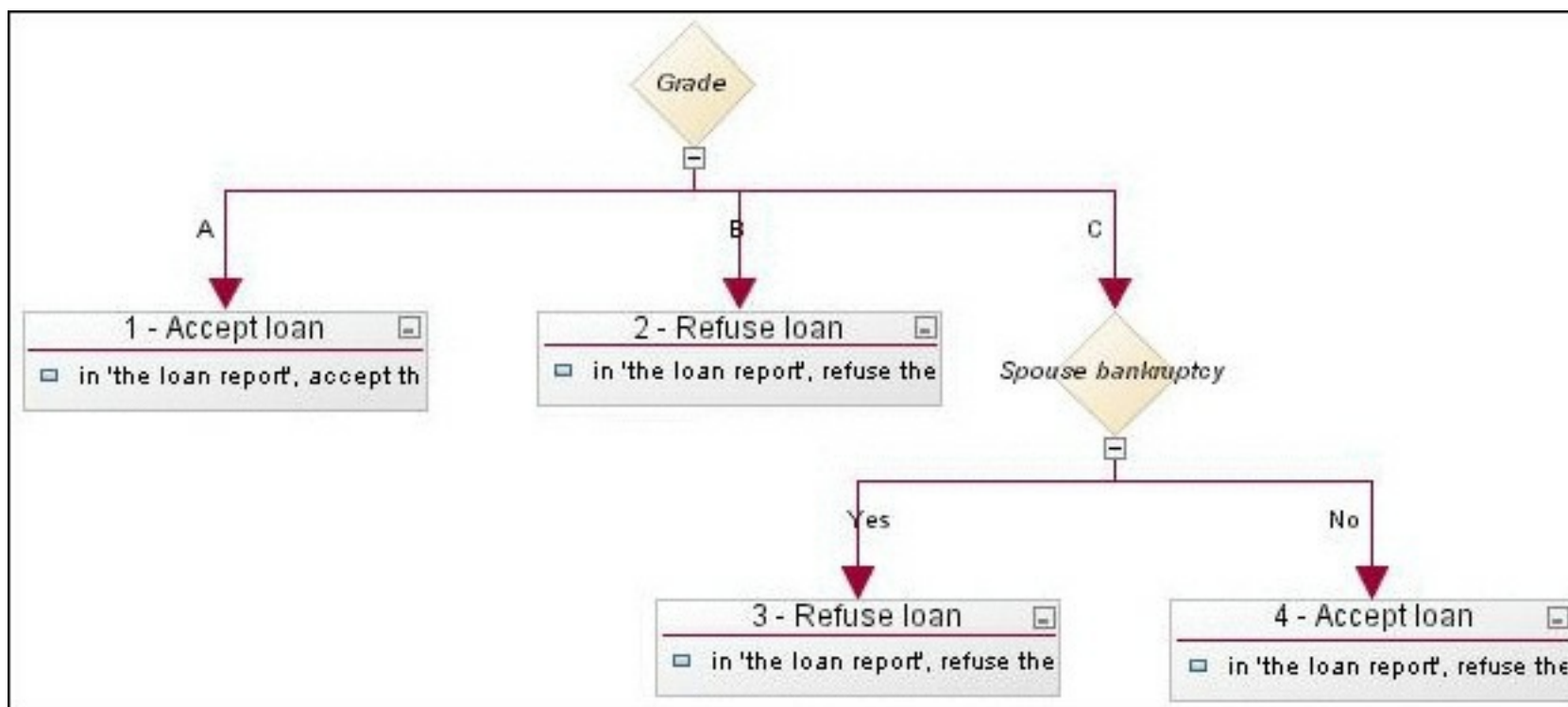
For example, the following decision tree forms three rules for a loan application (A, B, and C):



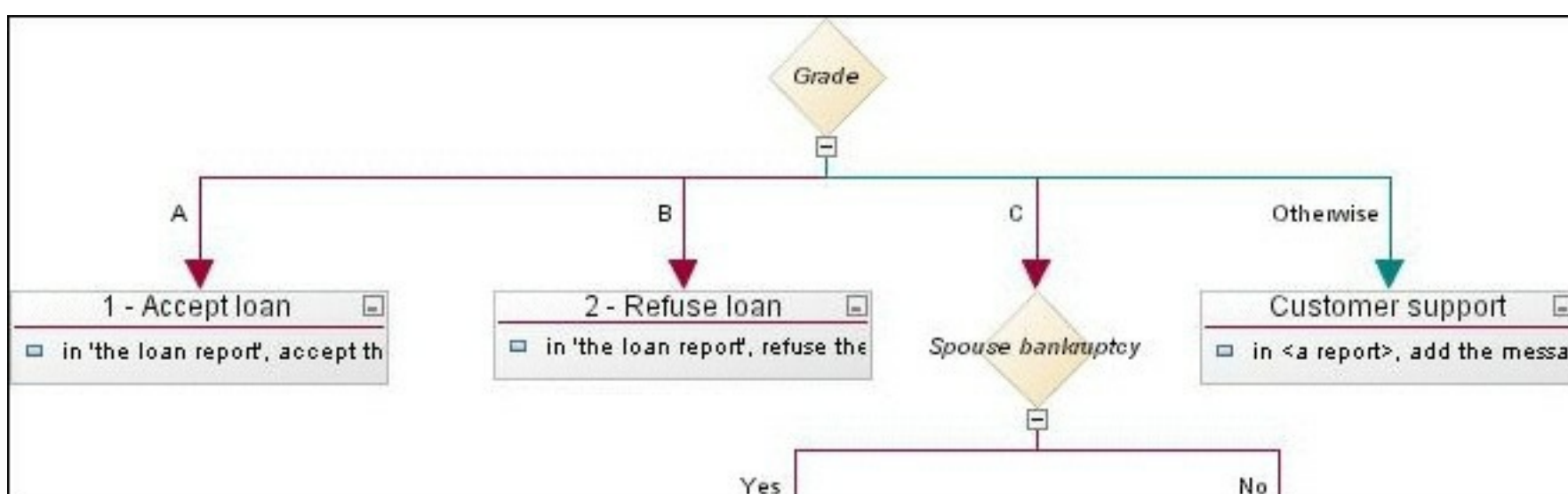
You can put as many actions as you want at the end of a branch, and add another condition to a branch.

For example, rules 1 and 2 in the following decision tree have only one condition (Grade), while rules 3 and 4

have a second condition to check (Spouse bankruptcy) before they do the actions:



Finally, you can add an Otherwise branch for condition values that are not covered by any of the other branches:



You can lay out your decision tree vertically or horizontally for optimal viewing, and set or remove consistency checking.

## Preconditions

You can set the following elements in a precondition section of a decision tree:

- Variables that can be used in the decision tree.
- Condition that is applied to an entire decision tree.

If the precondition is not satisfied, none of the rules in the decision tree can be evaluated.

For example, you can apply the following precondition to a decision tree:

### definitions

set 'wealthy customer' to a customer  
where the average monthly balance of this customer  
is more than \$1,000,000

### if

the state of residence of 'wealthy customer' is NY

Applying this precondition to a decision tree limits the scope of the decision tree rules to only those customers who have an average monthly balance of \$1,000,000 and live in New York. You can also use the variable `wealthy customer` in the tree.

Preconditions are tested before each rule in a tree is run.

**Parent topic:** [Decision Center basics](#)

## **(Deprecated) Templates**

A template is a partially written action rule or decision table that is used as a starting point to create multiple rules or decision tables with similar content and structure.

By defining it as such in the template, parts of rules that are created from a template can be frozen so that they cannot be edited.

Templates simplify the task of writing rules and decision tables, and are useful for:

- Creating action rules and decision tables that are already partially written.
- Restricting users from editing certain parts of partially completed rules.

A template applies only at the moment you create a rule or a decision table. If you subsequently modify the template, this will have no impact on rules or decision tables previously created from this template. You would need to update these rules or decision tables manually.

**Parent topic:** [Decision Center basics](#)



# Project element properties

A property provides additional information about a project element.

Most properties are useful to help you manage your business rules. For example, the query mechanism searches for project elements that match given criteria based on properties. You can also select which properties you want to display in your tables: click **Options** in the top banner and then select **Edit Display Options**.

The properties of a project element appear on the **Details** page. You can edit some properties in the **Properties** step of the Compose wizard (see [Compose: Create project elements](#)). You cannot edit properties that Decision Center automatically generates, such as the **Created By** property.

Although rule projects all contain certain properties, you can customize (extend) rule projects to capture information specific to the business domain of your company. The tables in the following sections describe properties marked as *extended properties*. These extended properties might not appear in your specific rule projects.

## Management properties

The following table describes properties that are useful for managing and authoring your project elements. You can normally edit them.

Property	Used to...
Categories	<p>Choose the categories of business terms that the rule displays in its drop-down lists when editing their content in the rule editor.</p> <p>By default, rules belong to the Any category, which means that they have all the available business terms of the project available in the drop-down lists.</p> <p>You can replace the Any category with another one to limit the number of entries in the drop-down lists. For example, you can limit the list to include only those entries that concern the Sales department. In this case, the Sales category must already be available next to the Categories property.</p>
Effective Date ( <i>extended property</i> )	<p>Specify the date on which the rule comes into effect. This property is normally used by a smart folder or a query to identify rules that come into effect.</p> <div><b>Note:</b> This is an extended property that might not be implemented in your application. Contact your development team for more information.</div>
Expiration Date ( <i>extended property</i> )	<p>Specify the date on which the rule expires. This property is normally used by a smart folder or a query to identify rules that become out of date.</p> <div><b>Note:</b> This is an extended property that might not be implemented in your application. Contact your development team for more information.</div>
Folder (Location)	<p>Set the location of the project element. Decision Center proposes separate folder hierarchies for business rules, resources, and templates (see <a href="#">Folders</a>).</p> <div><b>Note:</b> Folders correspond to rule packages in the developer environments.</div>
Group	<p>Associate the rule with the user's group. If the user belongs to more than one group, this property will be set to the first group of the user unless you specify otherwise.</p> <p>The Group property is used by the Administrator to set permissions.</p>
Name	Give a name to the project element.
Locale	Specify the language of the text displayed in the rules. The default language, <b>en_US</b> , corresponds to English.
Status ( <i>extended</i> )	Set the status of a rule, for example:



<i>property</i> )	<ul style="list-style-type: none"><li>• new</li><li>• defined</li><li>• validated</li><li>• rejected</li><li>• deployable</li></ul>
----------------------	-------------------------------------------------------------------------------------------------------------------------------------

The following table describes properties that are defined by Decision Center. You can display them but you cannot edit them.

Property	Used to...
Created By	See who created the project element.
Created On	See when the project element was created.
Last Changed By	See who last modified the project element.
Last Changed On	See when the project element was last modified.
Project	See to which project the element belongs. This property is useful when dealing with dependent projects.
Template	See from which template the rule was created. (Templates are deprecated.)
Type	See the type of project element: for example, folder, action rule, decision table.

### Execution properties

The following table describes the properties related to executing rules.

Property	Used to...
Active (extended property)	<p>Set whether or not the rule is active. You can deactivate a rule by deselecting this property.</p> <p><b>CAUTION:</b> When using this property, make sure that the extractor being used to generate the ruleset has been set up to extract active rules.</p>
Main Subflow Task	Identify the main ruleflow of the project. A project can contain many ruleflows but only one main ruleflow. You identify a main ruleflow by setting its Main Subflow Task property to true.
Priority (extended property)	<p>Set the priority of the rule. You use the Priority property to establish precedence between different rules.</p> <p>By default, the Priority property is set to 0. If you want a rule to be considered before another, set the priority of the first rule to a higher number than the second. To allow for any future rules that may be created, leave a gap between entries. For example: -100, 0, 100, 200.</p> <p>Rules with the same priority are processed according to the order in which they are added to the rule engine agenda. Rules more recently inserted into the agenda are chosen before those less recently inserted. For rules inserted at the same time, the order in which the rules appear in the agenda corresponds to the order in which the rules appear in the ruleset source file.</p> <div><p><b>Attention:</b></p><p>To enable accurate synchronization with Rule Designer, enter the priority value as an integer. If you do not, synchronization sets the priority to the default priority value of 0.</p></div>

### Technical properties

The following table describes the properties specific to technical rules, functions, and ruleflows:

Property	Used in...
Imports	

	Functions, technical rules, and ruleflows. Imports correspond to the classes used in the rule.
Return Type	Functions.
Initial Actions	Ruleflows.
Final Actions	Ruleflows.

**Parent topic:** [Decision Center basics](#)

**Related concepts:**  
[Overview: Ruleflows](#)  
[Functions](#)  
[Technical rules](#)

## Overview: Ruleflows

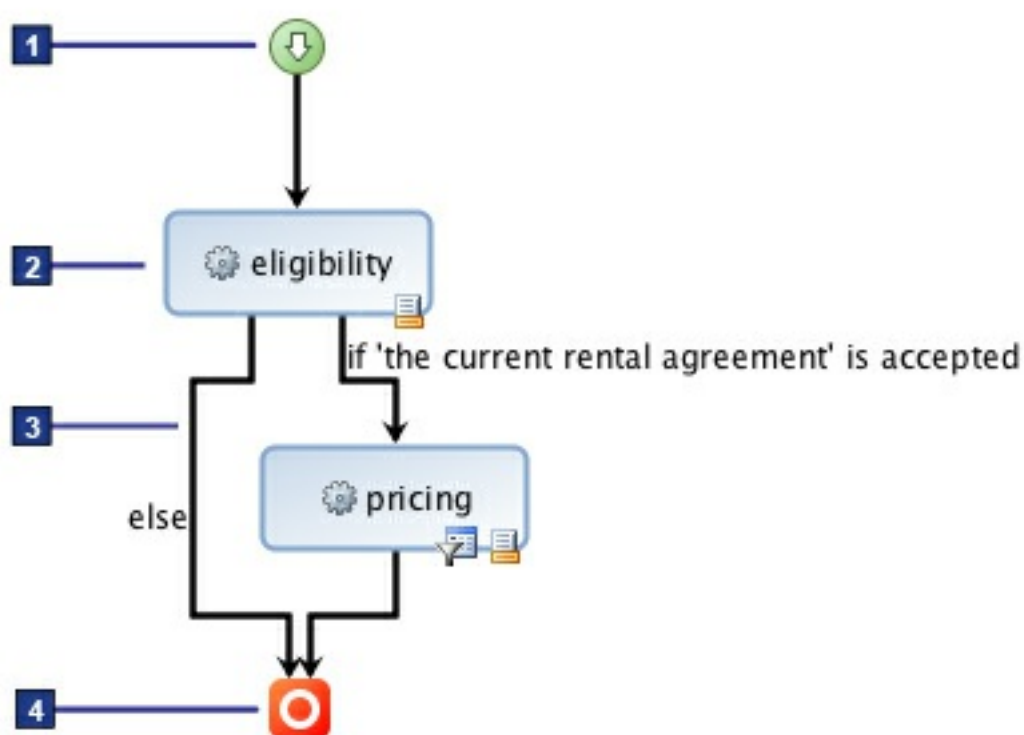
You can use different ruleflow elements to create your ruleflow in a project, such as start and end nodes, tasks, or transitions. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

You use ruleset parameters to transfer information between ruleflow tasks, and determine which path to follow through the transitions. For example, you can transfer a status variable, such as `isEligible`, to determine which task to go to next.

You can also use ruleset parameters to transfer information between the ruleflow and the application. The application obtains the results of ruleflow processing through ruleset parameters. For example, after the ruleflow ends, the parameter can return an aggregated report, diagnostics, a set of compliance exceptions, or computed values, such as prices, rates, or taxes.



The following diagram shows the main parts of a ruleflow:

- Start node **1**
- Task **2**
- Transition **3**
- End node **4**



**Note:** The appearance of ruleflows differs slightly across components. For example, the diagrams in this topic show ruleflows in Rule Designer. In the Business console, there are no icons below the rule task elements.

### Start nodes and end nodes


A start node  and an end node  are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

### Tasks

Between the start node and the end node, the ruleflow is made of tasks linked by transitions.

The tasks of a ruleflow contain the instructions for what to execute and in what order:

#### Rule task

A rule task  contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

- The execution mode of a rule task determines whether the rules are executed in the order shown (sequential mode), in order with some optimization at compile time (Fastpath mode), or in an order determined by the rule engine (RetePlus mode).
- The runtime rule selection filter determines which rules in a given rule task are executed. The rule selection filter is applied at run time and the rule engine evaluates only the rules that pass through the filter. You write a runtime rule selection filter like a condition in an action rule, except that in the filter


statement you can use only rules, rule properties, and any ruleset parameters defined for the rule project.

For example, you can write the following runtime rule selection filter:


`the author of 'the rule' is "Sally"`

With this filter, the rule engine evaluates only the rules written by Sally.

## Action task

An action task  contains rule action statements to be executed. You define the actions of an action task in the same way that you define the actions for business rules, by using BAL. Unlike the actions of business rules, however, action tasks can only use the action phrases associated with ruleset parameters. You can also define actions in IRL.

## Subflow task

A subflow task  references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the rule project. You can self-reference a ruleflow, but be careful to avoid infinite loops.

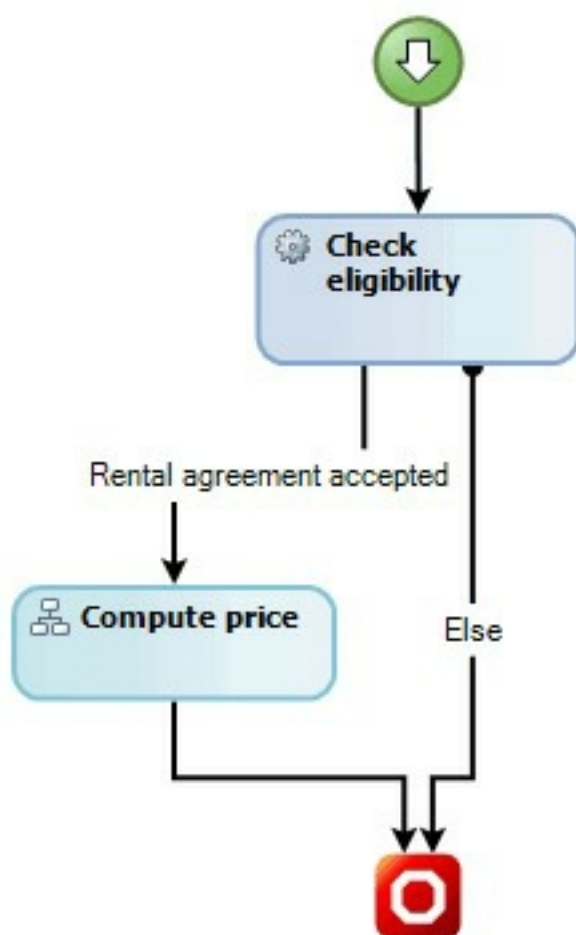
You can specify initial and final actions on subflow tasks.

## Transitions

Transitions connect tasks in a ruleflow and define the sequence of the ruleflow from one task to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in an action rule.

For example, with the following condition on the transition between the eligibility and pricing rule tasks, the pricing rule task can be performed only when the customer's rental agreement is accepted, otherwise the ruleflow ends.




A single transition from one task to another does not require a condition. However, when several transitions originate from a task, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

An Else transition specifies the path to take when a condition is not met. A task can be followed by several transitions but only one of them can be an Else transition. This means that all transitions except the Else transition must have conditions.


Whenever multiple transitions come from a choice, the transitions must have Boolean conditions that specify which path to choose during execution. The last remaining empty transition is automatically computed as an Else transition.


## Branches

A branch  is a node that organizes conditional transitions. For example, a branch for the age of a customer organizes transitions with conditions on a given age range. Several transitions can go to and from a branch.

Like transitions created from a task, all transitions created from a branch must have a condition, except the Else transition.

## Forks and joins

A fork  is a node that splits the execution flow into several parallel paths. The transitions created from a fork do not have conditions because all transitions created from the fork are followed.

A join  is a node that combines all the transitions created from a fork when the parallel paths are all completed.

## Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. You define initial and final actions in the same way as you define actions for an action task.

A task execution sequence consists of executing its initial actions, then its body, and then its final actions.

Initial and final actions are not mandatory and you can use them independently of each other.

You can also specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

**Parent topic:** [Decision Center basics](#)

## Rule verification

You can perform different levels of verification on your rules before you deploy them to the rule-based computer applications on which they act.

### Basic syntax checks

Decision Center verifies the syntax of any business rule that you edit.

### Decision table and decision tree checks

You can check decision tables and decision trees for gaps and overlap errors. You can also test decision tables for symmetry.

**Parent topic:** [Decision Center basics](#)

## Basic syntax checks

Decision Center verifies the syntax of any business rule that you edit.

When you save the rule, the **Details** page displays any errors or warnings resulting from the syntax checks. Any rule that contains any syntax error is excluded when the Configuration Manager generates a ruleset.

Decision Center checks for the following syntax errors:

- A rule statement uses a business term that is not recognized or no longer valid.
- A rule statement is ambiguous, that is, more than one correct interpretation exists. In this case, use parentheses to clarify the syntax.
- A variable in the definitions part is already declared or it is of the wrong type.
- A rule is incomplete, that is, some of the placeholders have not been filled in.
- A rule has an else part with no if part.

For a more exhaustive list of the types of checks available, see the **Rule Designer** online help.

**Parent topic:** [Rule verification](#)

**Related tasks:**

[Editing on the Explore tab](#)

## Decision table and decision tree checks

You can check decision tables and decision trees for gaps and overlap errors. You can also test decision tables for symmetry.

### Overlap checks

You can set Decision Center to check that the values you enter in your decision tables or decision trees do not overlap or are not identical.

For example, in the following condition statements, customers who are 29 years old satisfy the condition for both rules:

- Age is between 17 and 30
- Age is between 29 and 40

If these two conditions stem from the same condition node of a decision tree, or are part of the same column or within a partitioned groups of cells in a decision table, Decision Center signals it as an overlap error.

### Gap checks

You can set Decision Center to check that the values you enter in your decision tables or decision trees consider all possible cases. This ensures that there are no gaps in your conditions.

For example, in the following condition statements, customers who are 31 years old are never taken into account:

- Age is between 17 and 30
- Age is between 32 and 40

Decision Center reports a gap error.

In decision tables, Decision Center evaluates gaps for all the entries in a given column, or within partitioned groups of cells. In decision trees, Decision Center evaluates them for the branches of a condition node or for the entire tree.

### Symmetry checks

You can set Decision Center to check that all partitions of a decision table use the same set of items. You can verify symmetry for the whole table or for specified columns. Such analysis is useful to enforce that you have covered all choices in a decision table.

By default, symmetry analysis is off, allowing you to create nonsymmetrical tables.

**Parent topic:** [Rule verification](#)

**Related concepts:**

[\(Deprecated\) Decision tree overview](#)

**Related information:**

[Consistency checks in decision tables](#)

[Consistency checking in decision trees](#)

[Decision tables](#)



## Variable sets

Variable sets contain variables that you can use across the different elements of a rule project. You can also use these variables to pass data between tasks in a ruleflow.

Variable sets have a name and folder that you use to manage them.

Each variable in a variable set has the following components:

- A name to identify it
- An initial value
- A verbalization
- A BOM type

You can access variables from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors.

**Note:**

Variable sets are an advanced subject normally reserved for developers. For more information, refer to the Rule Designer documentation.

**Parent topic:** [Decision Center basics](#)

**Related concepts:**

[Rule projects](#)

[Overview: Ruleflows](#)

**Related tasks:**

[Creating a variable set](#)

## Functions

A function is a procedural item that is written in IRL. It is designed to share procedure code between the elements of a ruleset. A function is composed of a header and a statement part.

A function can be created in a rule project to share code procedures across more than one element of a ruleset. A function is expressed in ILOG Rule Language (IRL) and its code is evaluated when the ruleset runs.

**Note:** Functions are an advanced subject that is normally reserved for developers. See the Rule Designer documentation for more information.

**Note:** Rule Designer does not refactor functions. If you rename, move, or delete a function, the references in other rule project artifacts are not updated.

A function can be called either from the action part of a rule, from another function, or from an action task of a ruleflow.

You can add as many functions to the rule project as you like. The code that you write in a function in Rule Designer is mapped to the IRL `function` keyword.

Functions can be used to set up the working memory when you test ruleset execution.

The function is composed of a header and a statement part.

The header contains the function return type and its signature. A function signature consists of the function name and an argument list. This list can be either empty or composed of pairs that contain the argument type and name, which are separated by commas.

```
function returnType functionName (argumentList)
{
 code
}
```

Like the action part of a rule, the statement part can contain any action statements, plus a return statement.

You use `return` to exit from the current function. The flow of control returns to the statement that follows the original function call. The return statement has two forms: one that returns a value and one that does not. To return a value, put the value (or an expression that calculates the value) after the return keyword:

```
return ++count;
```

The data type of the value that is returned by `return` must match the type of the function that is declared as the return value. When a function is declared void, use the form of return that does not return a value:

```
return;
```

**Note:** A function is not required to declare in its `throws` statement any subclasses of `IlrRuntimeException` that might be thrown during the execution of the function.

**Parent topic:** [Decision Center basics](#)

## Technical rules

Technical rules are written using the ILOG Rule Language (IRL). IRL is a Java-like rule language that can be executed directly by the rule engine.

**Note:** Technical rules are an advanced subject normally reserved for developers. See the Rule Designer documentation for more information.

A technical rule is made of a condition part and an action part.

- The **condition** part, which begins with the keyword `when`, binds variables to objects and attribute values, and specifies tests on attribute values. This provides a filtering mechanism for objects.

Examples of IRL condition keywords are:

- `collect`
  - `evaluate`
  - `exists`
  - `from`
  - `in`
  - `instanceof`
  - `not`
  - `where`
- The **action** part, which begins with the keyword `then`, specifies the actions to be carried out if the rule is executed. It includes an optional second part which begins with the keyword `else`, that applies only if the last evaluated statement in the condition part is false.

Examples of IRL action keywords are:

- `break`
- `catch`
- `continue`
- `foreach`
- `modify`
- `retract`
- `try`
- `update`
- `throw`
- `while`

**Parent topic:** [Decision Center basics](#)

## Explore: Navigate your projects

The Explore tab provides a number of features that you can use to navigate within your rule projects.

### [Displaying project elements](#)

The Explore tab displays the smart folders and the project elements corresponding to these smart folders.

### [Displaying project element details](#)

You can display the details of any project element provided you have the relevant user permissions to do so.

### [Editing a project element](#)

You can edit a project element in a number of ways. Editing automatically locks the element so that others cannot edit it at the same time. You can also explicitly lock elements.

### [Deleting a project element](#)

You can delete an element from a project.

### [Copying a project element](#)

You can create copies of project elements.

### [Moving a folder or rule](#)

You can move project elements.

### [Locking project elements](#)

When you edit a project element, Decision Center automatically locks it so that others cannot edit it at the same time.

### [Exploring the history of an element](#)

Decision Center keeps a history of all the changes you make to any project element.

**Parent topic:** [Working with the Enterprise console](#)

## Displaying project elements

The Explore tab displays the smart folders and the project elements corresponding to these smart folders.

### About this task

You use the **Explore** tab to navigate through your smart folders and select what you want to work on.

#### Note:

The default smart folders display most of the project elements you need to work on. However, some more technical project elements, such as resources, variable sets, technical rules, and functions are not displayed in any of the default smart folders. You must create a new smart folder to display these types of project elements.

### Procedure

To display a project element:

1. Click the **Explore** tab.
2. Select the appropriate smart folder to display its elements in the table.

**Parent topic:** [Explore: Navigate your projects](#)

#### Related concepts:

[The Explore tab](#)  
[Smart folders](#)

## Displaying project element details

You can display the details of any project element provided you have the relevant user permissions to do so.

### About this task

The **Details** page for a project element contains the following information:

- Contents
- Properties
- Documentation
- Any attached items, such as tags, requirements, initial values, and overridden rules

### Procedure

To display project element details:

1. On the **Explore** tab, select the project element in the table.
2. Click **Details** in the toolbar or click the project element name in the table.

### Results

#### Important:

For smart folders, hover your mouse over the smart folder and click the Details icon  to access the Details page.

The **Details** page displays the information on the chosen element.

You can return to the main page of the **Explore** tab by clicking on the tab name.

**Parent topic:** [Explore: Navigate your projects](#)

#### Related concepts:

[Rule projects](#)

[Project element properties](#)

#### Related information:

[The Enterprise console environment](#)

## Editing a project element

You can edit a project element in a number of ways. Editing automatically locks the element so that others cannot edit it at the same time. You can also explicitly lock elements.

### Editing on the Compose tab

Use the Compose tab for editing if you want to access all the properties and documentation of an element or make minor changes to its contents.

### Editing on the Explore tab

Use the Explore tab for editing if you want to change the name or status of an element or make minor changes to its contents.

**Parent topic:** [Explore: Navigate your projects](#)

## Editing on the Compose tab

Use the Compose tab for editing if you want to access all the properties and documentation of an element or make minor changes to its contents.

### About this task

You can use this method to edit any project element as long as you have permission to do so.

When you edit a project element, the **Compose** tab displays with the steps of the wizard filled in with the current properties, content, and version information of the element.

### Procedure

To edit a project element on the Compose tab:

1. Select the check box for the element in the table on the **Explore** tab and click **Edit** in the toolbar above the table.

#### Note:

For smart folders, display its details first (see [Displaying project element details](#)) and then click **Edit** in the **Details** page toolbar.

The **Compose** tab displays the existing contents and properties for the element.

2. Click **Next** or **Previous** to step forwards or backwards through the wizard and edit the different steps as required.
3. Click **Finish** to save your changes.

If you click **Cancel**, none of your changes are saved.

**Parent topic:** [Editing a project element](#)

#### Related tasks:

[Selecting your preferred rule editor](#)  
[Locking project elements](#)

#### Related information:

[Compose: Create project elements](#)



## Editing on the Explore tab


Use the Explore tab for editing if you want to change the name or status of an element or make minor changes to its contents.

### About this task

You can edit any type of element that is listed in the table on the **Explore** tab, except for ruleflows.

### Procedure

To edit a project element on the Explore tab:

1. Click **Edit this element**  next to the name of the project element in the table.  
The element contents and some properties display in the editing area under the table.
2. Edit the properties or contents of the element in the editing area.

#### Note:

Use the **Add a comment to this version** field to summarize the changes you have made.

3. Click **Save** to save your changes.

**Parent topic:** [Editing a project element](#)

### Related tasks:

[Selecting your preferred rule editor](#)

# Deleting a project element

You can delete an element from a project.

## About this task

You can delete any project element as long as it is not locked by someone else and you have permission to delete project elements.

## Smart folders

Deleting a smart folder does not delete any of the folders or rules it displays. You can delete a smart folder even if it displays locked elements.


## Folders

Deleting a folder also deletes all the rules and any subfolders it contains. However, you cannot delete a folder if another user has locked anything contained in it.

Before you delete a folder, verify that the smart folder that you are using to display the folder has been set up to display all its content. This is to make sure that you do not delete any rules not visible in that view.

## Procedure

To delete a project element:

1. On the **Explore** tab, select the project element in the table, or access the **Details** page for smart folders (see [Displaying project element details](#)).
2. Click the **Delete** icon .

## Results

Project elements that you delete go to the recycle bin. You can restore deleted project elements from the recycle bin, or from any baseline that contains that project element.

**Parent topic:** [Explore: Navigate your projects](#)

## Related concepts:

[Smart folders](#)  
[Folders](#)

## Related tasks:

[Locking project elements](#)  
[Baselines](#)

# Copying a project element

You can create copies of project elements.

## About this task

You can copy any project element as long as you have permission to create those types of element. You can copy a project element to the current project or to another project, and to a folder of the same type (see [Folders](#)).

### Note:

Smart folders can only be copied to the root of the current project.

If you choose the current folder location, your copy keeps the same name but prefixed with **Copy of**. If you choose another location, the copy keeps the same name as the original.


## Procedure

To copy a project element:

1. On the **Explore** tab, select the project element you want to copy.

### Note:

For smart folders, display its details instead (see [Displaying project element details](#)).

2. Click the **Copy** icon  in the toolbar.
3. Select the project and folder to which you want to copy the project element.
4. Click **OK**.

**Parent topic:** [Explore: Navigate your projects](#)

### Related concepts:

[Rule projects](#)  
[Folders](#)

### Related tasks:

[Locking project elements](#)

### Related information:

[The Enterprise console environment](#)

## Moving a folder or rule

You can move project elements.

### About this task

You can move a project element to another folder of the same type, or move a folder to another location.

### Procedure

To move a folder or project element:

1. In the **Explore** tab, select the project element or folder you want to move and then click **Edit**.
2. In the **Properties** step of the wizard, click the folder icon next to one of the following fields and then use the drop-down list to set the destination folder:
  - Click the **Location** field, to move a folder.
  - Click the **Folder** field, to move a project element.
3. Enter the version information and then click **Finish** to save the change.

**Parent topic:** [Explore: Navigate your projects](#)

### Related concepts:

[Rule projects](#)

[Folders](#)

[Project element properties](#)


### Related tasks:

[Editing on the Compose tab](#)

# Locking project elements



When you edit a project element, Decision Center automatically locks it so that others cannot edit it at the same time.

## About this task

When people work concurrently on the same project, it is important to make sure that only one person at a time edits a given project element. When an element is locked, other users can open the element and view it, but they cannot edit it until you have saved your changes. A locked element displays the locked icon .

**Note:**

Hold the mouse over the locked icon to find out who has locked the element.

You can also manually lock most project element types. For example, if you want to work on all the rules contained in a folder, you can lock the folder manually. When you lock a folder, you also lock all the project elements it contains so that only you can edit them. A manually locked element displays the key icon  to you and the locked icon  to others.

**Note:**

The Administrator can release any locks by clicking **Release Lock** in the toolbar.

When you lock an element manually, you can also specify the following options to subfolders and sessions:

O p t i o n	Description
S u b f o l d e r s	<p>When you edit a folder, it locks automatically, along with all the project elements it contains. However, other users can still edit any subfolders unless, when you lock the folder, you select the option <b>Apply on subfolders</b>.</p> <p>To release all the locks set with this option, you only need to release the lock on the folder itself, and the locks on the subfolders will also be released.</p>
S e s s i o n	<p>By default, any lock (automatic or manual) is released when you end your working session (see <a href="#">Signing in to the Enterprise console</a>). To retain the lock, select the <b>Keep lock after the session closes</b> option when locking.</p> <p>If you select this option, the element remains locked until you unlock it, regardless of how many times you sign in and out.</p>

## Procedure

To lock an element manually:

1. On the **Explore** tab, select the project element. For folders, access the **Details** page (see [Displaying project element details](#)).
2. Click **Lock** in the toolbar.
3. On the **Lock** window, select the locking options for subfolders and session.
4. Click **OK**.

## Results

The locked element now displays the key icon, showing you have locked the resource, and other users see a lock icon with the element.

When you complete your work, you can unlock the folder manually by proceeding as above and clicking **Unlock**. There is no confirmation message.

**Parent topic:** [Explore: Navigate your projects](#)

**Related concepts:**

[Rule projects](#)  
[Folders](#)

**Related information:**

[Editing a project element](#)

# Exploring the history of an element

Decision Center keeps a history of all the changes you make to any project element.

## About this task

The change history displays a table listing all the versions of the element, including the current one, and shows the following details:

- Who created or changed the element, and when.
- Any comment entered as version information (see [Version information](#)).
- Which baselines, if any, contain the element.

**Note:**

The history retains a complete history of all elements. You cannot clean out previous versions, nor edit them.

From the history page you can explore the details of a version, restore a version, or copy a previous version. You can also compare the changes between two versions by displaying the **Compare 2 Versions** page:

Differences between versions 1.0 and 1.1			
Type	Property	Old Value	New Value
update	Name	insurance	My Decision Table
update	Content	...	...

The table displays the changes between the two versions you selected. Each row represents a change made to an element, and displays the following columns:

Type of change	Description
Type	The type of change made: create, update, delete.
Property	The property that changed.
Old Value	The value that was changed.
New Value	The replacement value.

If the change was made on the definition of the project element itself, the **Old Value** and **New Value** fields are empty. To display the changes in table form, click the hyperlink in the row.

The following image shows the differences between two versions of a decision table:

8		≥ 600,000		true	0.0075	9		< 100,000		true	0.0035
9		< 100,000		true	0.0035	10		100,000	300,000	true	0.006
10		100,000	300,000	true	0.006	11	C	300,000	600,000	true	0.0085
11		300,000	600,000	true	0.0085	12		600,000	800,000	true	0.012
12		≥ 600,000		true	0.0145	13		≥ 800,000		true	0.0145

## Procedure

To explore the history of a project element:

1. Select the project element in the **Explore** tab. For smart folders, access the **Details** page instead (see [Displaying project element details](#)).
2. Click **History** in the toolbar.

## Results

The **History** page opens and displays all previous versions of the selected project element.

From the history page you can explore the details of a version by selecting it and clicking **Explore Version Details** in the toolbar.

**Note:**

Alternatively, click the version number in the history table.

You can also select the two versions you want to compare by clicking the check boxes in the rows corresponding to the required versions and clicking **Compare 2 Versions** in the toolbar.

**Parent topic:** [Explore: Navigate your projects](#)

**Related tasks:**

[Versioning](#)  
[Baselines](#)  
[Displaying project element details](#)

# Compose: Create project elements

You use the Compose tab in the Enterprise console to create project elements.

## [Creating a smart folder](#)

You create smart folders in the **Compose** tab.

## [Creating a folder](#)

You create folders to organize your project elements.

## [Creating business rules](#)

You can create business rules, that is, action rules, decision tables, or decision trees.

## [\(Deprecated\) Creating a rule template](#)

You use a rule template as a starting point for creating a rule.

## [\(Deprecated\) Creating a decision table template](#)

You use a decision table template as a starting point for creating a decision table.

## [Creating a variable set](#)

Variable sets contain variables that you can use across the different elements of a branch.

## [Creating a function](#)

You create functions from the Compose tab.

## [Creating a technical rule](#)

You create technical rules from the Compose tab.

## [Creating a resource](#)

You create resources from the Compose tab.

## [Tags](#)

You use tags to store data with your project elements.

## [Rule overriding](#)

You use rule overriding to give rules precedence over other rules.

## [Project element documentation](#)

You can add project element documentation when you create or edit project elements.

## [Version information](#)

Version information is associated with the version to which you add it.

**Parent topic:** [Working with the Enterprise console](#)



## Creating a smart folder

You create smart folders in the **Compose** tab.

### About this task

You use smart folders to navigate through your projects according to criteria that you select.

You can change the order in which your smart folders are displayed on the **Explore** tab by clicking **Options** in the top banner.

### Procedure

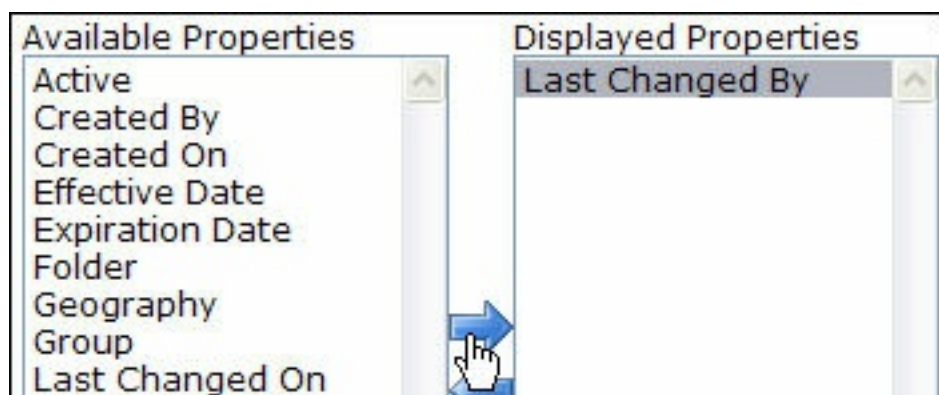
To create a smart folder:

1. On the **Compose** tab, select **Smart Folder** as the type of project element to create and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name of your smart folder and then click **Next**.
3. In **Step 2: Query**, specify how the smart folder searches through the project.

For information about queries, refer to [Query: Search your projects](#).

4. In **Step 3: Displayed Properties**, specify how you want the smart folder to display the elements that it finds in its query.

To select properties, in the **Available Properties** window, select the properties that you want displayed and then, using the arrow between the two properties windows, move the required properties to the **Displayed Properties** window.



#### Note:

Each property you move to the **Displayed Properties** window adds a level to the display.

5. Use the up and down arrows next to the **Displayed Properties** window to change the display order.
6. In **Step 4: Documentation**, document your smart folder.
7. In **Step 5: Version information**, enter the version information.
8. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

## Creating a folder

You create folders to organize your project elements.

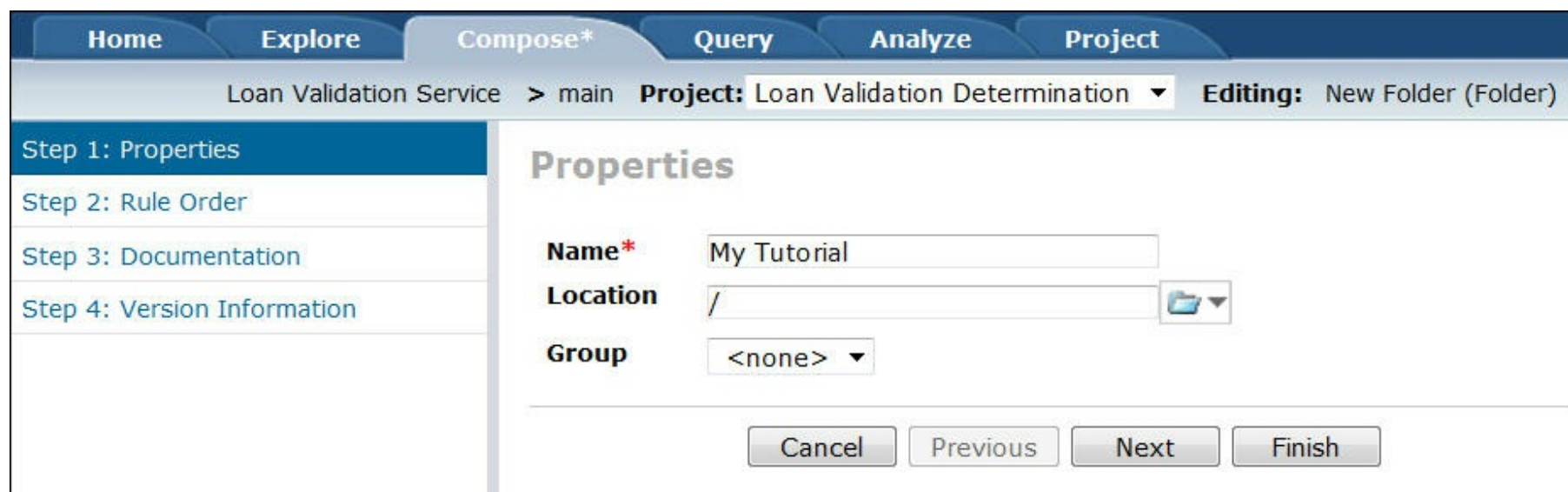
### About this task

You can create folders in the **Compose** tab.

### Procedure

To create a folder:

1. On the **Compose** tab, select **Folder** as the type of project element to create, choose the correct folder hierarchy, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name, location, and other properties of your folder and then click **Next**.

The screenshot shows the 'Compose\*' tab in a software interface. The breadcrumb trail is 'Loan Validation Service > main'. The 'Project' dropdown is set to 'Loan Validation Determination' and the 'Editing' dropdown is set to 'New Folder (Folder)'. On the left, a sidebar lists four steps: 'Step 1: Properties' (selected), 'Step 2: Rule Order', 'Step 3: Documentation', and 'Step 4: Version Information'. The main area is titled 'Properties' and contains three fields: 'Name\*' with the value 'My Tutorial', 'Location' with the value '/', and 'Group' with a dropdown menu showing '<none>'. At the bottom right, there are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

3. In **Step 2: Rule Order**, specify the order in which the business rules in the folder will be treated.

When you first create a new folder it contains no rules, so you can skip this step. When you have added some rules to the folder you can edit the folder to set the rule order. Refer to [Editing on the Compose tab](#).

The rule order is effective only in the literal ordering. In literal ordering mode, the rule order at folder level only has an impact if the folder itself is added to the task, and not when the rules are added.

Ordering applies only between rules in the same folder. If you add several folders to the same rule task, you must set their relative order in the ruleflow.

#### Note:

This step is only pertinent when creating folders of type Business Rule (see [Folders](#))

4. In the **Documentation** step, document your folder.
5. In the **Version information** step, enter the version information.
6. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

#### Related concepts:

[Folders](#)

[Project element properties](#)

[Overview: Ruleflows](#)

[Version information](#)

[Project element documentation](#)

# Creating business rules

You can create business rules, that is, action rules, decision tables, or decision trees.

## About this task

You create business rules in the **Compose** tab using the wizard.

## Procedure

To create a business rule:

1. On the **Compose** tab, select the type of project element to create and then click **OK**.

You can select action rule, decision table, or decision tree. For action rules and decision tables, you can also start from an existing template under **Start from a template**, if some are available.

2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your business rule and then click **Next**.
3. In **Step 2: Contents**, write the contents of the action rule, decision table, or decision tree.

To write rules, you must be familiar with both the structure of action rules (see [Action rules](#)) and how to use the different rule editors (see [Working with the editors](#)).

4. In **Step 3: Tags**, add any tags you want to associate to the rule.
5. In **Step 4: Override Rules**, declare any rules to override.
6. In **Step 5: Documentation**, document your business rule.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

## Related concepts:

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

[Tags](#)

[Rule overriding](#)

[Columns](#)

[Rows and cells](#)

[Preconditions](#)

# (Deprecated) Creating a rule template

You use a rule template as a starting point for creating a rule.

## About this task

Existing templates are available on the **Explore** tab in the **Templates** smart folder.

To use a template as a starting point for creating a rule, select the required template under **Start from a template** on the **Compose** tab.

You can create new rule templates if you have user permissions to do so. You create a new rule template in a similar way to creating a new rule, except for the following:

- You do not have to complete the rule statements. Instead, because the template is the starting point for creating similar rules, build the rule statements as you want them to appear.
- You can freeze any part of the rule statements so that they cannot be modified when someone uses the template to create their rules.
- You can set the properties that you want rules created using your template to have.

Any modifications to an existing template have no impact on rules previously created from this template. The modifications apply only when you use the template to create a new rule.

## Procedure

To create a rule template:

1. On the **Compose** tab, select the template as the type of project element to create, specify **Action Rule** as the rule type for this template in the drop-down list, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your template and then click **Next**.
3. In **Step 2: Content**, write the contents of an action rule as you want it to appear when using your template.

To freeze parts of a rule, right-click the part of the rule statement that you want to freeze and then select **Freeze** from the drop-down list.

To unfreeze a frozen part of a rule, right-click the frozen statement and then select **Unfreeze**.

4. In **Step 3: Initial Values**, set the properties that you want the rules created using your template to have. You set properties by building the **Initial Values** table. For example:

Name	Value
priority	high
status	defined

**Note:**

You cannot freeze these properties.

5. In **Step 4: Documentation**, document your template.
6. In **Step 5: Version information**, enter the version information.
7. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

**Related concepts:**

[\(Deprecated\) Templates](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

## (Deprecated) Creating a decision table template

You use a decision table template as a starting point for creating a decision table.

### About this task

Existing templates are available on the **Explore** tab in the **Templates** smart folder.

To use a template, select the required template under **Start from a template** on the **Compose** tab.

You can create new decision table templates if you have user permissions to do so. You create a new decision table template in a similar way to creating a new decision table, except for the following:

- You do not have to complete the condition and action columns. Because the template is the starting point for creating similar decision tables, you build the templates only with the content that is common to all decision tables to be created using this template.
- You can set the properties that you want the decision tables created using your template to have.

Any modifications to an existing template have no impact on decision tables previously created from this template. The modifications apply only when you use the template to create a new decision table.

### Procedure

To create a decision table template:

1. On the **Compose** tab, select the template as the type of project element to create, specify **Decision Table** as the type for this template in the drop-down list, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your template and then click **Next**.
3. In **Step 2: Content**, write the contents of a decision table as you want it to appear when using your template.
4. In **Step 3: Initial Values**, set the properties that decision tables created using your template will have.
5. In **Step 4: Documentation**, document your template.
6. In **Step 5: Version information**, enter the version information.
7. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[\(Deprecated\) Templates](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)

# Creating a variable set

Variable sets contain variables that you can use across the different elements of a branch.

## About this task

Each variable in a variable set consists of a name to identify it, an initial value, a verbalization, and a BOM type. Variables are accessible from business rules only if you verbalize them, that is, if you specify the text to be displayed in the rule editors. You can create new variable sets only if you have user permissions to do so.

### Note:

You must create a smart folder that displays variable sets to see them in Decision Center.

## Procedure

To create a variable set:

1. On the **Compose** tab, select **Variable Set** as the type of project element to create and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your variable set and then click **Next**.
3. In **Step 2: Variables**, create the set of variables.
4. In **Step 3: Documentation**, document your variable set.
5. In **Step 4: Version information**, enter the version information.
6. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[Project element properties](#)

[Variable sets](#)

[Smart folders](#)

[Project element documentation](#)

[Version information](#)

# Creating a function

You create functions from the Compose tab.

## About this task

You can create new functions if you have permission to do so.

### Note:

You must create a smart folder that displays functions to see them in the Enterprise console.

## Procedure

To create a function:

1. On the **Compose** tab, select **Function** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your function and then click **Next**.
3. In **Step 2: Contents**, write the function.
4. In **Step 3: Tags**, add any tags you want to associate with the function (see [Tags](#)).
5. In **Step 4: Arguments**, declare the function arguments: the type, name, and order.
6. In **Step 5: Documentation**, document your function.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[Functions](#)

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)



# Creating a technical rule

You create technical rules from the Compose tab.

## About this task

You can create new technical rules if you have user permissions to do so.

### Note:

You must create a smart folder that displays technical rules to see them in the Enterprise console.

## Procedure

To create a technical rule:

1. On the **Compose** tab, select **Technical Rule** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, enter the name and other properties of your technical rule and then click **Next**.
3. In **Step 2: Content**, write the technical rule.
4. In **Step 3: Tags**, add any tags you want to associate with the rule (see [Tags](#)).
5. In **Step 4: Override Rules**, declare any rules to override (see [Rule overriding](#)).
6. In **Step 5: Documentation**, document your technical rule.
7. In **Step 6: Version information**, enter the version information.
8. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[Technical rules](#)

[Smart folders](#)

[Project element properties](#)

[Project element documentation](#)

[Version information](#)



# Creating a resource

You create resources from the Compose tab.

## About this task

Resource elements let you store any type of file within the Decision Center repository. You can create new resources if you have user permissions to do so.

### Note:

You must create a smart folder that displays resources to see them in the Enterprise console.

## Procedure

To create a resource:

1. On the **Compose** tab, select **Resource** as the type of project element to create, and then click **OK**.
2. In **Step 1: Properties** of the Compose Wizard, browse to select your resource. The resource takes the name of the file you selected.

Enter the folder, group, and documentation of your resource, and then click **Next**.

3. In **Step 2: Version information**, enter the version information.
4. Click **Finish**.

**Parent topic:** [Compose: Create project elements](#)

### Related concepts:

[Smart folders](#)

[Project element properties](#)

[Version information](#)

### Related tasks:

[Storing domains in Excel](#)

[Updating dynamic domains](#)

# Tags

You use tags to store data with your project elements.

For example, you can store data related to the geography of the rule or its development phase. You can retrieve this information in a query or in a report.

Tags are presented as a table of names that you declare and values that you give them. You create them in the **Tags** step of the Compose wizard, where you build the table of tags for that project element. For example:

Name	Value
Geography	New York
Phase	1

For developers, tags can be useful when you have implemented a rule model extension in Rule Designer and you synchronize with Decision Center. In this case, extended properties are converted into tags so that the synchronization remains smooth. Later, if you implement the rule model extension in both Rule Designer and Decision Center, these tags are converted back to properties.

**Note:** If you defined tags on rule packages in Rule Designer, you cannot synchronize them with Decision Center, because Decision Center does not support tags on rule packages.

**Parent topic:** [Compose: Create project elements](#)

**Related tasks:**

- [Creating business rules](#)
- [Creating a function](#)
- [Creating a technical rule](#)

# Rule overriding

You use rule overriding to give rules precedence over other rules.

For example, you can set an entire decision table to override another decision table or decision tree.

A rule that overrides one or more other rules is executed in place of the rules that are overridden. Normally, rule overriding operates within a hierarchy of rules, with a local or more specific rule overriding a more general rule. For example, you can set a minimum customer age for a particular state to take precedence over a minimum customer age for the country.

The rule overriding hierarchy means that rules that are at the top of the hierarchy override rules at the bottom if they are found in the same rule task of a ruleflow.

You set rule overriding in the **Override Rules** step of the Compose wizard for the rule, by building the table of rules that you want the current rule to override. For example:

Rule	Type
bankruptcyScore	Overridden rule
defaultInsurance	Overridden rule

**Parent topic:** [Compose: Create project elements](#)

**Related concepts:**

[Overview: Ruleflows](#)

**Related tasks:**

[Creating business rules](#)

[Creating a technical rule](#)

## Project element documentation

You can add project element documentation when you create or edit project elements.

You add any information that you consider useful or pertinent to that project element.

You write documentation in a free text field. This field can remain empty, although it is not good working practice to do so.

**Note:**

Do not confuse element documentation with version information. You display version information when you consult the history of an element.

**Parent topic:** [Compose: Create project elements](#)

**Related concepts:**  
[Version information](#)

# Version information

Version information is associated with the version to which you add it.

You enter version information when creating or editing a project element.

Decision Center creates a new version of all project elements when they are saved, so it is good working practice to enter a comment in the text area of the **Version Information**. This comment is associated with the version and is displayed in the table when you consult the history of the element.

**Note:**  
  
For information about version numbers, refer to [Versioning](#).

**Parent topic:** [Compose: Create project elements](#)

## Query: Search your projects

You use queries to search through your rule projects to display the business rules or other project elements that correspond to criteria of your choice.

**Note:** Queries in the Enterprise console only pertain to classic rule projects. To use this feature with decision services, you must use the Decision Center Business console.

### [Introducing queries](#)

You can use queries to search through your rule projects.

### [Query conditions](#)

Query conditions are the criteria that you define for the search.

### [Query actions](#)

The default action of a query is to display the query results, but you can specify additional actions.

**Parent topic:** [Working with the Enterprise console](#)

## Introducing queries

You can use queries to search through your rule projects.

Queries display the business rules or other project elements that correspond to criteria of your choice.

You can use queries to evaluate the impact of changes to your rules. When you obtain the query results, you can perform actions on them, generate a report, or perform checks.

Queries are made up of two parts:

- The **query** part, for example, Find all business rules such that the creator of each business rule is me
- The **action** part, for example Do set the status property to deployable

You can use queries throughout the Enterprise console: in smart folders, reports, when generating rulesets. You create, run, and save them on the **Query** tab.

### Note:

Click the refresh icon  to check for updated queries run by concurrent users.

You construct queries in the same way that you construct rules. The query condition contains the criteria for the query action to be executed. The default query action is to just display the results, but you can also have the query carry out other actions such as moving the results of the query to another folder.

Queries run independently of what is currently selected in the **Explore** tab. They run against the current rule project, as well as any dependent projects if you specify this option.

To run a query, click the **Run Query** button. The query results display in the same tables as those on the **Explore** tab, which give you access to the same exploration features such as view details and history.

When your query completes, you can carry out the following tasks:

### Apply Actions

If your query has an action part, the actions are performed only when you click the link. The action is performed on all the rules retrieved by the query.

### Generate Report on Query Results

The report runs in another browser window and displays the content and properties of the results of the query, ready to print.

**Parent topic:** [Query: Search your projects](#)

# Query conditions

Query conditions are the criteria that you define for the search.

The conditions of your query appear under the Find part of the query.

By default, you run queries on the business rules but you can also search for specific types of business rules, such as action rules, decision tables, or another type of rule specific to your company.

You can also search for folders (rule packages), ruleflows, smart folders, functions, or variable sets.

**Note:**  
You can query only the working version of project elements, that is, the version referenced in the baseline currently selected in Decision Center.

You start by selecting the project element you want to query. You can then refine the query by adding a filter in the form of a `such that` statement, followed by condition statements.

## Example

*Find all business rules  
such that the effective date of each business rule is after 31/1/2016  
and the effective date of each business rule is before 28/2/2016*

You can refine queries using a `such that` statement to filter different project items:

- Properties: see [Filter by properties](#)
- Business terms: see [Filter by business terms](#)
- Rule behavior: see [Filter by behavior of rules during execution](#)
- Rule impact: see [Filter by impact of rules during execution](#)

**Note:**  
You can identify which rules have changed in the current session (by querying on `after my login time`) or which rules you have changed (by querying on `creator is me`).

## Filter by properties

You can filter the query according to one of the properties of the project elements being queried (see [Project element properties](#)).

For example, the following query displays only business rules whose status is new:

*Find all business rules  
such that the status of each business rule is new*

## Filter by business terms

You can filter the query according to the business terms used in the conditions and actions of your rules. This type of filter is useful when trying to find rules affected by a policy change, for example:

*Find all business rules  
such that each business rule modifies the value of the insurance rate*

The following query constructs find rules that use or modify business terms:

### uses the value of

Returns rules that use the values of certain business terms.

#### Example

The following query returns all business rules that use loan grade values:

*Find all business rules  
such that each business rule uses the value of the loan grade in 'a report'*



## uses the phrase

Returns rules that call a given business term.

### Example

The following query returns all business rules that use 'add to the corporate score in the loan report':

```
Find all business rules
such that each business rule uses the phrase [add 'a number' to the
corporate
score in 'a report']
```

## uses the phrase ... where

Returns rules that call a given business term to which a constraint is applied

### Example 1

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can equal 100:

```
Find all business rules
such that each business rule uses the phrase [set the credit score of 'a
borrower'
to 'a number' , where 'a number' equals 100]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 100;
```

### Example 2

The following query returns all business rules that call the term that sets the credit score of 'a borrower' to 'a number', where this number can be at least 100:

```
Find all business rules
such that each business rule uses the phrase [set the credit score of 'a
borrower'
to 'a number' , where 'a number' is at least 100]
```

This query would return the following rule:

```
if the age of 'the borrower' is 30
then set the credit score of 'the borrower' to 200;
```

### Example 3

The following query returns all business rules that call the term that gives the discount 'a number' to 'a borrower', where the credit score of 'a borrower' is at most 200:

```
Find all business rules
such that each business rule uses the phrase [give the discount 'a
number' to 'a borrower' where
the credit score of 'a borrower' is at most 200]
```

When the elements mentioned in the constraint do not appear in a rule, the rule is returned. In the following rule, the fact that the category of 'the borrower' is Gold does not exclude the possibility that the credit score of 'the borrower' is at most 200.

This query would return the following rule:

```
if the category of 'the borrower' is Gold then give the discount 10 to 'the
borrower';
```

This query would **not** return the following rule:

```
if the credit score of 'the borrower' equals 300 then give the discount 10
```

to 'the borrower';

### modifies the value of

Returns rules that modify certain business terms.

#### Example

The following query returns all decision tables that modify the value of the insurance rate:

*Find all decision tables  
such that each decision table modifies the value of 'the insurance rate'*

### Filter by behavior of rules during execution

You can filter the query according to the semantics of the rules, that is, their behavior during execution. This type of filter finds rules that could be applicable when certain conditions are true or become true. This filter is useful for testing the conditions under which rules would be executed to make sure that they behave as expected.

You can use the following semantic query constructs:

#### may apply when

Returns all rules whose condition part could meet the query condition, or where there is nothing in the rule condition that would contradict the query condition. This query can thus include rules with conditions that are not specifically relevant to the query condition.

#### Example

**Rule 1:** If the score of the borrower is at least 10 then...

**Rule 2:** If the age of the borrower is at least 21 then...

**Rule 3:** If the score of the borrower is less than 10 and the age of the borrower is more than 35 then...

#### Query 1:

*Find all business rules  
such that each business rule may apply when the score of the borrower is  
20*

#### Query 2:

*Find all business rules  
such that each business rule may apply when the score of the borrower is  
5*

**Query 1** returns Rule 1 and Rule 2. It returns Rule 1 because if the score of the borrower is at least 10, then it could be 20. It returns Rule 2 because there is nothing in Rule 2 that specifically says that the condition could not hold: the score of the borrower could be 20. It does not return Rule 3 because it contains a condition (score less than 10) that specifically indicates that the rule could not apply.

Query 2 returns Rule 2 and Rule 3. Rule 1 cannot apply because the query condition sets the score to less than 10. In Rule 2, nothing specifically stops the rule from being applicable when the score is 5. In Rule 3 at least one of the conditions could apply, and there is nothing in the other condition that negates the fact that the score could be 5.

#### may become applicable when

A more specific query that returns only those rules whose condition part could meet the query condition. It does not return rules with conditions that are not specifically relevant to the query condition.

#### Example

**Rule 3:** If the category of the customer is Platinum then...

**Rule 4:** If the category of the customer is not Platinum then...

**Rule 5:** If the age of the customer is at most 65 then...

**Rule 6:** If the age of the customer is at most 65 and the category of the customer is not Platinum...

#### Query 1:

*Find all business rules  
such that each business rule may become applicable when the category of  
'a customer' is Gold*

This query returns Rule 4 and Rule 6. It returns Rule 4 because if the category of the customer is not Platinum, it could be Gold. It returns Rule 6 for the same reason. The additional condition relating to the age of the customer does not contradict the condition in which the category may be Gold.

The query does not return Rule 3 because a rule that applies when the customer category is Platinum does not apply when the category is Gold. It does not return Rule 5 because the query looks for rules that could become applicable when the customer category is Gold, and the age of the customer is not relevant to this query. In other words, the condition of the rule cannot be affected by the fact that the customer category is Gold.

#### **Query 2:**

*Find all business rules  
such that each business rule may become applicable when [the age of  
'a customer' is at least 21]*

This query does not return Rule 5 because it searches for rules that could “become” applicable when the customer age is over 21. Rule 5 is applicable even if the customer age is under 21. Therefore Rule 5 does not “become” applicable, but it “remains” applicable even if the customer age changes from under 21 to over 21.

#### **may lead to a state where**

Returns rules that, when executed, could show a result that meets the query condition. This query takes into account both the condition and action parts of a rule. It filters rules based on their effect. It does not, therefore, return rules that have no influence on the query condition.

#### **Example**

**Rule 7:** If the age of the borrower is at least 25 then set the credit score of the borrower to 60

**Rule 8:** If the age of the borrower is more than 18 and less than 25 then set the credit score of the borrower to 20

#### **Query:**

*Find all business rules  
such that each business rule may lead to a state where the credit score  
of  
the borrower is more than 50*

This query returns Rule 7 but not Rule 8 because, when the age of the borrower has been checked and the credit score set, only Rule 7 shows a result of over 50.

#### **Filter by impact of rules during execution**

You can filter rules by identifying the links and dependencies between them. You can check how the execution of a rule impacts the applicability of other rules, and how a rule is impacted by the execution of other rules.

#### **may select**

Returns the ruleflows and rule tasks that may select a given rule.

#### **Example**

The following query returns the ruleflows and rule tasks that may select "Rule 1".

*Find all ruleflows  
such that each ruleflow may select "Rule 1"*

#### **may enable rule**

Returns rules that make a given rule applicable.

#### **Example**

##### **Rule 1:**

*if the age of the customer is 25*

then set the category of the customer to Bronze ;

**Rule 2:**

if the category of the customer is Bronze  
then give Champagne to the shopping cart of the customer with message:  
"Congratulations" ;

**Query:**

*Find* all business rules  
*such that* each business rule **may enable** "Rule 2"

This query returns Rule 1 because it sets the category of the customer to Bronze, and thus makes the condition of Rule 2 valid.

**may disable**

Returns rules that make a given rule inapplicable

**Example**

**Rule 1:**

if the age of the customer is 18  
then set the category of the customer to Copper ;

**Rule 2:**

if the category of the customer is Bronze  
then give Champagne to the shopping cart of the customer with message:  
"Congratulations" ;

**Rule 3:**

if the age of the customer equals 25 and the category of the customer is  
Bronze  
then set the category of the customer to Diamond ;

**Query:**

*Find* all business rules  
*such that* each business rule **may disable** "Rule 2"

This query returns Rule 1 and Rule 3. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It returns Rule 3 because although one of the conditions is that the category of the customer is Bronze, the action is to set the category of the customer to Diamond. Therefore, if the category of the customer is Diamond, Rule 2 is not applicable.

**may be enabled by**

Returns rules that are made applicable by a given rule

**Example**

**Rule 1:**

if the age of the customer is 18  
then set the category of the customer to Copper ;

**Rule 2:**

if the category of the customer is Bronze  
then give Champagne to the shopping cart of the customer with message:  
"Congratulations" ;

**Rule 3:**

```
if the age of the customer equals 25 and the category of the customer is
Gold
then set the category of the customer to Diamond ;
```

**Query:**

```
Find all business rules
 such that each business rule may disable "Rule 2"
```

This query returns Rule 1. It returns Rule 1 because if the category of the customer is set to Copper it cannot be Bronze. It does not return Rule 3 because the action is to set the category of the customer to Diamond, and it can only be executed from a state where the category is already not Bronze. Therefore Rule 3 does not disable Rule 1.

**may be disabled by**

Returns rules that are made inapplicable by a given rule

**Example**

**Rule 1:**

```
if the age of the customer is 25
then set the category of the customer to Bronze ;
```

**Rule 2:**

```
if the age of the customer equals 25 and the category of the customer is
Copper
then set the category of the customer to Gold
```

**Query:**

```
Find all business rules
 such that each business rule may be disabled by "Rule 1"
```

This query returns Rule 2 because it is made inapplicable by Rule 1. In Rule 1 the category of a customer whose age is 25, is set to Bronze, therefore it cannot be Copper.

**Parent topic:** [Query: Search your projects](#)

## Query actions

The default action of a query is to display the query results, but you can specify additional actions.

You can add further actions by specifying them under the **Do** part of the query.

Query actions can make the following changes:

- Copy or move the displayed elements to another folder
- Delete the displayed elements
- Add or remove a category to the displayed elements
- Set any of the properties of the displayed elements to a value that you specify

### Example

```
Find all business rules
 such that the status of each business rule is new
Do set the status of each business rule to validated
```

The actions are carried out on all the displayed elements when you click **Apply Actions** in the **Query** tab.

**Parent topic:** [Query: Search your projects](#)

**Related concepts:**  
[Project element properties](#)

## Analyze: Check your projects

You can check your projects for consistency and completeness and generate rule project reports.

**Note:** The topic in this section about [Managing merge reports](#) only pertains to classic rule projects.

### [Generating a project report](#)

You use HTML reports that show the content and properties of project elements.

### [Managing merge reports](#)

You can access and manage the reports created when you merge branches.

**Parent topic:** [Working with the Enterprise console](#)

## Generating a project report

You use HTML reports that show the content and properties of project elements.

If you are viewing a branch or baseline when you generate the report, the report describes the elements of the branch or baseline.

To generate a project report in HTML format:

- Select the **Analyze** tab and then click **Generate Project Report**.

You can base your report on any of the existing queries available on the **Query** tab. If the rule project has no queries, the report uses all the rules in the rule project.

The report is generated in another browser window.

**Note:**

You can also generate the report directly from the **Query** tab.

**Parent topic:** [Analyze: Check your projects](#)

**Related information:**

[Query: Search your projects](#)



# Managing merge reports

You can access and manage the reports created when you merge branches.

## About this task

Decision Center keeps an Excel version of the report when you merge branches. You can download these reports and manage them from the **Analyze** tab.

## Procedure

To manage the reports of merged branches:

1. On the **Analyze** tab, click **Manage Merge Reports**.
2. Select the branch for which you want to see the reports.

Decision Center lists the reports for that branch.

3. Click the name of the report in the table to download it, or select it in the table and click **Rename** to rename it or **Delete** to delete it.

**Parent topic:** [Analyze: Check your projects](#)

## Related tasks:

[Branches](#)

[Managing subbranches](#)

## Related information:

[Merging branches](#)

## Project: Manage your project

You use the Project tab to manage advanced features of your projects.

**Note:** The topics in this section about [Managing subbranches and baselines](#), [Merging branches](#), [Dynamic domains](#) only pertain to classic rule projects. To merge branches or use dynamic domains with decision services, you must use the Decision Center Business console.

### **[Managing subbranches and baselines](#)**

From any branch, including the main, you can manage subbranches, baselines, and deployment baselines.

### **[Merging branches](#)**

You can merge branches together.

### **[Project dependencies](#)**

When you set up a project to access the contents of other projects, you create a project dependency.

### **[Project options](#)**

You can set options on the project.

### **[BOM path](#)**

A BOM path entry defines a set of business elements in the business object model.

### **[Dynamic domains](#)**

Domains are visible as sets of values in your business terms.

**Parent topic:** [Working with the Enterprise console](#)

# Managing subbranches and baselines

From any branch, including the main, you can manage subbranches, baselines, and deployment baselines.

## **Managing subbranches**

You can create subbranches, and then manage them by renaming or deleting them.

## **Managing baselines**

You can create baselines, and then manage them by renaming, deleting, cloning them to a branch, or restoring them.

## **Managing deployment baselines**

You create deployment baselines when deploying RuleApps, and then manage them by renaming, deleting, or cloning them to a branch.

**Parent topic:** [Project: Manage your project](#)

### **Related tasks:**

[Baselines](#)

[Branches](#)

[Versioning](#)

### **Related information:**

[Merging branches](#)

# Managing subbranches

You can create subbranches, and then manage them by renaming or deleting them.

## About this task

Branches facilitate work done on parallel releases of a project.

Initially, a project contains only a main line in which you work. From the main, sometimes referred to as the main branch, you can create one or more subbranches. From any new branch you can create other subbranches, as many as you need to manage your releases.

The way you manage branches depends on how many releases you need to manage at a given time:

- If you only have **one release in production** at any given time, use the main line as the production branch. Then, to work in parallel on an upcoming release, create a branch for the new release. Changes to production continue in the main, from which you deploy, and changes for the upcoming release are made in the new branch. Periodically, you can merge the production changes into the upcoming release branch so that the upcoming release always has the most up-to-date rules. When you are ready for the upcoming release to go into production, merge the changes from the subbranch to the main, so the main continues to represent the production release. At this point, the branch receives no further edits and exists only for the historical record.
- If you have **multiple releases in production** at a given time, you need multiple active subbranches, each representing a production release. The main represents work in progress on an upcoming release. You may also be working in parallel on multiple upcoming releases, so some subbranches will also represent upcoming releases. By the time any given release goes to production, it must have its own branch. Production changes are made in each of these subbranches, and merged into other branches as needed. As you retire releases from production, these branches receive no further edits and exist for the historical record.

**Note:**

When keeping a branch for the historical record, you should baseline the branch and have the Administrator remove permissions to edit it.

## Procedure

To manage subbranches:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose subbranches you want to manage. By default, Decision Center selects the branch you are currently working in and displays its subbranches in the table.
3. In the **Subbranches** section of the page, click **New** to create a subbranch.

**Note:**

You can also create a subbranch directly from the **Home** tab.

If the subbranch already exists, select it and click one of the following:

- **Rename:** To rename the subbranch.
- **Delete:** To delete the subbranch. You cannot delete a branch if you are currently in it or one of its subbranches. You have to be viewing a parent branch.

**Parent topic:** [Managing subbranches and baselines](#)

**Related concepts:**

[The Home tab](#)

**Related tasks:**

[Branches](#)

[Versioning](#)

**Related information:**

[Merging branches](#)

# Managing baselines

You can create baselines, and then manage them by renaming, deleting, cloning them to a branch, or restoring them.

## About this task

Baselines are designed to capture the state of a project or branch at a given moment in time. Each branch of a project can contain many baselines.

You cannot edit a baseline; they are for consultation only. You consult them by selecting them in the **Home** tab.

**Note:**  
  
Note also the existence of deployment baselines, which you can create when deploying RuleApps (see [Managing deployment baselines](#)).

Baselines that you create appear in the table of baselines for that branch.

When you create a baseline, it includes any project dependencies.

In addition to creating baselines, you can do the following as part of managing baselines:

### Delete a baseline

When you delete a baseline you do not delete any project elements, only the snapshot itself.

### Clone to branch

You can create a branch from a baseline. You can then continue editing the content as part of a branch, which can then be merged with another branch. The original baseline remains intact when you clone it to a branch.

### Restore a baseline

You can restore a baseline so that it becomes the current state of the project or branch.

Project elements created after the baseline was created are overwritten when that baseline is restored. Consequently, before restoring a baseline, you might want to create a temporary baseline to keep a trace of any new project elements that have not been captured in the initial baseline.

**Note:**  
  
If you lose new project elements when restoring a baseline, you can retrieve them from the Decision Center recycle bin.

## Procedure

To manage baselines:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose baselines you want to manage. By default, Decision Center selects the branch you are currently working and displays its baselines in the table.
3. In the **Baselines** section of the page, click **New** to create a baseline.

If the baseline already exists, select it and click one of the following:

- **Rename:** To rename the baseline.
- **Delete:** To delete the baseline.
- **Clone to branch:** To create a new branch from the contents of the baseline.
- **Restore:** To restore the baseline.

**Parent topic:** [Managing subbranches and baselines](#)

**Related concepts:**  
[The Home tab](#)

**Related tasks:**  
[Baselines](#)  
[Versioning](#)  
[Recycle bin](#)

# Managing deployment baselines

You create deployment baselines when deploying RuleApps, and then manage them by renaming, deleting, or cloning them to a branch.

## About this task

Deployment baselines capture the version of the elements contained in the RuleApp at the time they were deployed. Then, to redeploy the RuleApp at some later time, you can choose whether the version of the elements corresponds to the initial deployment or to a subsequent redeployment.

Deployment baseline are similar to standard baselines, but have the following differences:

- You cannot create them manually; you create them as part of RuleApp deployment.
- Since they are created from the ruleset extraction, there is one deployment baseline created per ruleset contained in the RuleApp. Decision Center names deployment baselines **Name Given / RuleApp version / ruleset version**.
- Because RuleApps are independent of projects, deployment baselines created in a project are visible in any dependent project.

You can consult existing deployment baselines by selecting them in the **Home** tab. You can do the following as part of managing deployment baselines:

## Delete a deployment baseline

When you delete a deployment baseline you do not delete any project elements, only the snapshot itself.

## Clone to branch

You can create a branch from a deployment baseline. You can then continue editing the content as part of a branch. The original deployment baseline remains intact.

## Procedure

To manage deployment baselines:

1. On the **Project** tab, click **Manage Subbranches and Baselines**.
2. At the top of the page, select the branch whose deployment baselines you want to manage. By default, Decision Center selects the branch you are currently working and displays its deployment baselines in the table.
3. In the **Deployment Baselines** section of the page, select the entry in the table and click one of the following:
  - **Rename**: To rename the deployment baseline.
  - **Delete**: To delete the deployment baseline.
  - **Clone to branch**: To create a new branch from the contents of the deployment baseline.

**Parent topic:** [Managing subbranches and baselines](#)

## Related tasks:

[Baselines](#)

[Versioning](#)

# Merging branches

You can merge branches together.

## [Using branch merging](#)

You can merge branches together.

## [Using Diff and Merge](#)

Use Diff and Merge when there are several changes to a rule and you want to specify which of these changes to merge.

**Parent topic:** [Project: Manage your project](#)

### **Related tasks:**

[Branches](#)

[Managing subbranches](#)

[Versioning](#)

[Managing merge reports](#)

# Using branch merging

You can merge branches together.

## About this task

When you merge branches, you decide what to do with each project element that is different between the two branches:

- Replace the version in the target branch
- Update the source branch with the version found in the target branch
- Take no action at all

After you specify the two branches you want to merge, Decision Center looks at both branches and displays a table containing all the differences. This table contains the name and folder of project elements that are different between the branches, and their state in both branches. This table also contains an **Actions** column, containing a proposed action to take. You can click inside the cells of this column and change individual actions as you see fit.

Filter: <input type="text"/>				
Name ▲	Folder	main	Spring Updates	Action
approval	eligibility	Unmodified	Deleted	Delete from 'main'
<i>checkIncome</i>	<i>eligibility</i>	<i>In Conflict</i>	<i>In Conflict</i>	No action to perform
Rule 1	eligibility	-	New	Add in 'main'

Click next to the **Action** column when there are several changes to the rule and you want to specify which of these changes to merge (see [Using Diff and Merge](#)). This option is only available with action rules or decision tables.

The elements taken into account during a merge operation are:

- Rule Artifacts (BAL rules, technical rules, decision tables, decision trees, functions)
- Simulation artifacts (metrics, KPIs, simulation models, input data, simulations)
- Testing artifacts (test suites, test cases)
- Decision operations
- Deployment configurations
- Ruleflows
- Queries
- Variable sets
- BOM
- Vocabulary
- B2X
- Resources
- Folders
- DVS artifacts (scenario suites for testing and simulation)
- Templates

**Note:**

Changes to ruleset parameters, extractors, project dependencies, categories and dynamic XOM (schemas), cannot be merged. You must change these manually in the other branch.

To help Decision Center propose the correct action to take, you can specify your preferred direction for merging from the following options:

### Bidirectionally

This is the default option. The proposed action is based on the assumption that you want to reflect changes made in either branch as follows:

- If a new rule exists in one branch, you want to add it to the other branch.
- If a rule has been deleted in one branch, you want to delete it from the other branch.
- If a rule has been modified in one branch but not in the other, you want to update the unmodified one.
- If a rule has been modified in both branches, no action is proposed, and you must specify what action to take.

### Only to target branch



When you want all changes you made in the working branch to be pushed to the target branch, but none of the changes made in the target branch to be reflected in your working branch.

### **Only to source branch**

When you want all changes made in the target branch to be reflected in your working branch, but none of the changes made in your working branch to be pushed to the target branch.

### **Procedure**

To merge a branch:

1. From one of the branches to merge, go to the **Project** tab and click **Merge Branches**.
2. Select the target branch to merge with and click **Next**.
3. Change the direction in which to merge changes if different from **Bidirectionnally**.
4. Specify the actions to take and click **Apply Merge**.

### **Results**

Decision Center shows the details of the merge. You can access an Excel version of the report from the **Analyze** tab.

**Parent topic:** [Merging branches](#)

### **Related tasks:**

[Branches](#)

[Managing subbranches](#)

[Using Diff and Merge](#)

[Versioning](#)

[Managing merge reports](#)

# Using Diff and Merge

Use Diff and Merge when there are several changes to a rule and you want to specify which of these changes to merge.

## Before you begin

When merging decision tables in which columns have been added or removed, you must resolve column differences before you resolve row differences. If you try to copy a row before you have copied columns, a message prompts you to apply the requested change to the columns first.

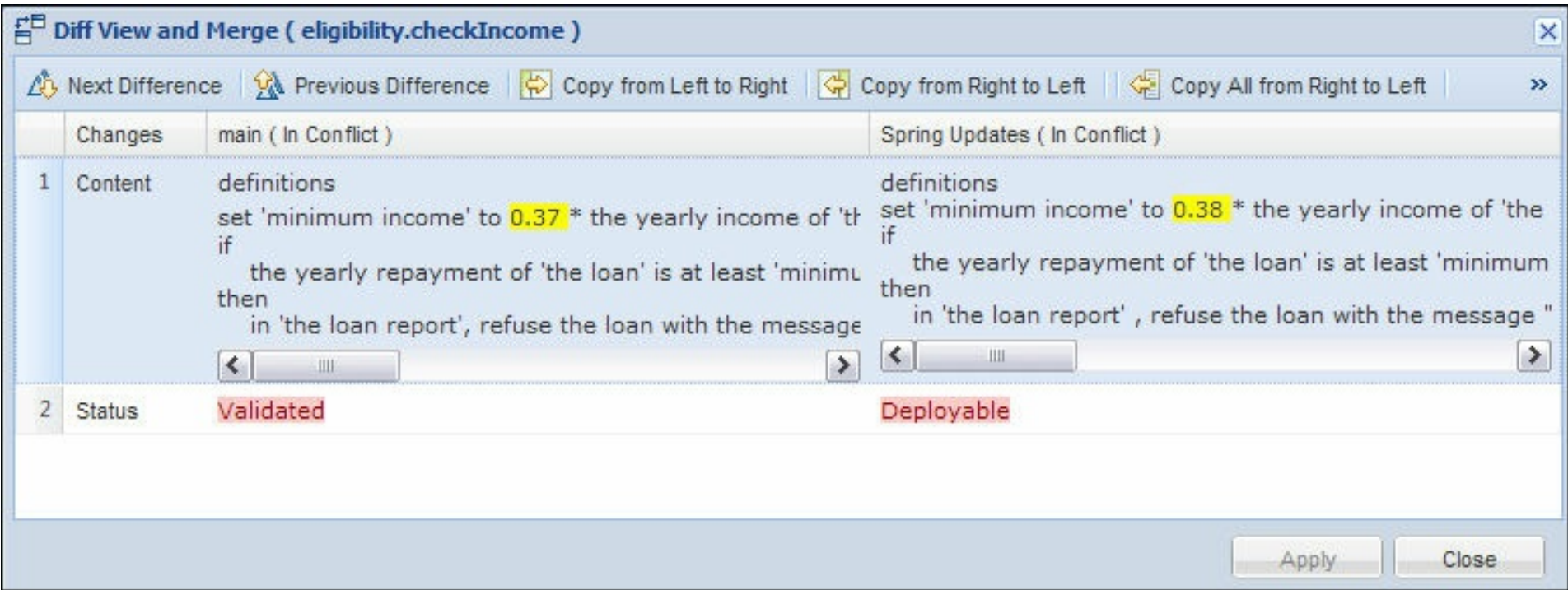
When merging decision table differences, the following differences are detected and collected together in a single row labeled Decision Table Properties in the Diff and Merge table. This means that the only option available to you is to copy them as a whole from one side to the other:

- Settings in the Table Properties editor, such as table view settings, locks, table checks, and preconditions.
- Column options such as column width, locks, and default value.

For example, you cannot copy table view settings from left to right and preconditions from right to left. This restriction does not apply to differences in properties that are defined in Step 1: Properties of the Compose wizard, such as name and status: these differences are listed in separate rows in the Diff and Merge table.

## About this task

Use Diff and Merge when there are several changes to the rule and you want to specify which of these changes to merge. Diff and Merge displays the contents and properties of each branch of the project element side by side. You can step through each difference in turn, visually compare the values, decide which value you want to retain, and copy it from one side to the other. You can request a partial merge by resolving some differences and retaining others.



## Procedure

To use Diff and Merge:

1. In the Diff and Merge menu bar, click **Next Difference**. Diff and Merge highlights the next available difference in the project element.
2. Take one of the following actions:
  - To merge the difference, click **Copy from Right to Left** or **Copy from Left to Right** depending on which value you want to retain. No changes are made to the project element at this stage.
  - To retain this particular difference, click **Next Difference**.
3. Repeat the previous steps to work through each instance of a difference that you intend to resolve. You do not need to resolve every difference.

**Note:**

To undo the most recent individual merge, click **Revert**.

4. When you have finished specifying how you want to resolve each difference, click **Apply**.

## Results

Diff and Merge closes and you return to the table displaying the list of project elements that contain differences in the two selected branches. At this point, the changes you have specified are not committed. The project element you have worked on displays the action **Apply Merge**, indicating that this project element is now ready for you to apply the merge. For information on how to apply the merge to commit the changes, see [Using](#)

[branch merging.](#)

**Parent topic:** [Merging branches](#)

**Related tasks:**

[Branches](#)

[Managing subbranches](#)

[Using branch merging](#)

[Versioning](#)

## Project dependencies

When you set up a project to access the contents of other projects, you create a project dependency.

You can set up a project or branch to access the contents of other projects or branches of that project. For example, you might want to place your BOM and vocabulary in a separate project and reference it, so that several rule projects can access the same BOM and vocabulary without duplication.

You set up project dependencies on the **Project** tab, under **Edit Project Dependencies**.

**Note:**

You must sign in to the Enterprise console with **Configuration Manager** permissions to be able to set up project dependencies.

When you create a project dependency, you establish a reference from your project to another project. When you have established a dependency, you can specify whether or not the smart folders and queries that you use in your projects include the dependent projects.

When you create a project dependency, you specify which project to reference and whether to reference the main of that other project or one of its branches or baselines. Keep in mind that when you create a baseline or branch of a project, the baseline or branch also keeps any dependencies. It stores the reference to the other project and not the actual contents of the other project.

**Parent topic:** [Project: Manage your project](#)

# Project options

You can set options on the project.

You can set the following project options on the **Project** tab, under **Edit Project Options**.

## Build options

Allow ruleset archive generation to be cancelled on warnings or errors, or not at all. When you select **Build automatically**, Decision Center generates the IRL for a rule as soon as you save it. Otherwise, this generation occurs when generating the ruleset. You might find enabling this option useful to accelerate ruleset generation.

## Rule editor

Select the default rule editor to be used for editing elements that belong to this project. Options: **Guided editor** and **Intellirule editor**. Users can override the default rule editor on the User Options page, which you access from the top banner of the Enterprise console.

## Show number of elements in the smart folders

Smart folders show the number of elements they contain.

## Refresh children on expand in the smart folders

Folders are refreshed automatically when you move from one to another in the smart folders.

**Parent topic:** [Project: Manage your project](#)

## Related tasks:

[Selecting your preferred rule editor](#)

## BOM path

A BOM path entry defines a set of business elements in the business object model.

In the Enterprise console you can display the BOM path by clicking **View BOM Path** on the **Project** tab.

**Note:**

You can view the BOM path only if you sign in with **Configuration Manager** permissions.

The business object model (BOM) makes business rule editing user-friendly by providing tools that you can use to set up a natural-language vocabulary. Policy managers can then use this vocabulary to describe their business logic in a business rule language.

A BOM path comprises one or more BOM path entries. BOM entries can target BOM files or BOMs from other projects.

You can order BOM entries so that if you have two business elements with the same name in two BOM entries, the one in the first BOM entry in the path overrides the other.

**Note:**

For more information, see the Rule Designer user documentation.

**Parent topic:** [Project: Manage your project](#)

**Related concepts:**  
[Project dependencies](#)

## Dynamic domains

Domains are visible as sets of values in your business terms.

With a dynamic domain, the set of values is stored and managed outside the business object model. Changes to the set of values are then reflected in the BOM. When you write business rules that use domains, the list of values that you choose from is created dynamically, so you never have to change the BOM manually.

Domains are available to Decision Center through a domain provider. The domain provider can be managed externally or by you in an Excel file. In either case, you must upload any changes to the domain to Decision Center.

### Storing domains in Excel

You can modify the values of dynamic domains that are stored in an Excel file.

### Updating dynamic domains

When your domain values change, you must update them in Decision Center.

**Parent topic:** [Project: Manage your project](#)

# Storing domains in Excel

You can modify the values of dynamic domains that are stored in an Excel file.

## About this task

When a project contains a dynamic domain defined in an Excel file, the Excel file is stored as a resource in the project. You can modify the Excel file and then upload the changes to Decision Center.

The Excel file has a specific structure with columns that map properties in the BOM:


- A column for the domain values
- A column for the label of each value
- A column that defines the BOM-to-XOM mapping for each value

Other optional columns for the documentation or labels in other locales might be defined in the Excel file.

You can modify the values in each column, but you must not change the columns or column headings.

## Procedure

To edit the Excel file:

1. In the smart folder dedicated to resources, browse to the resource corresponding to the Excel file. If the smart folder does not exist, create it. In **Step 1: Properties**, select **Include dependencies** if the Excel file is in a dependent project, and in **Step 2: Query**, use the query Find all resources to populate the smart folder.
2. Select the Excel file in the table, and click  **Download this element**.
3. Save the file on your local drive.
4. Open the Excel file, modify the values, and save the changes.

You can now upload the modified Excel file to Decision Center.

5. In the **Explore** tab, make sure that the Excel file is selected, and click **Edit**.
6. Click **Browse** to select the modified Excel file.
7. Click **Finish**.
8. To update the BOM with the new values, click **Update Dynamic Domains** in the **Project** tab.

**Parent topic:** [Dynamic domains](#)

### Related tasks:

[Creating a resource](#)

[Updating dynamic domains](#)



# Updating dynamic domains

When your domain values change, you must update them in Decision Center.

## About this task

Your domains can come from an outside provider or from an Excel file that you store as a resource. If your domain values change, you update Decision Center as follows:

## Procedure

1. Click **Update Dynamic Domains** in the **Project** tab. All domains available in the current project are displayed.
2. Select the domains that you want to update. Click **Ready to be updated** to see only the domains that changed.
3. Click **Update**.

**Parent topic:** [Dynamic domains](#)

## Related tasks:

[Creating a resource](#)

[Storing domains in Excel](#)

## Working with the editors

You can choose which editor to use when editing business rules.

### Using the rule editors

You can use the guided editor or the Intellirule editor to edit action rules in Decision Center.

### Editing decision tables

When you have created your decision table and its properties, you can use the decision table editor.

### (Deprecated) Editing decision trees

You can use the decision tree editor when the decision tree and its properties have been created at the project level, that is, in the Tree part of the Compose wizard.

### Previewing ruleflows

You cannot edit ruleflows in the Enterprise console.

**Parent topic:** [Working with the Enterprise console](#)

## Using the rule editors

You can use the guided editor or the Intellirule editor to edit action rules in Decision Center.

### Selecting your preferred rule editor

You can select the editor you use for editing action rules in Decision Center.

### Building rules using the guided editor

You use the guided editor to construct rules by following the predefined rule structure and adding rule fragments to it.

### Building rules using the Intellirule editor

You use the Intellirule editor to construct rules by inserting terms and phrases.

**Parent topic:** [Working with the editors](#)

## Selecting your preferred rule editor

You can select the editor you use for editing action rules in Decision Center.

### Procedure

To select your preferred rule editor:

1. In the top banner, click **Options**.
2. In the “Select the rule editor” area, do one of the following procedures:
  - Click **Default rule editor defined for the current project** to use whichever editor is selected for the current project.
  - Click **Guided editor** to use the Guided Editor every time you edit an action rule.
  - Click **Intellirule editor** to use the Intellirule editor every time you edit an action rule.

**Parent topic:** [Using the rule editors](#)

### Related information:

[Building rules using the guided editor](#)

[Building rules using the Intellirule editor](#)

# Building rules using the guided editor

You use the guided editor to construct rules by following the predefined rule structure and adding rule fragments to it.

## [Overview of the guided editor](#)

The guided editor presents you with a rule outline showing the different rule parts.

## [Adding or removing rule statements using the guided editor](#)

When using the guided editor, you can refine your rules by adding or removing rule statements.

## [Controlling how condition statements are combined](#)

You can control multiple condition statements in a rule.

## [Changing the grouping order of rule statements](#)

You can use parentheses to group rule statements.

## [Negating a condition statement](#)

You can negate a condition statement in the guided editor.

## [Inserting an arithmetic expression](#)

You can build complete arithmetic expressions to refine your rule statements.

## [Creating a new variable](#)

You can use variables to identify and subsequently reference an occurrence of something by a convenient name.

## [Opening an empty part of a rule](#)

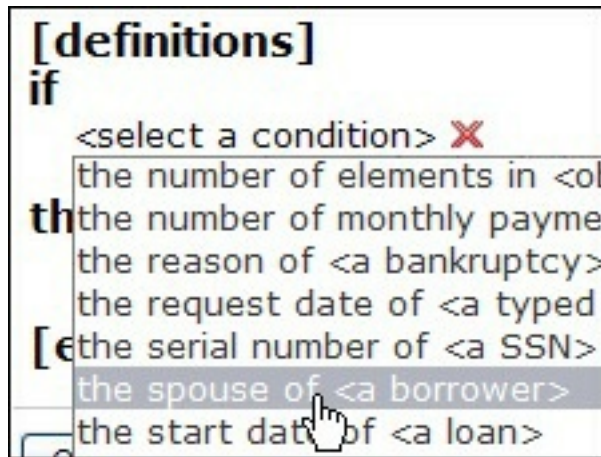
You can open an empty part of a rule and insert a statement in it.

**Parent topic:** [Using the rule editors](#)

## Overview of the guided editor

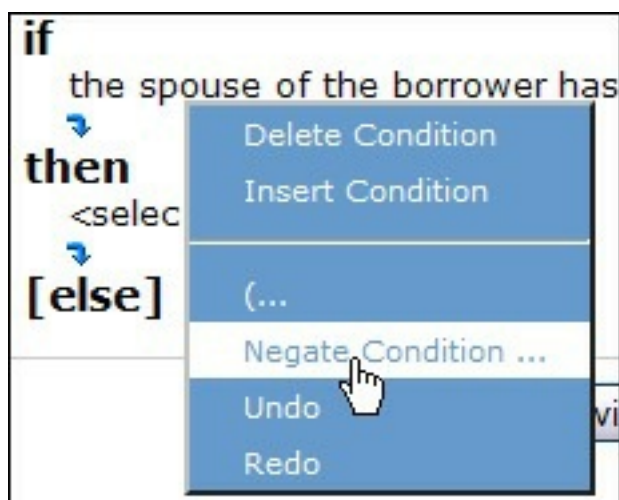
The guided editor presents you with a rule outline showing the different rule parts.

You construct rules from the predefined rule fragments by entering text or numbers in fields, or by selecting items in drop-down lists:



A business term can be part of a longer phrase. For example, if you select an item such as the spouse of <a borrower> from the drop-down list, you must subsequently complete the business term <a borrower> in that phrase.

You can right-click any part of a rule, including fields. Right-clicking provides a drop-down list appropriate to that part of the rule, known as a contextual menu.



**Parent topic:** [Building rules using the guided editor](#)

**Related concepts:**

[Action rules](#)

**Related tasks:**



[Creating business rules](#)

# Adding or removing rule statements using the guided editor

When using the guided editor, you can refine your rules by adding or removing rule statements.

## Procedure

To add or remove a rule statement after the last existing statement:

1. To add a rule statement after the last existing statement:
  - a. Click the add icon under any rule statement  .
2. To add a rule statement:
  - a. Right-click the first business term of the existing statement
  - b. Select **Insert Condition** from the drop-down list.
3. To delete a rule statement:
  - a. Click the red cross found next to the rule statement 
  - b. Alternatively, right-click the first business term and then select **Delete Condition** from the drop-down list.

**Parent topic:** [Building rules using the guided editor](#)

## Related concepts:

[Action rules](#)

## Related tasks:

[Creating business rules](#)

# Controlling how condition statements are combined

You can control multiple condition statements in a rule.

## About this task

When the condition part of a rule contains more than one statement, you can control whether the condition part of the rule is met if *all* (represented by the keyword and) or *any* (represented by the keyword or) of its rule statements are valid.

## Procedure

To change how condition statements are combined:

1. Click the **and** or **or** keyword.
2. Choose the appropriate keyword.



## Results

For more information about how and and or are interpreted, see [Condition negation](#).

**Parent topic:** [Building rules using the guided editor](#)

**Related concepts:**

[Action rules](#)

**Related tasks:**

[Creating business rules](#)



# Changing the grouping order of rule statements

You can use parentheses to group rule statements.

## About this task

By default, the and keyword takes precedence over the or keyword when you have several rule statements. Parentheses let you group rule statements to change the order in which they are processed.

## Procedure

To add or remove parenthesis to a rule statement:

1. To add an opening parenthesis to a rule statement:
  - a. Right-click the operator or business term.
  - b. Select **...(** from the drop-down list.
2. To add a closing parenthesis to a rule statement:
  - a. Right-click the last value of the statement (or right-click the keyword of the next statement).
  - b. Select **)...** from the drop-down list.
3. To remove a parenthesis:
  - a. Right-click the parenthesis.
  - b. Select **Delete** from the drop-down list.

**Parent topic:** [Building rules using the guided editor](#)

## Related concepts:

[Action rules](#)

## Related tasks:

[Creating business rules](#)

## Negating a condition statement

You can negate a condition statement in the guided editor.

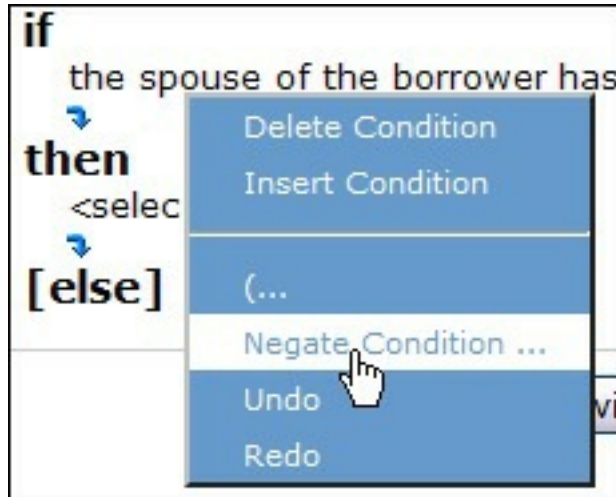
### About this task

You do so by adding `it is not true` at the beginning of the statement. For more information about negating condition statements, refer to [Condition negation](#).)

### Procedure

To insert or remove 'it is not true' in a condition statement:

1. To insert 'it is not true' in a condition statement:
  - a. Right-click the start of the statement (either the operator or the business term).
  - b. Select **Negate Condition** from the drop-down list.



2. To remove 'it is not true' from the statement:
  - a. Right-click **it is not true**.
  - b. Select **Delete** from the menu.

**Parent topic:** [Building rules using the guided editor](#)

**Related concepts:**

[Action rules](#)

**Related tasks:**

[Creating business rules](#)

## Inserting an arithmetic expression

You can build complete arithmetic expressions to refine your rule statements.

### Procedure

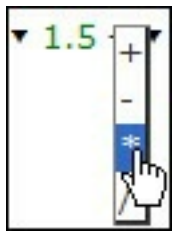
To insert the arithmetic operator:

1. Select  $\pm$  in a rule statement.



The multiplication operator (\*) displays.

2. Click the multiplication operator to change it to another operator.



**Parent topic:** [Building rules using the guided editor](#)

**Related concepts:**

[Action rules](#)

**Related tasks:**

[Creating business rules](#)

## Creating a new variable

You can use variables to identify and subsequently reference an occurrence of something by a convenient name.

### Procedure

To create a variable:

1. Click **definitions** to expand it if it is empty.
2. Click variable1 in the definitions part and then enter the name of your variable.
3. Click <select a choice> and then select how you want to set your variable.

For information on setting variables in the definitions part of a rule, see [Action rules](#).

**Parent topic:** [Building rules using the guided editor](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Creating business rules](#)

## Opening an empty part of a rule

You can open an empty part of a rule and insert a statement in it.

### About this task

Brackets [] around the title of a part of a rule means that it is currently empty. For example, the definitions and else parts are empty by default when you create a new rule.

### Procedure

To open an empty part of a rule:

1. Click the title.
2. Insert a statement in that part.

**Parent topic:** [Building rules using the guided editor](#)

### Related concepts:

[Action rules](#)

### Related tasks:

[Creating business rules](#)

# Building rules using the Intellirule editor

You use the Intellirule editor to construct rules by inserting terms and phrases.

## Overview of the Intellirule editor

When you use the Intellirule editor to build a rule, you add the appropriate terms and phrases to the editing area. You can add terms and phrases by typing or pasting text or by selecting terms and phrases from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, a value, or some other phrase.

## Displaying the completion menu

You can display the completion menu as you build your rule to view the list of terms and phrases that are valid in the current context of the rule, and to select a valid term or phrase.

## Setting completion menu options

You can set options for the way the Intellirule editor presents terms and phrases in the completion menu and for the way rules are parsed.

## Activating and deactivating keyword filtering

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

## Highlighting and correcting errors using the Problem List

The Intellirule editor highlights errors in red in the editing area and lists them in the Problem List under the editing area.

**Parent topic:** [Using the rule editors](#)

# Overview of the Intellirule editor

When you use the Intellirule editor to build a rule, you add the appropriate terms and phrases to the editing area. You can add terms and phrases by typing or pasting text or by selecting terms and phrases from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, a value, or some other phrase.

**Note:**  
Look for the **Rule editor flash demo** on the support site for an English only tour of the Intellirule editor features: [IBM® Customer Support site](#)

## Terms and phrases










The Intellirule Editor provides an editing area into which you add terms and phrases to build rules. You can add terms and phrases in the following ways:

- Type directly in the editing area
- Copy text from another editor or application, and paste it into the editing area
- Select predefined terms and phrases from a completion menu

## Completion menu

The terms and phrases available to you in the completion menu and the different ways in which you can combine them are defined in the business vocabulary of your company.

The following table displays the icons that the completion menu uses to identify the different types of elements that are available for selection.

I c o n	Description
	Term. A business term defined in the vocabulary of your business domain.
	Constant or text string.
	Predicate phrase. A phrase with a value of either true or false. See <a href="#">Rule conditions</a> .
	Navigation phrase. A phrase that refers to an attribute of a term. See <a href="#">Rule conditions</a> .
	Action phrase. See <a href="#">Rule actions</a> .
	Arithmetic operator.
	Automatic variable. A variable that is automatically declared for each term in the vocabulary of your business domain.
	Explicit variable or ruleset parameter. A variable that is declared in the definitions part of a rule, or a parameter that is declared by the rule application.
	Rule language construct.

## Placeholders

Placeholders indicate places in a term or phrase that you must complete by inserting the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder <a customer>:

the age of <a customer>

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain additional placeholders that you must also complete to construct a valid expression.

**Parent topic:** [Building rules using the Intellirule editor](#)

**Related tasks:**  
[Displaying the completion menu](#)  
[Setting completion menu options](#)

## Displaying the completion menu

You can display the completion menu as you build your rule to view the list of terms and phrases that are valid in the current context of the rule, and to select a valid term or phrase.

### About this task

If you do not want the completion menu to open automatically as you type, deactivate the **Enable on Spacebar** option.

### Procedure


To display the completion menu:

Choose one of the following methods:

- If you have selected the **Enable on Spacebar** option, click anywhere in the business rule and start typing. The completion menu opens when you press **Spacebar** or **Ctrl+Spacebar**.
- If you have not selected the **Enable on Spacebar** option, click anywhere in the business rule and then press **Ctrl+Spacebar**.
- Click a placeholder.
- If you have selected the **Enable on double-click** option, double-click a word in the editing area.

### Results

The completion menu displays a list of available insertion options based on your current position in the rule.

You can hide the completion menu by clicking **Close**  in the top corner of the completion menu or by pressing **Esc**.

**Parent topic:** [Building rules using the Intellirule editor](#)

### Related tasks:

[Setting completion menu options](#)




## Setting completion menu options

You can set options for the way the Intellirule editor presents terms and phrases in the completion menu and for the way rules are parsed.

### Procedure

To set completion menu options:

1. In the toolbar above the editing area, click **Completion Menu Options** . A pop-up window opens.
2. Select the check boxes for the options you want to set:
  - **Enable on Spacebar:**
    - Select this check box if you want to open the completion menu whenever you press the Spacebar while typing in the editing area.
    - Clear this check box if you want the completion menu to open only when you press **Ctrl+Spacebar**.
  - **Enable on double-click:**
    - Select this check box if you want to open the completion menu by double-clicking a word in the editing area.
    - Clear this check box if you want to suppress opening of the completion menu by double-clicking a word in the editing area. Double-clicking selects the current word instead.
  - **Enable auto-restart:**
    - Select this option to automatically reopen the completion menu when a term or phrase is selected.
    - Clear this check box if you want to suppress opening of the completion menu when a term or phrase is selected. You open the completion menu by pressing **Ctrl+Spacebar**.
  - **Enable smart mode:**
    - Select this check box to filter the completion menu entries in a smart way to reduce the number of entries.
    - Clear this check box to disable smart mode.
  - **Enable template mode:**
    - Select this check box if you want the editor to insert relevant placeholders along with the phrases you select from the completion menu. You complete phrases by clicking the placeholders and selecting options from the completion menu. Enable completion menu template mode if you prefer to insert longer phrases first and then substitute placeholders.
    - Clear this check box if you want the editor to insert only the section of the suggested phrase up to (but not including) the next placeholder. Deactivate completion menu template mode if you prefer to build the rule systematically from beginning to end in the same way that you might compose an email message.
  - **Use Hierarchical View:**
    - Select this check box if you want the completion menu to group terms and phrases together by type; for example, boolean expressions. Grouping items by type can make it easier to navigate long lists of options.
    - Clear this check box if you want the completion menu to list terms and phrases in alphabetical order.
  - **Filter unreachable phrases:**
    - Select this check box if you want the completion menu to hide phrases that cannot be used because no suitable ruleset parameter or rule variable is defined.
    - Clear this check box if you want the completion menu to display phrases even though they cannot be used because no suitable ruleset parameter or rule variable is defined.
  - **Display toolbar:**

- Select this check box if you want the completion menu toolbar to be displayed above the list of available terms and phrases.
- Clear this check box if you want to hide the completion menu toolbar.

- **Display documentation:**

- Select this check box if you want to display hover help next to the completion menu. When you hover your mouse pointer over a term or phrase, context-sensitive help is displayed in an adjacent panel.
- Clear this check box if you want to hide the hover help.

3. Click outside the pop-up window to save your settings.

**Parent topic:** [Building rules using the Intellirule editor](#)

**Related tasks:**

[Displaying the completion menu](#)

# Activating and deactivating keyword filtering

You can filter the list of completion menu options to display only those terms and phrases that contain a specific keyword.

## About this task

The completion menu automatically filters the list of terms and phrases as you type text and build your rule. The list becomes progressively more restricted as you continue to type. The way in which automatic filtering works depends on whether you activate or deactivate keyword filtering.

Keyword filtering is deactivated by default. Whenever you type while the completion menu is displayed, the text is entered directly in the editing area, and the list of completion menu options is restricted to terms and phrases that start with the text that you type until you have completed a term or have selected a term or phrase from the completion menu.

When you activate keyword filtering, the completion menu includes a filter field above the list of options. Whenever you type while the completion menu is displayed, the text is entered in the filter field instead of in the editing area and the list of completion menu options is restricted to terms and phrases that contain the filter text. The list includes all items that contain the keyword, not just those that start with the keyword.

You can toggle between activating and deactivating keyword filtering as you build your rule.

## Procedure

To toggle between activating and deactivating keyword filtering:

Click the **Toggle Keyword Filtering** button  at the top of the completion menu.

## Results

When you activate keyword filtering, the **Filter** field is displayed above the list of completion menu options. When you deactivate keyword filtering, the **Filter** field is hidden from the completion menu.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Highlighting and correcting errors using the Problem List

The Intellirule editor highlights errors in red in the editing area and lists them in the Problem List under the editing area.

### About this task

If the Problem List is too small or not visible at all, you can increase its size by dragging the horizontal dividing line of the editing area.

### Procedure

To identify and correct an error in a rule:

1. Do one of the following procedures:
  - Hover the mouse over a highlighted error in the Intellirule Editor to display an error tip icon.
  - Double-click the text displayed in the Message column on an error line in the Problem List to locate the error in the editing area.
2. Correct the error manually by typing directly in the editing area.

**Parent topic:** [Building rules using the Intellirule editor](#)

## Editing decision tables

When you have created your decision table and its properties, you can use the decision table editor.

### **Decision table editor**

You edit a decision table by clicking on the row, header, or cell you want to edit, which displays the editor with the appropriate content.

### **Adding, removing, and editing columns**

You can modify the column structure of your decision tables to define the conditions and actions of all the rules in the table.

### **Adding, removing, and editing rows**

You can add rows to and remove rows from a decision table, and edit the contents of a row using the decision table editor.

### **Editing cell values**

You can edit decision table cells directly in the table, or using the decision table editor.

### **Disabling action cells**

You can disable action cells so that the action is not carried out for that particular row.

### **Applying cell format**

You can implement special formatting on cells or columns in the Rule Designer developer environment.

### **Defining preconditions**

You can define preconditions in your decision tables.

### **Consistency checks in decision tables**

As you submit values into decision table cells, Decision Center automatically analyzes the rules for consistency.

**Parent topic:** [Working with the editors](#)

## Decision table editor

You edit a decision table by clicking on the row, header, or cell you want to edit, which displays the editor with the appropriate content.


Condition editor

✕ ✓ the amount of the loan [±] is at least ▼ 300000 [±] and less than ▼ 600000 [±]

📘 Use this editor to edit the selected cell.

	Grade	Amount of loan		Insurance required
		Min	Max	
1	A	< 100,000		false
2		100,000	300,000	true
3		300,000	600,000	true
4		≥ 600,000		true
5	B	< 100,000		false
6		100,000	300,000	true
7		300,000	600,000	true
8		≥ 600,000		true
9	C	< 100,000		true
10		100,000	300,000	true
11		300000	600000	true
12		≥ 600,000		true

The editor displays different buttons in its toolbar and different entries to modify, depending on whether you select a cell in a condition column, an action column, a row, or column header.

If you have many values to modify, you can edit the cells in “edit all” mode. To activate edit all mode, click  in the editor.

To commit all values that you modify, whether in the editor or in the table in edit all mode, click the **SUBMIT** button.

You can also use in-place editing, as described in [Editing cell values](#).

**Note:**

When a decision table opens, it shows only the first rows (1 - 16). Click **16 - 31** at the top of the table to display rows sixteen to thirty one, and so on. Click **All** to add a scroll bar so that you can scroll through all the rows.

## Locks

A lock on a column indicates restrictions on what can be done in this column. For example, a lock might be set to prevent editing values or adding rows. You set or release locks through the options panel for each column, or through table properties for global locks.

**Parent topic:** [Editing decision tables](#)

# Adding, removing, and editing columns

You can modify the column structure of your decision tables to define the conditions and actions of all the rules in the table.





## About this task

You can have any number of columns a decision table. However, tables with too many columns are difficult to read. As best practice, use several smaller tables instead of one very large table.

You start all changes by clicking on a column header to display the column information in the editor. You can then make various changes in the editor, such as modifying rule statements, adding and removing columns, and changing column headings. You commit changes globally by clicking **SUBMIT**.



## Procedure

To modify a column:



1. To modify the rule statement corresponding to the column:
  - a. Click the column header.
  - b. Click the rule statement in the Condition Column editor and modify the entries.
  - c. Click the **SUBMIT** button  to validate the change.
2. To change the header title:
  - a. Change the name that displays as the column title by typing in the title cell for that column.  
For columns with subtitles, change the subtitles by typing in the subtitle cell.
  - b. Click anywhere outside the cell to validate the change.
3. To add a new column:
  - a. Decide where you want to insert the new column and then click the header in the adjoining column.
  - b. In the toolbar, click the required icon:
    - Click  icon to add a column after the current one.
    - Click  to add a column before the current one.
  - c. Click the header of the new column and use the editor to build the statement.
4. To delete a column:
  - a. Click the header of the column you want to delete.
  - b. In the toolbar, click .

The column is immediately deleted.
5. To sort columns:

The editor sorts the entire column or within groups of related cells.

- a. Click the header of the column you want to sort.
  - b. In the toolbar, click the required sort icon:
    - Click  to sort the cells alphabetically in descending order.
    - Click  to sort the cells alphabetically in ascending order.
6. To move an action column:
  - a. Click the header of the column to move.

**Note:**  
You can move only action columns.

- b. In the toolbar, click the move icons  and  to move the action column sideways.
7. To set consistency checking options:
  - a. Click the header of the column.

**Note:**  
You set consistency checking at individual column level.


- b. Click **Show options** > > to display the option panel.

- c. Select the check boxes corresponding to the consistency checking you want: overlap, symmetry, and gaps.

For information about the checks you can set up, refer to [Consistency checks in decision tables](#).

- d. Click **SUBMIT** to validate.

8. To resize columns:

- a. In the editor toolbar, click  to access the table properties.
- b. Clear the **Auto resize table** check box and then click **SUBMIT**.
- c. Click the column headings.
- d. Click **Show options** > > to display the option panel.
- e. Enter the size in the **Width** field and then click **SUBMIT**.

**Parent topic:** [Editing decision tables](#)

**Related concepts:**

[Columns](#)

[Rows and cells](#)

[Preconditions](#)

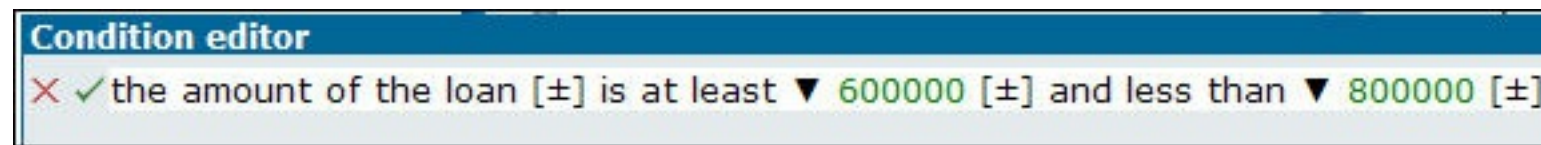


## Adding, removing, and editing rows

You can add rows to and remove rows from a decision table, and edit the contents of a row using the decision table editor.

### About this task

To edit a row, click in the decision table on the appropriate cell in the row. The editor opens, showing the information for the cell you clicked on, and the toolbar displays the tasks you can carry out:



#### Note:

You cannot edit rows if the column is locked. You set or release locks through the options panel for each column, or through table properties for global locks.

Click any row number to display the rule in the editor in its usual if-then form.

### Procedure





The editor toolbar provides various options for adding, deleting and editing rows in decision tables.

1. To add a row:

- a. Click the appropriate cell in the row before or after the position in which you want to add the new row.

Clicking the correct cell is particularly important when you add a row from a grouped cell.

- b. In the editor toolbar, select how to add the row:

-  - Inserts the row above the cell you select.
-  - Inserts the row under the cell you select.
-  - Inserts an Otherwise row under the cell you select.
-  - Inserts as many rows as there are remaining possible values for that column. This button is available only when the column contains grouped cells.

- c. Complete the cells of the new row.

2. To delete a row:

- a. Click the appropriate cell in the row you want to delete.

Clicking the correct cell is particularly important when you delete a row from a grouped (partitioned) cell.

- b. In the editor toolbar, click .

3. To move a row up or down:


- a. Click a cell in the row you want to move. The chosen cell must be in a condition column.

- b. In the editor toolbar, click  to move the row up or  to move the row down.

If you click a grouped cell, the row can be moved only within its partition.

4. To split a row:



- a. Click a cell in the row you want to split. The chosen cell must be in a condition column that has subcolumns.

- b. In the editor toolbar, click  to split the row.

- c. Complete the empty values.

5. To merge rows:

- a. Click a cell in the row you want to merge. The chosen cell must be in a condition column, and the row itself must be completed.

- b. In the editor toolbar, click  to merge the row with the one above it, or  to merge the row with the one under it.

When you merge rows, the values in the condition part merge, but the merged row retains the values of the row that was above for the action parts of the row.

**Parent topic:** [Editing decision tables](#)

**Related concepts:**

[Columns](#)

[Rows and cells](#)


[Preconditions](#)

# Editing cell values

You can edit decision table cells directly in the table, or using the decision table editor.


## About this task

You can edit cells in a number of different ways:

- Clicking the cell, changing the value in the editor (located above the decision table), and then clicking **SUBMIT**.
- Clicking the edit all table icon  in the editor, changing values in the table, and then clicking **SUBMIT**.
- Changing the value directly in the table. If you enter an erroneous value, the error displays in the Edit bar as you type.

**Note:**

You can lock a column to prevent users from editing or disabling it.

To delete the contents of a cell, click  in the toolbar.

## Procedure

To edit a cell:

1. Enter the value in the cell:

You can carry out the following types of edit:

- If the cell contains a text or number field, type in the new value in the field displayed.
- Select predefined values from a drop-down list.
- Change the operator that acts on a given cell.

You can use any of the following formats for values in a decision table:

- Value
- Numeric value with the following operators: =, !=, >, >=, <, <=
- Set of numeric or non-numeric values: {10, 15, 18} {New York, Rhode Island, Massachusetts}

**Note:**

You cannot select predefined values from a set directly in a cell. To select values, click the cell, and then use the editor (located above the decision table) to select the required values.

- Range of numeric or non-numeric values: [12..15] [A..F]

2. To commit the modified value, click **Next** or **Finish** in the Compose wizard. Click **Cancel** if you want to cancel all your current edits.

**Parent topic:** [Editing decision tables](#)

**Related tasks:**

[Creating business rules](#)

**Related information:**

[Decision tables](#)

## Disabling action cells


You can disable action cells so that the action is not carried out for that particular row.

### About this task

This is equivalent to clearing the contents of the cell, except that you keep the information the row contains in case you need to re-enable it later.

### Procedure

To disable an action cell:

1. Click in the action cell you want to disable.
2. In the editor toolbar, click  to disable the action cell.

This is a toggle key, so if you click it again you re-enable the action cell.

### Results

The action cell is disabled and the disabled icon displays in that cell.


**Parent topic:** [Editing decision tables](#)

## Applying cell format

You can implement special formatting on cells or columns in the Rule Designer developer environment.

### Procedure

To display existing cell or column formats:

1. In the editor toolbar, click  to access the table properties:
2. In the editor, check **Apply cell formatting**.
3. Click **SUBMIT** to validate.

**Parent topic:** [Editing decision tables](#)

## Defining preconditions

You can define preconditions in your decision tables.


### About this task

Decision tables contain preconditions that you can use to set variables and conditions:

- You can set variables that you use in the decision table.
- You can set a condition that applies to the entire decision table. If the precondition is not satisfied, none of the rules in the decision table are evaluated.

### Procedure

To define preconditions:

1. In the editor toolbar, click  to access the table properties:
2. In the editor, build your preconditions in the same way that you build them in the guided editor.
3. Click **SUBMIT** to validate.

**Parent topic:** [Editing decision tables](#)

### Related concepts:


[Columns](#)

[Rows and cells](#)

[Preconditions](#)

# Consistency checks in decision tables

As you submit values into decision table cells, Decision Center automatically analyzes the rules for consistency.

The affected cells display a squiggly red line. In addition, the column header displays the error icon  and a message appears in the option panel of the editor.

The following figure shows errors on some cells in a column:

300,000	600,000	true	0.008
600,000	800,000	true	0.012
≥ 600,000		true	0.014

You can specify which types of analysis to perform on a column by clicking the column header and selecting the appropriate check boxes displayed in the editor. You can check columns for overlap, gaps, or symmetry, as described in [Rule verification](#). You set consistency checking capabilities on a per-column basis, and only for condition columns.

A decision table that has a consistency error does not execute unless you disable consistency checking.

**Parent topic:** [Editing decision tables](#)

**Related tasks:**  
[Creating business rules](#)

**Related information:**  
[Decision tables](#)

## **(Deprecated) Editing decision trees**

You can use the decision tree editor when the decision tree and its properties have been created at the project level, that is, in the Tree part of the Compose wizard.

### **Decision tree editor**

You edit a decision tree by clicking on the part of the decision tree you want to edit.

### **Declaring condition nodes**

You must declare the initial condition of all your rules before you can add other conditions to a branch.

### **Working with branches**

You set up branches by setting the value of the condition for the branch, adding other conditions nodes if required, and setting actions.

### **Defining preconditions**

You can define preconditions that apply to all the rules of the decision tree.

### **Decision tree display**

Decision trees are designed to make it easier for you to view and manage large sets of business rules.

### **Consistency checking in decision trees**

You can set consistency checking on a decision tree or its individual conditions.

**Parent topic:** [Working with the editors](#)



## Decision tree editor

You edit a decision tree by clicking on the part of the decision tree you want to edit.

Clicking the tree highlights the part in blue and the editor displays the appropriate content.

The editor displays different buttons in its toolbar and different entries to modify, depending on whether you select a condition node, a branch, or the actions.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**  
[\(Deprecated\) Decision tree overview](#)

**Related tasks:**  
[Creating business rules](#)


# Declaring condition nodes

You must declare the initial condition of all your rules before you can add other conditions to a branch.

## About this task

Initially, a decision tree contains an empty condition node that you must declare so that it becomes the first condition of the rules in your tree.

When you have a condition node declared you can add branches to the node, and add and delete other condition nodes:

Action	Description
Add branches to the node	You can add different types of branches, as described in <a href="#">Working with branches</a> .
Add a condition node before an existing node	<p>You do this by selecting the node and clicking . You can access this icon on the toolbar or directly in the decision tree.</p> <p>When you add a condition node from an existing condition, you insert the new node before the existing one. The condition thus applies to all the branches of the existing node. To add a condition node after a node, so that it applies only to that branch, add it from the branch itself.</p>
Delete a condition node	<p>To delete a condition node, delete all the branches stemming from that node. When you delete the last branch, you also remove the condition node.</p>

## Procedure

To declare a condition node:

1. Click the condition node (anywhere in the yellow diamond).
2. In the editor, use the drop-down lists to declare the condition.  
  
Only the value of the condition remains empty, as this is done in the different branches.
3. In the **Label** field, enter the name you want to be displayed in the tree for the node.  
  
If you leave this field blank, the label adopts the entire name of the condition.
4. Set any consistency checking you want to have carried out on the branches of that node.  
  
These checks can be different from the checks on the rest of the tree.
5. Click **SUBMIT** to commit your changes.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**  
[\(Deprecated\) Decision tree overview](#)

**Related tasks:**  
[Creating business rules](#)

**Related information:**  
[Consistency checking in decision trees](#)



# Working with branches

You set up branches by setting the value of the condition for the branch, adding other conditions nodes if required, and setting actions.

When you create a condition node (see [Declaring condition nodes](#)), it has an empty branch stemming from it. You must set up this branch and any other branches that you add. For each branch, you must set the value of the condition for the branch and set actions. If required, you can also add other condition nodes.





## Set up a branch

You set up a branch as follows:


Action	Procedure
Set the value of the condition for the branch	<div><div>1. Select the branch by clicking on the arrow or the arrowhead.</div><div>2. In the editor, in the <b>Label</b> field, enter the name you want to appear on the arrow in the tree.</div><div>3. Set the value of the condition for that branch and then click <b>SUBMIT</b> to commit your changes.</div></div>
Set the actions of the branch	<div><div>1. Click the actions located at the end of the branch.</div><div>2. In the editor, in the <b>Label</b> field, enter the name you want to appear for the set of actions in the tree.</div><div>3. Create the actions and then click <b>SUBMIT</b> to commit your changes.</div></div> <div>When you have set the first action, you can add other actions to the branch by clicking . You can access this icon from the toolbar or the tree.</div> <div>You can remove individual actions from the set of actions by selecting the action in the editor and clicking the X next to it.</div>
Add other condition nodes you want to include in the branch	To add a condition node to the branch, select the branch (either the arrow or the actions) and click  . You can access this icon from the toolbar or the tree.

## Add branches

You must set up the branches for every branch that you add to your tree. You can add branches in various ways:

Branch	Description
Single branch	Select the condition node and then click  . You can access this icon from the toolbar or the tree.
Otherwise branch	<div>Select the condition node and then click  in the toolbar. The new branch displays the Otherwise label.</div> <div>An ‘Otherwise’ branch caters for the case where none of the other possibilities for that condition are correct. You can remove the ‘Otherwise’ status by clicking the same icon that you used to set it.</div>
Multiple branches	<div>Select the condition node and then click  in the toolbar. The new branches display the appropriate labels and empty actions are added to the end of each branch.</div> <div>You can create multiple branches only if the selected condition node allows it, that is, if a predefined list of items already exists. For example, the loan grade is ‘A’, ‘B’, ‘C’, or ‘D’.</div>
Duplicate a branch	<div>You can add a branch by duplicating an existing one and then modifying the new one. To duplicate an existing branch, select the branch and then click  in the toolbar.</div> <div>If the overlap check is active for the condition node, you get an error for both branches until you modify one of them.</div>

## Delete branches

You delete a branch by selecting the branch (the arrow or the actions) and then clicking  in the toolbar.

When you delete a branch, it immediately deletes any condition nodes and any subbranches that stem from them.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**  
[\(Deprecated\) Decision tree overview](#)

**Related tasks:**  
[Creating business rules](#)

# Defining preconditions

You can define preconditions that apply to all the rules of the decision tree.


## About this task

Decision trees can contain preconditions for you to perform the following actions:

- Set variables that can be used in the decision tree. For information about setting variables, see [Rule variables](#).
- Set a condition that applies to an entire decision tree. If the precondition is not satisfied, none of the rules in the decision tree can be evaluated.

## Procedure

To define preconditions:

1. Click the **Edit Properties** button  in the toolbar.
2. In the editor, build your preconditions as you would in the guided editor.
3. Click **SUBMIT** to validate.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**  
[\(Deprecated\) Decision tree overview](#)





**Related tasks:**  
[Creating business rules](#)

**Related information:**  
[Using the rule editors](#)



## Decision tree display

Decision trees are designed to make it easier for you to view and manage large sets of business rules.

Decision Center provides a number of display options to make it easier for you to view your decision trees:

- To display a decision tree horizontally, click the  icon on the toolbar.
- To display it vertically, click .
- You can also make viewing easier by minimizing the branches or actions: click the  symbol on a condition node. To restore the branch to normal, click the  symbol.

Similarly, you can minimize or restore actions by clicking  or  next to the rule heading.

- To change the position of a branch with respect to the other branches, click  or  to move the branch in the direction indicated.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**

[\(Deprecated\) Decision tree overview](#)

**Related tasks:**

[Creating business rules](#)

## Consistency checking in decision trees

You can set consistency checking on a decision tree or its individual conditions.

You can analyze the rules that make up your decision tree to make sure that there are no gaps and overlaps. You can do this at the tree level and for the branches of a given condition node.


**Note:**

You cannot check dates for gaps and overlaps.

If a decision tree has inconsistencies, for example, two branches that use the same value for a condition, Decision Center displays the 🚫 symbol to identify the affected condition node and its incorrect branches. To display the corresponding error message in the editor, click the condition node or branch.

You cannot execute a decision tree that has an inconsistency error unless you disable consistency checking.

### Tree and node verifications

You control consistency checking for the decision tree in the editing part, which displays when you click the Edit Properties button  in the toolbar. You select the condition node and set the checks. Decision Center then checks for gaps and overlaps on the selected condition nodes.

If you do not select consistency checking at tree level, no analysis is performed on any condition node, regardless of any checks you set for the individual condition nodes.

**Parent topic:** [\(Deprecated\) Editing decision trees](#)

**Related concepts:**

[\(Deprecated\) Decision tree overview](#)

**Related tasks:**

[Creating business rules](#)

## Previewing ruleflows

You cannot edit ruleflows in the Enterprise console.

To create and edit a ruleflow in Decision Center, you must use the Business console. In the Enterprise console, you can only view the ruleflow and its properties, by accessing the **Details** page of your ruleflow. The **Detail** page contains the following details:

- Ruleflow diagram: You can use the toolbar to zoom in and out of the diagram.
- Ruleflow properties
- Ruleflow task information, which is displayed at the bottom of the **Details** page.

**Parent topic:** [Working with the editors](#)

**Related concepts:**

[Overview: Ruleflows](#)

[Project element properties](#)

**Related information:**

[Editing ruleflows](#)



## Configure: Manage your project configuration

You use the Configure tab to carry out some administrative tasks.

The Permission manager can access the **Configure** tab to run diagnostics, clean the cache, export and erase current projects.

### Running diagnostics

In case of unexpected errors, the error message can include a link to the **Diagnostics** page. You can also run the diagnostics manually.

### Cleaning the cache

You can clean out the files cached on the server.

### Exporting the current project

You can export the working branch of the current project to a .zip file.

### Erasing the current project

You can erase the current project from Decision Center.

**Parent topic:** [Working with the Enterprise console](#)

## Running diagnostics

In case of unexpected errors, the error message can include a link to the **Diagnostics** page. You can also run the diagnostics manually.

### About this task

The diagnostics feature checks the configuration of Decision Center for Manager Bean access and connection to the data source. It shows your rule model extensions and verbalizers.

### Procedure

To run diagnostics:

On the **Configure** tab, click **Diagnostics**.

**Parent topic:** [Configure: Manage your project configuration](#)

## Cleaning the cache

You can clean out the files cached on the server.

### About this task

IRL files generated when you create rulesets are cached on the server in the location specified in `teamserver.war/WEB-INF/lib/teamserver-model-XXX.jar/ilog/rules/teamserver/preferences.properties`.

Over time these files can become obsolete, or you might want to modify the way the IRL is generated. In these cases, you can clean out the cache.

### Procedure

To clean the cache:

On the **Configure** tab, click **Clean Decision Center Cache**.

### Results

This action deletes the files cached on the server for the current project.

**Parent topic:** [Configure: Manage your project configuration](#)

## Exporting the current project

You can export the working branch of the current project to a .zip file.

### Before you begin

You can then import these files into your Eclipse workspace or import them into another Decision Center.

### Procedure

To export the current project:

1. On the **Configure** tab, click **Export Current Project State**.
2. Enter the location to save the file and click **OK**.

**Parent topic:** [Configure: Manage your project configuration](#)

## Erasing the current project

You can erase the current project from Decision Center.

### About this task

You can erase a rule project so that all its entries in the database are permanently removed and it no longer appears in Decision Center.

### Procedure

To erase a project:

1. On the **Configure** tab, click **Erase Current Project**.
2. Click **Yes** to confirm.

### Results

The project is deleted and you are returned to the **Home** page with the next available project displayed.

**Parent topic:** [Configure: Manage your project configuration](#)

## Storing and synchronizing rules

You can store and maintain rules in two different repositories, a Source Control System (SCC) and a database. Therefore, you must synchronize the two rules repositories periodically to bring them at the same level.

### Why synchronize?

As rules can be stored and maintained in two repositories in parallel, you must synchronize these repositories from time to time to keep them consistent.

### Sharing repositories between user profiles

To avoid conflicts when rules are shared between business users and developers, you must clearly establish which is the master repository: source code control (SCC) or the Decision Center database.

### Synchronization architecture

The synchronization process compares your local Rule Designer workspace, the remote Decision Center database, and a reference that computes the state of the synchronization.

### Branches and releases in synchronization

Synchronization between Rule Designer and Decision Center is done one branch at a time.

### Guidelines for synchronizing BOM changes

You can avoid synchronization errors by following these guidelines for synchronizing BOM changes in rule projects.

### Synchronizing from Rule Designer

With the synchronization commands in Rule Designer, you can create a project from an existing Decision Center project or publish an existing project to Decision Center.

### Synchronization and source code control

Use source code control (SCC) to share Rule Designer artifacts, and commit rule project resources.

### Rule project items

A rule project is a container for organizing rule artifacts and setting up the business object model (BOM) and rule authoring vocabulary. Each rule project item is associated with a folder.

## Why synchronize?

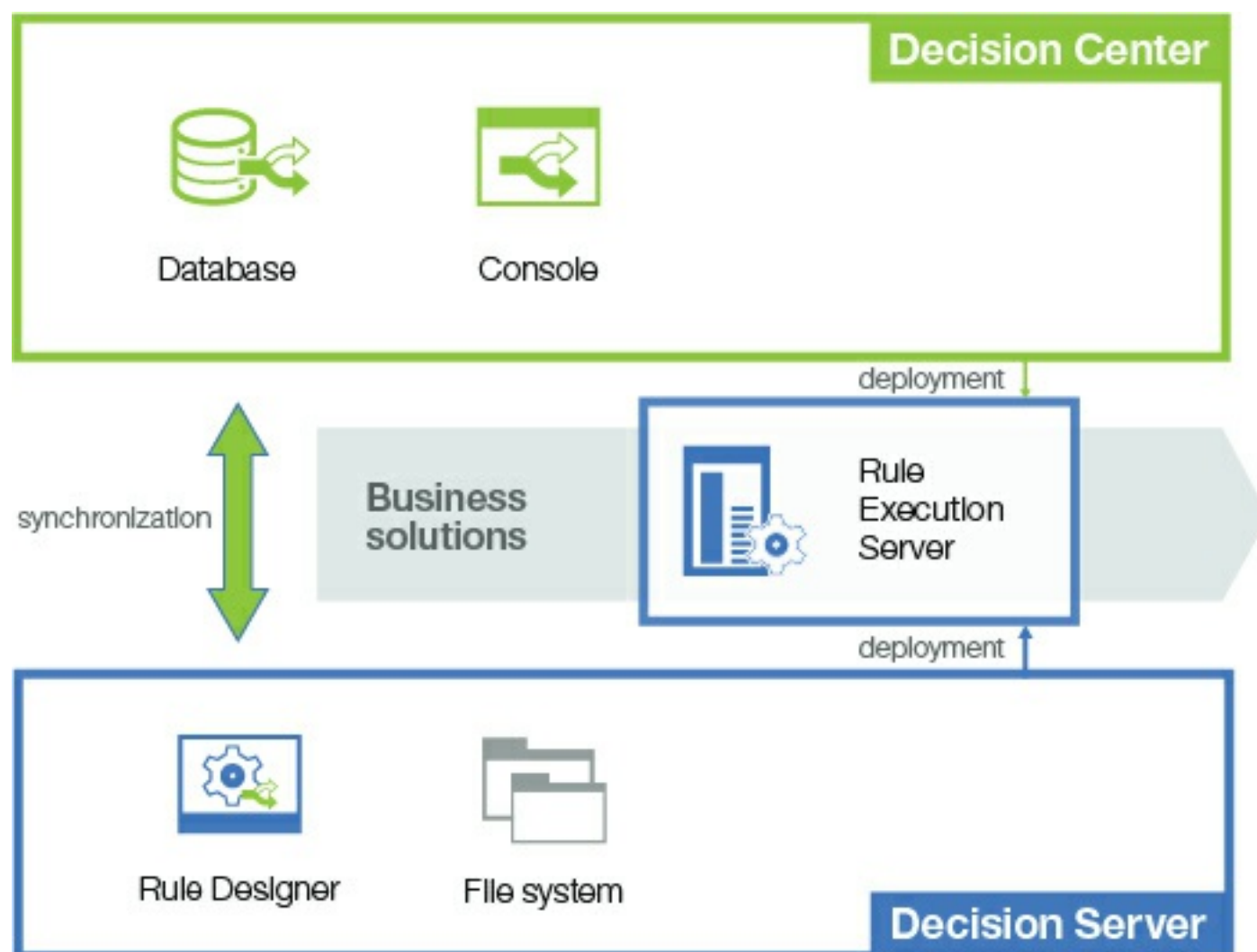
As rules can be stored and maintained in two repositories in parallel, you must synchronize these repositories from time to time to keep them consistent.

At the beginning of a decision rule development project, developers create rules using the Eclipse-based Rule Designer application, and store them in a **source control system** (SCS) to share files, handle versions, and resolve potential conflicts that can arise when several users are committing changes to the same repository .

At a later point in time, developers must push rule projects or decision services developed in Rule Designer to the Decision Center environment to make them available to business users (rule authors and policy managers). In Decision Center, rule projects or decision services are stored in a **database**. Decision Center handles concurrent accesses to the database and versioning.

When a rule project or decision service is both maintained in Rule Designer and Decision Center, it is necessary to synchronize them periodically to bring them at the same level. Synchronization is a process that can be done either manually or automatically with ant tasks, and is always initiated from Rule Designer.

The following illustration shows how rule projects or decision services are synchronized between Rule Designer and Decision Center.



**Parent topic:** [Storing and synchronizing rules](#)

### Related information:

[Sharing repositories between user profiles](#)

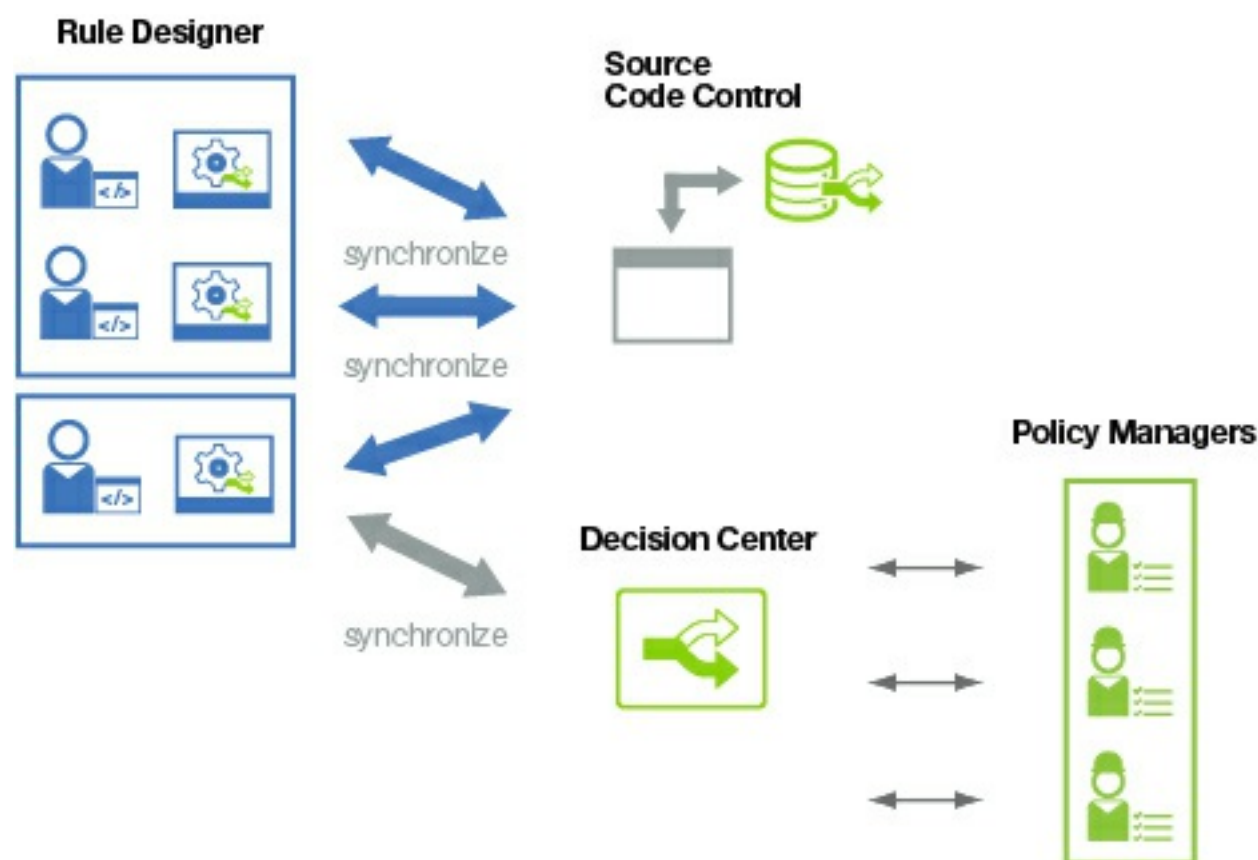
[Synchronizing by using Ant tasks](#)

[Synchronizing from Rule Designer](#)

## Sharing repositories between user profiles

To avoid conflicts when rules are shared between business users and developers, you must clearly establish which is the master repository: source code control (SCC) or the Decision Center database.

The interaction between the two user profiles is shown in the following figure. However, the choice of master repository depends on whether you are in the initial or later stages of project development:

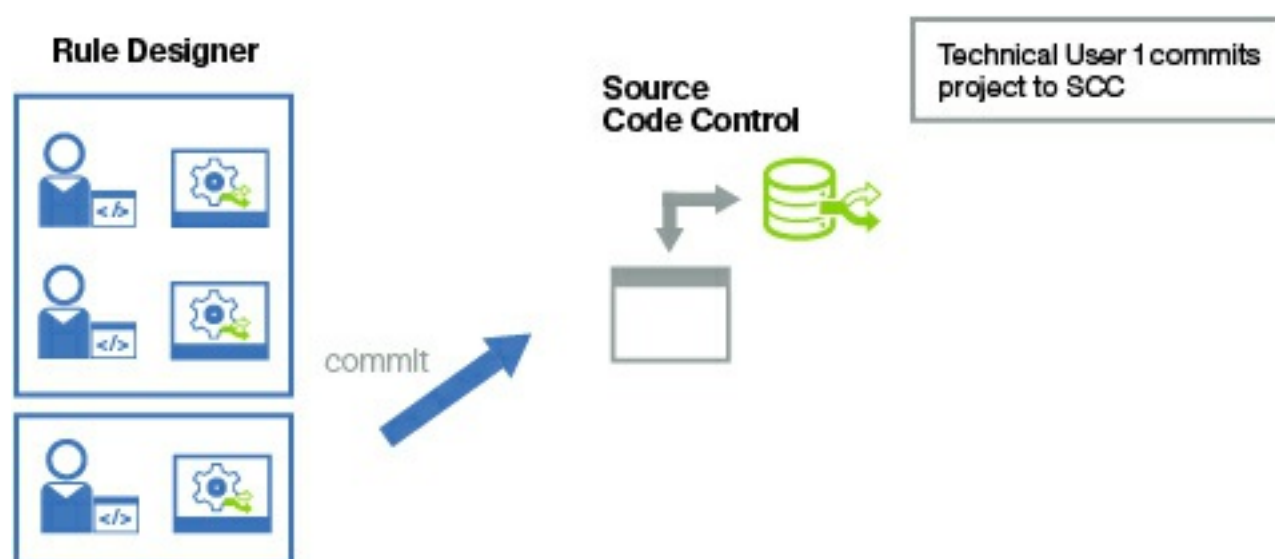


- A **developer-centric approach** is better adapted for an initial development situation. Technical users synchronize their rules through SCC. One dedicated technical user publishes to Decision Center for testing, and synchronizes any changes between the two repositories (see [Rule project items](#)).
- A **business user-centric approach** is better adapted for the later stages of project development, when the business object model stabilizes. You consider the Decision Center repository to be the source of truth and any Rule Designer developer copy in SCC as a temporary copy of the project.

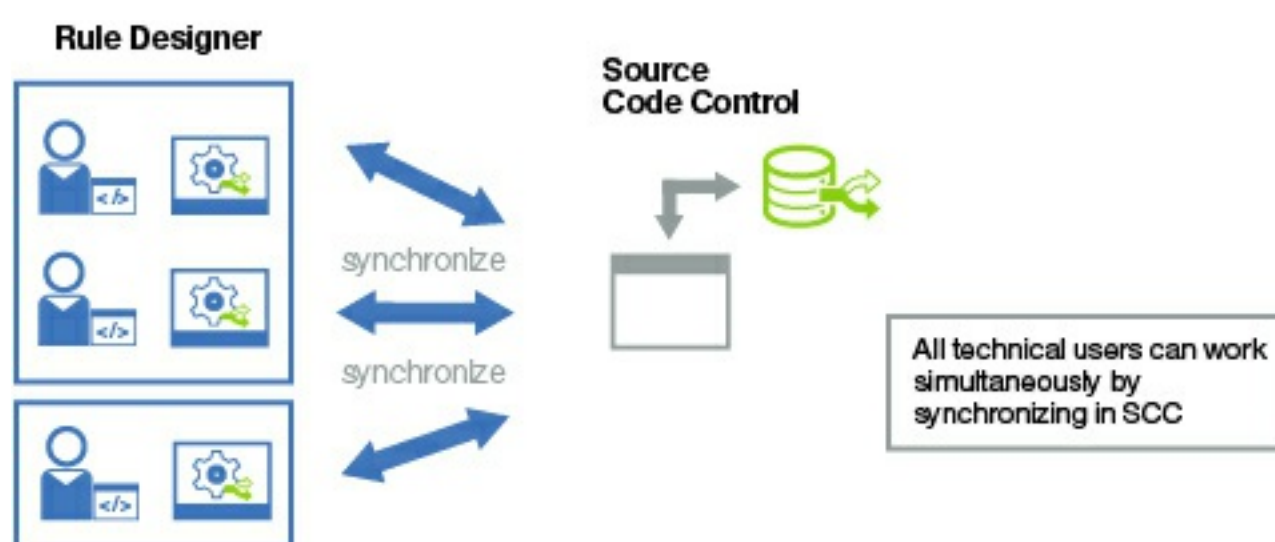
## Project lifecycle and the choice of master repository

In the early phases of a project, developers use Rule Designer to develop a business object model and other project artifacts. During this phase, you can maintain the project entirely in Rule Designer by using SCC to control multiuser updates as follows:

- A technical user (Technical User 1) creates the project in Rule Designer and then commits it to SCC.



- From that point, other developers can retrieve the project from SCC and start to work. All technical users work simultaneously by synchronizing through SCC.

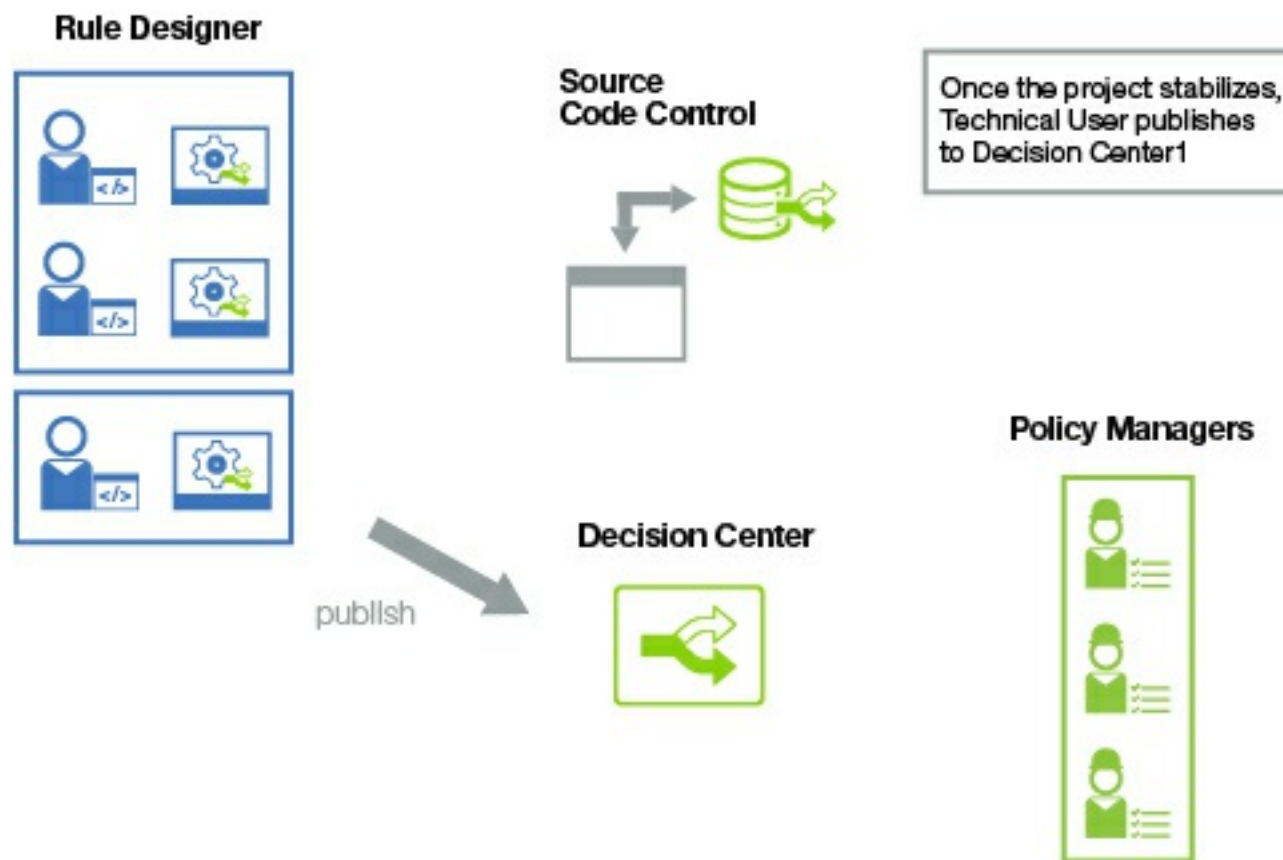




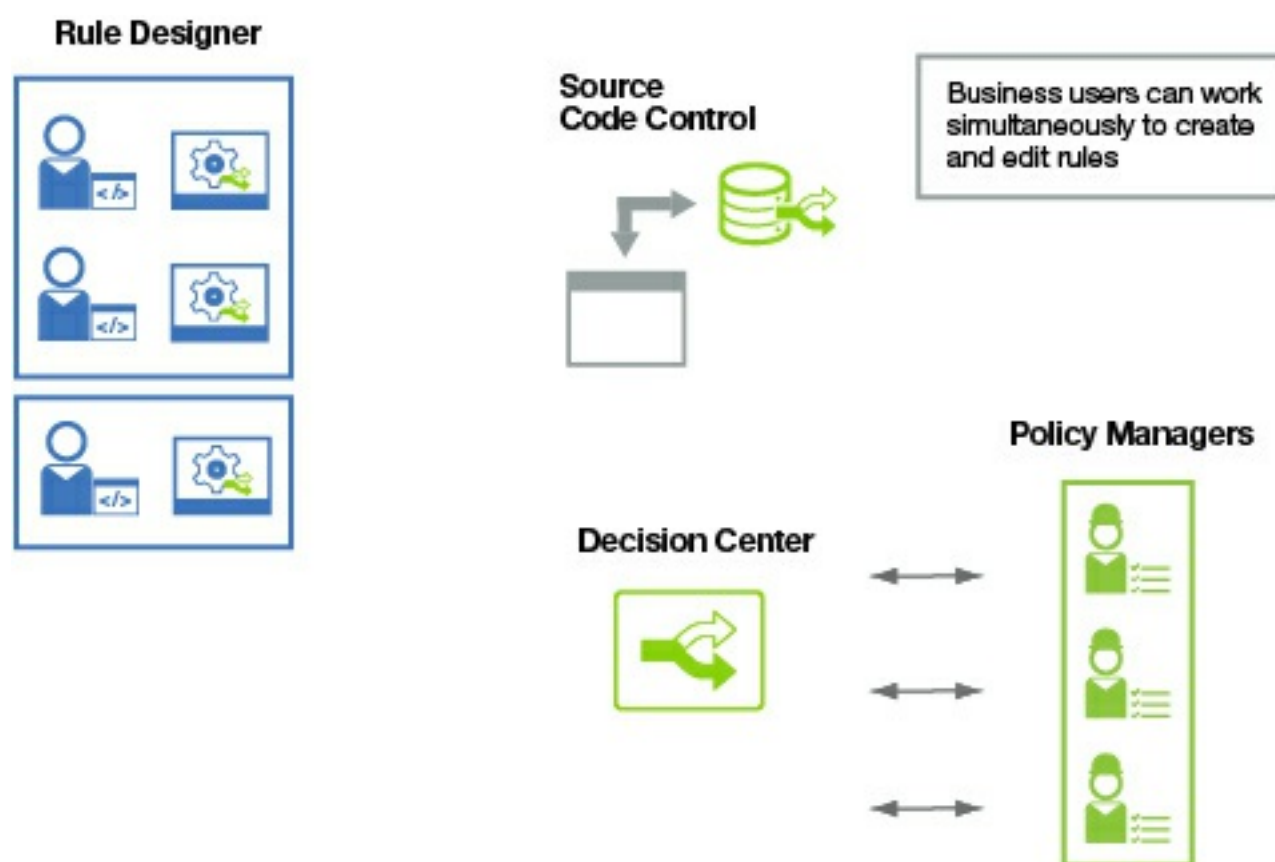
As the business object model begins to stabilize, business users start authoring rules.

To support the business user, Technical User 1 publishes the project to Decision Center, which becomes the copy of record for rules:

- Technical User 1 publishes to Decision Center.



- Business users can then work with the project in Decision Center to create and edit rules.



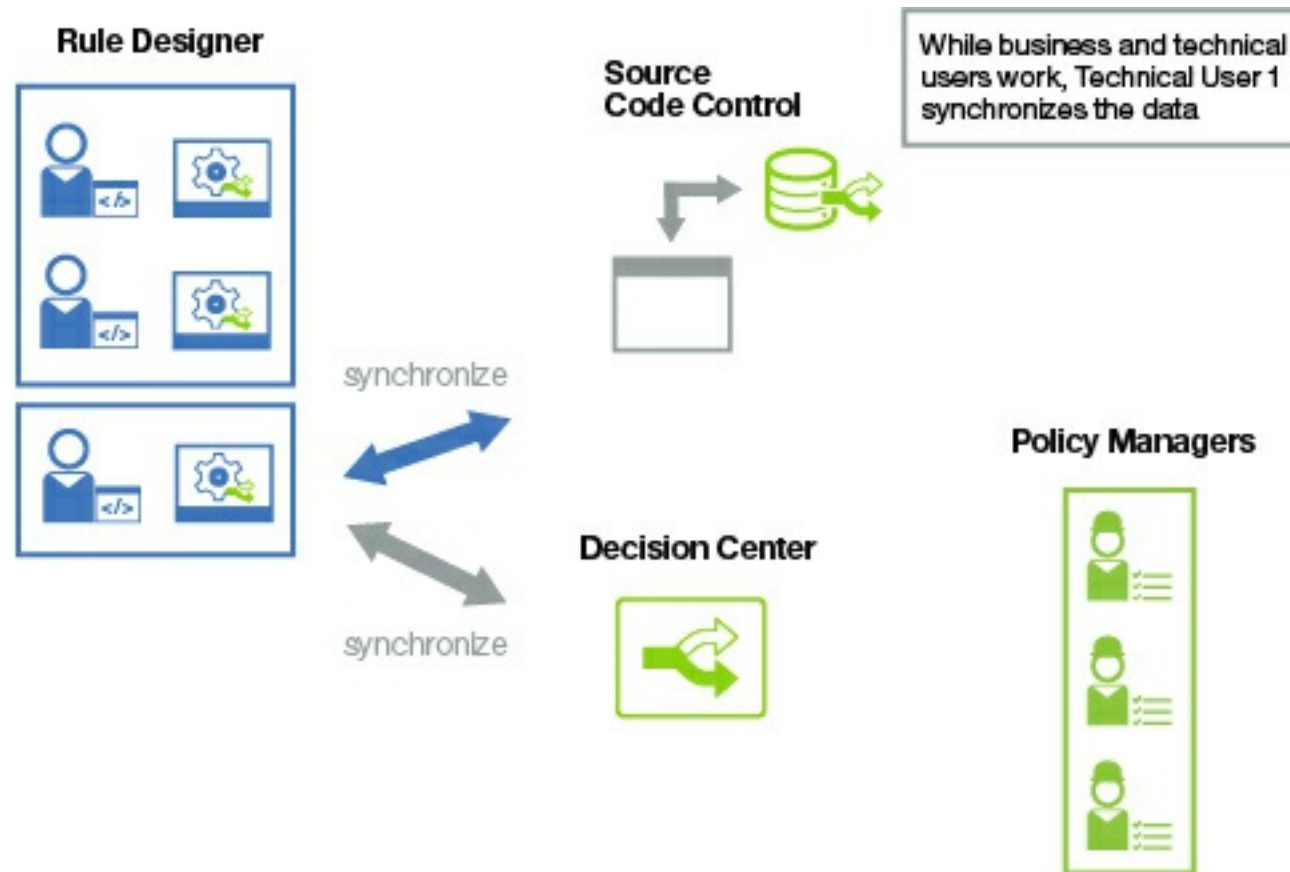
Control of the business object model now remains with the rule project in Rule Designer and developers must maintain it. To keep the developer view of the project in Rule Designer synchronized with the work of the business users, developers periodically update the rule project from Decision Center. Changes to the business object model are likewise published to Decision Center so that business users can work with the latest vocabulary.

**Note:**

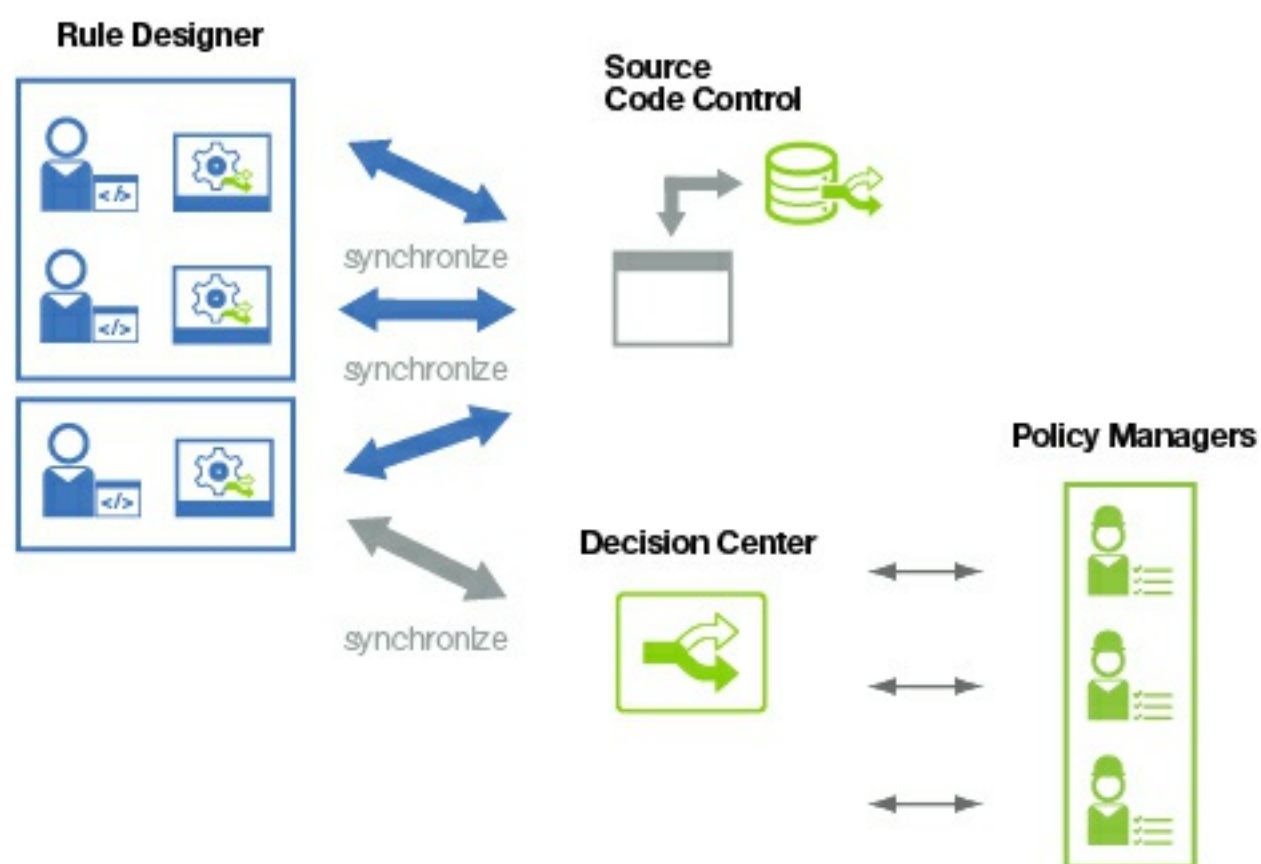
When BOM changes affect the vocabulary that is used in existing rules, you must bring these rules into Rule Designer for refactoring (see [Guidelines for synchronizing BOM changes](#)).

You maintain synchronization as follows:

- Technical User 1 periodically synchronizes the data for all the users.



- All the business and technical users work simultaneously, but Technical User 1 has the added role of synchronizing between the two user profiles.



When you send the project to production, the Rule Designer copy of the project is essentially in an archive state, and not required for day-to-day management of the rules. Afterward, business users continue to author and modify rules, and to deploy them to production as the business cycle dictates. The Decision Center repository remains the source for auditing all the policy changes that are implemented by rules.

When you want to develop the application further, you create a new branch in the development SCC system. You then populate it with the rule portion of the project by creating a new copy of the production rule project from Decision Center. From this point, development starts again.

**Parent topic:** [Storing and synchronizing rules](#)

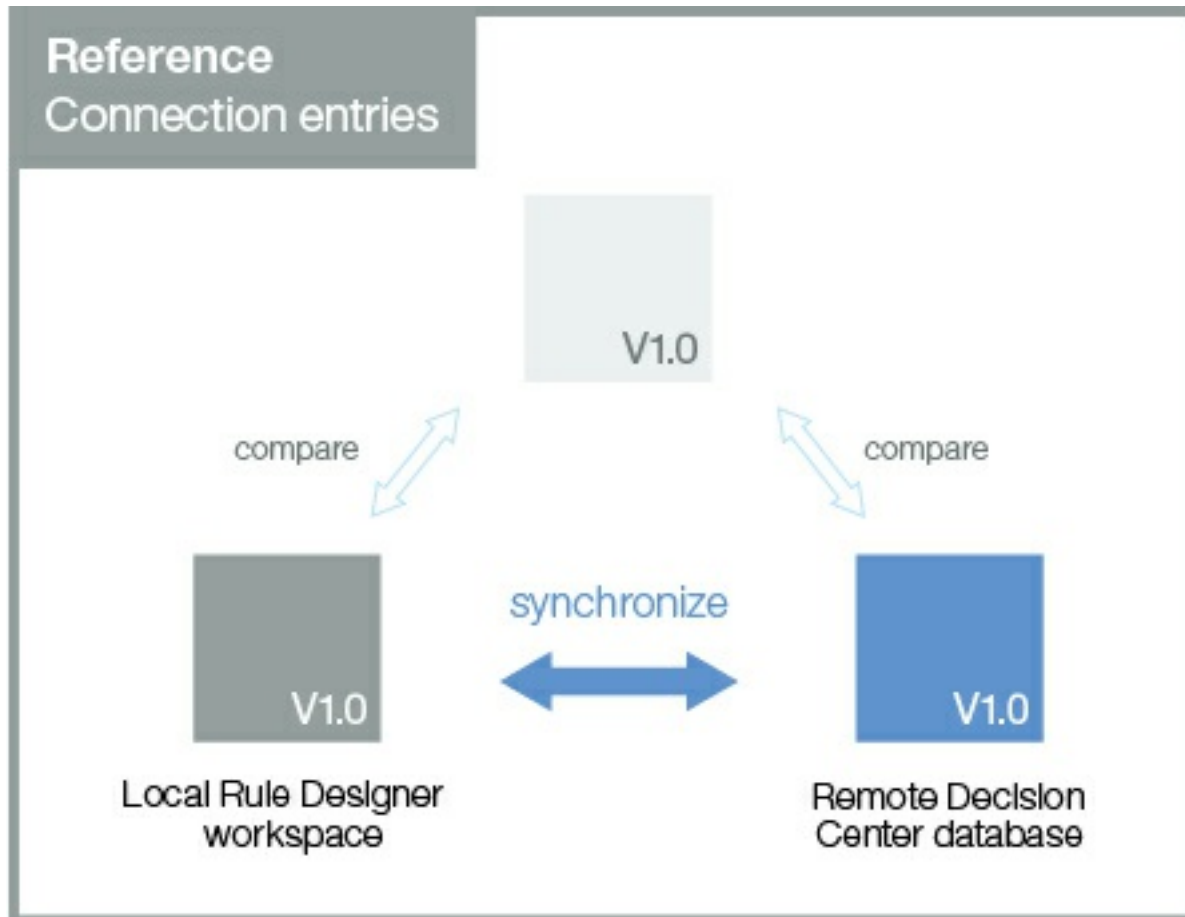
**Related concepts:**  
[Branches and releases in synchronization](#)

**Related information:**  
[Synchronization architecture](#)

## Synchronization architecture

The synchronization process compares your local Rule Designer workspace, the remote Decision Center database, and a reference that computes the state of the synchronization.

This reference state is created as a connection entry in your workspace when you connect to Decision Center.



The Synchronization tool in Rule Designer queries the Decision Center database remotely. To improve performance, the three-way comparison uses a checksum on both the remote and the local rules, and then compares them to the reference state.

Three-way comparison indicates the direction of changes:

### Incoming

Changes in Decision Center that you must update in Rule Designer.

### Outgoing

Changes that are made locally in Rule Designer that you must publish to the remote Decision Center.

### Conflict

Changes made to the local and remote versions.

If the Synchronization view advises any changes, you update or publish the changes as indicated, or override the proposed direction.

#### Note:

- If you rename a folder or subfolder in Rule Designer and synchronize, you obtain a new version in Decision Center for every artifact in that folder.
- The synchronization process is based on a user and a server. For example, if you synchronize between Rule Designer and Decision Center, then disconnect your project from Decision Center, and reconnect with another user credentials or connected to another server, then the synchronization can display conflicts even if no changes were made to your decision service or project.

## Connection entries in synchronization

When you connect to Decision Center, a connection file (.syncEntries) is created in your workspace for the project.

The connection file tracks the state of the synchronization with one or more Decision Centers if you connect to more than one.

#### Important:

You must keep this entry to retain the three-way comparison that gives you the state of the synchronization.

When you close your Eclipse session, the connection file remains in your workspace, so you do not need to reconnect the next time you open Rule Designer.

When you disconnect from one Decision Center to connect to another one, you must keep the connection file of the first, unless you plan never to reconnect.

If you delete the connection entry and later reconnect and synchronize with the same project, the Synchronize view displays conflicts, even though the rules are the same on both sides, because you lost the checksum information when you deleted the entry. In this case, you can delete the project in Rule Designer and reimport it from Decision Center.

Only delete the connection entry if you do not need to reconnect, or if you want to clean the connection files, such as when you send a rule project (not the workspace) to another user.

## UUIDs in synchronization

Synchronization uses Universal Unique Identifiers (UUIDs) to determine whether rules in Rule Designer and Decision Center are synchronized.

If you manually change the UUID of a rule in Decision Center, you get conflicts when you synchronize. For example, developers might change a UUID when they share rule projects through source code control (SCC) and want to work on different branches of a project in the same workspace. This approach breaks the synchronization. You must import different branches into separate workspaces to keep the synchronization.

When you copy a rule in Rule Designer as a starting point for another rule that you rename, copy it from the Rule Perspective to automatically change the UUID of the copied rule. Synchronization then detects the new rule on the Rule Designer side.

If you copy a rule in the Resource view or Windows Explorer, the copied rule has the same UUID as the original rule, and the copy process raises an error. To correct this error, right-click the copied rule in the Rule Explorer and click **Update UUIDs**.

### Note:

If you want to check the UUID of a rule, go to the `.brl` view in Rule Designer. To show the UUID in the Decision Center rule tables, click **Options** in the top banner of the Enterprise console.

**Parent topic:** [Storing and synchronizing rules](#)

### Related concepts:

[Synchronization commands in Rule Designer](#)

### Related tasks:

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

## Branches and releases in synchronization

Synchronization between Rule Designer and Decision Center is done one branch at a time.

In Rule Designer, you can have only one branch in your workspace at any given time. To synchronize with the different branches in use in Decision Center, connect, disconnect, and reconnect with each branch individually. When you disconnect, keep the connection entries for that branch. These entries are stored as part of the branch, allowing you to reconnect with that branch without introducing conflicts.

### Releases and change activities

In Decision Center, releases and change activities are used within the decision governance framework, a ready-to-use approach to change management and governance. The first publishing of a decision service creates an initial, closed release in Decision Center. Subsequent changes to the decision service in Decision Center take place in releases that are created from this initial release. In the Business console, changes to a release can only be done in change activities. Releases and change activities are branches of the decision service.

**Note:** If you do not want to use the decision governance framework with your decision services, you can publish to a branch called main.

After the initial publishing you publish changes to open change activities, and not directly to the release branch or to a closed changed activity. Of course you can update changes done in any branch in Decision Center back to your Rule Designer workspace so that all versions are synchronized.

A decision service is designed for several rule projects to be grouped as one entity under a main project. The decision service is assigned the name of the main project. All other projects contained in the decision service contain a direct or indirect dependency to the main project. When you publish from the main project, all dependent projects are also published to Decision Center.

If you need to add a project to the decision service, consider this as a redesign of the decision service. Consequently, you can only add a project to an open release, and not to a change activity. To publish a new project to the decision service, add the project in Rule Designer and set its dependency to the main project. Then connect to the release and publish the project. After the publish is completed, synchronize the decision service and publish the changes relating to the new dependencies.

### Administrator privileges

You can publish changes to a release branch if you connect with administrator rights. This privilege is offered for flexibility, but use it with restraint, because it circumvents the governance framework. Synchronizing to a release is the only way dependencies of a decision service can be changed.

**Parent topic:** [Storing and synchronizing rules](#)

**Related concepts:**

[Synchronization commands in Rule Designer](#)

**Related tasks:**

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

**Related information:**

[Change management](#)



## Guidelines for synchronizing BOM changes

You can avoid synchronization errors by following these guidelines for synchronizing BOM changes in rule projects.

Changes to the BOM in Rule Designer can cause corresponding changes in rules through the Rule Designer refactoring mechanism. You must handle these changes cautiously to maintain synchronization with Decision Center.

For example, consider a project that is worked on and managed simultaneously in Rule Designer and Decision Center:

- The project has a BOM class Customer verbalized as the `customer` with an integer member `age` verbalized as the `age of the {this}`
- The project has the following rule named `Legal Age`:

```
if
 the age of the customer is less than 21
then
 ...
```

If a Rule Designer user changes the verbalization of the class `Customer` to the `client`, Rule Designer refactors the rule `Legal Age` to read:

```
if
 the age of the client is less than 21
then
 ...
```

Suppose that, in the meantime, a Decision Center user creates a second rule named `Senior`:

```
if
 the age of the customer is more than 65
then
 ...
```

If the Rule Designer user updates from Decision Center, the new rule is added to the Rule Designer project, but is in error because “the age of the customer” is no longer part of the vocabulary in the Rule Designer project.

Similarly, if the Rule Designer user publishes the changes in the verbalization, all the rules in Decision Center referring to “the customer” are in error.

These occurrences are a natural consequence of simultaneous development of dependent artifacts; the rules depend on the BOM vocabulary. You can avoid these errors by following the development practices that are described here for synchronous and asynchronous changes.

### Simplest case: synchronous vocabulary change

The simplest approach to managing vocabulary change during simultaneous development is to synchronize changes to the BOM and vocabulary. This approach might not be practical for large projects, but works with smaller groups and small changes after the initial setup of the BOM.

In this case, you make all the changes to the BOM or vocabulary on a fully up-to-date copy of the rule project. Perform the following sequence of events:

1. Before you make the changes, all the developers commit their projects to SCC.
2. A single developer updates a copy of the project from SCC to make it current.
3. The developer updates from Decision Center to get a current copy of all the rule changes.
4. The developer makes the BOM and vocabulary changes, and refactors rules as required.
5. The developer commits the modified project to both SCC and Decision Center.
6. Simultaneous rule development in Rule Designer and Decision Center can then resume.

### Managing asynchronous changes

For large teams, or when you must manage multiple versions of a project for extended periods, implement the following practices:

- Developers who need to make BOM changes must not delete BOM classes or members. Deprecate obsolete classes and members (use the deprecated property in the BOM editor) during ordinary development. Remove them only when a fully synchronized copy of the project is available and all the rules are refactored.
- Developers must avoid changing argument numbers and types in the BOM. If you need a new argument number or a type change, create a new method with the “corrected” arguments and deprecate the old member.
- You do not need to place restrictions when you add classes or members to the BOM. However, you must establish processes to make sure that such additions are made only one time to a single copy of the project, and then propagated by synchronization to avoid having to merge incompatible additions.
- As much as possible, defer vocabulary changes (other than additions) to occasions when you can use a fully synchronized copy of the project so that all the rules are correctly refactored.

Change the vocabulary:

- Deprecate the member for which the vocabulary is to be changed (the “original” member)
- Add a new virtual member on the same class, with the required verbalization (the “new” member)
- Use BOM-to-XOM mapping to map the new virtual member to the appropriate real method in the XOM
- To resolve duplicate verbalizations when you have a fully synchronized version of the project:
  - Remove the new member and save the BOM entry that contains this member. All the rules that use the new verbalization will be temporarily in error.
  - Change the verbalization on the old member to the required new verbalization, and save the BOM entry. Examine all aspects of the verbalization carefully to make sure that it exactly matches the new verbalization that you want. Rule Designer refactors the rules to use the new verbalization and rebuilds the project, clearing the temporary errors from the first step.

**Parent topic:** [Storing and synchronizing rules](#)

**Related information:**  
[Synchronization architecture](#)

## Synchronizing from Rule Designer

With the synchronization commands in Rule Designer, you can create a project from an existing Decision Center project or publish an existing project to Decision Center.

**Note:** For synchronization to work correctly, you must use Rule Designer and Decision Center in the same persistence locale.

### [Synchronization commands in Rule Designer](#)

The Synchronize view in Rule Designer shows where changes in Rule Designer and Decision Center are not synchronized. You can publish, update, or override changes to achieve synchronization.

### [Creating projects from Decision Center](#)

You can create a project or in Rule Designer from an existing Decision Center project.

### [Use of Text Compare in synchronization](#)

Display conflicting rules and their differences in the Eclipse Text Compare Editor.

**Parent topic:** [Storing and synchronizing rules](#)



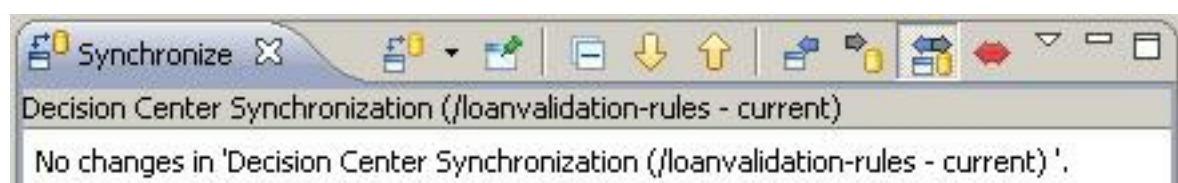
## Synchronization commands in Rule Designer

The Synchronize view in Rule Designer shows where changes in Rule Designer and Decision Center are not synchronized. You can publish, update, or override changes to achieve synchronization.

You synchronize rule projects from Rule Designer. Synchronization starts when you publish a Rule Designer project or decision service to Decision Center, or create a decision service or project from Decision Center.

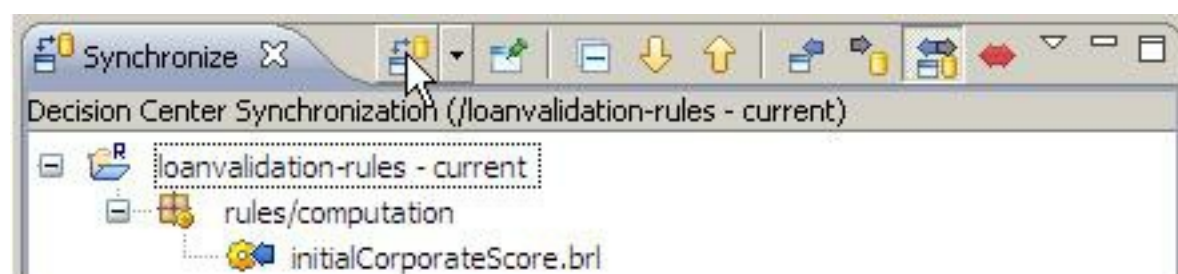
Projects and decision services are synchronized one branch at a time. You should be familiar with section [Branches and releases in synchronization](#) to avoid misuse.

The Synchronize view displays a rule project or decision service branch when you connect to Decision Center:



**Note:** When you upgrade to a new version of Operational Decision Manager, data structure discrepancies might occur when synchronizing for the first time. When a discrepancy occurs, override the older version.

The Synchronize view displays any modifications that are made to either the local Rule Designer or the remote Decision Center versions of the branch. Changes that are made in Rule Designer display automatically, and changes that are made in Decision Center display when you click the **Synchronize** button:



To open the Synchronize view:

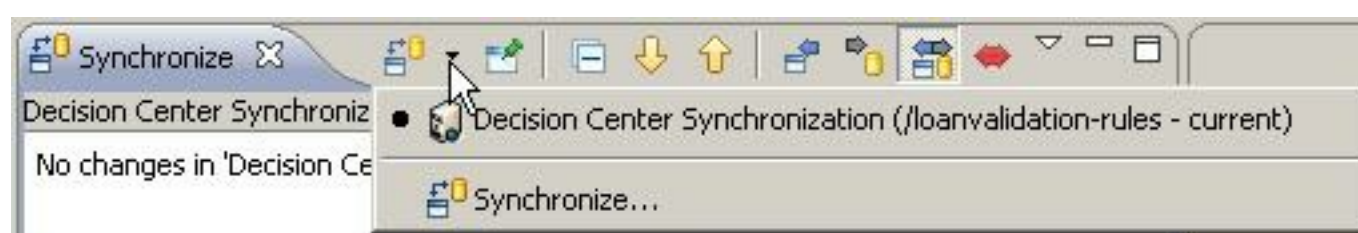
1. In Rule Designer, open **Window > Show View > Other > Team**.
2. Click **Synchronize**.
3. Click **OK** to open the view.

### Selecting a project for synchronization

In the Synchronization view, you can change the project and branch to be synchronized with Decision Center.

To select a project for synchronization:

1. In the Synchronization view, click the down arrow next to the **Synchronize** button to open the synchronization list:



2. In the list, click **Synchronize**.

The Synchronize - Rule Model Synchronization window opens with a list of available types of synchronization.

3. Select **Synchronization**, and click **Next**.

The synchronization window opens with a list of the projects that are currently in the workspace.

4. Select the project branch that you want to synchronize with Decision Center, and click **Finish**.

The Synchronizing message appears, and the selected project and a list of any changes that were made with Decision Center are displayed in the Synchronization view.

### Publish, Update, and Override


Entries in the Synchronize view can indicate that the local and remote versions of your project branch are no longer synchronized. You must publish or update the changes to synchronize the versions.

The Synchronize view includes the project branch and package of each changed artifact. You can publish one or more entries at a time, or all the changes that are detected in a package or project branch together.


Because project branches can be modified locally, remotely, or both concurrently, entries in the Synchronize

view show the direction of the change and an action:


**Publish**

Entries with a black arrow ( repayment.brl (1.0)) indicate that a change occurred in Rule Designer. Publish this change to Decision Center by right-clicking the entry in the Synchronize view and clicking **Publish**.

**Update**

Entries with a blue arrow ( repayment.brl (1.1)) indicate that a change occurred in Decision Center. Update this change back to Rule Designer by right-clicking the entry in the Synchronize view and clicking **Update**.

**Override**

Red entries with double arrows ( repayment.brl (1.1)) indicate that changes occurred in both Rule Designer and in Decision Center. This change corresponds to a conflict, and you must decide how to proceed. You cannot do automatic merging.

To resolve a conflict, you must **override** the proposed direction of a change by right-clicking the entry in the Synchronize view and clicking one of these options:










- **Override and Update** to keep the changes from Decision Center and update them back to Rule Designer.
- **Override and Publish** to keep the changes from Rule Designer and publish them to Decision Center.

**Note:**

If a change made on the BOM affects a rule, the Synchronization view displays both the changed BOM and the affected rule. You can decide whether to publish or update these changes.

In addition to its direction, the decoration of an entry also indicates the nature of the change, such as whether artifacts are added, deleted, or modified. The following table describes the options.

Table shows synchronization view decorations.

De cor ati on	Description
	An incoming <b>(+)</b> means that Decision Center contains a new artifact. <b>Updating</b> transfers the artifact to your Rule Designer workspace.
	An incoming <b>change</b> means that Decision Center contains a changed artifact. <b>Updating</b> transfers the new version to your Rule Designer workspace.
	An incoming <b>(-)</b> means that Decision Center no long contains an artifact. <b>Updating</b> deletes the artifact from your Rule Designer workspace.
	An outgoing <b>(+)</b> means that your Rule Designer workspace contains an artifact that is not in Decision Center. <b>Publishing</b> transfers the artifact to the remote project.
	An outgoing <b>change</b> means that Rule Designer contains a locally changed artifact that is not in the remote project. <b>Publishing</b> transfers the artifact to Decision Center and creates a new version.
	An outgoing <b>(-)</b> means that your Rule Designer workspace no longer contains an artifact. <b>Publishing</b> deletes the artifact from the remote Decision Center.
	A conflicting <b>(+)</b> means that Rule Designer and the remote Decision Center project contain a locally added artifact. You must resolve the conflict.
	A conflicting <b>change</b> means that Rule Designer and the remote Decision Center project contain a locally changed artifact. You must resolve the conflict.
	A conflicting <b>(-)</b> means that Rule Designer and the remote Decision Center project no longer contain an artifact. You must resolve the conflict.

**Parent topic:** [Synchronizing from Rule Designer](#)

**Related concepts:**  
[Rule project items](#)

**Related tasks:**  
[Creating projects from Decision Center](#)  
[Publishing decision services to Decision Center](#)

**Related information:**  
[Change management](#)  
[Synchronization architecture](#)

## Creating projects from Decision Center

You can create a project or in Rule Designer from an existing Decision Center project.

### About this task

Business users and developers can collaborate on rule projects by copying the project or a branch of the project from Decision Center to Rule Designer.

You cannot copy a project or a branch of a project into an Eclipse workspace that already contains the project. You must copy the project or branch to a different workspace, or delete the project from the target workspace before you copy the project or branch to the workspace.

#### Note:

Decision services are created in Rule Designer, and then published to Decision Center.

### Procedure

To create a project from an existing Decision Center project:

1. Click **File > New > Project**.
2. In the New Project wizard, select **Rule Project from Decision Center**, and then click **Next**.
3. In the **New Rule Project from Decision Center** window, complete the **URL** and **Data source** fields with information corresponding to your Decision Center session. By default, **Data source** is set to `jdbc/iLogDataSource`:

**New Rule Project from Decision Center**

**Decision Center Configuration**

Configure the project to be synchronized with a Decision Center project.

**Connection**

URL:

Not yet authenticated. Click the Connect button.

Data source:

**Connect**

**Synchronization Filter**

☐ Use the specified query to filter rule elements for synchronization

**Project configuration**

Remote project

**Refresh list**

Select a branch or release

Select a change activity

**Local project contents**

☒ Use defaults

Location  **Browse ...**

**< Back** **Next >** **Finish** **Cancel**

**Note:** In the URL, you can use the `/teamserver` extension, or `/decisioncenter` extension indifferently.

4. Indicate where you want the copy to be stored on your computer if the location differs from your Eclipse workspace.

Remember that you cannot copy the project or a branch of the project to a workspace that already contains the same project.

5. Click **Connect** to establish a connection with Decision Center.
6. Proceed with the authentication by entering your Operational Decision Manager on Cloud credentials.
7. To limit the rule elements that are synchronized, select the **Use the specified query to filter rule elements for synchronization** check box, and then select a query from the list of queries. If you do not use a query, all the elements are copied.
8. In **Project configuration**, use the **Remote project** list to select the Decision Center project that you want to import.

Click **Refresh list** if your project is not in the list.

9. In **Select a branch or release**, select a branch or release if there is more than one in your project.
10. In **Select a change activity**, select a change activity if you want to import an activity from a release.
11. Click **Finish**.

The **Problems** view displays any errors, but the import continues. When the import finishes copying the project to your workspace, you can see the project in the **Rule** perspective.

**Parent topic:** [Synchronizing from Rule Designer](#)

**Related concepts:**

[Synchronization commands in Rule Designer](#)

**Related tasks:**

[Publishing decision services to Decision Center](#)

**Related information:**

[Synchronization architecture](#)

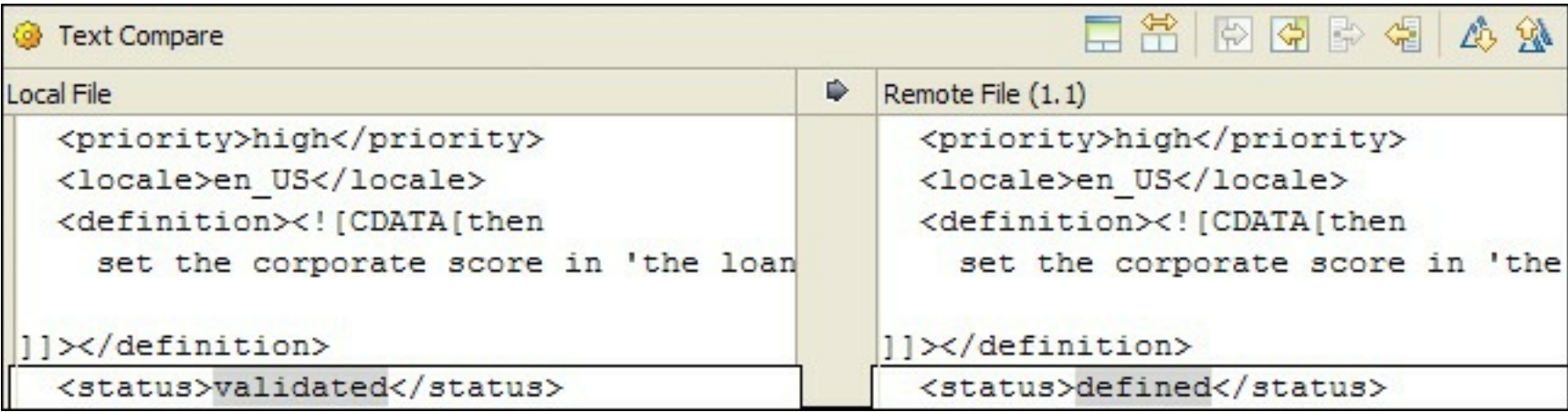
[Change management](#)



# Use of Text Compare in synchronization

Display conflicting rules and their differences in the Eclipse Text Compare Editor.

When the Synchronize view indicates a difference between the local Rule Designer and remote Decision Center versions of a rule, you can double-click the entry to display both rules and their differences in the Eclipse Text Compare Editor:



The Local File pane displays the local rule in its XML or modified text format. You can edit the rule in the Local File pane, but this practice is not the usual approach.

**Note:**

For details on the structure of the XML files, see [File types](#).

The Remote File pane displays the remote Decision Center version of the rule in read-only mode. The format of the remote version of the rule is an equivalent XML version, transformed so that you can compare them to find differences and resolve conflicts.

**Note:**

Saving a rule in Decision Center without changing the content increments the version number. As a result, the two different rules look the same in Text Compare.

After you check the contents of the local and remote versions, you can update, publish, or override to resolve any conflicts.

For details about the features available in the Text Compare Editor, see the Eclipse Help (**Workbench User Guide > Reference > User interface information > Views and editors > Compare editor**). In particular, use the **Show/Hide Ancestor Pane** button to verify the state of the common ancestor that is used in three-way comparisons.

**Parent topic:** [Synchronizing from Rule Designer](#)

**Related concepts:**

[Synchronization commands in Rule Designer](#)

**Related tasks:**

[Creating projects from Decision Center](#)

[Publishing decision services to Decision Center](#)

# Synchronization and source code control

Use source code control (SCC) to share Rule Designer artifacts, and commit rule project resources.

You must know your SCC well before you implement synchronization with Decision Center. Section [Rule project items](#) provides a list of the resources to commit to your SCC, and items not to commit, for example the output folder and connection entry files.

Some SCC systems use the Team Synchronization feature in Eclipse. You access the systems through **Team > Share**.

The Synchronize view can handle multiple participants. Therefore, you can have one Decision Center participant and one SCC participant for each rule project in Rule Designer.

To have two different branches of the same project, import the projects into separate workspaces. If you update the UUIDs manually, you break synchronization.

**Parent topic:** [Storing and synchronizing rules](#)

**Related concepts:**

[Rule project items](#)

**Related tasks:**

[Publishing decision services to Decision Center](#)

# Rule project items

A rule project is a container for organizing rule artifacts and setting up the business object model (BOM) and rule authoring vocabulary. Each rule project item is associated with a folder.

You implement a rule project in Rule Designer as an Eclipse project, which serves as a container for organizing rule-related items. A rule project can contain different types of folders:

## Source folder (rules)

A root container for the rule packages and artifacts.

## BOM folder (bom)

Contains the files that are related to the business object model (BOM). A BOM file that is stored in the BOM folder is part of the BOM path.

## Deployment folder (deployment)

Contains the deployment configuration for a decision service.

## Query folder (queries)

Contains query files.

## Resource folder (resources)

Contains resource files, that is, files or folders that are not part of the rule model.

## Template folder (templates)

Contains template files.

## Reports folder (reports)

Contains the reports that are created by different operations, including deployment.

The folders are registered in the project properties, and must be located directly under the rule project. The source folder contains rule packages that contain rule artifacts. It is also the root package, so it can contain rule artifacts.

**Note:**

If you rename a folder or subfolder in Rule Designer and synchronize with Decision Center, you obtain a new version in Decision Center for every artifact in that folder.

Each rule project item is associated with a folder, a file, or both. The following table describes these associations. The **Decision Center** column indicates which project items synchronize with Decision Center. The **SCC** column indicates which project items must be committed when you use a source code control system.

Item	As so cia te d wi th	Comments	D e ci si o n C e n t e r	S C C
Rule proje ct	Th e roo t fol der		⊗	✓
	.p ro je ct file	XML file that stores the general Eclipse project information, such as the nature of the project or the launch configuration.	⊗	✓
	.r ul ep ro je ct	XML file that stores the project properties that are specific to a rule project: <ul style="list-style-type: none"><li>• Categories</li><li>• BOM path</li></ul>	✓	✓

	file	<ul style="list-style-type: none"> <li>• XOM path</li> <li>• Output location</li> <li>• Relative path of the source, BOM, query, template, and resources folders.</li> </ul> <p>Displayed with the “Project properties” label in the Synchronize view.</p>		
Rule package	A rule package folder	Called a folder in Decision Center.	✓	✓
	.rulpackage file	XML file that stores the information for the rule package.  Displayed with the “Package properties” label in the Synchronize view.	✓	✓
Action rule	.brl file	XML file that stores the properties and the definition of an action rule.	✓	✓
Decision table	.dta file	XML file that stores the properties and the definition of a decision table.	✓	✓
Decision tree	.dtr file	XML file that stores the properties and the definition of a decision tree.	✓	✓
Function	.fct file	XML file that stores the properties and the definition of a function.	✓	✓
Ruleflow	.rfl file	XML file that stores the properties, the task definitions, and the description of a ruleflow diagram.	✓	✓
Technical rule	.trl file	XML file that stores the properties and the definition of a technical rule.	✓	✓
Deployment configuration	.dep file	The manner in which decision operations are packaged into RuleApps, managed, and then deployed.	✓	✓
Decision operation	.dop file	A function that defines the decision-making logic, and the input and output data for a decision. A decision operation is implemented as a ruleset.	✓	✓
Variable set	.var file	XML file that stores a list of variables.	✓	✓
Action rule template	.brt file	XML file that stores the template properties, and the properties and the definition of the action rule to instantiate.	✓	✓
Query	.qry file	XML file that stores the properties and the definition of a query.	✓	✓
BOM entry	.bom file	Plain text file that stores the structure of a BOM entry.	✓	✓
	<locale>.voc file	Key-value property file that stores the verbalization information that is attached to BOM elements. The first part of the keys corresponds to the fully qualified name of the BOM elements. The second part defines the verbalization of the BOM elements.	✓	✓



	.b2x file	XML file that stores the functions that map the BOM to the XOM.	✓	✓
Source folder	The source folder	The source folder is not a project item as such, but a container for the rule artifacts.	✗	✓
BOM folder	The bom folder	The BOM folder is not a project item as such, but a container for the BOM entries. All the BOM entries that are directly under the BOM folder or under folders in the BOM folder are part of the BOM and are referred to as the BOM path.	✗	✓
Deployment folder	The deployment folder	The deployment folder contains the deployment configuration for deploying a decision service.	✓	✓
Query folder	The queries folder	The queries folder is a container for the queries that can be used in the project.	✓	✓
Resource folder	The resource folder	<p>The resource folder is a container for files that are not part of the rule model. (See <a href="#">Defining a structure for rule project items</a> and <a href="#">Creating a resource</a>).</p> <p>The deployment.xml is created in the META-INF resource folder, and used with the managed XOM feature.</p> <p>If you selected the decision engine as rule engine, a B2X folder is created under the resources folder. The files in the B2X folder are taken into account during the synchronization with Decision Center and are required for the generation of ruleset archives from Decision Center.</p>	✓	✓
Template folder	The templates folder	The template folder is a container for the templates that can be used in the project and any dependent projects. The complete list of templates is computed by gathering the templates that are stored directly under the template folder or any of its subfolders.	✓	✓
Output folder	The output folder	The output folder stores compiled files that are generated when you build the project.	✗	✗
Reports folder	The reports folder	The reports folder holds the reports that are generated by different operations.	✗	✓
Decision Cent	The .s	File used to share the synchronization state between Rule Designer and Decision Center. Committing this file to SCC lets you work on the same project in a different workspace without generating	✗	✓

er conn ectio n entry	yn cE nt ri es file	conflicts.		
-----------------------------------	------------------------------------	------------	--	--

**Parent topic:** [Storing and synchronizing rules](#)

**Related information:**  
[Synchronization architecture](#)

# Setting up notifications from Decision Center with webhooks

You can use webhooks to receive notifications from Decision Center to the application of your choice.

A webhook is an HTTP custom callback, which you use to send notifications to external applications. A webhook is defined by the three following properties:

- Mandatory: A URL that specifies the endpoint where to send the event.
- Optional: A data source. If you do not specify one, the default data source is used.
- Optional: An authentication token to authenticate to your remote secure server.

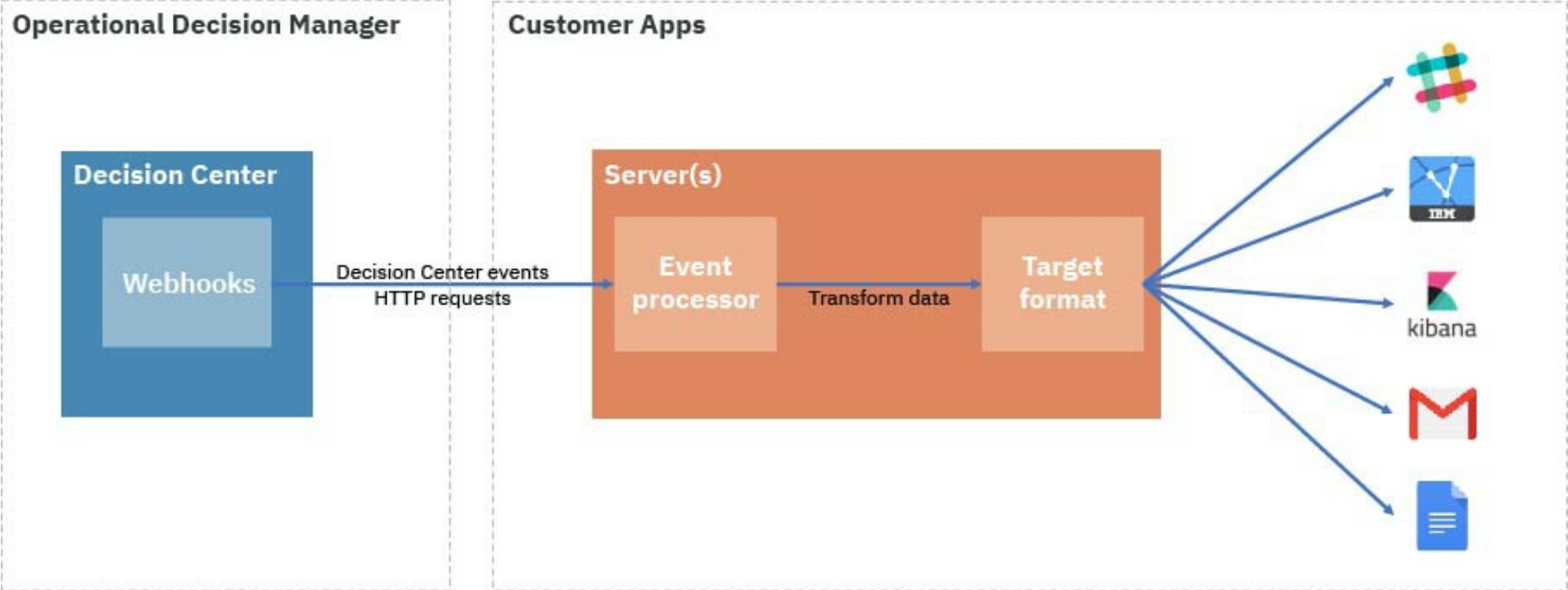
You can be notified in real time of the following types of events:

- When actions that are related to the decision governance framework happen in Decision Center, for example when a user creates a release, or approves an activity.
- When a deployment ends, you are notified of its status: completed, failed, or aborted.
- When a user creates or updates a rule or a decision table.

You register webhooks with the Decision Center REST API. Events are sent to the registered endpoints in JSON format. You must set up a server able to handle events and to convert them to a format compatible with the target application. For example, if you want to send notifications to Slack, your server must convert the events sent from Decision Center to a format that is compatible with Slack.

Notifications are asynchronous, which means they are done in the background.

Figure 1. Webhooks mechanism in Decision Center



## Configuring your webhooks

The Decision Center REST API provides you with four endpoints to handle webhooks:

- PUT /webhook/notify to define one or more URLs where notifications are sent, and optionally provide an authentication token.
- GET /webhooks/notify to see the list of URLs getting notifications from this instance of Decision Center.
- DELETE /webhooks/{webhookId}/notify to delete a webhook.
- POST /webhooks/{webhookId}/notify to update an existing webhook.

For more information, see [IBM® ODM Decision Center API](#)

## Receiving a webhook notification

The JSON payloads received by the server contain various information about your event. The following data is sent:

Table 1. Information sent by an event

Data	Description
id	Unique identifier that is given to an event. You use this ID to find the event in log files.
version	Version number of an event. This allows you to identify the version of the event.
author	The user that triggered the notification.
date	The date when an event was emitted, represented a Java™ date (long value).
sourcename	The name of the application that sends the event. This value is always Decision Center.
sourcelink	The URL of the Decision Center instance that sends the event.
type	The type of event that triggers a webhook, which allows you to filter events.

	<p>The following types are possible:</p> <p><b>Rule artifacts</b></p> <p>RuleCreated, when a rule is created. RuleUpdated, when a rule is updated. RuleDeleted, when a rule is deleted.</p> <p><b>Decision Governance Framework</b></p> <p>ReleaseCreated, when a release is created. ReleaseUpdated, when a release is updated. ReleaseDeleted, when a release is deleted. ActivityCreated, when an activity is created. ActivityUpdated, when an activity is updated. ActivityDeleted, when an activity is deleted.</p> <p><b>Deployment</b></p> <p>RuleAppDeployment, when a RuleApp is deployed.</p>
content	An array with information about the artifact associated with the event. For example, if you created a release, you have release details such as the approver’s name, due date, and so on.
Optional: details	An array with additional information about the event.
project	An array with information about the project associated with the event.
decisionService	An array with information about the decision service associated with the event.

Examples

The following example shows an event that is sent when a release is created:

```
{
 "version": "1.0",
 "id": "f3577511-c33b-4275-a5ef-de177598305b",
 "author": "rtsAdmin",
 "date": 1532519296400,
 "type": "ReleaseCreated",
 "content": [{
 "id": "faa9b2af-e2fe-42a4-9834-bb9cdd607334",
 "internalId": "brm.Release:866:866",
 "name": "New Release",
 "parentId": "7b360d2b-46ca-44c1-a945-fb91b9a0103e",
 "documentation": "",
 "buildMode": null,
 "status": "InProgress",
 "owner": "rtsAdmin",
 "targetDate": "2018-08-07T22:00:00.000Z",
 "approvers": [],
 "initial": false,
 "kind": "Release"
 }],
 "sourceName": "Decision Center",
 "sourceLink": "http://9.145.66.219:8081/decisioncenter?datasource=jdbc%2FilogDataSource",
 "project": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
 },
 "decisionService": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
 }
}
```

The following example shows an event that is sent when a change activity is updated. Here, the due date was changed. The details of the change are stored in the detail field.

```
{
 "version": "1.0",
 "id": "4eef8f4d-7c87-4022-b0aa-28bf88ffdb7f",
 "author": "rtsAdmin",
 "date": 1532519323387,
 "type": "ActivityUpdated",
 "content": [{
 "id": "0eade79c-14e5-4155-8a08-23c7d52a168b",
 "internalId": "brm.ChangeAct:881:881",
 "name": "Spring Activity",
 "parentId": "faa9b2af-e2fe-42a4-9834-bb9cdd607334",
 "documentation": "",
 "buildMode": "ClassicEngine",
 "status": "InProgress",
 "owner": "rtsAdmin",
 "targetDate": "2018-08-06T22:00:00.000Z",
 "approvers": [],
 "authors": [],
 "kind": "ChangeActivity"
 }],
 "details": [{
 "targetURL":
"http://9.145.66.219:8081/decisioncenter/t/library#overviewactivity?
datasource=jdbc%2FilogDataSource&baselineId=brm.ChangeAct%3A881%3A881",
 "updateType": "UPDATE_DUE_DATE",
 "oldValue": 1533679200000,
 "newValue": 1533592800000
 }],
 "sourceName": "Decision Center",
 "sourceLink": "http://9.145.66.219:8081/decisioncenter?
datasource=jdbc%2FilogDataSource",
 "project": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
 },
 "decisionService": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
 }
}
```

The following example shows an event that is sent when a RuleApp is deployed:

```
{
 "version": "1.0",
 "id": "e424a030-4aea-4452-bb5f-7450893a9803",
 "author": "rtsAdmin",
 "date": 1532519935332,
 "type": "RuleAppDeployment",
 "content": [{
 "id": null,
 "internalId": null,
 "name": "Report 2018-07-25_01-58-41-821",
 "status": "COMPLETED",
 "ruleAppName": "autoQuoteRuleApp",
 "messages": {
 "elements": [],
 }
 }]
```

```
 "totalCount": 0,
 "number": -1,
 "size": 0
 },
 "snapshot": {
 "id": "4758d793-e953-4f11-824f-9e9445cfa02a",
 "internalId": "dsm.DsDeploymentBsln:888:888",
 "name": "Deployment_to_QA_env_2018-07-25T11_58_39Z",
 "parentId": "0eade79c-14e5-4155-8a08-23c7d52a168b",
 "documentation": null,
 "buildMode": "ClassicEngine",
 "kind": "DeploymentSnapshot"
 },
 "servers": [],
 "archive": null
}],
"details": [{
 "targetURL":
"http://9.145.66.219:8081/decisioncenter/t/library#deploymentreport?
id=dsm.DSDeploymentReport%3A1%3A1&datasource=jdbc%2FilogDataSource&baselineId=
brm.ChangeAct%3A881%3A881"
}],
"sourceName": "Decision Center",
"sourceLink": "http://9.145.66.219:8081/decisioncenter?
datasource=jdbc%2FilogDataSource",
"project": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
},
"decisionService": {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:229:229",
 "name": "AutoQuote",
 "buildMode": "ClassicEngine"
}
}
```

# Deploying rule applications

You deploy a rule application, or RuleApp, from a decision service to an execution server. The RuleApp contains rulesets for validation or use with calling applications.

You use deployment configurations in decision services to deploy RuleApps. The deployment configurations apply decision operations that specify the rules to include in the rulesets of RuleApps. They also specify target execution servers, and the version policies for RuleApps and rulesets.

You also deploy the execution object model (XOM) that references the application objects and data on which you run your rules. You create and maintain the XOM in Rule Designer, and publish the XOM binary to Decision Center. You must keep the XOM synchronized in the components to ensure that the deployed RuleApps reference the correct XOM.

**Note:** For development purposes, you can deploy a RuleApp from Rule Designer to the Rule Execution Server in a development environment. However, you cannot deploy a RuleApp from Rule Designer directly to the Rule Execution Server in a test or production environment.

The following table summarizes the deployment tasks by user role:

User role	Deployment tasks
Rule developer	<div>Creates a decision service in Rule Designer and adds deployment configurations.</div> <div>Deploys a RuleApp for the decision service from Rule Designer or Decision Center to the development environment to verify and troubleshoot the deployment process.</div> <div>Publishes the decision service, its deployment configurations, and the XOM to Decision Center.</div>
Release manager	<div>Deploys RuleApps for decision services from Decision Center to the development, test, and production execution servers.</div> <div>Creates deployment configurations in the Decision Center Business console.</div>
Business user	<div>To validate a decision service, deploys RuleApps from Decision Center to the development and test execution servers.</div>
Integrator	<div>Deploys RuleApps to the development and test execution servers.</div>

## Deployment configurations

You use deployment configurations to deploy RuleApps from decision services. They include the decision operations, target servers, and version policies for deploying rulesets in RuleApps.

## Version policies

Version policies determine how client applications identify deployed rulesets.

## Configuring deployment in Rule Designer

In Rule Designer, you configure the deployment of decision services from Decision Center.

## Deploying from Rule Designer in the development environment

In an early stage of decision service development, you might want to deploy a decision service from your local Rule Designer to Rule Execution Server on the cloud to test the execution of the service before publishing it to Decision Center.

## Deploying from Decision Center

You can deploy any branch (release, change activity, or regular branch) of a decision service from the Decision Center Business console.

## XOM deployment

XOM deployment for decision services can be done with Rule Designer or with Decision Center. You synchronize the XOM binary file between Rule Designer and Decision Center to ensure that RuleApps deployed from either environment reference the correct XOM.

## Embedded managed Java XOM in Decision Center

Embedded managed Java™ XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can download RuleApp archives with embedded managed Java XOMs by using the Decision Center REST API. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

## Deploying to a hybrid cloud environment

To use Operational Decision Manager on Cloud in hybrid mode, you download and deploy a decision

service to a Decision Server that is outside the Operational Decision Manager on Cloud environment.

**Related information:**

[User roles, environments, and components](#)



# Deployment configurations

You use deployment configurations to deploy RuleApps from decision services. They include the decision operations, target servers, and version policies for deploying rulesets in RuleApps.

You can create or edit a deployment configuration in Rule Designer or the Decision Center Business console. In the Business console, only release managers can create or edit a deployment configuration. When working within the governance framework, a deployment configuration can be created or edited only within a change activity.

A deployment configuration generates a RuleApp archive that contains rulesets. These rulesets are defined in the decision operations in the decision service. You can select one or more rules by applying either a custom validator or a query. Decision operations are defined in Rule Designer and cannot be changed in Decision Center.

You create three deployment configurations, for development, testing, and production.

A deployment configuration includes the following information:

- Configuration type (production or nonproduction): limits deployment to certain servers
- RuleApp name: used by the client application to request a decision
- Decision operations: lists the rules to be deployed
- Target server: runs the rule applications
- Version information: shows the ruleset base version number and versioning policy

In the Business console, the target server of a deployment configuration determines which user groups can use the configuration. For example, the business user group can deploy a decision service to the development environment if the target server of the deployment configuration is the development server.

For each operation, the deployment configuration contains a base ruleset version number whose role depends on the version policy. In the increment version policy, it defines the minimal version, and in the base version policy, the deployed ruleset version. In the user-defined version policy, it is used to calculate a proposed ruleset version at deployment time.

You can also set ruleset execution options in the deployment configuration:

- Enabled: You can enable execution of a ruleset in a decision service. When you disable a ruleset, this ruleset is not taken into account by the rule engine at execution time.
- Debug: You can enable debug mode on a ruleset in a decision service.
- Trace: You can monitor ruleset execution by generating execution traces.
- Bytecode generation: You can generate the ruleset with Java™ bytecode from the Business console.

## Configuration type

You indicate whether a deployment configuration is for a production or nonproduction environment:

### Production

When a release of a decision service completes its lifecycle in the governance framework, it can be deployed to a production server.

### Nonproduction

When a release of a decision service is under development in the governance framework, it cannot be deployed to a production server.

## Target servers

When deploying from a decision service, you choose the target Rule Execution Server that is defined in the deployment configuration.

The user role and the release state determine the servers to which a user can deploy. For example, if you are working in a change activity, you can deploy to a test environment but not to production. Typically, business users are limited to the development and test environments, while release managers can deploy to production.

## Rule Execution Server Connections view

A deployment configuration specifies the target server for the deployment. Only a symbolic name for the target server is stored in the deployment configuration. The connection details for the server are defined in the Rule Execution Server Connections view of Rule Designer, which is separate from the deployment configuration.

**Parent topic:** [Deploying rule applications](#)

## Version policies

Version policies determine how client applications identify deployed rulesets.

In a deployment configuration for a decision service, you establish the version policies to define the paths of the rulesets in the RuleApp deployment. The ruleset path determines how an application calls a ruleset through Rule Execution Server.

The ruleset path can have the following structure:

DeploymentConfiguration/RAVmajor.RAVminor/Operation/RSVmajor.RSVminor

DeploymentConfiguration identifies the decision service. RAVmajor.RAVminor identifies the version of the decision service interface that is being used. Client applications use the version number to ensure that they are compatible with the decision service interface. Operation is the name of the ruleset. The major version number of a ruleset (Vmajor) often represents a specific release of the ruleset, and the minor version number (Vminor) represents a deployment version of the ruleset. Typically, client applications call the latest enabled ruleset, but what they actually call depends on the ruleset path that is used in the client application.

The version numbers use base numbers that you define in the deployment configuration. You manually define a base number for the RuleApp and each ruleset. Deployment increments the minor ruleset version number, or replaces or creates a ruleset version number (Vmajor.Vminor).

**Note:** The version policies that are set in a Rule Designer deployment configuration are synchronized with Decision Center.

### Select a policy by the type of deployment

The version policy that you use depends on the type of deployment:

Type of deployment	Version policy
Successive deployments of a release to a specific server	Increment minor version numbers
Development of a decision service	Use the base version numbers
Test fix to correct a deployed solution	Use the base version numbers
Test fixes or updates to an earlier release	Define the version numbers

#### Increment minor version numbers

This policy serves as the default setting. It deploys the decision operations of the release, changing the minor version number of each ruleset. It looks for the latest version number to increment on the server to which it has access, and the same deployed version number is used.

A client application uses a ruleset based on the ruleset path that is defined in the client application., but previous deployments remain available on Rule Execution Server. You can still use a previous deployment by specifying its full ruleset path. You use this policy with successive deployments of a release to a specific server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/2.1</div>

#### Use the base version numbers

This policy deploys the decision operations and replaces the base version on Rule Execution Server. The replacement ensures that the deployment configuration path corresponds exactly to what is deployed on the server.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0  /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0  /ruleset/2.0 (replaced)</div>

**Define the version numbers**

With this policy, you enter a specific ruleset version number at deployment time. You use this policy with test fixes or updates to an earlier release. Upon deployment, Rule Execution Server uses the specified ruleset version, and replaces the last version of the ruleset.

The following table shows an example of the results of this policy:

Deployed RuleApp	Base RuleApp	Results after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 (replaced) /ruleset/2.0</div>

**Parent topic:** [Deploying rule applications](#)

# Configuring deployment in Rule Designer

In Rule Designer, you configure the deployment of decision services from Decision Center.

The rule developer uses Rule Designer to do the following deployment-related tasks:

- Set up deployment configurations, which define the decision operations and target servers for deployment from Decision Center.
- Run a decision service locally by using a run configuration, and deploy to the development environment to verify the deployment process.
- Publish the decision service so that the deployment configurations and the XOM are available in Decision Center.

## Deployment artifacts

In Rule Designer, each rule project contained in a decision service has a **deployment** folder to store the deployment configurations and decision operations created by the rule developer. Typically, the deployment configurations are kept in the **main** project of the decision service.

See [Deployment configurations](#) for information about the different aspects of deployment configurations. Also, consider the following points for Operational Decision Manager on Cloud:

- Target servers are available for the development, testing, and production environments. The recommended approach is to create a deployment configuration for each environment.
- The configuration type is **Production** for the production server, and **Nonproduction** for the development and testing environments.
- From Rule Designer, you can only deploy to the development environment, and not to the test and production environments.

**Parent topic:** [Deploying rule applications](#)

### Related concepts:

[Deploying from Decision Center](#)

[XOM deployment with decision services](#)

### Related information:

[Synchronizing from Rule Designer](#)

## Deploying from Rule Designer in the development environment

In an early stage of decision service development, you might want to deploy a decision service from your local Rule Designer to Rule Execution Server on the cloud to test the execution of the service before publishing it to Decision Center.

### Procedure

1. In Rule Designer, right-click your decision service.
2. Click **Rule Execution Server > Deploy**.
3. In the RuleApp Deployment dialog, click the development deployment configuration, and then click **Next**.
4. Verify the deployment configuration settings, and then click **Next**.
5. Verify your authentication credentials.
6. Click **Connect** if you are not yet authenticated, and then click **Next**.
7. Verify the RuleApp and ruleset versions, and then click **Finish**.

When the RuleApp is deployed, a deployment report is displayed in Rule Designer.

<b>Note:</b> From Rule Designer, you cannot deploy to the test and production environments, but only to the development environment.
--------------------------------------------------------------------------------------------------------------------------------------

8. Use a test application to call the decision service that you have deployed.
9. Check that the behavior is as expected.

**Parent topic:** [Deploying rule applications](#)

## Deploying from Decision Center

You can deploy any branch (release, change activity, or regular branch) of a decision service from the Decision Center Business console.

All the information required for deployment is contained in a deployment configuration. Deployment configurations can be synchronized with Rule Designer.

Only a release manager can create, edit, or delete deployment configurations in the Business console. This ensures a basic level of security on deployment. Deployment configurations contain information that includes the target servers, the decision operations that define the business rules, and settings for versioning of rulesets and RuleApps. The following aspects of a deployment configuration relate to making sure that deployment is secure:

- The configuration type as Production or Nonproduction.
- The target server where this deployment configuration can deploy to.
- User groups that can deploy with the configuration depend on the selected server.

Any user who can access a decision service in the Business console can deploy its branches (release, change activity, or regular branch) from any deployment configuration available to that user in that branch. The deployment configurations that are available for a given user depend on this user's role (see [Deploying rule applications](#)). The following conditions apply:

- In a change activity, you can deploy only with a deployment configuration whose type is Nonproduction.
- In a release, you can deploy with a deployment configuration whose type is Production only if the release is completed.

When you deploy a decision service, the XOM is also deployed if the version of the XOM is not already present in the target Rule Execution Server.

When working within the governance framework, a deployment configuration can only be created or edited within a change activity.

Deployment snapshots can be found in the list of snapshots. You can then redeploy from the deployment snapshot or from the report of the deployment.

<b>Note:</b> You can also automate rule deployment by using the <a href="#">Decision Center REST API</a> .
------------------------------------------------------------------------------------------------------------

**Parent topic:** [Deploying rule applications](#)

**Related concepts:**

[XOM deployment from Decision Center](#)

## XOM deployment

XOM deployment for decision services can be done with Rule Designer or with Decision Center. You synchronize the XOM binary file between Rule Designer and Decision Center to ensure that RuleApps deployed from either environment reference the correct XOM.

### [XOM deployment with decision services](#)

A RuleApp that you deploy must reference the correct XOM.

### [XOM deployment from Decision Center](#)

When you deploy from Decision Center, the XOM is also deployed if the version is not already present in the target environment.

**Parent topic:** [Deploying rule applications](#)

## XOM deployment with decision services

A RuleApp that you deploy must reference the correct XOM.

When you deploy a decision service, Rule Designer goes through the following process:

1. Builds the XOM binary from the source files that are referenced by the decision service.
2. Deploys the XOM if it is not already present in Rule Execution Server, and retrieves the XOM URI.
3. Copies this URI to each ruleset that is contained in the RuleApp. Specifically, Rule Designer copies the URI to a property of the ruleset called `ruleset.managedxom.uri`.

With this mechanism, each ruleset in Rule Execution Server references the correct XOM.

**Note:** If the XOM is based on an XML schema (.xsd file), the XML schema is packaged into the ruleset archive and deployed with the RuleApp.

### XOM deployment in Decision Center

When Rule Designer builds the XOM from the source files, it places the XOM binary under `resources/xom-libraries`. The XOM binary is then published to Decision Center with the rest of the decision service.

With the XOM binary file synchronized between Rule Designer and Decision Center, RuleApps deployed from either environment reference the correct XOM.

### XOM modifications

You modify the XOM source files in Rule Designer. Rule Designer always builds the XOM binary from the source files. You can then publish the XOM to Decision Center.

If you import a decision service from Decision Center into an empty workspace:

- You do not obtain the XOM source files. You must obtain these source files in the usual way, for example through source code control.
- The XOM binary obtained from Decision Center is copied to the `resources/xom-libraries` folder. The files in this folder should not be used in the XOM path.

**Parent topic:** [XOM deployment](#)

**Related concepts:**

[XOM deployment from Decision Center](#)



# XOM deployment from Decision Center

When you deploy from Decision Center, the XOM is also deployed if the version is not already present in the target environment.

The XOM must first be published from Rule Designer to Decision Center. The XOM is stored in the resources of the BOM project of the decision service. All RuleApps created from this decision service reference this version of the XOM.

At deployment time, Decision Center creates a deployment snapshot, which contains the version of all rules and deployment artifacts at that moment in time. This ensures that any redeployed RuleApp references the correct XOM.

**Important:**

You cannot access the XOM source files from Decision Center. You modify XOM source files in Rule Designer.

In Decision Center, information on the XOM is visible in the deployment report. If you need to see the XOM files in Decision Center, create a smart folder that displays resources in the Enterprise console.

**Parent topic:** [XOM deployment](#)

**Related concepts:**

[XOM deployment with decision services](#)

**Related information:**



[Smart folders](#)

# Embedded managed Java XOM in Decision Center

Embedded managed Java™ XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can download RuleApp archives with embedded managed Java XOMs by using the Decision Center REST API. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

The embedded managed Java XOM feature can be operated from the Decision Center REST API.

Table 1. Embedded managed XOMs

Module or tool	Deploy	Create or retrieve	Description
REST API		 You can create a RuleApp archive with or without the embedded managed Java XOM.	For more information, see <a href="#">downloadUsingGET</a> . You set the parameter includeXOMInArchive to true to include the XOM in your RuleApp archive.

Parent topic: [Deploying rule applications](#)

Related concepts:  
[Embedded managed Java XOM in Rule Execution Server](#)

## Deploying to a hybrid cloud environment

To use Operational Decision Manager on Cloud in hybrid mode, you download and deploy a decision service to a Decision Server that is outside the Operational Decision Manager on Cloud environment.

### Before you begin

To work in a hybrid cloud environment, the following prerequisites apply:

- Access to a source Operational Decision Manager on Cloud environment where a decision service (RuleApp and its XOM) is deployed to a Rule Execution Server. The RuleApp must be linked to the XOM, and both must be working properly in Operational Decision Manager on Cloud. You need credentials and the URL of the Rule Execution Server.
- Access to a target environment, for example:
  - On-premises installation of IBM® Operational Decision Manager Decision Server
  - Another Operational Decision Manager on Cloud environment
- Administration credentials and the URL of the target Rule Execution Server.

**Important:** The target Decision Server must be compatible with the source Operational Decision Manager components for them to work together correctly. Refer to [IBM Operational Decision Manager on Cloud Compatibility](#).

### About this task

You can use the REST API or the Rule Execution Server console to download and deploy a decision service to another Decision Server.

#### [Using the Rule Execution Server REST API](#)

To deploy a decision service to a target Decision Server, you download the corresponding RuleApp and its XOM from the source Operational Decision Manager on Cloud environment, and use the REST API to place the RuleApp and XOM binary files on the target Decision Server. Then, you can run the decision service on this Decision Server.

#### [Using the Rule Execution Server console](#)

To deploy a decision service to a Decision Server that is outside Operational Decision Manager on Cloud, you download the corresponding executable assets (the RuleApp and its XOM) from the Rule Execution Server console of the cloud portal, and then deploy them to the target Decision Server through the Rule Execution Server console.

**Parent topic:** [Deploying rule applications](#)

#### **Related information:**

[IBM ODM on Cloud in a hybrid cloud environment](#)  
[Testing a ruleset for REST execution](#)

## Using the Rule Execution Server REST API

To deploy a decision service to a target Decision Server, you download the corresponding RuleApp and its XOM from the source Operational Decision Manager on Cloud environment, and use the REST API to place the RuleApp and XOM binary files on the target Decision Server. Then, you can run the decision service on this Decision Server.

### Before you begin

You must authenticate with Operational Decision Manager on Cloud and deployed the decision service.

### About this task

The example commands that are provided in the following procedure use cURL to authenticate with the cloud server and transfer archive files from the cloud server to another Decision Server. You can use a different program to send authentication credentials to the servers, access the REST API, and download files from the cloud server.

### Procedure

1. Download the RuleApp from Operational Decision Manager on Cloud.

The command includes basic authentication credentials and the path to the RuleApp. Replace the user name and password variables with your basic authentication credentials. Provide the host URL of your cloud instance, and specify the correct name of the RuleApp in the path:

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/v1/ruleapps/ruleapp_name/1.0/archive --
output ruleapp_name.jar
```

2. Download the XOM:

- a. In the `ruleset.managedxom.uris` property in the output of the previous command, look for the managed XOM URI. This property gets the RuleApp details.
- b. If the property points to a XOM resource (`resuri`), skip this step. If it points to a library (`reslib`), check the content property in the output of the previous command for the XOM URI. The content property gets the library details. Use the name and version of the library that is returned in the previous command.

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/libraries/library_name/1.0
```

- c. Download the XOM. Use the name and version of the resource returned in the previous command.

```
curl -k -u username@company_name.com:password
server_URL/odm/cloud_env/res/api/xoms/xom_name/1.0/bytecode --output
xom_name.zip
```

3. Upload the XOM.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/xoms/xom_name.zip --data-binary @xom_name.zip -H
"Content-Type:application/octet-stream"
```

If the RuleApp references a library rather than a XOM, re-create the library and specify the XOM resource that it is supposed to contain.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/libraries/library_name1.0 -d
"resuri://xom_name.zip/1.0" -H "Content-Type:text/plain"
```

<b>Note:</b> Keep the names of the resources when downloading and uploading them.
-----------------------------------------------------------------------------------

4. Upload the RuleApp to the other Decision Server.

The command includes the user name, password, and URL to access the Rule Execution Server console on the other Decision Server.

```
curl -X POST -k -u admin_username:admin_password
server_URL/res/api/v1/ruleapps --data-binary @ruleapp_name.jar -H
```

```
"Content-Type:application/octet-stream"
```

5. Test the execution of the ruleset (decision service) by using its REST API.

```
curl -X POST -k -u username@company_name.com:password
server_URL/DecisionService/rest/v1/ruleappname/ruleappversion/rulesetname/
rulesetversion -d jsonpayload -H "Content-Type:application/json"
```

**Parent topic:** [Deploying to a hybrid cloud environment](#)

## Using the Rule Execution Server console

To deploy a decision service to a Decision Server that is outside Operational Decision Manager on Cloud, you download the corresponding executable assets (the RuleApp and its XOM) from the Rule Execution Server console of the cloud portal, and then deploy them to the target Decision Server through the Rule Execution Server console.

### Before you begin

You must authenticate with Operational Decision Manager on Cloud and deployed the decision service.

### Procedure

1. Download the RuleApp from Operational Decision Manager on Cloud:
  - a. Launch the Rule Execution Server console in your Operational Decision Manager on Cloud environment: development, test, or production.
  - b. In the **Explorer** tab, click the RuleApp that you want to download.
  - c. Click **Download Archive** in the RuleApp view.
  - d. Select the directory where you want to store the RuleApp archive.
2. Download the XOM:
  - a. Back in the **Explorer** tab of the Rule Execution Server console, under **Resources**, click the XOM that you want to download.
  - b. Click **Download Resource** in the Resource view.
  - c. Select the directory where you want to store the XOM archive.
3. Upload the XOM:
  - a. In the Resources view, click **Deploy Resource**.
  - b. Browse to the local path of the XOM archive, and then click **Deploy**. The managed XOM is deployed under **Resources** in the Rule Execution Server console. If the XOM is referenced from a library, you must re-create the library first (**Navigators > Libraries > Add Library**). Then, in the Library View of the newly created library, add the XOM as the internal resource reference (**Library View > Add Internal Resource Reference**).
  - c. Select the deployed XOM in the list of internal resources, or the library that contains the XOM in the list of libraries, and then click **Add**.
4. Upload the RuleApp to the target Decision Server:
  - a. Launch the Rule Execution Server console of the target Decision Server.
  - b. In the RuleApps view, click **Deploy RuleApp Archive**.
  - c. Browse to the local path of the RuleApp archive, and click **Deploy**. The RuleApp archive is deployed under **RuleApps** in the Rule Execution Server console.
5. To verify the deployment of the decision service to the target Decision Server, see [Testing a ruleset for REST execution](#)

**Parent topic:** [Deploying to a hybrid cloud environment](#)

## Running rules

You run sets of rules, or rulesets, from decision services in Rule Execution Server to validate them and use them with calling applications.

### [Managing rule execution in Rule Execution Server](#)

The following topics cover the features in Rule Execution Server. You can open certain topics from specific points in the Eclipse interface.

### [Rule Execution Server console online help](#)

The Rule Execution Server console is a web-based graphical interface to access most of Rule Execution Server features.

# Managing rule execution in Rule Execution Server

Managing rule execution involves tasks such as creating and deploying RuleApps, developing client applications, debugging, installing a production environment, and administering the production environment.

## Introducing Rule Execution Server

Rule Execution Server architecture is based on a number of independent, but cooperating, software modules.

## Rule Execution Server components

Rule Execution Server is an environment for executing rules. It provides management, performance, security, and logging capabilities.

## Improving the performance of Rule Execution Server

You can improve the performance of Rule Execution Server.

## Rule Execution Server administration artifacts

System administrators use the web-based Rule Execution Server console to manage and monitor RuleApps, ruleset execution, and decision services.

## Rule artifact management and deployment

Rule Execution Server provides a comprehensive set of tools to manage the deployment and execution of rulesets.



# Introducing Rule Execution Server

Rule Execution Server architecture is based on a number of independent, but cooperating, software modules.

Architects, administrators, developers, and testers work together to create, deploy, monitor, and administer decision services that run on Rule Execution Server instances in a development, test, or production environment.

The following steps provide the ordered flow of tasks to execute a ruleset using Rule Execution Server:

1. **Create:** Design and create a RuleApp.

A RuleApp is a deployment and management unit for Rule Execution Server. A RuleApp contains one or more rulesets. You deploy your RuleApps to Rule Execution Server in order to make the ruleset available to a client application. RuleApps can be deployed from a decision service deployment configuration.

Rule Designer does not explicitly show RuleApps. You use the decision service deployment configuration to hold information to make a RuleApp.

2. **Develop:** Design and create a client application.

To call a deployed ruleset that is contained in a RuleApp on Rule Execution Server, you must create a client application that calls the decision service through the chosen HTDS endpoint (SOAP or REST).

3. **Deploy to the test environment:** From a decision service that is ready for deployment, you can deploy one or more RuleApps. You deploy the rulesets that you defined in a decision operation to the Rule Execution Server instance that you defined in a deployment configuration.
4. **Test:** Run your client application and correct any errors that you find.

Before you deploy your decision service to a production server, you must ensure, by testing with sufficient data, that the client application and the decision service are stable and that the results are correct.

5. **Deploy to the production environment:** You deploy your validated decision services to the production environment.

**Parent topic:** [Managing rule execution in Rule Execution Server](#)

# Rule Execution Server components

Rule Execution Server is an environment for executing rules. It provides management, performance, security, and logging capabilities.

With Rule Execution Server, you can change the business logic dynamically.

**Note:** For details of software requirements, see the [IBM® Operational Decision Manager on Cloud - Detailed System Requirements](#) page.

Rule Execution Server is a set of components that interact with the rule engine. The following table lists these components.

Table 1. Rule Execution Server components

Component	Features	For more information
Management console	<p>The management console provides the following services:</p> <ul style="list-style-type: none"><li>• 24x7 execution with hot ruleset deployment. Hot deployment is the ability to publish changed applications to a running server without having to restart the server.</li><li>• Web-based management of RuleApps</li><li>• Views of runtime statistics</li><li>• Views of runtime errors and warnings</li><li>• Run server diagnostics</li><li>• Lists of the transparent decision services that are linked to a ruleset, at the level of that ruleset. You can use the console to activate and deactivate a transparent decision service, or to view execution statistics.</li></ul>	<p>See the Rule Execution Server console online help.</p>
Transparent decision services	<p>Hosted transparent decision services are installed on the same application server as Rule Execution Server, then integrated with Rule Execution Server</p>	<p>See the following topics:</p> <ul style="list-style-type: none"><li>• <a href="#">Hosted transparent decision services reference</a></li></ul>

**Parent topic:** [Managing rule execution in Rule Execution Server](#)

# Improving the performance of Rule Execution Server

You can improve the performance of Rule Execution Server.

You improve the performance by designing and creating efficient rulesets. You can create efficient rulesets when you develop a business rule application. The goal of performance tuning is to decrease the amount of time and resources that your application server requires to process requests. Performance tuning allows your application server to complete more tasks in less time.

## Improving performance during development

During the development phase of your application, you can enhance the performance of ruleset execution by creating efficient artifacts and choosing the execution mode for your artifacts.

The following table describes how to optimize the performance by creating more efficient artifacts.

Development artifact	Recommendation
Executable object model (XOM)	The size of the Java™ XOM deployed in your EAR affects ruleset parsing and execution. To improve performance, design the class loader to be as small as possible. Include only the JAR files that hold classes that are directly called by the XOM in the managed XOM persistence.
Ruleset size	<p>The size of a ruleset affects the total execution time.</p> <p>The main cause of resource usage when generating a RuleApp archive is the number and size of rule artifacts that are deployed as part of the RuleApp. Business rule applications work better if you limit the size of the resources that are deployed to the execution environment.</p>
Ruleset resources	<p>To reduce the resource usage cost of a business rule application:</p> <ul style="list-style-type: none"><li>• Transform each large rule project into several smaller ones.</li><li>• Generate multiple smaller rulesets from each rule project.</li><li>• Use ruleset extractors, one per ruleset, to extract part of the rule project. In this case, you might have to change the application that executes the rulesets.</li></ul>
Rule flow	<p>Design the rule flow with mutually exclusive rule tasks to reduce the number of rules evaluated during the execution. In this way, the ruleset is segmented and only the rules from the relevant rule tasks are evaluated.</p> <p>This tip applies to rulesets that are built with the classic rule engine only.</p>
Threads	<p>The Java XOM must be thread-safe. The Java XOM must implement the Serializable API if you call the Rule Execution Server remotely.</p> <p>The toString method performance can have a huge influence.</p>
XML XOM	Configure rulesets that use an XML XOM to run in multiple simultaneous executions. Set the <b>ruleset.xmlDocumentDriverPool.maxSize</b> ruleset property to configure the execution pool of the XML document drivers. The default value is 1.
Rule flow	Keep rule flows simple. Complex rule flows reduce performance.
Rule tasks	Avoid dynamic filters in rule tasks. Use a fixed list of rules where possible.
Decision tables	Verify the size of the decision table in the Rule Designer technical view. Dividing a decision table into several smaller tables can reduce the size of the ruleset significantly.

For more information about the artifacts that can affect performance, see [Rule Execution Server administration](#)

[artifacts](#).

You can optimize the performance in specific circumstances by choosing an appropriate execution mode. For more information about execution modes, see [Choosing an execution mode](#).

**Improving performance at run time**

After you develop the application and deploy to an application server, a number of configuration steps can affect performance.

C on fi g ur at io n of	Recommendation
Gar b ag e col lec tor	On the IBM® JVM, the <b>gencon</b> garbage collector policy has good results on small and medium applications. Tune the garbage collector and memory size.
Rul es et up da te	Use asynchronous parsing to improve response times during ruleset updates. With asynchronous parsing, the parsing is done by a separate thread, which reduces the effect on concurrent ruleset execution threads. To enable this option, set the <b>asynchronousRulesetParsing</b> property to true in the ra.xml file.  Ruleset parsing applies to rulesets that are built with the classic rule engine only.  Rulesets that are built with the decision engine require the loading of Java classes only. The rules are already compiled to executable code before deployment.
Tra ce s	Execution without traces is the optimal mode in terms of memory usage and performance. For best performance, turn off the traces.  When you choose to run the engine with a trace, the following side effects might impair performance: <ul style="list-style-type: none"><li>• Memory usage increases as the execution trace is generated, the list of rules and tasks is cached, and the rule events are generated.</li><li>• The response of the execution takes longer when the execution trace is integrated. The size of the execution trace depends on the number of rules and tasks in the ruleset.</li></ul>
Jav a 2 Se cu rit y	Enabling Java 2 Security decreases performance. You can tune your security configuration to minimize this effect. You can integrate Java 2 Security to the secure mode of the application server, but it is not mandatory.  If Java 2 Security is enabled and an application is not prepared for it, the application is likely to run incorrectly and cause Java 2 Security access control exceptions at run time.  For best performance, turn off Java 2 Security, especially on the Java Platform, Enterprise Edition.

**Parent topic:** [Managing rule execution in Rule Execution Server](#)

## Rule Execution Server administration artifacts

System administrators use the web-based Rule Execution Server console to manage and monitor RuleApps, ruleset execution, and decision services.

System administrators use the Rule Execution Server console for the following tasks:

- To determine what rulesets are deployed
- To make real-time changes to business rules: typically, they can deploy, change, and manage business rules while the application is running.

The Rule Execution Server database stores all the changes that are made to a ruleset, such as information about the user who modified it, time of modification, and any comments that were added.

### RuleApps

A RuleApp is a deployable management unit that contains rulesets. RuleApps keep records of the following details:

- RuleApp name
- RuleApp version
- Number of rulesets in the RuleApp
- RuleApp creation date

### RuleApp archives

A RuleApp archive is an archive that stores RuleApps in a file.

**Attention:** RuleApp archives are saved in a strict directory structure: if you change that directory structure, you invalidate the RuleApp.

You search for a resource in a path that begins with the parent element in the descriptor:

- The path of a RuleApp is defined by the name and the version number of the RuleApp. No resources are associated with a RuleApp.
- The path of a ruleset is defined by the name and the version number of the RuleApp, followed by the name and the version number of the ruleset. This hierarchy is called the canonical ruleset path.

The following ruleset paths are all valid. The last example shows a canonical ruleset path.

- /rule8\_8app/6\_ruleset\_6/1.0
- /rule8\_8app/1.0/6\_ruleset\_6
- /rule8\_8app/6\_ruleset\_6
- /rule8\_8app/1.0/6\_ruleset\_6/1.0

### Rulesets

You can add a ruleset to a RuleApp in one of the following ways:

- From the RuleApp View of the Rule Execution Server console in the cloud development environment.
- By deploying the ruleset within a RuleApp file.

A ruleset is deployed to Rule Execution Server along with a ruleset archive.

RuleApp archives contain largely compiled rules:

- Java™ bytecode when you compile from Rule Designer and select the **Optimize ruleset loading (Java bytecode generation)** option upon ruleset archive creation.
- Rules and ruleflows in an intermediate representation when you clear the **Optimize ruleset loading (Java bytecode generation)** option or when you generate rulesets from Decision Center.

### Java XOM resources and libraries

When you set up Java XOM management, the deployment of Java XOM to Rule Execution Server generates Java XOM resources and libraries.

**Parent topic:** [Managing rule execution in Rule Execution Server](#)

**Related information:**

[Improving the performance of Rule Execution Server](#)

# Rule artifact management and deployment

Rule Execution Server provides a comprehensive set of tools to manage the deployment and execution of rulesets.

The Rule Execution Server console is a management tool designed for system administrators, developers, or business analysts to implement changes in business requirements quickly and easily. Such changes can be uploaded to the rule engine while the application is running.

## Management

Rule Execution Server provides an environment for you to manage the deployment and execution of your rules. Rule Execution Server includes a set of management facilities. System administrators can use them to change the behavior of the execution stack and disable rulesets in emergency situations.

**Note:**  
All managed entities are versioned.

The following table shows what management tasks are available in what environment.

Table 1. Management tasks

Entity Operations	Edited by
RuleApps	
Create	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Rule Designer in the development environment only.</li><li>Decision Center</li></ul>
Modify	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Decision Center</li></ul>
Remove	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Get RuleApp Archive	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Rulesets	
Create	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Rule Designer in the development environment only.</li><li>Decision Center</li></ul>
Modify (change ruleset archives and properties)	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Decision Center</li></ul>
Enable/Disable	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Decision Center</li></ul>
Remove	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Server	
Archive all RuleApps	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
XOM resources (resuri)	
Create	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Decision Center</li></ul>
Remove	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Get resource	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
XOM resources (reslib)	
Create	<ul style="list-style-type: none"><li>Rule Execution Server console</li><li>Decision Center</li></ul>
Modify	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Remove	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>
Get library	<ul style="list-style-type: none"><li>Rule Execution Server console</li></ul>

---

**Parent topic:** [Managing rule execution in Rule Execution Server](#)

## Rule Execution Server console online help

The Rule Execution Server console is a web-based graphical interface to access Rule Execution Server features.

Use the Rule Execution Server console to manage and monitor deployed rulesets and decision services.

### [Introducing the Rule Execution Server console](#)

The Rule Execution Server console provides a web-based graphical interface that you use for managing and monitoring RuleApps, ruleset archives, Java™ XOMs and libraries, and transparent decision services.

### [Managing RuleApps](#)

After you have deployed a RuleApp to Rule Execution Server, you can manage it from the Rule Execution Server console.

### [Managing rulesets](#)

You manage rulesets from the Rule Execution Server console.

### [Managing Java XOM resources and libraries](#)

If you deploy a Java XOM to Rule Execution Server as JAR or ZIP resources, you can manage these resources through the Resources View of the Rule Execution Server console. You can deploy an ordered list of Java XOMs to Rule Execution Server as a library. When you do so, you can manage such libraries through the Libraries View of the Rule Execution Server console.

### [Monitoring ruleset execution](#)

Using the monitoring capabilities of the Rule Execution Server console, you can enable the debug mode, enable ruleset monitoring, generate ruleset execution statistics, test ruleset execution, and view execution-related events in the XU log.

### [Viewing or downloading an HTDS description file](#)

You can view or download the description file of a hosted transparent decision service to Web Service Description Language (WSDL), Web Application Description Language (WADL), or OpenAPI format.



## Introducing the Rule Execution Server console

The Rule Execution Server console provides a web-based graphical interface that you use for managing and monitoring RuleApps, ruleset archives, Java™ XOMs and libraries, and transparent decision services.

You can also use the console to test ruleset execution and manage hosted transparent decision services.

### **User interface components**

The Rule Execution Server console includes different user interface components to accomplish different tasks.

### **Bookmarking a view**

To bookmark a RuleApp or ruleset view, you use the console permanent link feature and the bookmarking options of your browser.

**Parent topic:** [Rule Execution Server console online help](#)

# User interface components

The Rule Execution Server console includes different user interface components to accomplish different tasks.

## **Top banner**

From the top banner, you access the various features of the Rule Execution Server console.

## **Menu bar**

Each view that you can display in the Rule Execution Server console has a menu bar with specific command buttons.

## **Tables and filters**

In the Rule Execution Server console, you can sort the tables by column headers and you can filter their contents to view only certain artifacts.

## **Symbols**

Icons are used throughout Rule Execution Server console to represent execution artifacts.

**Parent topic:** [Introducing the Rule Execution Server console](#)

### **Related tasks:**

[Viewing debug traces for an execution unit \(XU\)](#)

[Testing ruleset execution](#)

## Top banner

From the top banner, you access the various features of the Rule Execution Server console.

After you sign in to the Rule Execution Server console, its Home page displays a top banner with several tabs and links. Use these tabs and links to access the features of the console for the following purposes:

- Know what artifacts are deployed on Rule Execution Server and when they were deployed.
- Work with deployed artifacts: RuleApps, Java™ XOM resources, and Java XOM libraries.
- Monitor rulesets and execution units (XU).
- Enable or disable deployed resources.
- Read documentation about the Rule Execution Server console.

The top banner gives access to the following tabs and windows:

### [The Home tab](#)

Displays a brief description of each of the other tabs.

### [The Navigator panel of the Explorer tab](#)

Gives access to the RuleApps, XOM resources, XOM libraries, and Service Information.



Shows your login name.

### **Sign Out**

Logs out the user from the current Operational Decision Manager on Cloud session.

### **Print View**

Displays a plain view of the current page for printing. The link toggles to **Normal View**.

### **Help**

Displays the documentation of the Rule Execution Server console in a separate window.

## The Home tab

The Home tab is displayed each time you sign in to the Rule Execution Server console. The Home page is a welcome page that provides another way to access the other tabs and a brief description of each.

## The Navigator panel of the Explorer tab

The **Explorer** tab gives access to views of the following execution artifacts:

- Deployed RuleApp and rulesets archives
- Java XOM JAR and ZIP resources
- Java XOM libraries

The Navigator panel is a side panel in the Explorer page. You can hide the Navigator panel by clicking the arrow handle. Click the arrow handle again to redisplay the Navigator panel.

Use the breadcrumbs that are provided in the Explorer pages to locate where you are and where you come from. The breadcrumbs are also links that you can click to move back along the same path.

Use the expandable tree to work with the artifacts.

### **RuleApps**

Click **RuleApps** to display a RuleApps View and, from it, access a RuleApp View on a deployed RuleApp and a Ruleset View on a ruleset. For more information, see [Managing RuleApps](#) and [Managing rulesets](#).

### **Resources**

Click **Resources** to display a Resources View and, from it, access a Resource View on a deployed Java XOM resource. For more information, see [Managing Java XOM resources and libraries](#).

### **Libraries**

Click **Libraries** to display a Libraries View and, from it, access a Library View on a Java XOM library. For more information, see [Managing Java XOM resources and libraries](#).

**Parent topic:** [User interface components](#)

# Menu bar

Each view that you can display in the Rule Execution Server console has a menu bar with specific command buttons.

At the top of each view in the Rule Execution Server console, a menu bar provides commands that are specific to that view. The menu commands available also depend on user rights.

Table 1. Explorer - RuleApps - Menu bar commands

Page and View	Command	Action	Comment
RuleApps View	Add RuleApp	Creates an empty RuleApp to which you can add rulesets later.	Available in the development environment only, not in test or production.
	Deploy RuleApp Archive	Imports the RuleApp archive that you select and deploys it using one of the version policies.	Available in the development environment only, not in test or production.
	Update RuleApps	Refreshes the RuleApps view to include the RuleApps that have just been deployed on the Rule Execution Server. The execution unit (XU) is updated.	
RuleApp View	Add Ruleset	Creates a ruleset within the selected RuleApp by importing a ruleset archive.	Available in the development environment only, not in test or production.
	Add Property	Adds a predefined or custom RuleApp property.	Available in the development environment only, not in test or production.
	Download Archive	Saves the RuleApp as a RuleApp archive.	
	Edit	Directly edits the display name and description of the RuleApp, and properties and included rulesets.	

Table 2. Explorer - Rulesets - Menu bar commands

Page and View	Command	Action	Comment
Ruleset View	View Statistics	Displays ruleset execution statistics, including all the visible execution units.	
	View Execution Units	Displays all execution units. Error and warning messages are attached to the selected ruleset.	
	Upload Ruleset Archive	Overrides the ruleset with an imported ruleset archive.	Available in the development environment only, not in test or production.
	Add Managed URI	Adds a reference to the URI of a managed Java™ XOM resource or library.	Available in the development environment only, not in test or production.
	Add Property	Adds a predefined or custom ruleset property.	Available in the development environment only, not in test or production.
	Edit	Edits the status and properties of the ruleset.	
	Retrieve WSDL	Displays or downloads the WSDL or WSDL code for a hosted transport	

	<b>RTDS Description n File</b>	WADL code for a hosted transparent decision service.	
Ruleset Statisti cs View	<b>Refresh</b>	Adds the latest execution in the statistics.	
	<b>Reset</b>	Resets execution statistics to null for the selected ruleset.	
Ruleset Executi on Units	<b>Refresh</b>	Refreshes the list of execution units for the selected ruleset.	
	<b>Reset Execution Unit Messages</b>	Resets the messages that pertain to the selected ruleset for all visible execution units.	

Table 3. Explorer - Resources - Menu bar commands

Pa ge and Vie w	Command	Action	Comment
Res ourc es Vie w	<b>Deploy resource</b>	Deploys the selected resource to the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	<b>Clean up resources</b>	Displays the list of invalid and unused resources.	
Res ourc e Vie w	<b>Remove</b>	Removes the selected resource from the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	<b>Download Resource</b>	Downloads the selected resource as a JAR or ZIP file. The location depends on your browser settings.	
Explorer - Libraries			
Libr arie s Vie w	<b>Add Library</b>	Adds a library to the libraries archive.	Available in the development environment only, not in test or production.
	<b>Clean up Libraries</b>	Displays the list of invalid and unused libraries.	
Libr ary Vie w	<b>Remove</b>	Removes the selected library from the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	<b>Add Internal Resource Reference</b>	Adds to this library a reference to a JAR or ZIP file that is already deployed to the Rule Execution Server instance.	Available in the development environment only, not in test or production.
	<b>Add Library Reference</b>	Adds a reference to another library.	Available in the development environment only, not in test or production.

Parent topic: [User interface components](#)

## Tables and filters

In the Rule Execution Server console, you can sort the tables by column headers and you can filter their contents to view only certain artifacts.

### Tables

In the various views, execution artifacts such as RuleApps, rulesets, and Java™ XOM resources and libraries are listed in tables, with one row for each artifact and columns for meaningful information such as name, version, ruleset path, URIs. You can sort the contents of these tables by clicking any column header. For example, in a RuleApp view, you can order the listed rulesets on their ruleset path by clicking the **Ruleset Path** column header.

### Filters

The Rule Execution Server console provides filters to limit what content is displayed inside a table. The filters are displayed at the top of each table.

The following filters are available:

#### View

The **View** filter is associated with the following check boxes, which you can select in combination:

- Select **Latest version** if you want to display only the latest version of each ruleset.
- Select **Enabled** if you want to display only the rulesets for which the **Enable** option is selected. This check box is available only in RuleApp Views.
- Select **Debug** if you want to display only the rulesets that have the debug mode enabled. This check box is available only in RuleApp Views.

#### Name

The **Name** field is not case-sensitive. It uses the [startsWith](#) method if you type directly into the field and also supports regular expressions, as documented on the [Class Pattern](#) page of the Java API documentation. For example, enter `.*xyz.*` in this field if you want to list only names that contain "xyz".
















#### Display by:

To select the number of artifacts to be displayed in the table, select **5**, **10**, **50**, or **100** from the drop-down menu.

**Parent topic:** [User interface components](#)

# Symbols

Icons are used throughout Rule Execution Server console to represent execution artifacts.

Symbol	Represents
	In a RuleApps View, several RuleApps  In a Resources View, several Java™ XOM resources  In a Libraries View, several Java XOM libraries
	In a RuleApps View, a RuleApp  In a Resources View, a Java XOM resource  In a Libraries View, a Java XOM library
	In a RuleApp View, gives access to the RuleApp properties table  In a Resource View or Library View, gives access to the tables of references
	In a RuleApp View, a ruleset  In a Resource View or Library View, a ruleset reference
	In a Ruleset View, ruleset properties
	In a Ruleset View, ruleset input parameters
	In a Ruleset View, ruleset output parameters
	In a Ruleset View, introduces the table of ruleset parameters
	In a Ruleset Statistics View, ruleset statistics
	In a Ruleset View, ruleset archive elements
	In a Ruleset View, execution unit (XU)
	In a Ruleset View, execution unit (XU) messages on a ruleset
	The resource or library is valid and used
	The resource or library is valid but not used
	The resource or library is not valid

Parent topic: [User interface components](#)

## Bookmarking a view

To bookmark a RuleApp or ruleset view, you use the console permanent link feature and the bookmarking options of your browser.

### About this task

Most web browsers support bookmarking to help you keep track of important web pages. In the Rule Execution Server console, pages do not ordinarily display individual URLs. The permanent link feature provides a way to navigate directly to a specified page.

### Procedure

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator pane.
3. Click the name of the relevant RuleApp.

The RuleApp View is displayed. You now have a full description of the RuleApp, including version, creation date, properties, and Rulesets.

4. Do one of the following steps:
  - For a RuleApp view, click **Permanent link** in the lower corner of the RuleApp description box.
  - For a ruleset view, click the relevant ruleset. In the ruleset view, click **Permanent link** in the lower corner of the ruleset description box.

The full URL is shown in the address bar of your browser.

5. Use the "bookmark" or "favorite" function of your browser to bookmark the page as you would any other web page.

### Results

#### Tip:

One reason for not using bookmarks is that they might become out of date. Rule Execution Server console web pages are no different in this respect from other web pages because over time, users remove, archive, or version RuleApps and rulesets.

**Parent topic:** [Introducing the Rule Execution Server console](#)



# Managing RuleApps

After you have deployed a RuleApp to Rule Execution Server, you can manage it from the Rule Execution Server console.

You can create, edit, and remove RuleApps and their properties. You can deploy and undeploy these RuleApps.

You can also archive a RuleApp to a JAR file. The RuleApp is not disabled or deleted from the server.

## **Archiving RuleApps**

You can package a RuleApp into a JAR file as a RuleApp archive and download it.

## **Deploying RuleApp archives**

You deploy a RuleApp archive from the RuleApps View. Managed Java™ XOMs can be included in the deployed RuleApp archive. You must choose versioning policies before you deploy the archive.

## **Adding RuleApps**

You add new RuleApps from the RuleApps View. Use only valid characters for RuleApp names.

## **Setting RuleApp properties**

You can add properties and rulesets to a deployed RuleApp, or remove existing properties or rulesets.

## **Versioning policy options**

You use versioning options to control how RuleApp, ruleset, resource, and library versions are numbered and replaced.

**Parent topic:** [Rule Execution Server console online help](#)

# Archiving RuleApps

You can package a RuleApp into a JAR file as a RuleApp archive and download it.

## About this task

You can package and download a RuleApp archive that includes all rulesets or selected rulesets in the RuleApp:

- Download a RuleApp archive with **all rulesets** in the RuleApp. This feature is available in the following views in the **Explore** tab:
  - RuleApps View
  - RuleApp View
- Download a RuleApp archive with **selected rulesets** in the RuleApp. This feature is available only in the RuleApp View page.

## Procedure

1. Click the **Explore** tab
2. In the Navigator panel, click **RuleApps** to display the RuleApps View page.
3. Download a RuleApp archive. Choose one of the options in the following table.

Table 1. How to download a RuleApp archive

Include all rulesets in the RuleApp archive		Include selected rulesets in the RuleApp archive
From RuleApps View	From RuleApp View	From RuleApp View
a. Select a RuleApp in the RuleApps table, and click <b>Download Archive with All Rulesets</b> next to a green arrow in the row.	a. Click a RuleApp in the RuleApps table. The RuleApp View page is displayed with a list of rulesets. b. Click <b>Download Archive with All Rulesets</b> next to a green arrow.	a. Click a RuleApp in the RuleApps table. The RuleApp View page is displayed with a list of rulesets. b. Select the rulesets that you want to include from the table, and click <b>Download</b> .

4. Save the file. The file is downloaded in the location that you specified.

## Results

The RuleApp archive is now created and downloaded in the specified location. The archive contains the RuleApp and its rulesets. The RuleApp itself is not changed or deleted from the server.

**Parent topic:** [Managing RuleApps](#)

# Deploying RuleApp archives

You deploy a RuleApp archive from the RuleApps View. Managed Java™ XOMs can be included in the deployed RuleApp archive. You must choose versioning policies before you deploy the archive.

## Before you begin

<b>Restriction:</b> This action is available in the development environment only.
-----------------------------------------------------------------------------------

## Procedure

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator panel to display the RuleApps View.
3. Click **Deploy RuleApp Archive** in the RuleApps View.
4. Click **Browse** and select the RuleApp file (.jar) to be deployed.

You can select the RuleApp archive file from your local computer or from the file system.

5. In the Deploy RuleApp Archive page, verify that appropriate versioning policies are selected in the **RuleApp Versioning Policy** section.
  - a. Optional: If there are managed Java XOMs that are included in the RuleApp archive, verify that appropriate versioning policies are selected in the **Resource Versioning Policy** and **Library Versioning Policy** sections.

<b>Note:</b> The selected resource and library versioning policy options are ignored when there is no managed Java XOM in the RuleApp archive.
------------------------------------------------------------------------------------------------------------------------------------------------

For more information about the versioning policy options, see [Versioning policy options](#)

6. Click **Deploy**.

Deployment Report is displayed in the RuleApps View. See the **Details** section in the report to check what was deployed.

7. Click **OK** to return to the RuleApps View.

## Results

When you deploy a RuleApp archive, you create a RuleApp on Rule Execution Server. The RuleApp is then listed in the RuleApp navigator.

**Parent topic:** [Managing RuleApps](#)

### Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

### Related tasks:

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM resources](#)

[Removing referenced Java XOM libraries](#)

# Adding RuleApps

You add new RuleApps from the RuleApps View. Use only valid characters for RuleApp names.

## About this task

This action is available in the development environment only.

## Procedure

To add a RuleApp:

1. Click the **Explorer** tab.
2. Click **RuleApps** in the Navigator panel to display the RuleApps View.
3. Click **Add RuleApp** in the RuleApps View.

The Add New RuleApp page is displayed.

4. Click in the **Name** field and type the name of the new RuleApp.

Use only a–z, A–Z, 0–9, and underscore (\_). For the first character, 0–9 is not allowed.

### Note:

If you set Rule Execution Server persistence to file, remember that Windows environments do not support case sensitivity.

5. Click the default version number and type the new version number, where the major version number is  $\geq 1$  and the minor version number is  $\geq 0$ .
6. Click **Add**.

## Results

The new RuleApp is added to the RuleApps list.

When you add a RuleApp, an empty container is created. It has a name and version number. You can edit the container to add one or more rulesets and a set of entities that specify how the rulesets are executed.

**Parent topic:** [Managing RuleApps](#)

# Setting RuleApp properties

You can add properties and rulesets to a deployed RuleApp, or remove existing properties or rulesets.

You can add properties and rulesets to a deployed RuleApp in the development environment only.

You can add predefined or custom properties to a RuleApp from the RuleApps View. A RuleApp property is a property that applies to individual RuleApps. You can set RuleApp predefined or custom properties in one of the following ways:

- Before rule deployment in Rule Designer.
- In the Rule Execution Server console after the rules are deployed to Rule Execution Server.

The following table lists the predefined RuleApp properties.

Table 1. RuleApp properties

Name	Description
<code>ruleapp.interceptor.classname</code>	The full name of the ruleset execution interceptor to be used
<code>ruleapp.interceptor.description</code>	A description of the ruleset execution interceptor class

Parent topic: [Managing RuleApps](#)

# Versioning policy options

You use versioning options to control how RuleApp, ruleset, resource, and library versions are numbered and replaced.

RuleApp version numbering is different depending on whether the RuleApp in the selected archive file exists in the server memory:

- If the RuleApp does not exist, it is deployed to Rule Execution Server with the version number 1.0, irrespective of the selected deployment policy.
- If the RuleApp already exists, it is deployed according to the versioning policy.

Here are examples that show the results of each versioning option.

## Increment RuleApp major version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /RuleApp/2.0 /ruleset/1.0</div>

## Increment RuleApp minor version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /RuleApp/1.1 /ruleset/1.0</div>

## Replace RuleApp version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>

## Increment rulesets major version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0 /ruleset/3.0</div>

## Increment rulesets minor version

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>	<div>/RuleApp/1.0 /ruleset/1.0</div>	<div>/RuleApp/1.0 /ruleset/1.0 /ruleset/2.0</div>

<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div> <div>/ruleset/1.1</div>
-------------------------------------------------------------------------	-------------------------------------------------	-------------------------------------------------------------------------------------------------

Replace rulesets versions

The following example shows the result after deployment if you select this option.

Deployed RuleApp	RuleApp to be deployed	Result after deployment
<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div>	<div>/RuleApp/1.0</div> <div>/ruleset/1.0</div> <div>/ruleset/2.0</div>

Increment resources major version

The following example shows the result after deployment if you select this option.

Deployed resource	Resource to be deployed	Result after deployment
<div>/resource/1.0</div>	<div>/resource/1.0</div> <div><b>Note:</b> The checksum is different from the one in the Deployed resource column.</div>	<div>/resource/1.0</div> <div>/resource/2.0</div>

Increment resources minor version

The following example shows the result after deployment if you select this option.

Deployed resource	Resource to be deployed	Result after deployment
<div>/resource/1.0</div>	<div>/resource/1.0</div> <div><b>Note:</b> The checksum is different from the one in the Deployed resource column.</div>	<div>/resource/1.0</div> <div>/resource/1.1</div>

Increment libraries major version

The following example shows the result after deployment if you select this option.

Deployed library	Library to be deployed	Result after deployment
<div>/library/1.0</div>	<div>/library/1.0</div>	<div>/library/1.0</div> <div>/library/2.0</div>

Increment libraries minor version

The following example shows the result after deployment if you select this option.

Deployed library	Library to be deployed	Result after deployment
<div>/library/1.0</div>	<div>/library/1.0</div>	<div>/library/1.0</div> <div>/library/1.1</div>

--	--	--

**Parent topic:** [Managing RuleApps](#)



## Managing rulesets

You manage rulesets from the Rule Execution Server console.

Managing rulesets can require you to enable, disable, or debug rulesets, add ruleset properties, set WSDL options, manipulate ruleset archives, and set references to Java™ XOM resources.

You disable a ruleset to prevent its execution. When you disable a ruleset, this ruleset is not taken into account by the rule engine at execution time.

You can use the remote debugging feature to have Rule Designer add breakpoints in the ruleset. The debug URL points to the remote machine where Rule Designer runs. This URL consists of the machine name and the selected debugger port.

You can add predefined or custom properties to a ruleset to specify execution information. A ruleset property is a property that applies to individual rulesets. You set predefined or custom properties on a ruleset to specify execution information, in one of the following ways:

- Before rule deployment in Rule Designer.
- In the Rule Execution Server console after you have deployed your rules to Rule Execution Server.

You can add ruleset properties in the development environment only.

You can add managed URIs to a ruleset and modify the order of precedence between managed URIs at run time. By doing so, you modify the index of the selected reference. The index of a managed URI determines its position in the `ruleset.managedxom.uris` ruleset property, hence its precedence order at run time. The managed URI with index 1 executes first. Modify the order of the URIs to update, test, or diagnose various execution behaviors.

### **Changing target namespaces**

When you generate the WSDL of a SOAP hosted transparent decision service (HTDS) or the WADL of a REST transparent decision service, you can change the namespace of the transparent decision service to avoid naming conflicts when more than one description file is generated at the same location.

### **Testing a ruleset for REST execution**

From the Rule Execution Server console, you test REST API requests for execution of a ruleset in XML or in JSON format.

### **Setting ruleset properties**

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

**Parent topic:** [Rule Execution Server console online help](#)

## Changing target namespaces

When you generate the WSDL of a SOAP hosted transparent decision service (HTDS) or the WADL of a REST transparent decision service, you can change the namespace of the transparent decision service to avoid naming conflicts when more than one description file is generated at the same location.

### About this task

When more than one description file is generated at the same location, naming conflicts can arise because the parameters might not be the same for different transparent decision services. Such is the case, for example, when more than one transparent decision service are added to a business integration project in IBM® Integration Designer. To avoid such conflicts, the URI for transparent decision service requests and responses, and their parameters, includes the ruleset name. The following example the URL for a ruleset named PreTradeChecks.

```
<wsdl:definitions name="PreTradeChecksDecisionService"
targetNamespace="http://www.ibm.com/rules/decisionservice/Execution/PreTradeChecks"
...

```

The schema importation mechanism references schema components from other schema documents.

```
...
<xsd:import
namespace="http://www.ibm.com/rules/decisionservice/Execution/PreTradeChecks/p
aram"/
...

```




The names of decision requests and responses are prefixed with the RuleApp name. For example:

```
<xsd:element name="PreTradeChecksRequest"
...
xsd:element name="PreTradeChecksResponse"

```

The value of the target namespace must be an absolute URI. To ensure the uniqueness of the target namespace, you can change the definition of this attribute in the Ruleset Parameters section of the Ruleset View.

### Procedure

1. In the Rule Execution Server console, click the **Explorer** tab, expand the RuleApp that you want, and select a ruleset to open the Ruleset View.
  2. At the bottom of the Ruleset View, click **Show HTDS Options**.
  3. To change the target namespace of a SOAP transparent decision service, proceed as follows:
    - a. Next to the **Target namespace (SOAP only)** URI, click the  **Edit** icon. In the option name, **SOAP only** means that changing the namespace makes sense only for WSDL code generation. When you generate the WADL code for a ruleset, you can change the target namespace for ruleset parameters.
    - b. Type the custom URL in the field.
    - c. Click the  **Save** icon.
- You can cancel by clicking the **Undo** arrow next to the **Save** icon.
4. Optional: For either a SOAP or a REST transparent decision service, if applicable, edit the URL for the ruleset parameters as follows:
    - a. Click **Parameter target namespace**.
    - b. Type the custom URL in the field.
    - c. Click the  **Save** icon.

You can cancel by clicking the **Undo** arrow next to the **Save** icon.

**Parent topic:** [Managing rulesets](#)

# Testing a ruleset for REST execution

From the Rule Execution Server console, you test REST API requests for execution of a ruleset in XML or in JSON format.

## Before you begin

### Restriction:

- Browser compatibility: The test feature is not supported by Internet Explorer 8 and earlier versions. If you work with Internet Explorer 9 or 10, make sure that the **Tools > Compatibility View** command is deactivated.
- Generation of OpenAPI description files is supported for only Java™ XOMs, while WSDL and WADL file generation are supported for XML XOMs and Java XOMs with JAXB annotations.

## About this task

Before you execute a ruleset in XML or in JSON form from a client, you can test your execution request in the Rule Execution Server console.

## Procedure

1. Select a ruleset and open the Ruleset View.
2. Click **Retrieve HTDS description file**.
3. Select **REST** in the **Service protocol type** section.
4. Select **WADL**, **OpenAPI - YAML**, or **OpenAPI - JSON** from the **Format** pull-down menu as explained in [Viewing or downloading an HTDS description file](#).

**Remember:** You can select **OpenAPI - YAML** and **OpenAPI - JSON** only for Java XOMs.

5. Click **Test**.


The execution request is displayed in a separate browser window.

If you selected WADL in the previous step, you can choose **XML** or **JSON** for Java XOMs as an execution format from the **Execution Request** drop-down menu. For XML XOMs, you have XML as an execution format.

If you selected OpenAPI, you have only one execution format: JSON.

6. Click **Execute Request**.

The response is displayed in the Server Response frame. If you selected **Decision trace information** before you click **Test**, the trace filter is added to the request and the decision trace is included in the response.

7. Optional: If you selected XML as an execution format, click the  **Validate XML** icon to validate the generated XML code.

**Note:** The JSON format does not support validation.

If the request is invalid, an error message indicates the reason and location of the failure. A red dot is displayed next to the incorrect line in the request.

8. Optional: Use the **Undo** and **Redo** icons to erase or restore what you type in the text areas.

**Parent topic:** [Managing rulesets](#)

### Related tasks:

[Testing ruleset execution](#)

## Setting ruleset properties

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

Ruleset properties apply to individual rulesets. You can set ruleset properties in one of the following ways:

- In Rule Designer before you deploy.
- In the Decision Center consoles before you deploy.
- In the Rule Execution Server console after deployment.
- Using the Rule Execution Server REST API.

**Note:** You can add or edit ruleset properties in the Rule Execution Server console only in the development environment.

To set values for the built-in ruleset properties in the Rule Execution Server console:

1. Click **Add Property** in the Ruleset View page.
2. In the New Ruleset Property page, select a ruleset property from the menu. Check **Predefined** in the **Property Type** section. Set a value for the selected ruleset property.

**Tip:**

- If the built-in ruleset property that you want to use is not listed in the menu when you select **Predefined**, select **Custom** instead and enter the name and the value of the property.
- You cannot rename ruleset properties. Instead, you must remove the property and create a new one.

### [Built-in ruleset properties](#)

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

### [Built-in ruleset properties for OpenAPI](#)

Some information in the OpenAPI description file can be overridden by specifying the ruleset properties for OpenAPI.

### [Setting the ruleset property for increasing rule execution performance](#)

You can significantly increase rule execution performance by increasing the maximum size of the document driver pool.

**Parent topic:** [Managing rulesets](#)

# Built-in ruleset properties

You can use built-in ruleset properties to stock deployment or execution information for a particular ruleset.

The following tables provide the list of built-in ruleset properties.

Table 1. Built-in ruleset properties

Name	Valid values	Default value	Description
ilog.rules.teamserver.permalink.report	A URL as a string		This property contains the URL to access the Decision Center report.
decisioncenter.url	A URL		The URL of the Decision Center server from which the deployment was done.
decisioncenter.buildnumber	A string		The number of the Decision Center build.
decisionservice.name	A string		The name of the decision service that was deployed, for example, loanvalidation-rules-service.
decisionservice.id	A string		The ID of the root project of the decision service that was deployed, for example, brm.RuleProject:44:44.
decisionservice.branch.name	A string		The name of the deployment snapshot if the user chose to create one, the branch, or the snapshot in the decision service that was deployed, for example, b or d-20150231-158621.
decisionservice.branch.id	A string		The ID of the deployment snapshot, branch, or snapshot in the decision service that

			was deployed, for example, brm.Branch:224:224.
<b>decisionservice.branch.url</b>	A URL		A link to a page in the Decision Center Business console that displays the deployed branch or snapshot.
<b>decisionservice.deploymentConfiguration.name</b>	A string		The name of the deployment configuration that was deployed, for example, d.
<b>decisionservice.deploymentConfiguration.id</b>	A string		The ID of the deployment configuration that was deployed, for example, dsm.Deployment:3:3.
<b>decisionservice.deployer.name</b>	A string		The display name of the Decision Center user who did the deployment.
<b>decisionservice.deployer.id</b>	A string		The login name of the Decision Center user who did the deployment.
<b>ruleset.engine.version</b>	A string		The MMU of the rule engine. for example, 1.20.0.
<b>ilog.console.wsdl.endpoint</b>	A URL as a string		Use this property to override the default HTDS option for the web service endpoint.
<b>ilog.console.htds.context</b>	A URL as a string		Use this property to override the default HTDS option for the location.
<b>wsdl.targetnamespace</b>	A URL as a string		Use this property to override the default HTDS option for the target namespace.
<b>wsdl.paramtargetnamespace</b>	A URL as a string		Use this property to override the default HTDS options for the parameter

			<p>target namespace.</p> <div><p><b>Note:</b> This parameter can be used for both WSDL and WADL code generation , although <b>wsdl</b> is used as part of the parameter name. You cannot change <b>wsdl</b> to <b>wadl</b> in the parameter name even when this parameter is used for the WADL code generation .</p></div>
<b>ruleset.managedxom.uris</b>	<p>A comma-separated list of URIs. For example:</p> <p>resuri://common-classes.jar/1.0,resuri://LoanValidation.jar</p>	No default value	<p>This ruleset property controls Java™ XOM management. It locates the Java XOM resources for the ruleset.</p> <ul style="list-style-type: none"><li>• If you do not set this property and the ruleset uses a Java XOM, the Java classes are loaded by the application class loader.</li><li>• If you set this property (as a list of URIs) for a ruleset , the execution unit (XU)</li></ul>

			<p>creates and stores a dedicated class loader for the execution of the ruleset. Each URI must target a Java Archive (.jar file or a .zip archive of classes and resources for Java execution.</p> <p>Only internal URIs are supported: resuri and reslib protocol</p>
<b>ruleset.maxIdleTime</b>	<p>Three possible values:</p> <ul style="list-style-type: none"><li>• <b>-1</b> or undefined : the ruleset is removed from the cache, depending on its usage by the JCA Connection Pool.</li><li>• <b>=&gt;0</b> : the ruleset is removed from the cache</li></ul>		<p>This property enforces the ruleset pool policy on a ruleset. A ruleset stays in memory until the maximum idle time (in seconds) has reached the specified value set for this property. To avoid ruleset reparsing, you can use the special value of 0 to ensure that the ruleset is never released from memory.</p> <div><p><b>Important:</b> Use the 0 value with caution as it might introduce a significant memory leak if the</p></div>



	<div>after the time out (in seconds) is reached and no SPI connections reference it any more.</div> <ul style="list-style-type: none"><li>• 0: the ruleset is never removed from the cache except if the ruleset is redeployed.</li></ul>		<div>ruleset is not used any more.</div>
<b>ruleset.trace.enabled</b>	true, false		This property enables or disables the rule engine trace mode.
<b>ruleset.xmlDocumentDriverPool.maxSize</b>	>=0	1	<div>The 0 value means that an XMLDocumentDriver instance is created for each XMLObject transformation.</div> <div>Use a strictly positive value to specify the maximum size of the IlrXMLDocumentDriver pool, that is,</div> <div>the maximum number of used and unused IlrXMLDocumentDriver</div>

			objects per ruleset.
<code>ruleset.xmlDocumentDriverPool.reserveTimeout</code>	<code>&gt;=0</code>		This property specifies the number of milliseconds after which the call to reserve an <code>IlrXMLDocumentDriver</code> instance times out.

Parent topic: [Setting ruleset properties](#)

# Built-in ruleset properties for OpenAPI

Some information in the OpenAPI description file can be overridden by specifying the ruleset properties for OpenAPI.

Table 1. Built-in ruleset properties for OpenAPI

Name	Valid values	Default value	Description
openapi.info.title	A string		See the description in the <a href="#">Info Object</a> section in the OpenAPI Specification.
openapi.info.description	A string		See the description in the <a href="#">Info Object</a> section in the OpenAPI Specification.
openapi.execute.operation.summary	A string		See the description in the <a href="#">Operation Object</a> section in the OpenAPI Specification.
openapi.execute.operation.description	A string		See the description in the <a href="#">Operation Object</a> section in the OpenAPI Specification.
openapi.execute.operation.operationId	A string		See the description in the <a href="#">Operation Object</a> section in the OpenAPI Specification.

Parent topic: [Setting ruleset properties](#)

# Setting the ruleset property for increasing rule execution performance

You can significantly increase rule execution performance by increasing the maximum size of the document driver pool.

## About this task

By default, the **ruleset.xmlDocumentDriverPool.maxSize** ruleset property value is set to 1. This value might cause a bottleneck if multiple clients execute a hosted transparent decision service concurrently. By increasing the value of this property, you can significantly increase the performance of ruleset execution.

Set the value of the ruleset property in the Ruleset View of the Rule Execution Server console.

## Procedure

1. Log in to the Rule Execution Server console.
2. Click the **Explorer** tab.
3. Click **RuleApps** in the Navigator panel to display the RuleApps View.
4. In the RuleApps View, click the name of the RuleApp that contains the ruleset.
5. In the RuleApp View, click the relevant ruleset.
6. In the Ruleset View, click **Add Property**.
7. In the New Ruleset Property form, select the predefined property **ruleset.xmlDocumentDriverPool.maxSize** in the drop-down list.
8. Set the required value.

The optimal value depends on the number of concurrent executions of the ruleset.

For example, for five concurrent clients, 5 can be a good value.

9. Click **Add**.

**Parent topic:** [Setting ruleset properties](#)

## Managing Java XOM resources and libraries

If you deploy a Java™ XOM to Rule Execution Server as JAR or ZIP resources, you can manage these resources through the Resources View of the Rule Execution Server console. You can deploy an ordered list of Java XOMs to Rule Execution Server as a library. When you do so, you can manage such libraries through the Libraries View of the Rule Execution Server console.

Managed Java XOM resources (JAR) and libraries can be included in a deployed RuleApp archive. These resources and libraries are called embedded managed Java XOMs.

### **Java XOM resources**

After you have deployed a Java XOM as JAR or ZIP resources, you can view these resources in the Rule Execution Server console.

### **Java XOM libraries**

After you have deployed an ordered list of Java™ XOMs as a library, you view this library in the Libraries View of the Rule Execution Server console.

### **Embedded managed Java XOM in Rule Execution Server**

Embedded managed Java XOMs are managed Java XOM resources (.jar) and libraries that are included in a RuleApp archive. You can deploy RuleApp archives with embedded managed Java XOMs in the Rule Execution Server console. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

### **Removing referenced Java XOM resources and libraries**

You want to remove a Java XOM resource or library at one point, but it might be referenced from rulesets or libraries. Before you remove a resource or library that is referenced, you must remove all links that point from these rulesets and libraries to the target resource or library. The library that you want to remove might be referencing resources or other libraries as well. Before you remove a library, all links that point from the library to these resources or libraries are removed as well.

**Parent topic:** [Rule Execution Server console online help](#)

## Java XOM resources

After you have deployed a Java™ XOM as JAR or ZIP resources, you can view these resources in the Rule Execution Server console.

From the Resource View, you can:

- Deploy new Java XOM JAR or ZIP resources and increment the major version or minor version of the resource.
- Remove a previously deployed JAR or ZIP resource. Execution units (XU) are notified of the removal, which is logged as a message in the console.

**Attention:** When the value of the **Checksum** section of a resource is empty, it means that the resource file is empty. Using the empty resource file can corrupt the database.

You can deploy and remove resources in the development environment only.

Use the **Show references** buttons to show or hide the list of rulesets and the list of libraries that use the resource. The `ruleset.managedxom.uri` ruleset property allocates the resources to the ruleset at run time. Click **Show references from rulesets** to display the list of the rulesets that, directly or indirectly, use the resource in the `ruleset.managedxom.uri` ruleset property.

You can view invalid and unused resources so that you can clean up the resources archive. The Resources Clean-up View contains information that can help you correct the invalid references or remove the invalid or unused resources. Resources are diagnosed as invalid based on the algorithm supported by the **res-diagnose-xom-uri** Ant task.

Each deployed JAR or ZIP resource is flagged with an icon that reflects its status.

### Green

The icon is green  if the following conditions are all satisfied:


- The data from the URI can be read.
- The checksum that is stored in the persistence layer is the same as the checksum that is computed from that data.
- At least one ruleset uses the resource. A ruleset can use the resource transitively through a library that might depend on another library.

### Yellow

The icon is yellow  if the following conditions are all satisfied:

- The data from the URI can be read.
- The checksum that is stored in the persistence layer is the same as the checksum that is computed from that data.
- The resource is never used in any ruleset, even transitively through interdependent libraries.

### Red

The icon is red  in one of the following cases:

- The data from the URI cannot be read.
- The checksum that is stored in the persistence layer is not the same as the checksum that is computed from that data.

**Parent topic:** [Managing Java XOM resources and libraries](#)

## Java XOM libraries

After you have deployed an ordered list of Java™ XOMs as a library, you view this library in the Libraries View of the Rule Execution Server console.

From the Libraries View, you can:

- Add a library to an archive of libraries.
- Add references to custom resources, to internal resources, or to other libraries to a library.
- Remove a library from an archive of libraries. When you remove a library, you erase the selected library from the managed Java™ XOM but do not delete the internal resources and libraries that the library was referencing. To remove an internal reference to a resource or library, you remove its URI manually in the value of the **ruleset.managedxom.uris** property.

You can add a library, add references to a library, and remove libraries in the development environment only.

Use the **Show references** buttons to show or hide the list of rulesets that use this library in its **ruleset.managedxom.uris** property, directly or indirectly, through one or several libraries.

You can view invalid and unused libraries and then clean up the libraries archive. The Libraries Clean-up View contains information that can help you correct the invalid references or remove the invalid or unused libraries. Libraries are diagnosed as invalid based on the algorithm that the **res-diagnose-xom-uri** Ant task supports.

Each deployed library is flagged with an icon that reflects its status.

### Green

The icon is green  if the following conditions are satisfied:


- All the referenced URIs are valid.
- At least one ruleset uses the library or is used by a ruleset-referenced library. Any action to remove this library leads to an inconsistent state.

### Yellow

The icon is yellow  if the following conditions are satisfied:

- All referenced URIs are valid.
- No ruleset uses the library.

### Red

The icon is red  if at least one referenced URI is not valid.



**Parent topic:** [Managing Java XOM resources and libraries](#)

# Embedded managed Java XOM in Rule Execution Server

Embedded managed Java XOMs are managed Java™ XOM resources (.jar) and libraries that are included in a RuleApp archive. You can deploy RuleApp archives with embedded managed Java XOMs in the Rule Execution Server console. By embedding managed Java XOMs in a RuleApp archive, you can easily maintain the version of the managed Java XOM and the version of the corresponding ruleset together.

The embedded managed Java XOM feature can be operated from the Rule Execution Server console.

Table 1. Embedded Managed Java XOMs

Module or tool	Deploy RuleApp archive with embedded managed Java XOM	Create or retrieve RuleApp archive with embedded managed Java XOM	Description
Rule Execution Server console			For more information, see the following topics: <ul style="list-style-type: none"><li>• <a href="#">Deploying RuleApp archives</a></li><li>• <a href="#">Removing referenced Java XOM resources and libraries</a></li></ul>

Parent topic: [Managing Java XOM resources and libraries](#)



## Removing referenced Java XOM resources and libraries

You want to remove a Java™ XOM resource or library at one point, but it might be referenced from rulesets or libraries. Before you remove a resource or library that is referenced, you must remove all links that point from these rulesets and libraries to the target resource or library. The library that you want to remove might be referencing resources or other libraries as well. Before you remove a library, all links that point from the library to these resources or libraries are removed as well.

### About this task

Remove all links to/from the resource or library, and then remove the resource or library itself.

**Note:** If you do not remove the resource or library after removing these links, references might not be updated when you redeploy the RuleApp. If it occurs, you must update them manually.

#### Removing referenced Java XOM resources

Before you remove a Java XOM resource that is referenced from rulesets or libraries, you must first remove all links that point to the resource.

#### Removing referenced Java XOM libraries

Before you remove a Java XOM library that is referenced from rulesets or other libraries, you must first remove all links that point to the library. When the links pointing to the library are removed, links that are pointed from the library are removed at the same time.

**Parent topic:** [Managing Java XOM resources and libraries](#)

#### **Related concepts:**

[Embedded managed Java XOM in Rule Execution Server](#)

#### **Related tasks:**

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources](#)

[Removing referenced Java XOM libraries](#)

# Removing referenced Java XOM resources

Before you remove a Java™ XOM resource that is referenced from rulesets or libraries, you must first remove all links that point to the resource.

## About this task

You want to remove a Java XOM resource at one point, but it might be referenced from rulesets or libraries. Before you remove a resource that is referenced from rulesets or libraries, you must remove the following links:

- Links pointing from rulesets to the resource that you want to remove
- Links pointing from libraries to the resource that you want to remove

## Procedure

1. In the **Explorer** tab, open Resource View of the Java XOM resource that you want to remove.

See the following sections to check whether the resource is referenced from other artifacts:

- **Show References from Rulesets**
- **Show References from Libraries**

2. Click **Remove All References**.

A warning message is displayed.

**Remember:** Only the links that are pointing to the resource are deleted. The artifacts that reference the resource, such as rulesets and libraries, are not deleted.

3. Click **Confirm**.

Now the resource is not referenced from any ruleset or library. Check the sections that are listed in Step 1.

4. Click **Remove** to remove the resource.

**Important:** If you do not remove the resource, references might not be updated when you redeploy the RuleApp. You might need to update them manually.

## Results

The resource is now removed cleanly.

You can remove multiple resources by using the **Remove All References** and **Remove** buttons respectively in Resources View.

**Parent topic:** [Removing referenced Java XOM resources and libraries](#)

### Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

### Related tasks:

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM libraries](#)

# Removing referenced Java XOM libraries

Before you remove a Java™ XOM library that is referenced from rulesets or other libraries, you must first remove all links that point to the library. When the links pointing to the library are removed, links that are pointed from the library are removed at the same time.

## About this task

You want to remove a Java XOM library at one point, but it might be referenced from rulesets or other libraries. Before you remove a library, the following links must be removed:

- Links pointing from rulesets to the library that you want to remove
- Links pointing from other libraries to the library that you want to remove

The library that you want to remove might be referencing other libraries or resources as well. In the Rule Execution Server console, the preceding links as well as the following links can be removed in one go:

- Links pointing from the library that you want to remove to Java XOM resources
- Links pointing from the library that you want to remove to other libraries

Therefore, the library can be removed cleanly without any link attached to it.

## Procedure

1. In the **Explorer** tab, open Library View of the Java XOM library that you want to remove.

See the following sections to check whether the library is referenced from other artifacts, or if it references other artifacts:

- **Show Referenced Internal Resources**
- **Show Library References**
- **Show References from Rulesets**
- **Show References from Libraries**

2. Click **Remove All References**.

A warning message is displayed.

**Remember:** Only the links to the library are deleted. The artifacts that reference the library, such as rulesets and libraries, are not deleted. The artifacts that are referenced from the library are not deleted, either.

3. Click **Confirm**.

Now the library is not referenced from any ruleset or library, and it does not reference any resource or library. Check the sections that are listed in Step 1.

4. Click **Remove** to remove the library.

**Important:** If you do not remove the library, references might not be updated when you redeploy the RuleApp. You might need to update them manually.

## Results

The library is now removed cleanly.

You can remove multiple libraries by using the **Remove All References** and **Remove** buttons respectively in Libraries View.

**Parent topic:** [Removing referenced Java XOM resources and libraries](#)

### Related concepts:

[Embedded managed Java XOM in Rule Execution Server](#)

### Related tasks:

[Deploying RuleApp archives](#)

[Removing referenced Java XOM resources and libraries](#)

[Removing referenced Java XOM resources](#)

## Monitoring ruleset execution

Using the monitoring capabilities of the Rule Execution Server console, you can enable the debug mode, enable ruleset monitoring, generate ruleset execution statistics, test ruleset execution, and view execution-related events in the XU log.

### [Enabling and disabling the debug mode](#)

You can enable or disable the debug mode on a ruleset. When this mode is enabled, you can use Rule Designer to put breakpoints in rulesets by using the remote debugging feature.

### [Generating ruleset statistics](#)

You can generate execution statistics for a particular ruleset from the Ruleset View.

### [Ruleset statistics views](#)

Ruleset execution statistics are provided for each execution unit (XU) across the entire cluster.

### [Viewing logged events on execution units \(XU\)](#)

You can view the execution events logged for each execution unit (XU) in the Rule Execution Server console. You can also modify how the events information is displayed.

**Parent topic:** [Rule Execution Server console online help](#)

#### **Related tasks:**

[Monitoring transparent decision services](#)

## Enabling and disabling the debug mode

You can enable or disable the debug mode on a ruleset. When this mode is enabled, you can use Rule Designer to put breakpoints in rulesets by using the remote debugging feature.

### About this task

The debug URL points to the remote server where Rule Designer runs. This URL consists of the server name and the chosen debugger port.

### Procedure

To enable/disable the debug mode:

1. In the Navigator panel, click the relevant ruleset.
2. In the **Ruleset View**, click **Edit**.

The **Ruleset Properties** are now editable.

3. Select the check box next to the **Debug** property to enable the debug mode.
4. Click **Save** in the **Ruleset View**.

### Results

When you enable the debug mode for a specific ruleset, that ruleset, when called, opens to a remote debugger.

**Parent topic:** [Monitoring ruleset execution](#)

# Generating ruleset statistics

You can generate execution statistics for a particular ruleset from the Ruleset View.

## About this task

Ruleset statistics provide information on ruleset execution such as the number of times a ruleset was executed and how long the execution took. When the ruleset is in debug mode, statistics are only on debug executions. If you disable the debug mode, ruleset statistics are reset. Debug execution statistics are not mixed with regular execution statistics.

## Procedure

### To generate statistics on the previous executions of a ruleset:

1. In the Navigator panel, click the relevant ruleset.
2. In the Ruleset View, click **View Statistics**.

## Results

The Ruleset Statistics View is displayed.

**Parent topic:** [Monitoring ruleset execution](#)

### Related information:

[Ruleset statistics views](#)

# Ruleset statistics views

Ruleset execution statistics are provided for each execution unit (XU) across the entire cluster.

The Ruleset Statistics View displays a table of information about the execution of rulesets, both for each execution unit (XU) in the configuration and for the entire cluster.

The ruleset statistics table contains the following columns: Metric and Ruleset Execution.

The statistics table rows provide the following information. Durations are expressed in milliseconds.

Table 1. Ruleset statistics

Labels	Description
Count	The number of times the ruleset has been executed during this session.
Total time (ms)	The total time to execute the ruleset.
Average time (ms)	The average time to execute the ruleset. This figure is derived from the total execution time (Total time (ms)) and the number of executions (Count).
Max / Min time (ms)	The longest and shortest ruleset execution times
First / Last Execution Date	The dates and times of the first and last ruleset executions.
Last Execution Time (ms)	The time of the last ruleset execution.

Parent topic: [Monitoring ruleset execution](#)

## Viewing logged events on execution units (XU)

You can view the execution events logged for each execution unit (XU) in the Rule Execution Server console. You can also modify how the events information is displayed.

### Procedure

To view logged events on execution units (XU):

1. Click the **Explorer** tab.
2. In the Navigator panel, click the relevant ruleset.
3. In the Ruleset View page, click **View Execution Units**.

The Execution Units page is displayed. The execution unit (XU) table provides the number of warnings and errors logged on each server.

4. Click the name of the server in the list of deployed execution units.

### Results

You can display events information in different ways:

- To limit the displayed log, use the **Display by: 5, 10, 50, 100** filter.
- To sort the log messages by column in ascending order, click the column name: **Level, Creation Date, Message**. To sort in descending order, click the column name again.
- To update the model and the log, click **Refresh** at the top of the page.

**Parent topic:** [Monitoring ruleset execution](#)



# Viewing or downloading an HTDS description file

You can view or download the description file of a hosted transparent decision service to Web Service Description Language (WSDL), Web Application Description Language (WADL), or OpenAPI format.

## Before you begin

### Restriction:

- Generation of OpenAPI description files is supported for only Java™ XOMs, while WSDL and WADL file generation are supported for XML XOMs and Java XOMs with JAXB annotations.
- You cannot use the **Retrieve HTDS Description File** feature for rulesets that are built with an incompatible engine version. In case of incompatibility, the Ruleset View provides more information.

## About this task

When you present a ruleset as a SOAP hosted transparent decision service, the description file is generated in WSDL. When you create a transparent decision service to execute a ruleset through the REST API, the description file is generated in WADL or OpenAPI.

## WSDL

To invoke a hosted transparent decision service, you can generate a WSDL file from a ruleset from the Rule Execution Server console. You can download the generated WSDL for the following purposes:

- Import the transparent decision service into any product or application that supports WSDL code, such as Rule Designer or IBM® Integration Designer with XSD files.
- Host the WSDL on another computer.

## WADL

When you want to execute a ruleset by using the REST API, you can generate a description file in .wadl format for the ruleset from the Rule Execution Server console.

## OpenAPI

Another way to execute a ruleset by using the REST API is to generate an OpenAPI definition file in .yaml or .json format for the ruleset from the Rule Execution Server console, and execute it through an OpenAPI tool or framework.

When you download a WSDL or WADL file, you can select the ruleset path, add a decision trace request, and download the XSD files separately.

When you download an OpenAPI file, you can select the ruleset path and add a decision trace request.

## Procedure

1. In the Rule Execution Server console, click the **Explorer** tab, expand the RuleApp you want, and select a ruleset to open the Ruleset View.
2. At the bottom of the Ruleset View, click **Show HTDS Options**.
3. Optional: If applicable, edit the transparent decision service target name space as explained in [Changing target namespaces](#).
4. Click **Retrieve HTDS Description File**. The ruleset path is displayed above the options. For example: /execution/1.0/PreTradeChecks/1.0
  - To generate a WSDL file, keep the **SOAP** option selected. If you select no options for WSDL generation, the WSDL code is generated as a single .wsdl file from the ruleset that you selected in the RuleApp View, with dynamic namespaces and decision identifiers, and with no decision trace filters.
  - To generate a WADL file, select the **REST** option, and then choose **WADL** from the **Format** pull-down menu.
  - To generate an OpenAPI file, select the **REST** option, , and then choose **OpenAPI - YAML** or **OpenAPI - JSON** from the **Format** pull-down menu.
5. Optional: For either format, select the appropriate options.

### Latest ruleset version

Select this check box if you want to generate the WSDL, WADL, or OpenAPI file from the latest ruleset of the RuleApp displayed above the options.

### Latest RuleApp version

Select this check box if you want to generate the WSDL, WADL, or OpenAPI file from the latest ruleset in the latest version of the RuleApp.

### Decision trace information

Select this check box if you want the WSDL, WADL, or OpenAPI file to include decision trace filter definitions and trace.

In the WSDL and WADL code, descriptions of the decision trace filters are documented in `<xsd:documentation>` elements.

For information about the decision trace filters, see [HTDS decision trace filters](#)

**Important:**

The order between filters is meaningful: when a filter conflicts with another filter, the filter in the higher position is taken into account.

**Inline types in separate XSD files**

Select this check box if you want to download the WSDL or WADL code as a .zip archive in which the .xsd files are saved separately from the .wsdl or .wadl file. This option is useful if you import more than one decision from Decision Server into an IBM Integration Designer project, or if you turn a rule flow into a process. The separation of the object types from the WSDL avoids the duplication of these objects in IBM Integration Designer.

**Proxy for API Connect**

Select this check box if you want to use the OpenAPI file in API Connect.

6. Click **View** or **Download**.

**Results**

The WSDL, WADL, or OpenAPI file is displayed in a separate browser tab or window and saved to the disk to your default download directory. If you select **Inline types as separate XSDs**, the **View** button is not available and you can only download the generated files.

**[HTDS decision trace filters](#)**

You can include decision trace filter definitions when you view or download the WSDL, WADL, or OpenAPI description file of a hosted transparent decision service in the Rule Execution Server console.

**Parent topic:** [Rule Execution Server console online help](#)

**Related tasks:**

[Changing target namespaces](#)

[Configuring a web service endpoint](#)

[Configuring the hosted transparent decision service location](#)

# HTDS decision trace filters

You can include decision trace filter definitions when you view or download the WSDL, WADL, or OpenAPI description file of a hosted transparent decision service in the Rule Execution Server console.

**Important:**  
  
The order between filters is meaningful: when a filter conflicts with another filter, the filter in the higher position is taken into account.

The following decision trace filter definitions are included when you select the **Decision trace information** option for viewing or downloading the WSDL or WADL description file for a selected ruleset in the Rule Execution Server console:

Table 1. Decision trace filter definitions that are included in the WSDL or WADL description file

Filter	Type	Default value	Description
all	boolean	false	Records everything if it is set to true.
executionDuration	boolean	false	Records the duration of the execution.
executionDate	boolean	false	Records the date of the execution.
rulestProperties	boolean	false	Records static data about the ruleset.
systemProperties	boolean	false	Records static data about the execution server.
inetAddress	boolean	false	Records static data about the internet address of the server.
totalRulesFired	boolean	false	Records the number of rules executed.
totalRulesNotFired	boolean	false	Records the number of rules that have not been executed.
rules	boolean	false	Records the complete list of rules in the executed ruleset.
rulesFired	boolean	false	Records any rule that has been executed at least once.
rulesNotFired	boolean	false	Records the rules that have not been executed.
totalTasksExecuted	boolean	false	Records the number of tasks executed.
totalTasksNotExecuted	boolean	false	Records the number of tasks that have not been executed.
tasks	boolean	false	Records the complete list of tasks in the executed ruleset.
tasksExecuted	boolean	false	Records any task that has been executed at least once.
tasksNotExecuted	boolean	false	Records the tasks that have not been executed.
outputString	boolean	false	Records the

			stream output or the engine execution.
<b>inputParameters</b>	boolean	false	Records ruleset input parameters.
<b>outputParameters</b>	boolean	false	Records ruleset output parameters.
<b>workingMemory</b>	boolean	false	Records the state of the working memory.
<b>workingMemoryFilter</b>	string		Holds the working memory filter.
<b>executionEvents</b>	boolean	false	Records the events that have been executed. This filter is a shortcut to select the <b>tasksExecuted</b> and <b>rulesFired</b> filters.
<b>boundObjects</b>	boolean	false	Records the list of matched objects.
<b>boundObjectsSerializationType</b>	string  enum: <ul style="list-style-type: none"><li>• JAXB</li><li>• ToString</li><li>• ClassNa me</li></ul>	ClassName	Choose the type of serialization for bound objects.

**Parent topic:** [Viewing or downloading an HTDS description file](#)

## Testing decision services

You can check the results and performance of a decision service by testing it before you put it into production.

As you develop a decision service, you can run it through tests and simulations to ensure that its rules produce the right results. In creating the decision service in Rule Designer, you can run its rules in Rule Execution Server. After publishing the decision service to Decision Center, you can run tests and simulations to validate its rules.

### Testing decision services in the development environment

You can test a decision service by deploying its rules from Rule Designer to the development Rule Execution Server, and then running them with a test application.

### Testing and simulation in Decision Center

Decision Center includes testing and simulation features to verify rules and evaluate their results.

### Working with Excel scenario files

You store testing and simulation scenarios in Excel spreadsheets. Start by making sure the business object model (BOM) is configured correctly. Then, you can generate a scenario file and populate it with data.

### Testing rule execution

You can test the execution of a deployed ruleset through a SOAP or REST web service.

## Testing decision services in the development environment

You can test a decision service by deploying its rules from Rule Designer to the development Rule Execution Server, and then running them with a test application.

### Procedure

1. In Rule Designer, right-click your decision service.
2. Click **Rule Execution Server > Deploy**.
3. In the RuleApp Deployment dialog, click the development deployment configuration, and then click **Next**.
4. Verify the deployment configuration settings, and then click **Next**.
5. Verify your login credentials.
6. Click **Connect** if you are not yet authenticated, and then click **Next**.
7. Verify the RuleApp and ruleset versions, and then click **Finish**.

After deployment, Rule Designer displays a report.

<b>Note:</b> Rule Designer can only deploy to the development environment. It cannot deploy to the test or production environment.
------------------------------------------------------------------------------------------------------------------------------------

8. Use a test application with Rule Execution Server to call the ruleset deployed from your decision service.
9. Check the results to determine whether the rules run as expected.

**Parent topic:** [Testing decision services](#)

## Testing and simulation in Decision Center

Decision Center includes testing and simulation features to verify rules and evaluate their results.

You can improve the performance of a business application by validating its rules before deployment. Decision Center offers you two ways to check rule behavior:

- Testing: You run a set of rules on scenarios that are defined by you to compare the actual results with your expected results.
- Simulation: You run rules on real or fictitious operational data, and use the results to assess and refine the behavior of the rules. You apply key performance indicators (KPIs) in analyzing the rules.

You do rule testing and simulation in the Decision Center Business console. You create scenario files in Excel, and for simulations, you define KPIs, report formats, and simulation configurations.

<b>Note:</b> You can also run tests with the <a href="#">Decision Center REST API</a> .
-----------------------------------------------------------------------------------------

### Related information:

[Testing sets of rules in the Business console](#)

[Simulating business application results](#)

## Testing rulesets in Rule Designer

To validate rulesets against scenarios, you set up and run a testing configuration.

### About this task

You can generate a template of the scenarios file where you enter the values that you want to test. Then, you set up the testing configuration by specifying a decision operation and the Excel file that contains the scenarios.

### Procedure

1. If you don't already have one, create a scenario file. To generate a template scenario file, see [Generating an Excel scenario file](#).
2. Optional: You can add breakpoints in various artifacts of the project to pause the execution of the test in strategic places for debugging purposes. To learn more about breakpoints, see [Breakpoints](#).
3. Right-click the project that you want to test and select **Debug as > Debug configurations....**
4. In the list of configurations, right-click **Testing decision operation** and click **New**.
5. Give this configuration a name and specify the Excel file that contains the scenarios and the decision operation to use.
6. Click **Debug**.

### Results

The results of the execution display in the Console view: you see whether each scenario was successful, the time it took to execute it, and the number of executed rules and tasks.

The results for the scenarios use the following statuses:

- **Successful:** A test is successful when the expected results match the actual results.
- **Unsuccessful:** A test is unsuccessful when the expected results are different from the actual results.
- **Error:** An error is reported when the test cannot run the scenarios, for example, when an entry in the scenario file is not correctly formatted.

If there are breakpoints in the rules of the project, the execution stops when it encounters one and the Debug view opens. You can then see which breakpoints stopped the execution and you can continue the resume the execution until the next breakpoint.

### Example

Here is an example of the results of a test with three scenarios:

```

Scenario Number 0 : 'Rejected loan' = Success
Time = 16, Nb Executed Rules = 1, Nb Executed Tasks = 3

Scenario Number 1 : 'Big amount loan' = Failure
Expected result 'the loan is approved equals ': Failed
Time = 0, Nb Executed Rules = 1, Nb Executed Tasks = 2

Scenario Number 2 : 'Approved loan' = Success
Time = 0, Nb Executed Rules = 0, Nb Executed Tasks = 3

Nb Executed 3
Nb Failures 1
Nb Errors 0
End running the JVM for testing
```

The Rejected loan scenario and the Approved loan scenario were successful, meaning that the expected values defined in the scenarios match the actual results of the tests. The Big amount loan scenario failed because the rule that approves the loan did not produce the expected result.



## Working with Excel scenario files

You store testing and simulation scenarios in Excel. Before you generate an Excel scenario file, make sure that the business object model (BOM) is configured correctly. You can then generate the file and populate it with scenario data.

### [Excel scenario files](#)

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

### [Relationship between the BOM and the Excel file](#)

You use Excel to format scenarios for tests and simulations. Some columns in the scenario file use the information contained in the BOM of the rule project.

### [Adding columns to the Excel file](#)

You configure the BOM to add columns to the **Scenarios** and **Expected Results** sheets.

### [Removing columns from the Excel file](#)

You configure the BOM to remove columns from the **Scenarios** and **Expected Results** sheets.

### [Validating the project](#)

You verify that a rule project or decision operation is suitable for generating Excel scenario files.

# Excel scenario files

After you generate an Excel scenario file, you must add the scenario data that is used to validate your rules in tests or simulations.

An Excel scenario file can contain the following sheets:

- Scenarios: Contains the input data for scenarios. Both testing and simulation scenario files have this sheet.
- Data entry: Regroups information that is used in other sheets.
- Expected Results: Holds the results that you expect from tests.
- Expected Execution Details: Holds the execution details that you expect from tests.

**Important:**

Never modify the column structure of the sheets.

## Scenarios sheet

Each row in the Scenarios sheet represents one scenario. Each scenario must contain all the information needed to process a transaction.

		the borrower					the loan		
Scenario ID	description	first name	last name	b	S	c	y	z	
Very low risk	Very low risk loan accepted	Sam	Adams	4			95	8/1/2009	# 100000
Low risk	Low risk loan accepted	Bob	Schwartz	4			94	8/7/2010	# 500000
Average risk	Average risk loan accepted	John	Johnson	4			85	9/1/2010	# 900000
Amount too high	Loan rejected when amount too high	Mary	Doe	4			32	8/15/2010	# 1100000

The columns indicate the information that you must provide for each scenario:

- **Scenario ID:** The name identifies the scenario and must be unique.
- **description:** A free text cell to describe the scenario.
- **business terms:** Values for the scenarios. Bold columns or subcolumns are mandatory.

**Note:**

The reduced rows between the column headers and the first scenario contain information that might be useful to developers. Never edit these rows.

The following visual aids help you complete your scenarios:

- Red triangle: When shown in a column header, it indicates the presence of a comment that explains the type of values to be entered, such as text, numbers, dates, or true or false. Hover over a triangle to display a comment as follows:

the borrower				
first name	last name	credit score	yearly income	
John	Smith	600	80000	
John	Smith	600	80000	

- Dark green triangle: When shown in a cell, it suggests a better format for the cell. For example, if you enter a numerical value in a text cell, Excel suggests that you format the cell for numerical values. To make sure that the scenario runs correctly, ignore this suggestion.
- Arrow: When shown before a column name, it indicates that the values must correspond to entries that are specified in a separate **data entry** sheet.

## Data entry sheets

You create data entry sheets when you generate a scenario file. Data entry sheets regroup data to be used in other sheets.

The name of each data sheet serves as the type of data that is expected in the column of the other sheet that uses the data. For example, in the following figure, the address sheet **1** contains different addresses that are used in the addresses column **2** of the Scenarios sheet.

Entry ID	Street	City	State	Zip Code
billing address	2207 7th avenue	New York	NY	10027
shipping address	25 Elm Street	Dallas	TX	75201
personal address	110 Cactus Street	Phoenix	AZ	85003

1

	the borrower						th
description	first name	last name	credit score	yearly income	amount	→ addresses	st
Loan rejecte	John	Smith	600	80000	500000	personal address	
Loan approve	John	Smith	600	80000	25000	billing address	

Enter the name of an object defined in the sheet address

### Note:

A column that uses values from a data entry sheet has an arrow in its column header.

You create an entry in a data entry sheet by completing a row that includes a unique name to identify the entry in the other sheets.

## Expected Results sheet

The Expected Results sheet contains the results that you expect to obtain when you run tests that use the scenarios.

You can test many aspects of a set of rules, but the Expected Results sheet contains only the tests that you specified when you generated the scenario file. For example:

Scenario ID	the loan report is approved equals	the messages of the loan report contains
Very low risk	TRUE	Very low risk loan
Low risk	TRUE	Low risk loan
Average risk	TRUE	Average risk loan
Amount too high	FALSE	The loan cannot exceed 1000000

Each blue column in the Expected Results sheet corresponds to a test. If you do not enter a value in a column, the corresponding test is skipped.

### Tip:

You cannot create a new column in Excel. However, you can generate another empty scenario file template, and then copy and paste a column from it.

You link the sheets that contain the scenarios and their expected results by entering names in their respective Scenario ID columns. In addition to making sure that the names match between the two sheets, it is good practice to keep the scenarios and their expected results in the same order.

When you test for a list of values, you enter each item in the corresponding cell on a separate row. You do not have to duplicate the Scenario ID for each row.

## Expected Execution Details sheet

The Expected Execution Details sheet contains the execution details that you expect to obtain when you run tests that use the scenarios.

The following are examples of execution details:

- List of rules fired
- List of executed ruleflow tasks
- Duration of execution

You can test many aspects of a run, but the Expected Execution Details sheet contains only the tests that you specified when you generated the scenario file. Each green column in the Expected Execution Details sheet

corresponds to a test.

You link the sheets that contain the scenarios and expected execution details by entering names in their respective Scenario ID columns.

When you test for a list of values, you enter each item in the corresponding cell on a separate row.

4			
5		Scenario ID	the list of fired rules contains
9		Big Loan	eligibility.checkIncome
10			eligibility.approval
11			eligibility.checkCreditScore
12		Small Loan	
13			
▶▶ Scenarios Expected Execution Details HELP			

You do not have to duplicate the Scenario ID for each row because the last ID entered is used.

**Parent topic:** [Working with Excel scenario files](#)

**Related concepts:**  
[Relationship between the BOM and the Excel file](#)



## Relationship between the BOM and the Excel file

You use Excel to format scenarios for tests and simulations. Some columns in the scenario file use the information contained in the BOM of the rule project.

When you use Excel as the format for your scenarios, columns in the **Scenarios** and **Expected Results** sheets of the Excel scenario file are generated using information contained in the BOM of the rule project.

This section explains how information retrieved from the BOM affects the sheets of the Excel scenarios file.

### Scenarios sheet

The columns in the **Scenarios** sheet are defined by the BOM classes that define instances of input parameters required to execute the rulesets.

The **Scenarios** sheet is composed of:

#### Required columns

These provide the data that is required to execute your rulesets. The name of a required column displays in bold in the Excel sheet.

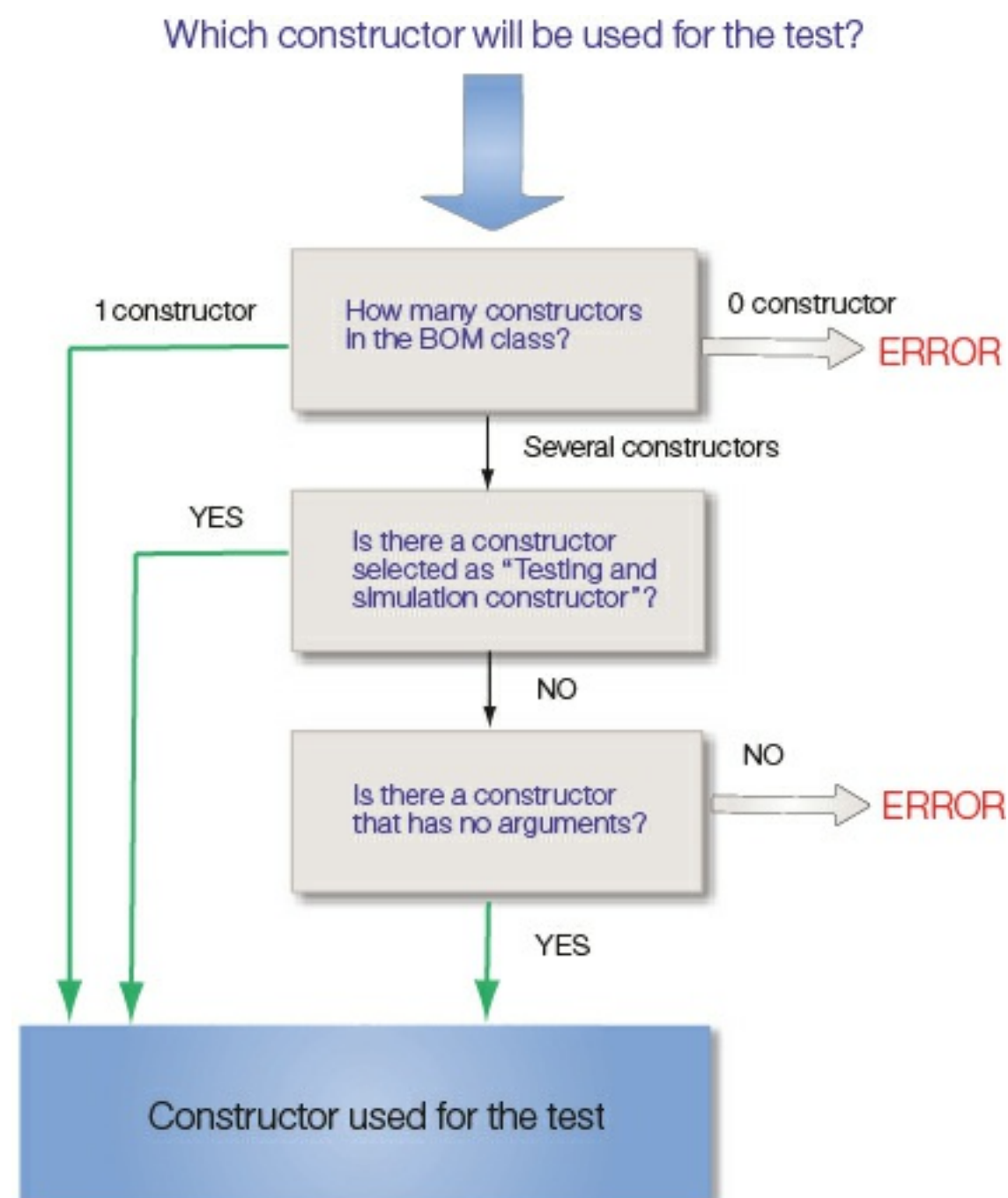
#### Optional columns

These provide additional information on the scenarios. The attributes of the BOM class define the optional column headings. Each optional column corresponds to an attribute. If the attribute is verbalized, the name in the column heading is the verbalization of the attribute.

**Note:** You can use only attributes that are non-static, and writable ("read/write" and "write only") to create optional columns.

A class or an interface in the BOM requires a constructor for it be useful in an Excel scenario file. A constructor of the BOM class defines the required columns. Choose the constructor that you want to use for the Excel scenario file.

A class sometimes has a default constructor that does not have any arguments. When there are several constructors in a class, the class uses the constructor with no arguments by default. If a class has several constructors, you must specify the one you want to use in your tests by selecting the **Testing and simulation constructor** option in the **General Information** section of the BOM editor.



**Note:** If the constructor has no arguments, a required column does not show in the Excel scenario file.

The name of the arguments in the constructor define the required column headings.

By default, when you generate a BOM from a XOM, the constructor arguments have generic names such as arg1, arg2, and arg3, as shown in the following figure:

Arguments

Edit the arguments of this member.

Name	Type
arg1	java.lang.String
arg2	java.lang.String
arg3	java.util.Date
arg4	java.lang.String
arg5	int

To obtain meaningful column headings, you must rename these arguments. Make sure that the name of the constructor arguments are meaningful to a business user.

You must rename each argument that references an attribute with the name of the attribute. For example, if arg1 sets the value of the attribute firstName, you must rename arg1 to firstName. You must rename other arguments with a name that is meaningful to a business user. The following figure shows an example of arguments that have been renamed:

Arguments

Edit the arguments of this member.

Name	Type
firstName	java.lang.String
lastName	java.lang.String
birthDate	java.util.Date
SSN	java.lang.String
creditScore	int

**Note:** You can also use annotations in your XOM to give a business name to the arguments, see [@BusinessName](#).

The name displayed in the column heading is the verbalized name of the attribute or constructor argument:

- If the attribute has an explicit verbalization, the verbalization is used as column heading.
- If the attribute does not have an explicit verbalization, or if the argument is an implementation parameter that does not correspond to an attribute, the column heading uses a default verbalization. For example, if the argument name is dateOfBirth, the default verbalization is date of birth.

Expected Results sheet

The ruleset output parameters and the BOM classes define the columns in the **Expected Results** sheet of the Excel scenario file.

The attributes of the BOM class define the column headings. Each column corresponds to an attribute.

You must verbalize each attribute because the name in the column heading is the verbalization of the attribute. If the attribute is a collection, you must define the domain.

**Note:** You can use only attributes that are non-static and readable ("read/write" and "read only") to create columns.

**Parent topic:** [Working with Excel scenario files](#)

**Related concepts:**  
[Excel scenario files](#)

# Adding columns to the Excel file

You configure the BOM to add columns to the **Scenarios** and **Expected Results** sheets.

## About this task

The columns contained in the Excel scenario file are generated using information from the BOM. The input parameters, the attributes, and the constructor arguments are used to create the input columns in the Excel scenario file. You can also configure the BOM to add columns in the **Scenarios** and **Expected Results** sheets.

You can create attributes in the BOM to add new columns in the **Scenarios** sheet or in the **Expected Results** sheet.

## Procedure

To add columns to the **Scenarios** and **Expected Results** sheets:

1. Create virtual attributes:
  - Writable attributes (“Read/Write” or “Write Only”) for optional columns in the Scenarios sheet
  - Readable attributes (“Read/Write” or “Read Only”) for columns in the Expected Results sheet
2. Define the BOM to XOM mapping for these virtual attributes:
  - Edit the setter part for a new column in the Scenarios sheet
  - Edit the getter part for a new column in the Expected Results sheet
3. Select the new available **Expected Results** options when you generate the Excel scenario file.

## Results

To add **required columns**, you must modify the list of arguments of the testing and simulation constructor, or choose a different testing and simulation constructor.

**Parent topic:** [Working with Excel scenario files](#)

## Related concepts:

[Relationship between the BOM and the Excel file](#)

## Related tasks:

[Removing columns from the Excel file](#)

[Generating an Excel scenario file](#)

# Removing columns from the Excel file

You configure the BOM to remove columns from the **Scenarios** and **Expected Results** sheets.

## About this task

The Excel scenario file uses information from the BOM to generate columns. The input parameters, the attributes, and the constructor arguments are used to create the input columns in the Excel scenario file. You can also configure the BOM to remove columns from the **Scenarios** sheet.

By default, there is a column for each constructor argument and for each attribute (providing it is non-static and writable). If you do not want to display all the attributes as columns in the **Scenarios** sheet, you can exclude them.

## Procedure

To remove optional columns from the **Scenarios** sheet:

1. In the BOM editor, double-click the attribute that you want to exclude from the Excel scenario file.
2. In the **General Information** area, select the **Ignore for testing and simulation** option.

## Results

This option ensures that there is no column in the Excel scenario file for the attribute you have excluded.

To remove **required columns**, you can either modify the list of arguments of the testing and simulation constructor, or choose a different testing and simulation constructor (see [Relationship between the BOM and the Excel file](#)).

**Parent topic:** [Working with Excel scenario files](#)

### Related concepts:

[Excel scenario files](#)

[Relationship between the BOM and the Excel file](#)

### Related tasks:

[Adding columns to the Excel file](#)



# Validating the project

You verify that a rule project or decision operation is suitable for generating Excel scenario files.

## Before you begin

**Important:**  
Before validating a project, make sure that you select the correct constructor in the BOM. For more information about the constructors and how to use them, see [Relationship between the BOM and the Excel file](#).

## About this task

The default format for entering scenarios is Excel. When you use the default Excel format for your scenarios, you must validate that projects on which you want to run tests are suitable for generating the correct columns in the Excel scenario file.

You validate a project in the Testing and Simulation Project Validation view in Rule Designer. This view raises errors and warnings if the BOM or input parameters require modification. You must fix any errors in the view before you generate an Excel scenario file.

## Procedure

To validate the project:

1. In the Rule Explorer, right-click the project on which you want to enable testing, and then click **Testing and simulation > Check project**.

The Testing and Simulation Project Validation view opens and lists the errors and warnings.

2. To view the solution description, select the warning or error.

The Solution Description section displays a description of the warning or error and indicates what you can do to solve the problem.

3. Double-click the warnings or errors to open the class or member to fix in the BOM editor.
4. In the Testing and Simulation Project Validation view, click the **Refresh View** button to view any remaining errors or warnings.

## Results

If there are no remaining errors, you can generate the Excel scenario file in the Decision Center Business console.

**Important:**  
Make sure that you publish to Decision Center the project containing any BOM changes that you made during project validation.

**Parent topic:** [Working with Excel scenario files](#)

**Related concepts:**  
[Excel scenario files](#)

## Testing rule execution

You can test the execution of a deployed ruleset from a client application through a SOAP web service or a REST web service.

### [Executing rules by using the REST service](#)

Decision Server provides a Representational State Transfer (REST) service for ruleset execution. You can use it to execute rulesets through the HTTP protocol by using the XML or the JSON format.

### [Executing rules by using the SOAP service](#)

Through a Simple Object Access Protocol (SOAP) web service, Rule Execution Server automatically presents as a web service any deployed ruleset that uses an XML schema or a Java™ XOM.

## Executing rules by using the REST service

Decision Server provides a Representational State Transfer (REST) service for ruleset execution. You can use it to execute rulesets through the HTTP protocol by using the XML or the JSON format.

### Benefits

The REST service for ruleset execution provides XML and JSON generation, XSD validation, and execution services. The service is designed for the following benefits:

- You do not need a client library or a complex configuration to interact with a remote Rule Execution Server instance.
- You can work across environments or from various client applications, typically from JavaScript clients.
- You can easily move from local to remote Rule Execution Server execution.

### Workflow

To use the REST service for a ruleset execution, use the following process. See [Executing a ruleset from a client](#).

1. Deploy a valid ruleset.
2. Use the REST service to generate an XML or a JSON payload. The REST service provides XML and JSON generation, XSD validation, and execution services. You can test the generation of the payload and its execution from the Rule Execution Server console. See [Testing ruleset REST execution](#).
3. After you are familiar with the input and output format, send the request as the payload of an HTTP call through a POST method to the corresponding URI. See [Executing a ruleset by creating a REST request](#).

#### [REST resources](#)

Resources for the Decision Server REST service for execution are rulesets.

#### [Endpoint URIs](#)

Endpoint URIs represent rulesets as Rule Execution Server resources for the REST execution service.

#### [HTTP methods](#)

The HTTP methods of the REST API provide the operations that are available on Rule Execution Server artifacts.

#### [HTTP headers and URI parameters](#)

The URIs for Rule Execution Server REST resources support some HTTP header fields and generic URI parameters.

#### [Content types](#)

The Content-Type field in HTTP headers indicates in which format the data is sent to, or returned by, the HTTP methods of the Rule Execution Server REST service.

#### [Request and response schema](#)

In the REST service for ruleset execution, requests and responses follow specific schemas, depending on whether the ruleset XOM is based on XML classes or on Java™ classes. The schema determines how the types are serialized.

#### [XML serialization of ruleset XOMs](#)

In the REST service for ruleset execution, primitive Java types, XSD types, and Java XOM classes are serialized differently.

#### [JSON serialization of ruleset XOMs](#)

Ruleset parameters of primitive Java types, arrays, and Java XOM classes can be serialized to JSON through a Jackson process.

#### [Executing a ruleset by creating a REST request](#)

You can execute a valid ruleset by creating an XML or a JSON request and using the POST method.

#### [Generating a WADL representation](#)

You can generate a WADL representation of request and response elements, and their documentation.

#### [Generating an OpenAPI representation](#)

OpenAPI is a standardized version of the Swagger specification. It provides a language-independent way to present REST APIs. You can generate OpenAPI representations for executing decision services. This capability is part of ODM Decision Connect which enables reusable decisions to be exposed as OpenAPI based public APIs to be invoked directly by applications. The OpenAPI based decisions can also be published to an IBM API Connect catalog for managed APIs. IBM API Connect Essentials is also an optional part of Decision Connect.

**Parent topic:** [Testing rule execution](#)

**Related concepts:**

[Endpoint URIs](#)

[Executing rulesets in the decision engine](#)

[Hosted transparent decision services](#)

**Related tasks:**

[Calling a decision service by using the REST API](#)  
[Calling a decision service by using OpenAPI](#)  
[Calling a decision service from API Connect](#)

## REST resources

Resources for the Decision Server REST service for execution are rulesets.

Developers can use the REST service to execute rulesets. As in hosted transparent decision services, rulesets are identified by their signature. Short ruleset paths and canonical ruleset paths are both supported.

The primary artifact for a REST service is the resource. Each resource is represented as a unique resource identifier (URI), and every resource has an HTTP method that clients use to request the execution of a ruleset and receive the response. WADL and OpenAPI formats can be used to describe the signature of the REST service that can be called to execute the ruleset.

Examples of ruleset paths:

- Short ruleset path with no version numbers: `miniloanruleapp/miniloanrules/`
- Canonical ruleset path: `miniloanruleapp/1.0/miniloanrules/1.0/`

**Parent topic:** [Executing rules by using the REST service](#)

# Endpoint URIs

Endpoint URIs represent rulesets as Rule Execution Server resources for the REST execution service.

URIs for REST endpoints have the following format:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetpath}/{filetype}?{options}
```

URIs are defined as follows:

Table 1. Items that make up an URI

Item	Description
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService	<p>The endpoint of the hosted transparent decision service (HTDS) application.</p> <p>The value of &lt;odm_on_cloud_environment&gt; must be either dev, test, or prod.</p> <p>Example: https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService .</p>
/rest	The REST service context root.
/v1	The version number of the REST service. This parameter is optional. The current version is numbered v1. If subsequent versions are released, the version number is incremented but the changes added by each version remains compatible with any code that implements the current /v1 version.
{rulesetpath}	<p>The short ruleset path or the canonical ruleset path.</p> <ul style="list-style-type: none"><li>• A canonical ruleset path indicates the RuleApp name and version number and the ruleset name and version number. For example: myRuleApp/1.0/myRuleset/1.0</li><li>• A short ruleset path leaves out one or both version numbers: For example: myRuleApp/myRuleset</li></ul>
{filetype}	The file type: wadl or openapi.
{options}	You can add options at the end of the URI. See Table 2 for details.

Table 2. Options

Option	Used for	Description
format	OpenAPI	<p>Select YAML or JSON. If no format is specified, YAML is used by default.</p> <p>Example: format=JSON</p>
schemes	OpenAPI	<p>Specify HTTP or HTTPS, separated by comma.</p> <p>Example: schemes=HTTP, schemes=HTTPS, or schemes=HTTP,HTTPS</p> <p>If this parameter is not specified, the same protocol as the request URL is used.</p>

extension	OpenAPI	Specify apiconnect to generate an OpenAPI file for IBM API Connect®.  Example: extension=apiconnect
trace	OpenAPI WADL	If it is set to true, the trace is enabled. The trace filter is added to the request and the corresponding trace is added to the response.  Example: trace=true
download	OpenAPI WADL	If it is set to true, the generated file is downloaded to the file system.  Example: download=true
zip	WADL	If it is set to true, the WADL code and its XSD files are in a compressed file.  When you specify the zip option, the inline option is not necessary.  Example: zip=true
inline	WADL	If it is set to true, the XSD files are added in the WADL code.  Example: inline=true

Example of adding an option:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?zip=true
```

Use & when you want to add more than one option:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetpath}/{filetype}?{option1}&{option2}&{option3}
```

**Parent topic:** [Executing rules by using the REST service](#)

## HTTP methods

The HTTP methods of the REST API provide the operations that are available on Rule Execution Server artifacts.

The REST API for ruleset execution supports the GET and POST methods.

- You use the GET method to generate a sample XML or JSON payload or to retrieve WADL or OpenAPI for a specified ruleset. For example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/xml
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/json
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/ruleapp/1.0/miniloanrules/1.0/wadl
```

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/ruleapp/1.0/miniloanrules/1.0/openapi
```

- You use the POST method to create a request for execution of a ruleset or to validate an XML payload. The request body contains the XML or JSON payload. For example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0
```

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniload/1.0/miniloan/1.0/validate
```

**Parent topic:** [Executing rules by using the REST service](#)



## HTTP headers and URI parameters

The URIs for Rule Execution Server REST resources support some HTTP header fields and generic URI parameters.

The REST execution service supports the **Accept-Language** field name in HTTP headers. All other fields are ignored.

### The Accept-Language header field

The Accept-Language field sends the list of languages that are valid for the response message. For example, Accept-Language: fr, pt-BR means that the preferred language is French but Brazilian Portuguese is also accepted. You can use the equivalent URI parameter instead of the HTTP header field.

### The Accept-Language parameter

The **accept-language** URI parameter is equivalent to the **Accept-Language** HTTP header field. You can use it in request messages to send the list of languages that are valid for the response message.

**Parent topic:** [Executing rules by using the REST service](#)

## Content types

The Content-Type field in HTTP headers indicates in which format the data is sent to, or returned by, the HTTP methods of the Rule Execution Server REST service.

The REST service for ruleset execution supports the `application/xml` and the `application/json` content types. XML is the default content type of the response. You can also find the object definitions by retrieving the Web Application Description Language (WADL).

For OpenAPI service definitions, only the `application/json` content type is supported. See the consumes and produces sections in the OpenAPI definition file.

**Parent topic:** [Executing rules by using the REST service](#)

## Request and response schema

In the REST service for ruleset execution, requests and responses follow specific schemas, depending on whether the ruleset XOM is based on XML classes or on Java™ classes. The schema determines how the types are serialized.

The request and response schema results from the signature of the target ruleset.

- The request part is composed of the following elements:
  - The **IN** and **INOUT** parameters of the ruleset, in alphabetical order.
  - Optionally, a decision ID if you want to set it to a specific value.
  - Optionally, a trace filter.
- The response part is composed of the following elements:
  - The **INOUT** and **OUT** parameters of the ruleset, in alphabetical order.
  - The decision ID: either the default identifier or the value that you set in the request.
  - The returned trace, depending on the filter that you set in the request.

The XML payload is analyzed against the generated XSD files. The execution response is sent in the same format as the execution request (XML or JSON).

### XML request validation

The validation response is returned in JSON format:

- If the request is valid, the response is an empty JSON list [ ].
- If the request is invalid, the tool returns the list of errors. Each error contains the following fields:
  - Type: the type of the error. Possible values are "Error", "Fatal" and "Warning".
  - Line: the number of the line that contains the error in the .xml file
  - Column: the number of the column that contains the error in the .xml file
  - Message: the error message itself

The JSON payload cannot be validated.

Here are examples of error messages:

```
{"type": "Error", "line": 8, "column": 32, "message": "cvc-datatype-valid.1.2.1: 'falseee' is not a valid value for 'boolean'."}
```

```
{"type": "Error", "line": 9, "column": 22, "message": "cvc-datatype-valid.1.2.1: '5d' is not a valid value for 'integer'."}
```

```
{"type": "Fatal", "line": 39, "column": 24, "message": "The element type\n\"par:longParam\" must be terminated by the matching end-tag\n\"</par:longParam>\"."}
```

**Parent topic:** [Executing rules by using the REST service](#)

#### Related concepts:

[XML serialization of Java types](#)

[XML serialization of ruleset XOMs](#)

#### Related tasks:

[Executing a ruleset by creating a REST request](#)

## XML serialization of ruleset XOMs

In the REST service for ruleset execution, primitive Java™ types, XSD types, and Java XOM classes are serialized differently.

Ruleset parameters are handled differently depending on whether they are expressed as XML elements or as Java types.

### Primitive types

Primitives types are Java types and classes that can be written as inline strings. The XML element that is used to serialize a primitive type as a ruleset parameter is the name of the ruleset parameter in the parameter namespace.

You can find more information here:

- [Changing target namespaces](#) explains how to customize the parameter namespace from the Ruleset View in the Rule Execution Server console.
- [XML serialization of Java types](#) shows the XML types and Java Architecture for XML Binding (JAXB) annotations for Java primitive types and simple types.

### Dynamic XOM

For dynamic XOMs, the root element depends on how the ruleset parameters are defined. The XML description is serialized from the original XSD code of the dynamic XOM on which the ruleset is based.

#### Name of the root element

For rulesets that are based on a dynamic XOM, the root element name is determined as follows:

- If the parameter is an XML element in the ruleset signature, that element is used as the root element name of the parameter in requests and responses.
- If the parameter is a complex or primitive Java type, the parameter name is used as the root element name of the parameter in requests and responses, within the configured param namespace.

**Note:** A ruleset parameter is declared from an XML element if it meets the following condition: that parameter uses an XML complex type for which an XML element was declared from that complex type in the original imported XSD file. For example, starting from the following XSD definition:

```
<xsd:element name="MyConference">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element name="session" type="session"
maxOccurs="unbounded"/>
 <xsd:element name="participant" type="participant"
minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

the ruleset parameter is declared from the MyConference XML element:

```
<par:Request
xmlns:par="http://www.ibm.com/rules/decisionsservice/myruleapp/myruleset
/param" xmlns:jav="http://www.acme.com/myconference">
 <jav:MyConference>
 <jav:session>
 [...]
 <jav:session>
 <jav:MyConference>
</par:Request>
```

### Serialization

The root element depends on your XML types.

- For XML complex types, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.
- For XML elements, the parameter declaration uses the defined XML element as the root element in the original namespace. In this case, the name of the parameter that is attached to each node is provided in the WADL code as a comment message.

When multiple elements have the same name, element nodes are differentiated alphabetically. In this case, if you want one of the elements to be null, you must set the **xsi:nil** attribute to true. You can do that only if

the **nillable** attribute is set to `true` for the root element in the original XSD schema of the dynamic XOM. The default value of the **nillable** attribute is `false`.

## Java XOM

For Java XOMs, the root element derives from the parameter declaration and XML serialization uses JAXB annotations. Arrays are supported for ruleset parameters.

### Name of the root element

For rulesets that are based on a Java XOM, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.

### Serialization

The XML structure of Java XOM classes is serialized through the standard JAXB process. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using JAXB annotations.

**Note:** A parameter is visible as an XML element only if you specify it with an `XmlElement` annotation or if valid `getXXX` and `setXXX` methods apply to the parameter.

### Arrays as ruleset parameters

In Java XOM, you can specify ruleset parameters as simple or multidimensional arrays. XML serialization is processed differently for each type of arrays:

- If the ruleset parameter is a simple array, the parameter name repeats as a standard array in XML.
- If the ruleset parameter is a multidimensional array, the parameter name repeats as a standard array type in XML for the first dimension and then uses `item` elements for the subsequent dimensions. For example:

```
<myMultiDimArrayParam>
 <item>1</item>
 <item>2</item>
</myMultiDimArrayParam>
<myMultiDimArrayParam>
 <item>3</item>
 <item>4</item>
 <item>5</item>
</myMultiDimArrayParam>
```

**Parent topic:** [Executing rules by using the REST service](#)

#### Related concepts:

[Request and response schema](#)

[XML serialization of Java types](#)

## JSON serialization of ruleset XOMs

Ruleset parameters of primitive Java™ types, arrays, and Java XOM classes can be serialized to JSON through a Jackson process.

Each ruleset parameter is written as a JSON name/value pair. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using Jackson or JAXB annotations. Jackson annotations are applied before JAXB annotations with matching precedence. Some JAXB annotations are not supported or do not have a JSON equivalent.

An empty constructor, either public or private, is needed to deserialize with Jackson.

### Jackson version

Jackson currently exists as two major versions, 1.x and 2.x. Packaging and API differences make the two releases incompatible. The REST service for ruleset execution uses Jackson 2.x, which means that you must use Jackson 2.x to annotate the Java XOM classes.

### Some known limitations

Static nested classes are supported. Nonstatic nested classes, also known as inner classes, are not supported.

Cyclic links between three or more classes are not supported.

The official Jackson documentation recommends not to use `java.sql.date`.

For more information, see the [Jackson JSON Processor Wiki](#).

If you use Java primitive types as input values in your execution requests, make sure that each value falls within its corresponding data type range. Otherwise, you might get unexpected output.

**Parent topic:** [Executing rules by using the REST service](#)

## Executing a ruleset by creating a REST request

You can execute a valid ruleset by creating an XML or a JSON request and using the POST method.

### About this task

To execute a deployed ruleset by using the REST service, you generate the XML or JSON payload and post the request. You can also generate a WADL representation of request and response elements. See [Generating a WADL representation](#).

In your client, you construct the request message as an XML or JSON packet, which depends on the XML signature in WADL format of the REST service. The client must specify the web service URL, pass in the request, and specify a variable to hold the response from the service.

### Procedure

1. Generate an XML or JSON fragment by posting your request to the following URI.

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/{format}
```

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/xml
```

Then, you can use the result as a starting point to write an XML or JSON request.

2. Validate the XML structure of the request by posting your request to the following URI.

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/validate
```

Example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/validate
```

**Tip:** You can test the generation of the XML code also from the Rule Execution Server console as explained in [Testing a ruleset for REST execution](#).

If the request is not valid, error messages are returned in JSON format. For more information, see [Request and response schema](#). Validation (/validate) is not available for the JSON format.

3. Post the execution request to the following URI:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}
```

REST requests for execution use the POST method. The request body contains the XML or JSON payload. Example:

```
POST
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0
```

The execution response is sent in the same format as the request.

**Parent topic:** [Executing rules by using the REST service](#)

**Related concepts:**  
[Request and response schema](#)

## Generating a WADL representation

You can generate a WADL representation of request and response elements, and their documentation.

### About this task

To write XML input for the POST method to a specific ruleset endpoint URI, you can generate the WADL representation. The WADL representation contains two entries for a POST method and one for a GET method. A set of XSD files are attached to this method to describe the XML representation of the input and output objects.

To generate a WADL representation, you can use the endpoint URIs. The WADL format is described in the [Web Application Description Language](#) page of the W3C website.

### Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/wadl
```

The URI variables are defined in [Endpoint URIs](#).

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl
```

2. Optional: To add the XSD files in the WADL code, use the **inline** query parameter. Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?inline
```

If you do not specify the **inline** parameter, the external .xsd files that are referenced by the .wadl file remain external. The .wadl file might not work correctly with some tools from independent software vendors. If you specify the **inline** parameter, the .xsd files are included in the .wadl file.

**Note:** In WebSphere® Application Server, you must explicitly set the **inline** value to true.

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?inline=true
```

3. Optional: To generate the WADL code and its XSD files to a compressed file, add the **zip** parameter.

If you specify the **zip** parameter, the **inline** parameter is not necessary.

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/wadl?zip=true
```

### Results

If you try to generate WADL representation for an invalid URL or if an error occurs during the WADL generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

**Note:** The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

**Parent topic:** [Executing rules by using the REST service](#)

**Related concepts:**



[Endpoint URIs](#)  
[Content types](#)

## Generating an OpenAPI representation

OpenAPI is a standardized version of the Swagger specification. It provides a language-independent way to present REST APIs. You can generate OpenAPI representations for executing decision services. This capability is part of ODM Decision Connect which enables reusable decisions to be exposed as OpenAPI based public APIs to be invoked directly by applications. The OpenAPI based decisions can also be published to an IBM API Connect catalog for managed APIs. IBM API Connect Essentials is also an optional part of Decision Connect.

### About this task

To write JSON input for the POST method to a specific ruleset endpoint URI, you can generate the OpenAPI representation. The OpenAPI representation contains one entry for the POST method.

To generate an OpenAPI representation, you can use the [Endpoint URIs](#). For more information about OpenAPI, see [The OpenAPI Specification](#) page of the Open API Initiative website.

**Note:** Version 2.0 of the OpenAPI specification is supported.

### Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/{rulesetPath}/openapi?format={format}
```

Set *{format}* to the format of the OpenAPI definition file, which is either YAML or JSON. The default format is YAML.

The URI variables are defined in [Endpoint URIs](#).

The following example shows a request to generate an OpenAPI representation in YAML format:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/openapi?format=YAML
```

2. Optional: To use the OpenAPI file in IBM API Connect®, use the **extension** query parameter and specify apiconnect.

The following example shows a request to generate an OpenAPI representation in YAML format and use it in IBM API Connect:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/miniloanruleapp/1.0/miniloanrules/1.0/openapi?format=YAML&extension=apiconnect
```

### Results

You generated an OpenAPI representation in YAML or JSON format.

If you used an invalid URL or if an error occurs during the OpenAPI generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

**Note:** The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, a status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

**Parent topic:** [Executing rules by using the REST service](#)

### Related concepts:

[Content types](#)

 [Decision Connect](#)

 [IBM API Connect](#)

## Executing rules by using the SOAP service

Through a Simple Object Access Protocol (SOAP) web service, Rule Execution Server automatically presents as a web service any deployed ruleset that uses an XML schema or a Java™ XOM.

The SOAP web service automatically generates a Web Services Description Language (WSDL) file for each deployed ruleset archive.

SOAP web services support the first-level elements of the simple Java types, and the corresponding XML binding parameters in the WSDL code. A simple Java type is the finest-grain type from which a complex Java type can be expressed. A mapping table is provided in XML serialization of Java types.

The WSDL file references the XML schema types that are packaged in the RuleApp. You can use a tool such as Web Tools Platform (WTP) to generate a web service Java client. You can configure the WSDL generation through the Rule Execution Server console. You can configure the location and the web service endpoint.

### **Endpoint URIs**

Endpoint URIs represent rulesets as Rule Execution Server resources for the SOAP execution service.

### **XML serialization of ruleset XOMs**

In the SOAP service for ruleset execution, primitive Java types, XSD types, and Java XOM classes are serialized differently in the SOAP request and SOAP response.

### **Generating a WSDL representation**

You can generate a WSDL representation of request and response elements, and their documentation.

**Parent topic:** [Testing rule execution](#)

**Related concepts:**

[XML serialization of Java types](#)

**Related tasks:**

[Calling a decision service as a SOAP web service](#)

## Endpoint URIs

Endpoint URIs represent rulesets as Rule Execution Server resources for the SOAP execution service.

The endpoint URI for a decision service in the Operational Decision Manager on Cloud server uses the following format:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/ws/<ruleset_path>
```

The following list explains the parts of the URI:

- *<vhostname>*: Serves as the name of the host for the decision service.
- *<odm\_on\_cloud\_environment>*: Uses dev, test, or prod depending on whether the decision service ruleset is in the development, test, or production environment.
- *<ruleset\_path>*: Provides the path for the decision service rule execution and must take one of the following formats:
  - ruleAppName/rulesetName
  - ruleAppName/ruleAppVersion/rulesetName
  - ruleAppName/ruleAppVersion/rulesetName/rulesetVersion

The following example shows a URI to MiniloanServiceRuleset:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/MiniloanService/MiniloanServiceRuleset
```

**Parent topic:** [Executing rules by using the SOAP service](#)

## XML serialization of ruleset XOMs

In the SOAP service for ruleset execution, primitive Java™ types, XSD types, and Java XOM classes are serialized differently in the SOAP request and SOAP response.

Ruleset parameters are handled differently depending on whether they are expressed as XML elements or as Java types.

### Primitive types

Primitives types are Java types and classes that can be written as inline strings. The XML element that is used to serialize a primitive type as a ruleset parameter is the name of the ruleset parameter in the parameter namespace.

You can find more information here:

- [Changing target namespaces](#) explains how to customize the parameter namespace from the Ruleset View in the Rule Execution Server console.
- [XML serialization of Java types](#) shows the XML types and Java Architecture for XML Binding (JAXB) annotations for Java primitive types and simple types.

### Dynamic XOM

For dynamic XOMs, the root element depends on how the ruleset parameters are defined. The XML description is serialized from the original XSD code of the dynamic XOM on which the ruleset is based.

#### Name of the root element

For rulesets that are based on a dynamic XOM, the root element name is determined as follows:

- If the parameter is an XML element in the ruleset signature, that element is used as the root element name of the parameter in requests and responses.
- If the parameter is a complex or primitive Java type, the parameter name is used as the root element name of the parameter in requests and responses, within the configured param namespace.

**Note:** A ruleset parameter is declared from an XML element if it meets the following condition: that parameter uses an XML complex type for which an XML element was declared from that complex type in the original imported XSD file. For example, starting from the following XSD definition:

```
<xsd:element name="MyConference">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element name="session" type="session"
maxOccurs="unbounded"/>
 <xsd:element name="participant" type="participant"
minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

the ruleset parameter is declared from the MyConference XML element:

```
<par:Request
xmlns:par="http://www.ibm.com/rules/decisionsservice/myruleapp/myruleset
/param" xmlns:jav="http://www.acme.com/myconference">
 <jav:MyConference>
 <jav:session>
 [...]
 <jav:session>
 <jav:MyConference>
</par:Request>
```

### Serialization

The root element depends on your XML types.

- For XML complex types, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.
- For XML elements, the parameter declaration uses the defined XML element as the root element in the original namespace. In this case, the name of the parameter that is attached to each node is provided in the WSDL code as a comment message.

When multiple elements have the same name, element nodes are differentiated alphabetically. In this case, if you want one of the elements to be null, you must set the **xsi:nil** attribute to true. You can do that only if

the **nillable** attribute is set to `true` for the root element in the original XSD schema of the dynamic XOM. The default value of the **nillable** attribute is `false`.

## Java XOM

For Java XOMs, the root element derives from the parameter declaration and XML serialization uses JAXB annotations. Arrays are supported for ruleset parameters.

### Name of the root element

For rulesets that are based on a Java XOM, the parameter declaration uses the parameter name (in the parameter namespace) as the root element.

### Serialization

The XML structure of Java XOM classes is serialized through the standard JAXB process. If the default serialization does not provide the results that you expect for your classes, you can customize the serialization process by using JAXB annotations.

**Note:** A parameter is visible as an XML element only if you specify it with an `XmlElement` annotation or if valid `getXXX` and `setXXX` methods apply to the parameter.

### Arrays as ruleset parameters

In Java XOM, you can specify ruleset parameters as simple or multidimensional arrays. XML serialization is processed differently for each type of arrays:

- If the ruleset parameter is a simple array, the parameter name repeats as a standard array in XML.
- If the ruleset parameter is a multidimensional array, the parameter name repeats as a standard array type in XML for the first dimension and then uses `item` elements for the subsequent dimensions. For example:

```
<myMultiDimArrayParam>
 <item>1</item>
 <item>2</item>
</myMultiDimArrayParam>
<myMultiDimArrayParam>
 <item>3</item>
 <item>4</item>
 <item>5</item>
</myMultiDimArrayParam>
```

**Parent topic:** [Executing rules by using the SOAP service](#)

## Generating a WSDL representation

You can generate a WSDL representation of request and response elements, and their documentation.

### About this task

To write the SOAP payload for the request to a specific ruleset endpoint URI, you must generate the WSDL representation. The WSDL representation contains the description of the SOAP request and SOAP response, as well as the endpoint of the web service. A set of XSD files are attached to the WSDL file to describe the XML representation of the input and output objects.

To generate a WSDL representation, you can use the format of SOAP resource URIs. The WSDL format is described in the [Web Services Description Language](#) page of the W3C website.

### Procedure

1. Define the request:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/ws/{rulesetPath}/wsdl
```

The URI variables are defined in [Endpoint URIs](#).

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/miniloanruleapp/1.0/miniloanrules/1.0/wsdl
```

2. Optional: To generate the WSDL code and its XSD files to a compressed file, add the **zip** parameter.

Use the **zip** parameter if you want to download the WSDL as a ZIP archive in which the XSD files are saved separately from the WSDL file. This option is useful if you import more than one decision from Decision Server into an IBM® Integration Designer project, or if you migrate a ruleflow to a process. The separation of the object types from the WSDL avoids the duplication of these objects in IBM Integration Designer.

Example:

```
GET
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/ws/miniloanruleapp/1.0/miniloanrules/1.0/wsdl?zip=true
```

### Results

If you try to generate WSDL representation for an invalid URL or if an error occurs during the WSDL generation, an error status code is returned.

- If you sent the request from a browser, an HTML page displays the error message.

**Note:** The status code is 200 despite the error message because it applies to the HTML page, not to the request result.

- If you sent the request from a client other than a browser, status code 404 (Not Found) is returned and the error is described in XML.

You can find a description of error messages in [HTTP status codes](#).

**Parent topic:** [Executing rules by using the SOAP service](#)

## Integrating business rules

After you deploy a ruleset from a decision service, you can call the ruleset as a SOAP web service or by using the REST API. You can also interact with it by using OpenAPI or API Connect. You must provide service credentials for authentication, and should keep them separate from the application code.

### Calling decision service rulesets

Your application can call a decision service ruleset as a SOAP/XML, REST/XML, or REST/JSON web service. Incoming data is passed as HTTP content in the XML or JSON format. Security is provided over HTTPS through basic HTTP authentication that includes a user name and password. Use service credentials to authenticate client applications.

### Authentication for REST and SOAP invocation

Operational Decision Manager on Cloud supports the use of basic authentication in invoking REST and SOAP APIs. Instead of hardcoding login credentials into your application, pull them in from a separate file.



## Calling decision service rulesets

Your application can call a decision service ruleset as a SOAP/XML, REST/XML, or REST/JSON web service. Incoming data is passed as HTTP content in the XML or JSON format. Security is provided over HTTPS through basic HTTP authentication that includes a user name and password. Use service credentials to authenticate client applications.

Operational Decision Manager on Cloud uses the HTTPS secure communication protocol. It has the security level of TLS (TLS1.0, TLS1.1, or TLS1.2). If your client application does not support TLS, an exception occurs during the connection process.

You must use a cloud password in calling a decision service ruleset. Use service credentials because they are highly secure, and exempt from cloud portal policies for updating passwords (see [Service credentials for client applications](#)).

### **Using the REST API to call a ruleset**

Your client application calls a decision service ruleset through the REST API of Operational Decision Manager on Cloud. It uses standard HTTP GET, POST, PUT, and DELETE commands.

### **Using a SOAP web service**

A client application can invoke a decision service ruleset as a SOAP web service.

### **Using OpenAPI**

You can generate a client from an OpenAPI file that is made for a decision service ruleset.

### **Using API Connect**

You can expose a decision service ruleset to other organizations via IBM API Connect®, which manages access and uses built-in security mechanisms.

**Parent topic:** [Integrating business rules](#)

## Using the REST API to call a ruleset

Your client application calls a decision service ruleset through the REST API of Operational Decision Manager on Cloud. It uses standard HTTP GET, POST, PUT, and DELETE commands.

### Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud(see [Service credentials for client applications](#)).

### About this task

A client application calls a decision service ruleset by connecting with an execution server and calling the ruleset through the REST API.

The endpoint URI for the ruleset in the server looks as follows:

```
https://<vhostname>.bpm.ibmcloud.com/odm/<odm_on_cloud_environment>/DecisionService/rest/v1/<ruleset_path>
```

The following list explains the parts of the URI:

- *<vhostname>*: Serves as the name of the host for the decision service.
- *<odm\_on\_cloud\_environment>*: Uses dev, test, or prod depending on whether the decision service ruleset is in the development, test, or production environment.
- *<ruleset\_path>*: Provides the path for the decision service rule execution and must take one of the following formats:
  - ruleAppName/rulesetName
  - ruleAppName/ruleAppVersion/rulesetName
  - ruleAppName/ruleAppVersion/rulesetName/rulesetVersion

The following example shows a URI to MiniloanServiceRuleset:

```
https://vhost005.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/MiniloanService/MiniloanServiceRuleset
```

The execution of the ruleset is performed through an HTTP POST at the specified endpoint URI. The user name and password for the authentication are passed with an authorization header (see [Authentication for REST and SOAP invocation](#)).

### Example

The following example shows Java™ code that calls MiniloanServiceRuleset on the Operational Decision Manager on Cloud server. The example uses Apache HttpClient v4.3.

```
import java.io.IOException;

import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpStatus;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class DecisionServiceExecution {

 public static void main(String[] args) throws ClientProtocolException,
 IOException {

 // Replace <vhostname> with the name of the host of the cloud
 portal

 String endpointURI =
 "https://<vhostname>.bpm.ibmcloud.com/odm/dev/DecisionService/rest/v1/Miniloan
 Service/MiniloanServiceRuleset";

 // Replace "loginID" with the functional ID of the service
 account you are using to authenticate with your tenant in the Cloud
```

```

 String userName = "loginID";

 // Replace "password" with the password of the service account
 you are using to authenticate with your tenant in the Cloud
 String password = "password";

 String credentials = userName + ":" + password;
 String encodedValue =
Base64.encodeBase64String(credentials.getBytes());
 String authorization = "Basic " + new String(encodedValue);

 String contentType = "application/json";

 // Set the borrower and the loan
 String payload = "{\"borrower\": {" +
 "\"name\": \"John\", " +
 "\"creditScore\": 600, " +
 "\"yearlyIncome\": 80000 " +
 "}, " +
 "\"loan\": {" +
 "\"amount\": 500000, " +
 "\"duration\": 240, " +
 "\"yearlyInterestRate\": 0.05 " +
 "}" +
 "}";

 CloseableHttpClient client = HttpClients.createDefault();

 try {
 HttpPost httpPost = new HttpPost(endpointURI);
 // Add the basic authentication header
 httpPost.addHeader("Authorization", authorization);
 httpPost.setEntity(new StringEntity(payload));
 httpPost.addHeader("Content-Type", contentType);
 CloseableHttpResponse response =
client.execute(httpPost);
 try {
 if (response.getStatusLine().getStatusCode()
!= HttpStatus.SC_OK) {
 System.err.println("Status Code: " +
response.getStatusLine().getStatusCode());
 System.err.println("Status Line: " +
response.getStatusLine());

 String responseEntity =
EntityUtils.toString(response.getEntity());
 System.err.println("Response Entity: "
+ responseEntity);

 throw new RuntimeException(
 "An error occurred
when invoking Decision Service at: "
 +
endpointURI
 + "\n"
 +
response.getStatusLine() + ": " + responseEntity);
 } else {
 String result =
EntityUtils.toString(response.getEntity());
 System.out.println("Result: " +
result);
 }
 } finally {
 if (response != null) {
 response.close();
 }
 }
 }
 }
}

```

```
 }
 }
 } finally {
 client.close();
 }
}

}
```

**Parent topic:** [Calling decision service rulesets](#)

## Using a SOAP web service

A client application can invoke a decision service ruleset as a SOAP web service.

### Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#) and [Authentication for REST and SOAP invocation](#)).

### About this task

For a client application to invoke a decision service ruleset as a SOAP web service, you must create proxy classes from a Web Services Description Language (WSDL) file generated for the ruleset path. The format of a WSDL file is independent of the language that generates it.

### Procedure

1. Obtain a WSDL file for a decision service ruleset.
2. Generate proxy classes from the WSDL file.
3. Use the proxy classes to invoke the ruleset from your client application.

### Example

To get a WSDL file from Rule Execution Server:

1. Deploy the ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click the ruleset that corresponds to your decision service.
4. In the Ruleset View, click **Retrieve HTDS Description File**.
5. Select **SOAP** as the service protocol type.
6. Check **Latest ruleset version** and **Latest RuleApp version** to generate the WSDL file for the latest versions.
7. Click **Download**.

To generate the proxy classes with Apache Axis:

1. Download and install [Eclipse IDE for Java™ EE Developers](#), which includes the Web Tools Platform (WTP) to generate the proxy classes for invoking the web service from a Java application.
2. Start Eclipse IDE for Java EE Developers and create a new Java Project ( **File > New > Java Project**) to host the proxy classes.
3. Copy the WSDL file into this Java project.
4. Click **File > New > Other > Web Services > Web Service Client**.
5. In the Web Service Client wizard, click **Next**.
6. For the service definition, browse to the WSDL file.
7. Move the slider to select **Develop client**.
8. Under Configuration, ensure **Apache Axis** is selected as the Web service runtime, and then select the Java project as the Client project where you want to host the proxy classes that are generated.
9. Click **Finish**.

The following Java code sample imports the proxy classes for MiniloanServiceRuleset and calls the ruleset from a Java application:

```
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceMiniloanServiceRulesetBindingStub;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetDecisionServiceProxy;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetRequest;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Minil
oanServiceRulesetResponse;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.param
.Borrower;
import
com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.param
.Loan;

public class DecisionServiceExecution {
```

```

 public static void main(String[] args) {

 // Replace <vhostname> with the name of the host of the Cloud
 portal
 // NB: endpointURI is defined in the location attribute of the
 WDSL file
 String endpointURI =
 "https://<vhostname>.bpm.ibmcloud.com/odm/dev/DecisionService/ws/MiniloanService/MiniloanServiceRuleset/v75";

 MiniloanServiceRulesetDecisionServiceProxy proxy = new
 MiniloanServiceRulesetDecisionServiceProxy(endpointURI);

 MiniloanServiceMiniloanServiceRulesetBindingStub stub =
 (MiniloanServiceMiniloanServiceRulesetBindingStub)proxy.getMiniloanServiceRule
 setDecisionService_PortType();

 // Replace "loginID" with the functional ID of the service
 account you are using to authenticate with your tenant in the Cloud
 stub.setUsername("loginID");

 // Replace "password" with the password of the service account
 you are using to authenticate with your tenant in the Cloud
 stub.setPassword("password");

 // Set the borrower

 com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Borro
 wer borrower =
 new
 com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Borro
 wer(
 "John", // name
 600, // credit score
 80000); // yearlyIncome

 Borrower borrowerParam = new Borrower(borrower);

 // Set the loan

 com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Loan
 loan =
 new
 com.ibm.www.rules.decisionservice.MiniloanService.MiniloanServiceRuleset.Loan(
 500000, // amount
 240, // duration
 0.05, // yearlyInterestRate
 0, // yearlyRepayment (to be
 computed by the decision service)
 true, // approved (set to true
 by default, to be computed by the decision engine),
 null); // messages (to be
 computed by the decision service)

 Loan loanParam = new Loan(loan);

 // Set the decision ID
 String decisionID = "1";

 MiniloanServiceRulesetRequest request = new
 MiniloanServiceRulesetRequest(decisionID, borrowerParam, loanParam);

 try {
 MiniloanServiceRulesetResponse response =

```

```

proxy.miniloanServiceRuleset(request);
 System.out.println("Rules executed.");
 System.out.println("Approved: " +
response.getLoan().getLoan().isApproved());
 System.out.println("Yearly interest rate: " +
response.getLoan().getLoan().getYearlyInterestRate());
 System.out.println("Yearly repayment: " +
response.getLoan().getLoan().getYearlyRepayment());
 String[] messages =
response.getLoan().getLoan().getMessages();
 if (messages != null) {
 System.out.println("Messages: ");
 for (String message : messages) {
 System.out.println(message);
 }
 }
 }
 catch (Exception e) {
 throw new RuntimeException("An error occurred when
invoking Decision Service at: "
+ endpointURI, e);
 }
}
}

```

**Parent topic:** [Calling decision service rulesets](#)

# Using OpenAPI

You can generate a client from an OpenAPI file that is made for a decision service ruleset.

## Before you begin

The client application must pass authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#)).

### Important:

You can use Swagger Editor or other [open source Swagger tools](#) to generate a client in your preferred language, for example: C#, Go, Apache Groovy, JavaScript, PHP, Python, Ruby, Scala, Swift, or TypeScript.

## About this task

To call a decision service ruleset from a client application, you create proxy classes from an OpenAPI file generated for a ruleset path.

## Procedure

1. Obtain a OpenAPI file for a decision service ruleset.
2. Generate proxy classes from the OpenAPI file.
3. Use the proxy classes to invoke the ruleset from your client application.

## Example

To get the OpenAPI file from Rule Execution Server:

1. Deploy the decision service ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click a ruleset for your decision service.
4. In Ruleset View, click **Retrieve HTDS Description File**.
5. Select **REST** as a service protocol type.
6. Select **OpenAPI - YAML** or **OpenAPI - JSON** as a format to generate an OpenAPI file in YAML or JSON.
7. Check **Latest ruleset version** and **Latest RuleApp version** to generate the OpenAPI file for the latest versions.
8. Click **Download**.

To generate the proxy classes in Swagger Editor:

1. Open <http://editor.swagger.io/> in a web browser.
2. Click **File > Import File**.
3. Click **Browse** to select the OpenAPI file, and then click **Open**.
4. Click **Import**.
5. Click **Generate Client > Java**.

The following Java™ code sample imports the proxy classes generated in Swagger Editor for a ruleset and calls the ruleset from a Java application:

```
import java.util.List;

import io.swagger.client.ApiClient;
import io.swagger.client.ApiException;
import io.swagger.client.model.Borrower;
import io.swagger.client.model.Loan;
import io.swagger.client.model.Request;
import io.swagger.client.model.Response;

public class DecisionServiceExecution {

 public static void main(String[] args) {

 ApiClient apiClient = new ApiClient();

 // Replace "loginID" with the ID of a user who has access to
the Cloud portal
 apiClient.setUsername("loginID");

 // Replace "password" with the password of a user who has
access to the Cloud portal
 apiClient.setPassword("password");
```



```

DefaultApi api = new DefaultApi(apiClient);

// Create the request
Request request = new Request();

// Set the borrower
Borrower borrower = new Borrower();
borrower.setName("John");
borrower.setCreditScore(600);
borrower.setYearlyIncome(80000);
request.setBorrower(borrower);

// Set the loan
Loan loan = new Loan();
loan.setAmount(500000);
loan.setDuration(240);
loan.setYearlyInterestRate(0.05);
// approved (set to true by default, to be computed by the
decision engine)
loan.setApproved(true);
request.setLoan(loan);

// Retrieve the response
try {
 Response response =
api.callDecisionOperation(request);
 System.out.println("Rules executed.");
 System.out.println("Approved: " +
response.getLoan().getApproved());
 System.out.println("Yearly interest rate: " +
response.getLoan().getYearlyInterestRate());
 System.out.println("Yearly repayment: " +
response.getLoan().getYearlyRepayment());
 List<String> messages =
response.getLoan().getMessages();
 if (messages != null) {
 System.out.println("Messages: ");
 for (String message : messages) {
 System.out.println(message);
 }
 }
} catch (ApiException e) {
 throw new RuntimeException("An error occurred when
invoking Decision Service", e);
}
}
}

```

**Parent topic:** [Calling decision service rulesets](#)

# Using API Connect

You can expose a decision service ruleset to other organizations via IBM API Connect®, which manages access and uses built-in security mechanisms.

## Before you begin

You must use authentication credentials for Operational Decision Manager on Cloud (see [Service credentials for client applications](#)).

### Important:

IBM API Connect can also generate client code for Swagger APIs. For more information, see [Viewing and testing APIs in the Developer Portal](#), and for API Connect, see [IBM API Connect](#).

## About this task

To expose a decision service ruleset in API Connect, you must generate an OpenAPI file for API Connect, import it into API Connect, and configure the credentials.

## Procedure

1. Obtain a OpenAPI file specific to API Connect for a decision service ruleset.
2. Import the OpenAPI file into API Connect, and set the credentials used when calling the ruleset.

## Example

To get the OpenAPI file specific to API Connect from Rule Execution Server:

1. Deploy the decision service ruleset to Rule Execution Server.
2. In the Rule Execution Server console, go to the **Explorer** tab.
3. In the Navigator pane, click a RuleApp, and then click a ruleset that corresponds to your decision service.
4. In the Ruleset View, click **Retrieve HTDS Description File**.
5. Select **REST** as a service protocol type.
6. Select **OpenAPI - YAML** or **OpenAPI - JSON** as a format to generate an OpenAPI file in YAML or JSON.
7. Check **Latest ruleset version** and **Latest RuleApp version** to generate the OpenAPI file for the latest versions.
8. Check **Proxy for API Connect** to generate an OpenAPI file for a proxy so that the decision service can be called from API Connect.
9. Click **Download**.

To import the OpenAPI file into API Connect and set the credentials:

1. Download and install API Designer for IBM API Connect, or use it on IBM® Bluemix®.
2. In the **APIs** tab in API Designer, click **Add > Import an existing OpenAPI**.
3. Click **Select File**.
4. Navigate to the OpenAPI file, and then click **Open**.
5. Click **Import**.
6. Click the **Assemble** tab to edit the policy assembly.
7. Click the **Calls Decision Service Operation** invoke policy.
8. In the property sheet, enter the user name and password of a user who has access to Operational Decision Manager on Cloud.
9. Click the **Save** icon.

Your decision service ruleset is now ready to be called from API Connect.

You can now continue configuring your API at the API Connect level by adding a custom security.

**Parent topic:** [Calling decision service rulesets](#)

# Authentication for REST and SOAP invocation

Operational Decision Manager on Cloud supports the use of basic authentication in invoking REST and SOAP APIs. Instead of hardcoding login credentials into your application, pull them in from a separate file.

The recommended practice for using login credentials is to keep them in a separate file, and to call them from your application during authentication. With this approach, you do not have to alter the code of your application every time you change your credentials. The primary alternative is to add the credentials directly to your application. This approach risks the introduction of mistakes to your code, and requires you to redeploy the application every time you update the credentials.

The cloud portal offers two types of credentials:

- Service credentials: These credentials are exempt from the password renewal policies of the cloud portal. They are only valid for calling REST and SOAP APIs. Users cannot use them to log in to the portal or its components (see [Service credentials for client applications](#)).
- Local accounts: These user accounts have credentials that are subject to the password renewal policies of the cloud portal. Users and applications can use these credentials to log in to the portal and its components (see [Managing your account](#)).

**Tip:**

Use service credentials for authenticating your applications. They provide better security, and you can apply your own policies for updating login credentials.

## SAML users

You cannot use SAML login credentials to authenticate a decision service. If you use SAML to log in to Operational Decision Manager on Cloud, you must request service credentials for REST or SOAP invocation (see [Getting a set of service credentials](#)). IBM® internal users automatically use SAML to log in to Operational Decision Manager on Cloud, and must request service credentials.

## Calling service credentials

The following steps show you how to set up your application to call service credentials from a separate file. Depending on your IT strategy, you can use the credentials of a local account in place of the service credentials.

1. Obtain your service credentials from your cloud administrator. Only a cloud administrator can generate service credentials. The service credentials include a functional ID and a password.
2. Place the service credentials in a property file, for example, `sc.properties`:

```
odmoc.url=https://mytenant.bpm.ibmcloud.com/odm/dev
odmoc.functionalId=api1.fid@t1023
odmoc.password=xfmptNJsQ0NCvRxB1bYS1AxlIcYyh1UR2y/LMpyx
```

The properties in this example are defined as follows:

Property	Description
<code>odmoc.url</code>	The URL of your tenant of the cloud portal
<code>odmoc.functionalId</code>	The functional ID in the service credentials
<code>odmoc.password</code>	The password in the service credentials

3. In your application, add the code that calls the service credentials from the file `sc.properties`. The following code is from a Java™ client:

```
package test;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Properties;

import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpResponse;
import org.apache.http.Header;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.client.HttpClients;
```

```

import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.util.EntityUtils;

import com.ibm.json.java.JSONObject;

public class OdmRestClient {

 public static void main(String[] args) throws Exception {
 // The builder should be configured to connect using HTTPS to ODM
on Cloud,
 // and must use a secure cipher with hostname verification. The
following
 // code does not reflect all the required settings.
 HttpClientBuilder builder = HttpClientBuilder.custom();
 builder.disableRedirectHandling();
 HttpClient client = builder.build();

 // This code calls the service credentials from the file
sc.properties.
 //It reads the url, functional Id and password from a property
file.
 String configFile = "sc.properties";
 FileInputStream fistream = null;
 try {
 fistream = new FileInputStream(configFile);
 }
 catch (FileNotFoundException e) {
 System.out.println("Could not open file: " + configFile);
 System.exit(0);
 }
 Properties props = new Properties();
 props.load(fistream);

 String baseUrl = props.getProperty("odmoc.url");
 String fid = props.getProperty("odmoc.functionalId");
 String pwd = props.getProperty("odmoc.password");
 String url = baseUrl + "/res/api/v1/ruleapps?count=true";

 // Basic Authentication header value
 String baHeader = fid + ":" + pwd;
 System.out.println("BA header is: " + baHeader);
 baHeader = Base64.encodeBase64String(baHeader.getBytes());

 // execute request
 HttpResponse response = null;
 System.out.println("** Calling: " + url);

 HttpGet request = new HttpGet(url);
 request.setHeader("Authorization", "Basic " + baHeader);
 try {
 response = client.execute(request);
 }
 catch (Exception e) {
 throw new Exception("Could not execute API call, reason: " +
e.getMessage());
 }

 int httpRespCode = response.getStatusLine().getStatusCode();
 System.out.println("Response code is: " + httpRespCode);

 if (httpRespCode==302) {
 // We are in one of these 3 cases: wrong ID, wrong password,
account locked
 System.out.println("Basic auth access denied.");
 System.out.println("Reason1: Wrong ID or password");
 }
 }
}

```

```
 System.out.println("Reason2: Account locked after some
unsuccessful attempts");
 }
 else if (httpRespCode==401) {
 // Aithorization is not granted to access the URL
 System.out.println("ODM API access was denied: you are not
authorized");
 }
 else if (httpRespCode==200){
 // Correct execution result, extract the HTTP response
 String json = EntityUtils.toString(response.getEntity(),
"UTF-8");
 System.out.println("Successful invocation, result : " +
json);
 }
 else {
 // Diagnose why such a code. Contact IBM support with all the
details.
 System.out.println("Unexpected return code: " +
httpRespCode);
 }
}
}
```

The HTTP client side can return the following codes during authentication:

Code	Description
Code 200	This code denotes a correct execution result. The return result is the actual response to the API call. You can get Code 200 in the case of wrong credentials when the redirect handling is not deactivated. To diagnose a connection problem accurately, deactivate the redirect handling.
Code 302	The HTTP request returns Code 302 when the functional ID or password is wrong, or the account is locked. The code does not provide the cause of the authentication failure. If the account is locked, you can try waiting one hour for the account to be unlocked automatically. If the account is not unlocked automatically, contact your cloud administrator. This code is returned when the redirect handling is deactivated
Code 401	You get this code when the credentials are correct, but they are not authorized to access the URL. You must ask your cloud administrator to grant the required authorization. (Note: This code is not currently returned but authorization could be enhanced in the future.)
Other codes	If you get any other code, contact IBM for assistance (see <a href="#">Troubleshooting and support</a> ).

Parent topic: [Integrating business rules](#)

## Troubleshooting and support

Use the troubleshooting resources to resolve any technical problem that you might encounter while using Operational Decision Manager on Cloud.

### Identifying problems

Start by using this troubleshooting process to try to identify the source of the problem. It includes recording the symptoms, re-creating the problem, and eliminating potential causes.

### Support for Operational Decision Manager on Cloud

Customer support is available for the users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.

### Limitations on OpenAPI specifications and tools

You might face some restrictions concerning the OpenAPI specifications and tools when you use the hosted transparent decision service (HTDS) through OpenAPI-based clients.

## Identifying problems

Troubleshooting is the process of finding and eliminating the cause of a problem. The first steps for identifying a problem include recording the symptoms, re-creating the problem, and eliminating potential causes.

### Recording the symptoms

When you get an error message in the product, make sure that you record this message. You might also receive multiple error messages that look similar, but have subtle differences. By recording the details of each one you can learn more about where your problem exists.

You can get error messages in the following sources:

- **Problems** view in the Eclipse workbench
- **Console** view in the Eclipse workbench
- **Error Log** view in the Eclipse workbench
- Error dialogs
- Log files in your workspace: `<workspace>/ .metadata/ .log`

### Re-creating the problem

To reproduce the problem, try to identify the steps that you performed. If you have a consistently repeatable test case, you can have an easier time determining what solutions are necessary.

You can ask yourself the following questions:

- How did you first notice the problem?
- What was the first symptom of this problem?
- Were there other symptoms occurring around that time?
- Did you do anything different that made you notice the problem?
- Is this process a new procedure, or has it worked successfully before?
- If this process worked before, what has changed?

The change can refer to any type of change made to the system, ranging from adding new hardware or software, to configuration changes you might have made to existing software.

- Does the same problem occur elsewhere?

The problem could be on one machine only or on multiple machines.

### Eliminating possible causes

You can narrow the scope of your problem by eliminating components that are not causing the problem. Consult the information that comes with the product and other available resources to help you with your elimination process.



# Support for Operational Decision Manager on Cloud

Customer support is available for the users of subscription instances of Operational Decision Manager on Cloud. Other online support resources are also available.

## Before contacting customer support

Start by collecting data on your problem (see [Collecting Data: Read first for all IBM Operational Decision Management Components](#)). Customer support needs this information to help you resolve your problem. In gathering the data, you can refer to online resources that might present a solution to your problem.

## Business hours

Support is available during the following hours:

- 8:00 AM - 5:00 PM Eastern Standard Time (EST), Monday through Friday (excluding US holidays)
- 8:00 AM - 5:00 PM Central European Time (CET), Monday through Friday (excluding German holidays)

Submit your questions or issues through the [Client Success Portal](#). You must sign in to use the service.

## After hours

For support outside normal business hours, open a ticket in an [IBM Service Request](#). Refer to the following table for severity levels.

Table 1. Severity levels and their definitions

Severity	Severity definition	Response time objectives	Response time coverage
1	Critical business impact or service down: Business critical functionality is inoperable or a critical interface has failed. This usually applies to a production environment and indicates an inability to access services, resulting in a critical impact on operations. This condition requires an immediate solution.	Within 1 hour	24 hours, 7 days a week
2	Significant business impact: A service business feature or function of the service is severely restricted in its use, or the client is in jeopardy of missing business deadlines.	Within 2 business hours	Monday-Friday, business hours
3	Minor business impact: The service or functionality is usable, and there is no critical impact on operations.	Within 4 business hours	Monday-Friday, business hours
4	Minimal business impact: An inquiry or nontechnical request.	Within 1 business day	Monday-Friday, business hours

## Online support resources

- [dW Answers for Operational Decision Manager on Cloud](#): Answers common questions that are related to the product.
- [IBM Operational Decision Manager Developer Center](#): Serves as an online community for developers of reusable decisions for smarter processes and applications.
- [IBM Operational Decision Manager on Cloud: Terms of Use](#): Provides information about the service level agreements (SLAs) that apply to technical support.
- [Service requests and PMRs](#): Shows the status of service requests.
- [IBM Software as a Service \(SaaS\) Support Handbook](#): Provides information about support response times.
- [Directory of worldwide contacts](#): Lists the countries where IBM has offices. Click your country in the directory to obtain contact information about local IBM services.



# Limitations on OpenAPI specifications and tools

You might face some restrictions concerning the OpenAPI specifications and tools when you use the hosted transparent decision service (HTDS) through OpenAPI-based clients.

## Symptoms

You might encounter an issue in some specific use cases due to an incorrect behavior in some of the OpenAPI reference tools.

**Note:** You might encounter some of the issues that are described in the Table 1 when you use one or more of the following tools:

- swagger-core Java™ library (embedded in the product): V1.5.10
- Swagger Codegen code generation tool: V2.2.1
- Swagger Editor: V2.10.3

## Diagnosing the problem

Table 1. Data types and limitations

Data type	Limitation
short type	<p>The OpenAPI specification has only <code>int32</code> and <code>int64</code> representations for integer, and standard integer numbers (<code>int</code> and <code>java.lang.Integer</code>) and long numbers (<code>long</code> and <code>java.lang.Long</code>) respectively for mapping.</p> <p>Short numbers (<code>short</code> and <code>java.lang.Short</code>) are mapped to <code>int32</code> and treated as regular integer numbers.</p> <p>As a consequence, when you use <code>short</code> or <code>java.lang.Short</code> in rule projects or decision services, clients or interfaces that are generated from the OpenAPI definition file provided by the HTDS allow you to input integer numbers instead of short numbers. Thus, entering a number bigger than the max value for short ends up in an execution error.</p>
password data type annotation	<p>Customizing Java XOM fields with the <code>@ApiModelProperty</code> annotation might be reflected on the OpenAPI definition file that is generated by the HTDS.</p> <p>In particular, you might use the <b>datatype</b> property to specify or override the type of the field of the Java object in OpenAPI. However, if you set password as a data type, the OpenAPI definition file might not be generated. You will see the following error:</p> <div>Unrecognized Type: [null]</div> <p>This problem is due to an incorrect behavior in the swagger-core Java library that is used for generating OpenAPI definition files. Avoid using the password data type.</p> <p>As a workaround, you can use a <code>String</code> field without an annotation (or at least without the password data type), and modify the generated OpenAPI definition file to specify <code>"format": "password"</code> in the corresponding field. It works with the Swagger Editor (the test form takes it into account, hiding characters as they are entered), and the Swagger Codegen tool can generate a client properly (the field is represented with a standard <code>String</code> type).</p>
binary data type annotation	<p>Similar to the password data type, a <code>String</code> field in a Java XOM can be annotated with the <code>@ApiModelProperty</code> annotation having</p>

	<p>binary as a data type property.</p> <p>Unlike the password data type, this binary data type does not prevent the OpenAPI definition file from being properly generated by the HTDS.</p> <p>However, if you use the Swagger Codegen tool to generate a Java client, the annotated field is represented as a <code>byte[]</code> field, which leads to a ruleset execution error at run time.</p> <p>In this particular use case, no such issue arises when you use the Swagger Editor.</p>
byte type	<p>When <code>byte</code> or <code>java.lang.Byte</code> is used as a XOM field or as a ruleset parameter, it becomes <code>byte[]</code> in a Java client that is generated by the Swagger Codegen tool, which leads to a ruleset execution error at run time.</p> <p>In this particular use case, no such issue arises when you use the Swagger Editor, since bytes are represented as strings, and the HTDS REST accepts and converts them to real bytes.</p>
arrays	<p>Arrays become lists when you use the Swagger Codegen tool to generate Java clients.</p> <p>For example, a field of type <code>String[]</code> becomes <code>List&lt;String&gt;</code>.</p>

## Trial version

The no-charge, 30-day trial version introduces you to Operational Decision Manager on Cloud.

The trial version contains the full version of Operational Decision Manager on Cloud, and is kept up to date with the latest features. You use it anonymously and securely, without risk of exposure to users from other organizations. You can take a decision service or a decision model service completely through its lifecycle.

**Note:** The platform is fully contained, so you cannot connect it to a production application outside the trial instance.

## Introduction

The trial version gives you an opportunity to work on an instance of Operational Decision Manager on Cloud. It provides all the components you need to create, develop, and deploy a decision service or a decision model service. If you use the trial version with other users from your organization, you can discover the project management features along with the development features.

You can also use a sample client application to call business rules from an execution server. The application is included in the download files, and works with the product tutorials. You can see how to connect decision services with client applications.

While the trial version is fully functional, you cannot use it to develop rule applications for the production applications of your organization. You can use the tutorials and sample application to get a feel for the platform.

Access to the trial version is limited to 30 days. After 30 days, you must register again to continue using the trial version.

You use the trial version by following the information in this knowledge center. For additional information, you can contact the cloud team or visit [IBM Operational Decision Manager Developer Center](#). There is no customer support for the trial version, and you cannot obtain assistance through the Client Success Portal or IBM Service Request.

**Important:** The trial version includes data security and privacy (see [Data Processing and Protection Datasheet](#)).

## Signing up

To use the trial version:

1. Sign up for the free trial version at [IBM® Operational Decision Manager on Cloud Trial](#). Your application form is checked, and an invitation is sent to your email address.
2. Follow the instructions in the invitation:

IBM Operational Decision Management on Cloud is available now

[Click here to activate your account](#)

Or copy and paste this link into your browser  
<https://www.blueworksccloud.com/login/?token=0240b89a-0077-4652-8adb-434d65491e55&activation>

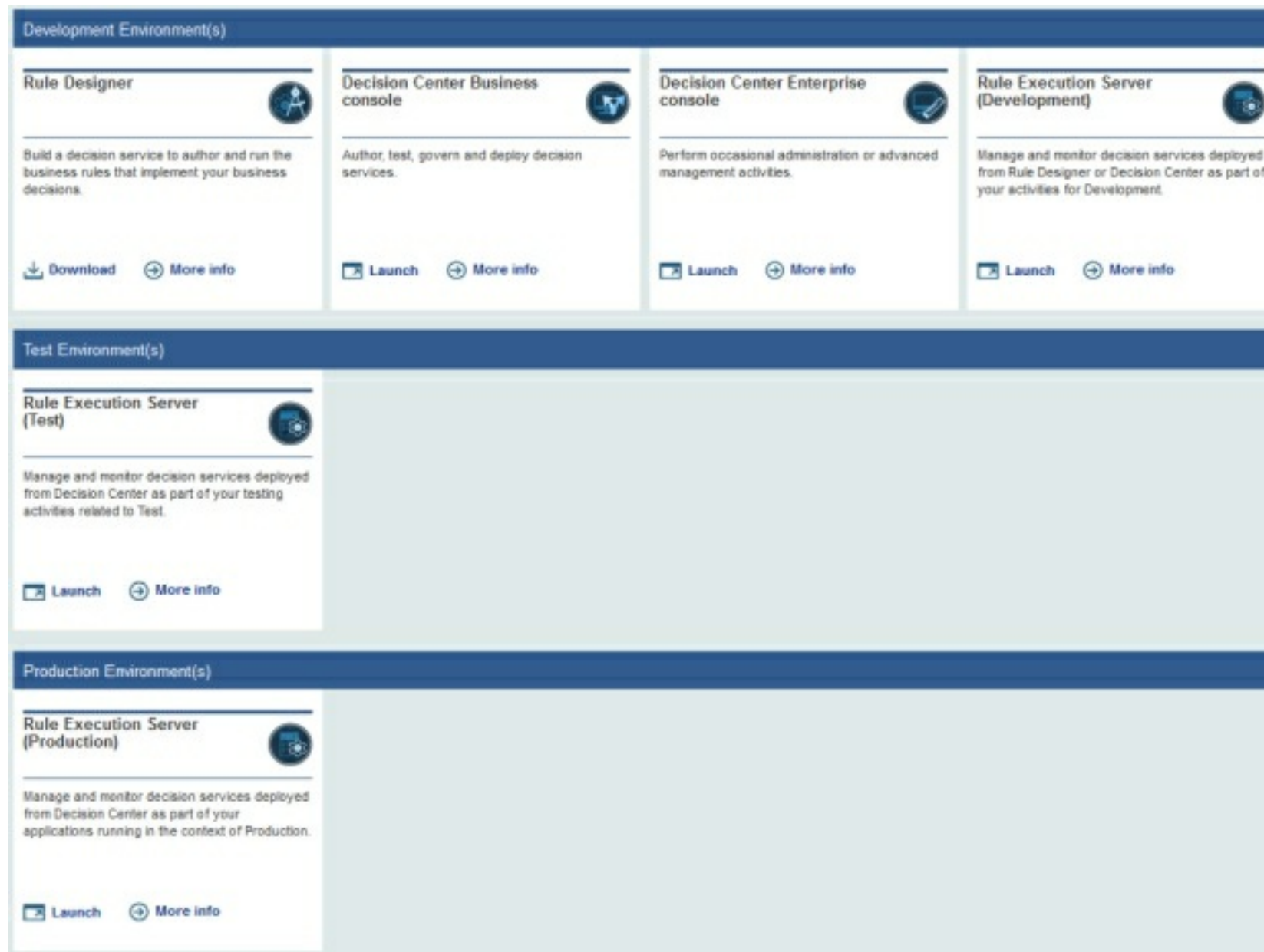
"Use the link to validate your email address and activate your account. Then, you can access the service at  
<https://vhost108.blueworksccloud.com>

Use your email address as your login name.

3. When the Activate Your Account page opens, enter the requested information and click **Activate**. A message informs you that your account is being provisioned.

**Tip:** When prompted to enter a password, follow the guidelines at [Resetting your password](#).

4. Enter your email address as your user ID in the sign-in page, and click **Continue**.
5. Enter your password, and then click **Continue** again. The trial instance of Operational Decision Manager on Cloud opens. It contains the full version of the portal with the development, test, and production environments (see [Cloud environments](#)):



You can now use the trial version.

## Tutorials

Use the following tutorials to become familiar with Operational Decision Manager on Cloud:

- [Getting started in Operational Decision Manager on Cloud](#): This tutorial shows you how to work on a decision service in the Operational Decision Manager on Cloud portal.
- [Creating a decision service in Rule Designer](#): This tutorial shows you how to create a decision service in Rule Designer. You also deploy the decision service to Rule Execution Server, and publish it to Decision Center.
- [Getting started with decision modeling](#): This tutorial shows the basics of creating a decision model service. You do all the modeling and authoring of a decision service in the Decision Center Business console.

You can find more tutorials at [Tutorials](#).

# Reference

A set of reference topics for IBM® Decision Server.

## [Rule Designer reference](#)

Refer to these topics for libraries, rule languages, and user interfaces.

## [Decision Center reference](#)

Refer to these topics for messages.

## [Rule Execution Server reference](#)

Refer to these topics for predefined ruleset and RuleApp properties.

# Rule Designer reference

A set of reference topics that includes libraries, rule languages, and user interfaces.

## [File types](#)

It can be useful to understand the different file types available in a rule project.

## [Rule languages](#)

Decision Server provides three rule languages: Business Action Language (BAL), ILOG® Rule Language (IRL), and Advanced Rule Language (ARL).

## [Messages](#)

Each listed message provides a detailed description about the error message and some steps to try and resolve the error.

## File types

It can be useful to understand the different file types available in a rule project.

### XML rule artifact files

The different types of XML rule artifact file all have the same structure.

### BOM files

A BOM file ( . bom) is a text file that defines part of the business object model (BOM)

### Vocabulary files

A vocabulary file ( \_<locale>.voc) is a key-value property file that stores the verbalization data for the BOM.

### BOM to XOM mapping files

A BOM to XOM mapping file ( . b2x) is an XML file that stores the mapping between the BOM and the XOM.

### Rule project files

A rule project file ( . ruleproject) is an XML file that stores the properties that are specific to a rule project.

**Parent topic:** [Rule Designer reference](#)

# XML rule artifact files

The different types of XML rule artifact file all have the same structure.

XML rule artifact files have the following extension: .rulepackage, .brl, .dta, .dtr, .fct, .rfl, .trl, .var, .brt, .qry. A description of each file type is provided in [Rule project items](#).

All XML rule artifact files are based on the following structure:

XML version header
Element type
Name
UUID
General properties (one per line)
Specific properties

**Parent topic:** [File types](#)

**Related information:**  
[Rule project items](#)



## BOM files

A BOM file (.bom) is a text file that defines part of the business object model (BOM)

The format of a BOM file is similar to the Java™ class definition format.

Here is an example of a .bom file:

```
public class MyClass
{
 public int attribute;
 public ilog.rules.brl.String method(float arg);
}
```

**Parent topic:** [File types](#)

**Related information:**

[business object model \(BOM\)](#)

[Rule project items](#)

## Vocabulary files

A vocabulary file (`_<locale>.voc`) is a key-value property file that stores the verbalization data for the BOM.

You have one vocabulary file per locale, so a BOM entry can be associated with several vocabulary files. All the vocabulary files are published to Decision Center , which displays the rules in the user locale.

Here is an example of a `_<locale>.voc` file:

```
MyClass
MyClass#label = my class
MyClass.attribute#sentence.action = set the attribute of {this} to {attribute}
MyClass.attribute#sentence.navigation = {attribute} of {this}
MyClass.method(float)#sentence.navigation = {this}.method({0})
```

**Parent topic:** [File types](#)

**Related information:**

[Vocabulary](#)

[Rule project items](#)

## BOM to XOM mapping files

A BOM to XOM mapping file (.b2x) is an XML file that stores the mapping between the BOM and the XOM.

Here is an example of a BOM to XOM mapping file:

```
<translation>
 <class>
 <businessName>MyClass</businessName>
 <executionName>java.util.Vector</executionName>
 <import>java.util.Vector</import>
 <tester language="irl"><![CDATA[
 String type = (String)this.get(0);
 return type.equals("MyClass");
]]></tester>
 <attribute>
 <name>attribute</name>
 <getter language="irl"><![CDATA[
 Integer val = (Integer)this.get(1);
 return val.intValue();
]]></getter>
 <setter language="irl"><![CDATA[
 this.set(1, new Integer(value));
]]></setter>
 </attribute>
 </class>
</translation>
```

**Parent topic:** [File types](#)

**Related information:**

[Defining BOM to XOM mapping](#)

[Rule project items](#)

## Rule project files

A rule project file (.ruleproject) is an XML file that stores the properties that are specific to a rule project.

Here is an example of a rule project file:

```
<?xml version="1.0" encoding="UTF-8"?>
<model.base:RuleProject xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:model.base="http://ilog.rules.studio/model/base.ecore"
xmlns:model.bom="http://ilog.rules.studio/model/bom.ecore"
xmlns:model.xom="http://ilog.rules.studio/model/xom.ecore">
 <name>rproject</name>
 <uuid>_srANAD1JEdqCPM8pR7aTDA</uuid>
 <outputLocation>output</outputLocation>
 <categories>any</categories>
 <paths xsi:type="model.xom:XOMPath" pathID="XOM"/>
 <paths xsi:type="model.bom:BOMPath" pathID="BOM"/>
 <modelFolders xsi:type="model.base:SourceFolder">
 <name>rules</name>
 </modelFolders>
 <modelFolders xsi:type="model.bom:BOMFolder">
 <name>bom</name>
 </modelFolders>
</model.base:RuleProject>
```

The structure of the files has been designed to be handled easily by diff tools, which you use to identify differences between different versions of the file. Being able to identify differences is an important feature when using a source code control system or when updating files from Decision Center.

**Parent topic:** [File types](#)

**Related information:**  
[Rule project items](#)

## Rule languages

Decision Server provides three rule languages: Business Action Language (BAL), ILOG® Rule Language (IRL), and Advanced Rule Language (ARL).

### **Business Action Language (BAL)**

You use constructs to build rules. You use operators to do arithmetic operations, associate or negate conditions, and compare expressions. You use literals to declare values as being of a particular type. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

### **ILOG Rule Language (IRL)**

The ILOG Rule Language (IRL) contains a set of keywords, and has its own syntax to structure each part of the rule. IRL does not support generics because IRL is not Java™. As a result you cannot use generics in: IRL, functions, BOM to XOM mapping, and initial/final actions.

### **Advanced Rule Language (ARL)**

For each Business Action Language (BAL) expression that you create for BOM-to-XOM mapping in the decision engine, you can review its equivalent Advanced Rule Language (ARL) expression. ARL is a read-only rule language that is similar in syntax to Java 7.

**Parent topic:** [Rule Designer reference](#)

## Business Action Language (BAL)

You use constructs to build rules. You use operators to do arithmetic operations, associate or negate conditions, and compare expressions. You use literals to declare values as being of a particular type. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

### **BAL constructs**

You use BAL constructs to build business rules.

### **BAL operators**

You use operators in rule statements to do arithmetic operations, associate or negate conditions, and compare expressions.

### **BAL literals**

The Business Action Language (BAL) supports number, string, date, and time data types.

### **Punctuation in rules**

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules (for example, in the Intellirule editor), mandatory punctuation is usually predicted in the completion menu.

**Parent topic:** [Rule languages](#)

# BAL constructs

You use BAL constructs to build business rules.

## all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

## any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

## called (variable)

This construct provides a name for a variable declared in a `for each action` statement.

## definitions

This construct introduces the part of the rule where you define variables.

## else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

## for each (object) in (list)

This construct specifies the actions that must be performed for every element of a collection.

## from (object)

This construct accesses data from objects that are related to known objects.

## if

This construct introduces the part of the rule in which you define conditions.

## in (list)

This construct accesses data from object collections that are related to known objects.

## it is not true that (a condition)

This construct negates a condition statement.

## none of the following conditions are true

This construct negates a group of conditions.

## print (string)

This construct prints a string, phrase or rule name to the standard output.

## set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

## set (variable) to (value)

This construct changes the value of a ruleset parameter or variable.

## the name of this rule

This construct returns the name of the currently executed rule in the `then` part of a business rule.

## the number of (objects)

This construct returns the number of objects in the current data set or in the specified collection.

## then

This mandatory construct introduces the part of the rule that defines the actions that are executed if the `if` part of a rule is satisfied.

## there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the current data set or the specified collection.

## there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the current data set or the specified collection.

## there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in the current data set.

## there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

## there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

## there is at least one (object)

This construct tests whether there is at least one object of a given type in the current data set.

**there is at most one (object)**

This construct tests whether there is at most one object of a given type in the current data set.

**there is no (object)**

This construct tests whether a data set contains no objects of the given type.

**there is one (object)**

This construct tests whether the current data set contains one and only one object of a given type.

**where (test)**

This construct matches objects against tests.

**Parent topic:** [Business Action Language \(BAL\)](#)

**Related information:**  
[Working with action rules](#)



## all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

### Purpose

The construct `all of the following conditions are true` provides a more compact and convenient alternative to the logical operator `and` when you want to combine multiple conditions. The resulting statement is considered true only if each of the listed conditions is true.

### Syntax

```
all of the following conditions are true:
 - <condition>* [,]
```

### Description

This is equivalent to writing `if <condition 1> and <condition 2> and ... <condition n>`. However, if you have a large number of conditions, using the `all of the following conditions are true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This separates those conditions that form part of the construct from any additional condition statements that might follow.

### Example

The following group of conditions tests that a customer is Gold junior member.

```
if
 all the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

**Parent topic:** [BAL constructs](#)

## any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

### Purpose

The construct `any of the following conditions are true` provides a more compact and convenient alternative to the logical operator `or` when you want to combine multiple conditions. The resulting statement is considered true if at least one of the listed conditions is true.

### Syntax

```
any of the following conditions is true:
- <condition>* [,]
```

### Description

This is equivalent to writing `if <condition 1> or <condition 2> or ... <condition n>`. However, if you have a large number of conditions, using the `any of the following conditions is true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

#### Note:

Indentation in rules is for readability only; it does not influence the way the rule is processed.

### Example

This example shows how to test if a customer is a Gold member or a senior customer.

```
if
 any of the following conditions is true:
 - the category of the customer is Gold
 - the age of the customer is more than 60
then...
```

The following more complex example shows the use of a comma to mark the end of a group of conditions. In this example, the order must be over \$50, the customer must be either a Gold customer or a senior customer, and the order must be shipping to NJ.

```
if
 all of the following conditions are true:
 - the order total is greater than $50
 - any of the following conditions is true:
 - the category of the customer is Gold
 - the age of the customer is more than 60,
 - the shipping address is in NJ
then...
```

Without the comma to mark the end of the `any of the following conditions are true` block, the final condition (the shipping address is in NJ) would be considered part of the previous group, and the rule would be interpreted to mean that the order must be over \$50 and (either the customer is a Gold customer or a senior customer or the shipping address is in NJ).

**Parent topic:** [BAL constructs](#)

## called (variable)

This construct provides a name for a variable declared in a `for each` action statement.

### Purpose

You use this construct in the outer loop of nested `for each` loops so that you can reference the named item from within the inner loop.

### Syntax

```
called <variable>,
```

### Description

When using a `for each` statement to perform an action on a set of items, a variable is created automatically to allow you to refer to each item in turn as 'this item' (see the first example below). However, if you nest one `for each` loop inside another, you must give the variable in the outer loop a name so that you can reference it from within the inner loop (see the second example below).

### Example

The following rule shows a simple `for each` action statement. There is no need to use the `called` construct in this case.

```
if
 the category of the customer is Gold
then
 for each customer in 'senior customers':
 - send flowers to this customer;
```

The following rule shows two nested `for each` statements. The `called` construct is used so that each customer can be explicitly referred to from within the nested `for each` loop.

```
if
 the category of the customer is Gold
then
 for each customer called c, in 'senior customers':
 - for each account in 'new accounts':
 - send a statement for this account to the mailing address of c;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[else](#)

[then](#)

[for each \(object\) in \(list\)](#)

# definitions

This construct introduces the part of the rule where you define variables.

## Purpose

You use this construct to define local variables that you can use elsewhere in the same rule. Use local variables to produce more concise and readable rules.

## Syntax

```
definitions
 set <variable> to <definition> [in <list> / from <object>] [where <test>*]
;*
```

## Description

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules. You define local variables using the [set \(variable\) to \(definition\)](#) construct.

You can also define global input/output variables (called ruleset parameters) at the application or rule project level. Ruleset parameters are accessible to any of the rules in your project. Ruleset parameters are created and edited in Rule Designer.

## Example

The following example declares the local variable 'preferred customer' in the definitions part of the rule and uses it in the if and then parts of the rule.

```
definitions
 set 'preferred customer' to a customer;
if
 the age of 'preferred customer' is less than 18
then
 apply a 10% discount to the shopping cart of 'preferred customer';
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[from \(object\)](#)

[in \(list\)](#)

[set \(variable\) to \(definition\)](#)

[the number of \(objects\)](#)

[where \(test\)](#)

## else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

### Purpose

You use this construct to declare actions that are executed if the `if` part of a rule has not been satisfied. These actions are ignored if all local variables are not correctly initialized due to preconditions in the definitions part failing to be satisfied.

### Syntax

```
else
 <action>;*
```

### Description

The `else` part of a rule is optional and allows you to specify one or more actions to perform if the conditions in the `if` part of the rule are not met.

The actions in the `else` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

If one or more variables with preconditions are defined in the definitions part of a rule, the rule will only be processed if these preconditions are satisfied and all local variables are correctly initialized. This means that if these preconditions are not satisfied, the rule is not be processed at all, and the actions in the `else` part of the rule are never executed, whether or not the conditions in the `if` part of the rule are met.

Adding preconditions in the definitions part of a rule rather than in the `if` part of a rule can provide a small performance improvement, since it potentially reduces the number of conditions to be checked. However, since the logic of the rule is not the same in each case, the method to use depends on the result you want to obtain.

### Example

In the following basic example, a customer will get a 10% discount if the customer is in the Gold category. Otherwise, the customer gets a 5% discount.

```
definitions
 set 'valued customer' to a customer
if
 the category of 'valued customer' is Gold
then
 apply a 10% discount;
else
 apply a 5% discount;
```

In the following example, a second condition has been added in the `if` part of the rule. Now, a customer receives a 10% discount only if the customer is in the Gold category and is aged over 20. All other customers receive a 5% discount.

```
definitions
 set 'valued customer' to a customer
if
 the category of 'valued customer' is Gold and 'valued customer' is older
 than 20
then
 apply a 10% discount;
else
 apply a 5% discount;
```

Now consider a third example. In the following rule, a minimum age requirement is added as a pre-condition in the definitions part of the rule. This means that a customer must first be over 20 years old to be considered a valued customer. Otherwise, the 'valued customer' variable is not initialized and the rest of the rule is not executed. With this rule, a customer under 20 receives no discount at all. A customer who is over 20 and in the Gold category receives a 10% discount. The `else` part of the rule is executed and a 5% discount applied only if the customer is over 20 but not in the Gold category.

```
definitions
 set 'valued customer' to a customer
 where this customer is older than 20;
```

```
if
 the category of 'valued customer' is Gold
then
 apply a 10% discount;
else
 apply a 5% discount;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**  
[then](#)  
[for each \(object\) in \(list\)](#)  
[set \(variable\) to \(value\)](#)

## for each (object) in (list)

This construct specifies the actions that must be performed for every element of a collection.

### Purpose

You use this construct to apply one or more actions to all objects in a collection.

### Syntax

```
for each <object> [called <variable>,) in <list>:
 - <action>*;
```

### Description

The construct should be immediately followed by a colon (:) and a list of one or more actions, each on a separate line, each preceded by a dash. End each action statement with a semi-colon (;).

The implicit variable 'this <object>' can be used in each action statement within the for each loop to refer to the current object. If you nest a for each statement inside another, you can use the called construct to define an explicit name for the object in the 'outer' for each loop in order to reference it clearly from within the nested for each loop.

### Example

The following rule shows how to apply a discount to all the expensive items in a customer's shopping cart, if the customer is a Gold customer.

```
definitions
 set 'smith' to a customer ;
 set 'expensive items' to all items
 in the items of the shopping cart of smith
 where the price of 'each item' is greater than 1000;
if
 the category of smith is Gold
then
 for each item in 'expensive items':
 - apply 5% discount to this item;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[else](#)

[then](#)

[called \(variable\)](#)

## from (object)

This construct accesses data from objects that are related to known objects.

### Purpose

You use this construct to expand the set of data that can be used by a rule.

### Syntax

```
from <object>
```

### Description

By default, rules are used to process a set of data established by the developer of an application. If you need to access data that is not in this set, but related to a data object in this set, you can use the `from` construct to define a variable that pulls a data object in from a related object.

The `from` construct is used to retrieve one distinct object from another one. You cannot use it to retrieve a collection of objects. To access a collection of objects, use the [in \(list\)](#) construct.

### Example

The following rule declares a variable based on the relationship between a customer and the customer's preferred item.

```
definitions
 set 'preferred CD' to a CD
 from the preferred item of the customer;
if
 the price of 'preferred CD' is more than 25
then...
```

By default, the rule engine does not know what a "CD" object is. To write a rule that uses a CD object, you must first declare a variable of type CD that pulls in data from the preferred item object of the customer. This is possible because the customer object (and therefore the customer's preferred item) is already known to the rule engine.

**Parent topic:** [BAL constructs](#)

**Related reference:**

[if](#)  
[where \(test\)](#)  
[definitions](#)



## if

This construct introduces the part of the rule in which you define conditions.

### Purpose

You use this construct in a rule to define one or more condition statements.

### Syntax

```
if
 <condition>*
```

### Description

If the conditions in the `if` part are met, the actions in the `then` part of the rule are executed. The `if` part of a rule is optional. If there is no `if` part, all actions in the `then` part of the rule are performed each time the rule is executed.

If the conditions in the `if` part of the rule are not met, the actions in the `else` part of the rule are executed. If the conditions are not met and the rule does not have an `else` part, the rule does nothing.

### Example

The following rule shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if
 the age of the customer is less than 18
then
 apply 10% discount to the shopping cart of the customer
```

**Parent topic:** [BAL constructs](#)

#### Related reference:

[from \(object\)](#)

[in \(list\)](#)

[the number of \(objects\)](#)

[where \(test\)](#)

[there are \(number\) \(objects\)](#)

[there is at most one \(object\)](#)

[there is no \(object\)](#)

[there is one \(object\)](#)

## in (list)

This construct accesses data from object collections that are related to known objects.

### Purpose

You use this construct to expand the set of data that can be used by a rule.

### Syntax

```
in <list>
```

### Description

By default, rules are used to process a set of data established by the developer of an application. If you need to access data that is not in this set, but related to data in this set, you can use the `in` construct to define a variable that pulls data in from a related data collection.

The `in` construct is used to retrieve data from a collection of objects related to an existing object. The `<list>` parameter of the `in` construct must be an expression that denotes a collection. To retrieve one distinct object from another distinct object, use the [from \(object\)](#) construct.

The `in` construct can be used in the definitions part of a rule and in count conditions specified in the `if` part of a rule.

### Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
 set 'item' to a CD
 in the items of the shopping cart of the customer ;
if
 there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
 there is an item
 in the items of the shopping cart of the customer ;
then...
```

**Parent topic:** [BAL constructs](#)

#### Related reference:

[if](#)  
[where \(test\)](#)  
[definitions](#)  
[there are \(number\) \(objects\)](#)  
[there is at most one \(object\)](#)  
[there is no \(object\)](#)  
[there is one \(object\)](#)

## it is not true that (a condition)

This construct negates a condition statement.

### Purpose

You use this construct when there is no sentence in the vocabulary or no operator to express the opposite of a condition.

### Syntax

```
it is not true that <condition>
```

### Description

When there is no sentence in the vocabulary or no operator to express the opposite of a condition, you can insert the `it is not true that` construct in front of an existing condition to negate it.

This can be useful in cases where a boolean (true/false) attribute defined in your vocabulary is verbalized as an expression such as 'customer is a loyalty member', but where you want to test for the opposite of this expression ('customer is not a loyalty member'). If the opposite expression has not been defined in your vocabulary, you could write 'it is not true that customer is a loyalty member' to achieve the required result.

In general, the logic of a negative expression is more difficult to interpret, so it is advisable to avoid them where possible.

### Example

The following example uses the `it is not true that` construct to check the age of the customer, assuming that the boolean property 'is a minor' has been verbalized in the vocabulary, but that the opposite of the expression 'is a major' has not been defined.

```
if
 it is not true that the Customer is a minor
```

The following example shows how to negate a set of conditions that have been grouped using the `all of the following conditions are true` construct. The resulting expression will return True except if the customer is both a gold member and over 60.

```
if
 it is not true that all of the following conditions are true:
 - the category of the Customer is gold
 - the age of the customer is more than 60
```

**Parent topic:** [BAL constructs](#)

## none of the following conditions are true

This construct negates a group of conditions.

### Purpose

You use this construct to groups together a series of condition statements that you want to negate.

### Syntax

```
none of the following conditions are true:
 - <condition>* [,]
```

### Description

The group evaluates to true only if none of the conditions listed are true.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

### Example

The following group of conditions tests that a customer is not a Gold senior member and not under 60.

```
if
 none of the following conditions are true:
 - the category of the customer is gold
 - the age of the customer is least 60
then...
```

**Parent topic:** [BAL constructs](#)

## print (string)

This construct prints a string, phrase or rule name to the standard output.

### Purpose

You use this construct to define an action phrase that by default outputs a string to the standard output.

### Syntax

```
print "<string>;
```

### Description

You can modify this behavior by setting a note handler on the note method of the rule engine.

### Example

The following action prints the string "Hello World!" to standard output.

```
print "Hello World!" ;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**  
[the name of this rule](#)  
[the number of \(objects\)](#)

## set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

### Purpose

You use this construct in the definitions part of a rule to declare a local variable.

### Syntax

```
set '<variable>' to <definition> [in <list> / from <object>] [where <test>*] ;
```

### Description

Variables produce more concise rules by replacing a value or the result of an expression with a short, convenient identifier. A local variable can be used anywhere within the rule in which it is defined, but is not available in other rules.

Enclose the name of each variable in single quotes. This is compulsory for variable names that contain spaces. While it is not essential to enclose one-word variable names in single quotes, it makes them easier to identify and to reduce the risk of confusion.

### Example

The following examples show how to define a variable as an object.

```
definitions
 set 'i' to an item;
 set 'house' to a house
 where the price of this house is more than 1000;
```

The following examples show how to define a variable as a literal, string, or collection.

```
definitions
 set 'category' to Gold ;
 set 's' to "a string";
 set 'expensive items' to all items
 in the items of the shopping cart of the customer
 where the price of each item is more than 200;
```

The following example shows how to define a variable as the result of an expression.

```
definitions
 set 'expr' to the price of the car of the customer + 100;
 set 'h2' to the house of the customer
 where the price of this house is more than 1000;
```

The following example shows how to define a variable as another variable.

```
definitions
 set 'expr2' to 'expr';
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[from \(object\)](#)  
[in \(list\)](#)  
[where \(test\)](#)  
[definitions](#)

## set (variable) to (value)

This construct changes the value of a ruleset parameter or variable.

### Purpose

You use this construct to modify the value of a ruleset parameter or variable in the action part of a rule.

### Syntax

```
set <variable> to <value>;
```

### Description

The variable can be set to a literal value, an expression, or a list.

### Example

The following action changes the value of a ruleset parameter to the age of a customer.

```
then
 set param1 to the age of the customer;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[else](#)  
[then](#)

## the name of this rule

This construct returns the name of the currently executed rule in the then part of a business rule.

### Purpose

You use this construct to provide an audit trail, for example, by logging a message that quotes the name of the rule whenever it is executed.

### Syntax

```
the name of this rule
```

### Description

This construct displays the name of the current rule as a string. It can be useful as a means of providing an audit trail. Each time a rule is executed, you can log a message that this rule was executed.

The variable is of type String and contains the name of this rule. It is only available in the action part of the rule.

### Example

The following action would return the name of the executed rule, Retired\_Customer\_Policy for example.

```
if
 the age of the customer is greater than 65
then
 print the message "A 5% discount has been applied to policy: " + the name of this rule;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[else](#)

[then](#)

[for each \(object\) in \(list\)](#)



## the number of (objects)

This construct returns the number of objects in the current data set or in the specified collection.

### Purpose

You use this construct to count the number of objects of the specified type and return the total as a number.

### Syntax

```
the number of <objects> [in <list>] [where <test>,*]
```

### Description

You can use the optional `in <list>` clause to count the objects in a specific list. You can use one or more optional `where` clauses to apply additional conditions to the objects that are included in the returned count.

This construct cannot be used in the definitions part of the rule.

### Example

The following condition is met if there are at least 6 vehicles:

```
if
 the number of vehicles is more than 5
then...
```

**Parent topic:** [BAL constructs](#)

#### Related reference:

[if](#)  
[where \(test\)](#)  
[in \(list\)](#)  
[definitions](#)

## then

This mandatory construct introduces the part of the rule that defines the actions that are executed if the `if` part of a rule is satisfied.

### Purpose

You use this construct to introduce one or more actions to be executed if the conditions of the rule are met.

### Syntax

```
then
 <action>;*
```

### Description

Every rule must have a `then` part. If a rule does not have an `if` part, the actions in the `then` part of the rule are always executed whenever the rule is executed.

The actions in the `then` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

### Example

The following example shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if
 the age of the customer is less than 18
then
 apply 10% discount to the shopping cart of the customer;
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[else](#)

[for each \(object\) in \(list\)](#)

[set \(variable\) to \(value\)](#)

## there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the current data set or the specified collection.

### Purpose

You use this construct to determine whether the current data set contains exactly the specified number of occurrences of a particular object.

### Syntax

```
there are <number> <objects> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests if the number of Gold customers is equal to 8.

```
if
 there are 8 customers
 where the category of each customer is Gold
then...
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[if](#)  
[where \(test\)](#)  
[in \(list\)](#)

## there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the current data set or the specified collection.

### Purpose

You use this construct to determine whether the current data set contains at least the specified number of occurrences of a particular object.

### Syntax

```
there are at least <number> <objects> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if there are at least 10 gold customers.

```
if
 there are at least 10 customers
 where the category of each customer is gold
then...
```

**Parent topic:** [BAL constructs](#)

# there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in the current data set.

## Purpose

You use this construct to determine whether the current data set or the specified collection contains no more than the specified number of occurrences of a particular object.

## Syntax

```
there are at most <number> <objects> [in <list>] [where <test>,*]
```

## Description

Use the optional in clause to apply the test to a specific collection of objects. Use one or more optional where clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the if part of a rule.

**Note:**  
  
The <number> argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

## Example

The following condition tests if the number of Gold customers is lower than or equal to 3.

```
if
 there are at most 3 customers
 where the category of each customer is Gold
then...
```

Parent topic: [BAL constructs](#)

## there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains fewer than the specified number of occurrences of a particular object.

### Syntax

```
there are less than <number> <objects> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if the number of Gold customers is lower than 3.

```
if
 there are less than 3 customers
 where the category of each customer is Gold
then...
```

**Parent topic:** [BAL constructs](#)

## there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains more than the specified number of occurrences of a particular object.

### Syntax

```
there are more than <number> <objects> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if there are more than 10 customers with the gold category.

```
if
 there are more than 10 customers
 where the category of each customer is Gold,
then...
```

**Parent topic:** [BAL constructs](#)

## there is at least one (object)

This construct tests whether there is at least one object of a given type in the current data set.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains at least one occurrence of a particular object.

### Syntax

```
there is at least one <object> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests whether or not a Customer object exists.

```
if
 there is at least one customer
 where...
then...
```

The following condition tests if a senior Gold customer exists.

```
definitions
 set 'gold customers' to all customers
 where the category of each customer is Gold;
if
 there is at least one customer in 'gold customers'
 where the age of this customer is at least 60,
then...
```

**Parent topic:** [BAL constructs](#)



## there is at most one (object)

This construct tests whether there is at most one object of a given type in the current data set.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains zero or one occurrence of a particular object.

### Syntax

```
there is at most one <object> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests if there is at most one Gold customer in the set of objects provided to the rule engine for execution.

```
if
 there is at most one customer
 where the category of this customer is Gold,
then...
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[if](#)  
[where \(test\)](#)  
[in \(list\)](#)

## there is no (object)

This construct tests whether a data set contains no objects of the given type.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains no occurrences of a particular object.

### Syntax

```
there is no <object> [in <list>] [where <tests>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests that there is no object of type `Customer` in the object set provided to the rule engine for execution.

```
if
 there is no customer where...
then...
```

**Parent topic:** [BAL constructs](#)

**Related reference:**

[if](#)  
[where \(test\)](#)  
[in \(list\)](#)

## there is one (object)

This construct tests whether the current data set contains one and only one object of a given type.

### Purpose

You use this construct to determine whether the current data set or the specified collection contains one and only one occurrence of a particular object.

### Syntax

```
there is one <object> [in <list>] [where <test>,*]
```

### Description

Use the optional `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests if there is one Gold customer in the set of objects provided to the rule engine for execution.

```
if
 there is one customer
 where the category of this customer is Gold
then...
```

**Parent topic:** [BAL constructs](#)

#### Related reference:

[if](#)  
[where \(test\)](#)  
[in \(list\)](#)

## where (test)

This construct matches objects against tests.

### Purpose

Matches objects against tests.

### Syntax

```
where <test>[,]*
```

### Description

In the definitions part of a rule, you can use one or more where clauses with the set construct to test that an object meets one or more conditions in order to be a valid value for the variable being declared. In the definitions part of the rule, the where statement does not end with a comma. Instead, each set construct should end with a semi-colon (;).

In the if part of a rule, you can use one or more where clauses in count conditions (there is more than, there is less than, and so on) to test that an object meets one or more conditions before being counted.

In a where construct, an implicit variable that refers to the current instance of the object being tested is automatically declared. In the test, you can thus refer to potential instances of the variable as this <object type> (or each <object type> if they are part of a collection.)

### Example

The following example shows the definitions part of a rule that defines the local variable 'expensive items' as a collection of items whose price is greater than 100. The implicit variable each item is used in the where clause to refer to each item object in the collection.

```
definitions
 set 'expensive items' to all items
 where the price of each item is more than 100;
```

The following example shows the where clause used together with the there is one construct in the if part of a rule to test if there is a customer older than 100. The implicit variable this customer is used in the where clause to refer to the customer object being tested.

```
if
 there is one customer
 where the age of this customer is more than 100,
```

**Parent topic:** [BAL constructs](#)

#### Related reference:

[if](#)  
[from \(object\)](#)  
[in \(list\)](#)  
[set \(variable\) to \(definition\)](#)  
[definitions](#)  
[there are \(number\) \(objects\)](#)  
[there is no \(object\)](#)  
[there is one \(object\)](#)

# BAL operators

You use operators in rule statements to do arithmetic operations, associate or negate conditions, and compare expressions.

## Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

## Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

## Date operators

Date operators define conditions based on dates.

## Number operators

Number operators define conditions based on numbers.

## Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

## Text operators

Text operators define conditions based on strings.

**Parent topic:** [Business Action Language \(BAL\)](#)

### **Related information:**

[Inserting an arithmetic expression](#)

[Controlling how condition statements are combined](#)

[Decision trees](#)

[Working with action rules](#)

# Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

Table 1. Arithmetic operators

Operator	Description	Example
- <number >	Makes a number negative.	<pre>if   there is more than one error then   set 'status code' to - the   number of errors;</pre>
<number > + <number >	Adds a number to another number.	<pre>if   there is at least one additional   driver then   set the rental cost of the car   to the base rental cost of the car   + 100;</pre>
<number > - <number >	Subtracts a number from another number.	<pre>if   the number of 'items purchased'   - the number of 'items returned' is   more than 0 then...</pre>
<number > / <number >	Divides a number by another number.  If both numbers are integers, the result is also an integer (the result of the division without the remainder.)	<pre>if   there is at least one order then   set the average order quantity to   the number of items ordered / the   number of orders;</pre>
<number > * <number >	Multiplies a number by another number.	<pre>if   the price of the item * 'sales   tax' is more than 100 then...</pre>
<text > + <text >	Concatenates two strings.	<pre>definitions   set 'full name' to 'first name'   + ' ' + 'last name'</pre>
<number > + <text >	Concatenates a number and a string. The result is treated as a string.	<pre>if   the order total is less than   'discount threshold' then   display the message: "Spend just   \$" + ('discount threshold' - order   total) + " more to receive a 10%   discount!";</pre>
<text > +	Concatenates a string and a number. The result is treated as	<pre>if</pre>

<code>&lt;number&gt;</code>	a string.	<pre>if     the order total is more than     'discount threshold' then     apply a 10% discount;     display the message: "You     received a 10% discount because     your order was over \$" + 'discount     threshold' ;</pre>
-----------------------------	-----------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Parent topic: [BAL operators](#)

# Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Table 1. Logical operators

Operator	Description	Example
<condition > and <condition >	Associates two conditions such that the resulting expression is only true if both conditions are true.	<pre>if   the category of the customer is Gold   and the age of the customer is at least 60 then...</pre>
<condition > or <condition >	Associates two conditions such that the resulting expression is true if either (or both) conditions are true.	<pre>if   the category of the customer is Gold   or the age of the customer is at least 60 then...</pre>

Parent topic: [BAL operators](#)



# Date operators

Date operators define conditions based on dates.

Table 1. Date operators

Operator	Description	Example
<a day of week> is after <a day of week>	Tests that a day of the week comes after another day of the week. <div><b>Note:</b> Sunday is considered to be the first day of the week.</div>	<pre>if   the day of the return date of 'rented car'   is after Friday then...</pre>
<a day of week> is before <a day of week>	Tests that a day of the week comes before another day of the week. <div><b>Note:</b> Sunday is considered to be the first day of the week.</div>	<pre>if   the day of the return date of 'rented car'   is before Friday then...</pre>
<date> is after <date>	Tests that a date comes after another date.	<pre>if   the return date of 'rented car' is after 05/07/2007 05:00:00 PM then...</pre>
<date> is after <date> and on or before <date>	Tests that a date is in a range in which the first date is excluded, and the last date is included.	<pre>if   the return date of 'rented car' is after 'pickup date' and on or before 'scheduled return date' then...</pre>
<date> is after <date> and before <date>	Tests that a date is in a range in which both dates are excluded.	<pre>if   the return date of 'rented car' is after 'pickup date' and before 'scheduled return date' then...</pre>
<date> is after or the same as <date>	Tests that a date comes after another date, or is the same as that other date.	<pre>if   the return date of 'rented car'   is after or the same as 05/07/2007 05:00:00 PM then...</pre>
<date> is at <time>	Tests that the time part of a date is at the specified time.	<pre>if   the return date of 'rented car' is at 05:00:00 PM then...</pre>
<date> is before <date>	Tests that a date comes before another date.	<pre>if   the return date of 'rented car' is before 05/07/2007</pre>

		car' is before 05/07/2007 05:00:00 PM then...
<date> is before or the same as <date>	Tests that a date comes before another date, or is the same as that other date.	if the return date of 'rented car' is before or the same as 05/07/2007 05:00:00 PM then...
<date> is in <month>	Tests that a date is in a specific month.	if the return date of 'rented car' is in October then...
<date> is in <month> the year <year>	Tests that a date is in the specified month of the specified year.	if the return date of 'rented car' is in October the year 2007 then...
<date> is in the year <year>	Tests that a date is in the specified year.	if the return date of 'rented car' is in the year 2007 then...
<date> is on <a day of week>	Tests that a date is on a specific day of the week.	if the return date of 'rented car' is on Thursday then...
<date> is on <simple date>	Tests that a date is on a specific date, specified without a time.	if the return date of 'rented car' is on 05/07/2007 then...
<date> is on <day of week> at <time>	Tests that a date is on a specific day of the week, at a specific time.	if the return date of 'rented car' is on Thursday at 05:00:00 PM then...
<date> is on or after <date> and before <date>	Tests that a date is in a range in which the first date is included, and the last date is excluded.	if the return date of 'rented car' is on or after 'pickup date' and before 'scheduled return date' then...
<date> is on or after <date>	Tests that a date is in a range in which both dates	

after <date> and on or before <date>	range in which both dates are included.	if the return date of 'rented car' is on or after 'pickup date' and on or before 'scheduled return date' then...
--------------------------------------------	--------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------

Parent topic: [BAL operators](#)

# Number operators

Number operators define conditions based on numbers.

Table 1. Number operators

Operator	Description	Example
<number> does not equal <number>	Tests that a number is not equal to another number.	<pre>if     the number of rentals does not equal 3 then...</pre>
<number> equals <number>	Tests that a number is equal to another number. This is equivalent to the is <number> operator	<pre>if     the number of rentals equals 3 then...</pre>
<number> is <number>	Tests that a number is equal to another number. This is equivalent to the equals <number> operator	<pre>if     the number of rentals is 3 then...</pre>
<number> is at least <number>	Tests that a number is greater than or equal to another number.	<pre>if     the number of rentals is at least 3 then...</pre>
<number> is at least <number> and less than <number>	Tests that a number is included in a range in which the first value is included and the last value is excluded.	<pre>if     the age of the customer is at least 12 and less than 25 then...</pre>
<number> is at most <number>	Tests that a number is less than or equal to another number.	<pre>if     the number of rentals is at most 3 then...</pre>
<number> is between <number> and <number>	Tests that a number is included in a range in which both values are included.	<pre>if     the age of the customer is between 12 and 25 then...</pre>
<number> is less than <number>	Tests that one number is less than another.	<pre>if     the number of rentals is less than 3  then...</pre>

<number> is more than <number>	Tests that one number is greater than another.	if the number of rentals is more than 3 then...
<number> is more than <number> and at most <number>	Tests that a number is included in a range in which the first value is excluded and the last value is included.	if the age of the customer is more than 12 and at most 25 then...
<number> is strictly between <number> and <number>	Tests that a number is included in a range in which the first value is excluded and the last value is excluded.	if the age of the customer is strictly between 12 and 25 then...

Parent topic: [BAL operators](#)

# Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Table 1. Object operators

Operator	Description	Example
<object> is <object>	Tests that two objects are equivalent.	<pre>if     the current customer is the customer on the rental agreement then...</pre>
<object> is not <object>	Tests that two objects are not equivalent.	<pre>if     the current customer is not the customer on the rental agreement then...</pre>
<object> is one of <list>	Tests that an object is part of a set.	<pre>if     the item of the order is one of 'discounted items' then...</pre>
<object> is not one of <list>	Tests that an object is not part of a collection.	<pre>if     the customer category is not one of 'all categories' then...</pre>
<list> contain <object>	Tests that a collection contains an object. This operator is functionally equivalent to <object> is one of <list>.	<pre>if     the customer categories contain Platinum then...     apply a 10% discount;</pre>
<list> do not contain <object>	Tests that a collection does not contain an object. This operator is functionally equivalent to <object> is not one of <list>.	<pre>if     the customer categories do not contain Normal then...     apply a 10% discount;</pre>

Parent topic: [BAL operators](#)

# Text operators

Text operators define conditions based on strings.

Table 1. Text Operators in BAL

Operator	Description	Example
<text> contains <text>	Tests that a string contains another string.	<pre>if     the name of the customer contains "Smith" then...</pre>
<text> does not contain <text>	Tests that a string does not contain another string.	<pre>if     the name of the customer does not contain "Smith" then...</pre>
<text> does not end with <text>	Tests that a string does not end with another string.	<pre>if     the rental agreement code of the customer does not end with "XG5" then...</pre>
<text> does not start with <text>	Tests that a string does not start with another string.	<pre>if     the rental agreement code of the customer does not start with "XG5" then...</pre>
<text> ends with <text>	Tests that a string ends with another string.	<pre>if     the rental agreement code of the customer ends with "XG5" then...</pre>
<text> is empty	Tests that a string is empty. An empty string contains no characters and is equivalent to "". A string is not empty if it contains a space (" ")	<pre>if     the rental agreement code of the customer is empty then...</pre>
<text> is not empty	Tests that a string is not empty. A string is not empty if it contains a space (" ")	<pre>if     the rental agreement code of the customer is not empty then...</pre>
print <text>	Prints a string to standard out.	<pre>if</pre>

		<pre>11.... then     print the message     "Hello World"</pre>
<p>&lt;text&gt; starts with &lt;text&gt;</p>	Tests that a string starts with another string.	<pre>if     the rental     agreement code of the     customer starts with     "XG5" then...</pre>
<p>the length of &lt;text&gt;</p>	Returns the number of characters in a string.	<pre>if     the length of     'customer name' is     more than 3 then...</pre>

Parent topic: [BAL operators](#)



# BAL literals

The Business Action Language (BAL) supports number, string, date, and time data types.

## Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

## Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

## Dates and times

You can use the Date, Simple Date, and Universal Date value types to express date values. You can use the Time and Universal Time value types to express time values.

**Parent topic:** [Business Action Language \(BAL\)](#)

### **Related information:**

[Overview: Ways to express business rules](#)

[Decision trees](#)

[Working with action rules](#)

[Working with decision tables](#)

[Working with decision trees](#)

# Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

- 10 (integer)
- 10.5 (decimal, with decimal point separator)
- 150,500.51 (decimal, with '000 grouping separator and decimal point separator)
- -10 (negative integer)
- 10E-5 (exponent)

The lexical representation of numbers (the decimal point separator and the grouping separator) is locale-specific.

**Note:**

Big number literals are not supported in the business rule language. All literals manipulated in a rule must be within the range of the Java™ Double class, otherwise they are rounded to the closest Double value. The internal representation of numbers corresponds to Java doubles, and precision can be lost in conversion everywhere a number is passed as a parameter to a BOM method expecting a narrow type. This also happens when assigning an expression to a variable or a field. The IRL view of the rule editor allows you to see exactly where these conversions – represented by C-style casts – take place.

**Parent topic:** [BAL literals](#)

# Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

To include certain special characters within a String, you must prefix them with a backslash (\). For example, to include a double quotation mark within a String, you would write \". The backslash is required here to indicate that the String has not yet ended.

The following special character sequences are accepted within text strings:

- \n (new line character)
- \t (tab character)
- \b
- \r (carriage return character)
- \f (line feed character)
- \' (single quotation mark)
- \" (double quotation mark)
- \\ (backslash)

**Parent topic:** [BAL literals](#)

# Dates and times

You can use the Date, Simple Date, and Universal Date value types to express date values. You can use the Time and Universal Time value types to express time values.

Several types of date and time are available.

In the following table, a is the symbol for AM or PM, and z is the time zone information, for example +0200.

Date	Standard date format that includes time, and is written as:  m/d/y h:mm:ss a
Simple Date	Simple date format that does not include time, and is written as:  m/d/y
Time	Time format that is written as either:  h:mm:ss a  h:mm a
Universal Time	Time format that includes the time zone, and is written as either: h:mm:ss a z  h:mm a z
Universal Date	Date and time format that includes the time zone, and is written as:  m/d/y h:mm:ss a z

**Note:**

The business rule language supports only the Gregorian calendar format.

**Parent topic:** [BAL literals](#)

## Punctuation in rules

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules (for example, in the Intellirule editor), mandatory punctuation is usually predicted in the completion menu.

### Backslash \

The backslash is used to prefix special characters in text strings. For further details, see [Strings](#).

### Comma ,

The comma marks the end of Where statements.

#### Example

This example shows a Where statement that is closed by a comma.

```
if
 there is at least one car
 where
 the color of this car is blue,
then ...
```

Some language constructs may be terminated by an optional comma.

#### Example

This example shows a construct that is terminated by an optional comma. This may prevent the construct becoming ambiguous in a particular context.

```
all following conditions are true:
- the color of the car is blue
- the price of the car is more than 1000,
```

### Double quotation mark "

Double quotation marks are used to enclose string literals; that is, text strings. Avoid using them in the verbalization of BOM classes and members.

#### Example

This example shows an action phrase in which the message to be displayed is a text string enclosed in double quotation marks.

```
then
 apply a 10% discount;
 display the message: "You received a discount!";
```

### Parentheses ( )

Parentheses can be used to group any expression. They can help clarify the default precedence of logical operators linking conditions to one another. You can also use parentheses to regroup condition statements and hence change the order in which they are interpreted.

#### Example

This example shows nested parentheses in an action phrase.

```
then
 print "the customer: " + (the name of the customer of (the shopping cart))
```

### Semicolon ;

The semicolon marks the end of variable definitions and action phrases.

### Single quotation mark '

Single quotation marks are used to enclose variables. Single quotation marks are optional for single-word variables and mandatory for variables that consist of several words. Automatic variables are not enclosed in single quotation marks. Because single quotation marks are frequently used in the rule language, avoid using them in the verbalization of BOM classes and members.

## Example

In this example, the variables `customer` and `the cart` are defined. The `customer` variable does not need to be enclosed in single quotation marks because it is a single word, whereas `the cart` needs to be delimited because it consists of two words.

*definitions*

```
set customer to a customer;
set 'the cart' to the shopping cart of customer;
```

**Parent topic:** [Business Action Language \(BAL\)](#)

# ILOG Rule Language (IRL)

The ILOG® Rule Language (IRL) contains a set of keywords, and has its own syntax to structure each part of the rule. IRL does not support generics because IRL is not Java™. As a result you cannot use generics in: IRL, functions, BOM to XOM mapping, and initial/final actions.

## [IRL keywords](#)

The ILOG Rule Language defines IRL keywords to support rule authoring, management, and execution.

## [IRL grammar](#)

The ILOG Rule Language (IRL) grammar conforms to a specific notation and has naming restrictions. It is composed of several operators that are a subset of the operators provided in the Java programming language.

**Parent topic:** [Rule languages](#)

# IRL keywords

The ILOG® Rule Language defines IRL keywords to support rule authoring, management, and execution.

## **after**

The `after` keyword defines a binary temporal constraint in the condition part of a rule in an event condition.

## **agendafilter**

The `agendafilter` keyword defines an agenda filter in a rule task.

## **algorithm**

The `algorithm` keyword defines an execution mode.

## **allrules**

The `allrules` keyword sets all the rules in a rule task as eligible to be executed.

## **as**

The `as` keyword defines a type conversion.

## **before**

The `before` keyword defines a binary temporal constraint.

## **body (in flowtask)**

The `body` keyword defines the body of the ruleflow.

## **body (in action task)**

The `body` keyword defines the body of an action task.

## **body (in ruletask)**

The `body` keyword defines the body of the rule task.

## **break**

The `break` keyword exits a loop.

## **case**

The `case` keyword identifies a statement block in a switch statement.

## **catch**

The `catch` keyword designates exceptions that are caught if thrown.

## **collect**

The `collect` keyword constructs a collection object.

## **continue**

The `continue` keyword skips an iteration.

## **default (in ruletask)**

This `default` keyword selects the RetePlus execution mode for a rule task.

## **default (in switch)**

This `default` keyword executes the `else` condition statement.

## **dynamic**

The `dynamic` keyword specifies the dynamic ordering of rules.

## **dynamicselect**

The `dynamicselect` keyword specifies the dynamic selection of a rule.

## **else (in if)**

In functions or rule actions, the `else` keyword executes an alternate statement block.

## **else (in rule)**

In rule definitions, the `else` keyword executes an alternate action block depending on the value of an `evaluate` statement.

## **evaluate**

The `evaluate` keyword specifies tests on objects.

## **event (in rule conditions)**

The `event` keyword declares an event condition or an event object.

## **event (in rule actions)**

The `event` keyword declares an event condition or an event object.

## **exists**

The `exists` keyword tests whether a class condition is true.

## **filter**

The `filter` keyword defines an agenda filter.



### **finalaction**

The finalaction keyword declares a final action.

### **finally**

The finally keyword introduces a control block that is executed after a try block.

### **firing**

The firing keyword determines whether all the rules are executed.

### **firinglimit**

The firinglimit keyword specifies the number of rules to be executed.

### **flowtask**

The flowtask keyword declares a subflow task.

### **for**

The for keyword introduces an execution loop.

### **foreach**

The foreach keyword executes a statement block several times.

### **fork**

The fork keyword executes several statement blocks sequentially.

### **from**

The from keyword supports relations between objects.

### **function**

The function keyword defines a function to be called.

### **functiontask**

The functiontask keyword declares an action task.

### **goto**

The goto keyword jumps to another ruleflow statement.

### **hasher**

The hasher keyword defines a hashing expression.

### **hierarchy**

The hierarchy keyword defines a hierarchical property.

### **if**

The if keyword executes one of two statement blocks.

### **if (in ruleflow)**

The if keyword executes one statement block or another, depending on a Boolean expression value.

### **import**

The import keyword imports a Java class or package into a rule file.

### **in**

The in keyword supports relations between object values.

### **in — !in (predicates)**

In any expression, the in keyword tests whether a value belongs to a collection and or the !in keyword tests whether a value does not belong to a collection.

### **in (ruleset parameter)**

The in keyword defines a ruleset parameter.

### **initialaction**

The initialaction keyword declares an initial action.

### **inout**

The inout keyword defines a ruleset parameter that is both an input parameter and an output parameter.

### **insert**

The insert keyword inserts an object into the working memory.

### **instanceof**

The instanceof keyword tests whether an expression can be cast into a type.

### **instances**

The instances keyword declares class instances.

### **isknown**

The isknown keyword tests whether a class attribute has an initialized value.

### **isunknown**

The `isunknown` keyword tests whether a class attribute has an initialized value.

#### **logical (in wait)**

In a `wait` statement, the `logical` keyword designates that conditions must remain true for a `wait` statement to become true.

#### **match**

The `match` keyword deals with hierarchical property values.

#### **modify**

The `modify` keyword modifies objects and updates the agenda.

#### **new**

The `new` keyword creates a new object or array.

#### **not**

The `not` keyword tests the negation of a rule condition.

#### **occursin**

The `occursin` keyword defines a unary temporal constraint in an event condition.

#### **ordering**

The `ordering` keyword sets the order of the rules executed in a rule task.

#### **out**

The `out` keyword defines a ruleset output parameter.

#### **overriding, overrides**

The `overriding` and `overrides` keywords declare overriding relations.

#### **package**

The `package` keyword declares a package.

#### **property**

The `property` keyword declares a rule, ruleset, or ruleflow property.

#### **propertydefinition**

The `propertydefinition` keyword predeclares the types of rule properties.

#### **refresh**

The `refresh` keyword requests a refresh of the agenda.

#### **retract**

The `retract` keyword removes an object from the working memory.

#### **return**

The `return` keyword returns to the caller.

#### **rule**

The `rule` keyword declares a rule.

#### **rule (in ruletask)**

The `rule` keyword sets a single rule as eligible for execution.

#### **ruleset**

The `ruleset` keyword declares a ruleset.

#### **ruletask**

The `ruletask` keyword declares a rule task.

#### **scope**

The `scope` keyword defines a scope in a rule task.

#### **select**

The `select` keyword selects a rule in a rule task.

#### **sequential**

The `sequential` keyword specifies the execution mode of a rule task.

#### **switch**

The `switch` keyword executes one statement block or another according to an integer expression value.

#### **then**

The `then` keyword declares the action part of a rule.

#### **throw**

The `throw` keyword throws an exception.

#### **timeof**

The `timeof` keyword accesses an event timestamp.

#### **timeout**

The timeout keyword is associated with a wait statement.

**try**

The try keyword executes control statements.

**until**

The until keyword specifies an absolute time.

**update**

The update keyword updates an object.

**use**

The use keyword imports an IRL package or artifact.

**variables**

The variables keyword declares variables.

**wait**

The wait keyword incorporates time as a rule parameter.

**when**

The when keyword declares the condition part of a rule.

**where**

The where keyword defines a constraint in a collect statement.

**while**

The while keyword executes a statement block.

**while/break/continue (in ruleflow)**

The while keyword executes a loop.

**Parent topic:** [ILOG Rule Language \(IRL\)](#)

## after

The after keyword defines a binary temporal constraint in the condition part of a rule in an event condition.

### Purpose

A binary temporal constraint in an event condition.

### Context

Rule conditions

### Syntax

```
[?var:] event className (?eventVar1 after [interval] ?eventVar2);
```

### Description

#### Deprecated as of V7.5.

The after keyword is used in the condition part of a rule in an event condition. You express a temporal constraint between two events with the after and before keywords. An interval is of the form [lowerBound, upperBound], where a bound is either an expression evaluating to an integer or the \$ sign which denotes infinity.

Considering that the timestamp of ?event <sub>1</sub> is t<sub>1</sub> and that the timestamp of ?event <sub>2</sub> is t<sub>2</sub>, you can express the after operator as follows:

- ?event <sub>2</sub> after[min,max] ?event <sub>1</sub>  
is satisfied if the value of t<sub>2</sub> - t<sub>1</sub> is between the values of min and max, inclusive.
- ?event <sub>2</sub> after[min,\$] ?event <sub>1</sub>  
is satisfied if the value of t<sub>2</sub> - t<sub>1</sub> is greater than, or equal to, the value of min.
- ?event <sub>2</sub> after[\$,max] ?event <sub>1</sub>  
is satisfied if the value of t<sub>2</sub> - t<sub>1</sub> is less than, or equal to, the value of max.
- ?event <sub>2</sub> after ?event <sub>1</sub>  
is satisfied if the value of t<sub>2</sub> - t<sub>1</sub> is positive or null.

### Example

This example assumes that a User instance is in the working memory and that an Alarm instance is inserted. A partial instance of the ReportAlarmPairsToUsers rule is created and maintained for four ticks. If the User instance is then retracted, the partial instance of the rule is immediately deleted. An instance of the ReportAlarmsToUsers rule is created on each instance of the User class in the working memory when a second alarm event ?a2 is inserted no later than four ticks after ?a1.

```
rule ReportAlarmPairsToUsers {
 when {
 ?u: User();
 ?a1: event Alarm();
 ?a2: event Alarm(?this after[1, 4] ?a1);
 }
 then {
 ?u.report(?a1, ?a2);
 }
};
```

**Parent topic:** [IRL keywords](#)

#### Related reference:

[before](#)  
[event \(in rule conditions\)](#)  
[event \(in rule actions\)](#)  
[occursin](#)  
[timeof](#)

## agendafilter

The `agendafilter` keyword defines an agenda filter in a rule task.

### Purpose

This agenda filter is used if the execution algorithm selected for the rule task is `RetePlus`.

### Context

Rule task definitions

### Syntax

```
(1) ruletask ruleTaskName
{
 agendafilter = filter (variableName)
 {
 action1
 ...
 actionn
 }
};

(2) ruletask ruleTaskName
{
 agendafilter = value;
};
```

### Description

You can write agenda filters in two different syntaxes:

#### Syntax (1)

You write the code of the agenda filter within the task definition. This code is similar to an IRL function with an [IlrRuleInstance](#) parameter and a Boolean return type. The *variableName* placeholder represents this `IlrRuleInstance` object. It can be referenced in the inline code of the agenda filter. Ruleset variables are accessible from the agenda filter code. This agenda filter is applied on each instance of the rules that compose the task body. Any instance for which the agenda filter returns `true` is executed. The other instances are not.

#### Syntax (2)

You write the agenda filter as an instance of a class that implements the [IlrAgendaFilter](#) interface. This filter is applied on each instance of the rules that compose the task body. The instance for which the agenda filter returns `true` is executed. The other instances are not.

### Example

These two examples illustrate the use of the `agendafilter` keyword.

#### Example 1

This example defines a rule task `ruletask1` in which the body is composed of all the rules of the ruleset. The agenda filter defined on the rule task filters the rule instances contained in that task. Only the rule instances whose names are contained in the input parameter of the ruleset are executed. The others are not.

```
ruletask ruletask1
{
 body = select(?rule) {return true;}
 agendafilter = filter(?instance)
 {
 String name = ?instance.ruleName;
 return inputParam.contains(name);
 }
};
```

#### Example 2

This example defines another rule task in which the body is composed of all the rules of the ruleset. The

agenda filter defined on the task is an instance of the Java™ class `MyAgendaFilter`, which implements the [llrAgendaFilter](#) interface.

```
ruletask ruletask1
{
 body = select(?rule) {return true;}
 agendafilter = new MyAgendaFilter();
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruletask](#)

# algorithm

The `algorithm` keyword defines an execution mode.

## Purpose

This key is used to define an execution mode for a rule task, either `RetePlus`, `sequential`, or `Fastpath`.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 algorithm = default|sequential;
};
```

## Description

The value that follows the `algorithm` keyword indicates which execution mode is used to execute the rules in that rule task. You can apply one of the following execution modes to rule tasks:

- The `RetePlus` mode: the value of the `algorithm` property is then `default` because `RetePlus` was the default execution mode in versions earlier than Operational Decision Manager V8.6.
- The `sequential` mode: the value of the `algorithm` property is then `sequential`.
- The `Fastpath` mode: `Fastpath` is the default algorithm. To use it, you must set the `algorithm` property to `sequential` and the custom property `ilog.rules.engine.sequential.fastpath` to `true`. For example:

```
ruletask R1
{
 property ilog.rules.engine.sequential.fastpath = true;
 algorithm = sequential;
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[ruletask](#)

# allrules

The allrules keyword sets all the rules in a rule task as eligible to be executed.

## Purpose

This keyword is used to set all the rules as eligible for execution in a rule task.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 firing = allrules|rule;
};
```

## Description

The value that follows the firing keyword indicates whether all the rules are executed (allrules, the default value) or whether only one rule is executed (rule).

### Note:

All instances of one rule are executed. If only one rule is executed, use the keyword [firinglimit](#). See also [ruletask](#) for more information.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruletask](#)



## as

The as keyword defines a type conversion.

### Purpose

This keyword is used to convert the type of an expression to another type.

### Context

Any expression

### Syntax

```
expression as type ;
```

### Description

The as statement attempts to convert the type of the expression into the requested type, without raising a `ClassCastException` instance. If the conversion is possible, the as statement returns an expression with the correct type. It returns null if the conversion failed.

### Example

```
rule AsStatement {
 when {
 ?a: Alarm(?e : equipment);
 }
 then {
 Multiplex m = ?e as Multiplex;
 if (m != null);
 System.out.println("Alarm occurs on Multiplex");
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[instanceof](#)

# before

The before keyword defines a binary temporal constraint.

## Purpose

This keyword is used to define a binary temporal constraint in an event condition.

## Context

Rule conditions

## Syntax

```
[?var:] event className(?eventVar1
before [interval] ?eventVar2);
```

## Description

### Deprecated as of V7.5.

Use the before keyword in the condition part of a rule in an event condition. To express a temporal constraint between two events, you use the after and before keywords. An interval is of the form [lowerBound, upperBound], where a bound is either an expression evaluating to an integer, or the \$ sign which denotes infinity.

Considering that the timestamp of ?event 1 is t1 and that the timestamp of ?event 2 is t2, you can express the before operator as follows:

1. ?event 1 before[min,max] ?event 2  
is satisfied if the value of t2 - t1 is between the values of min and max, inclusive.
2. ?event 1 before[min,\$] ?event 2  
is satisfied if the value of t2 - t1 is greater than, or equal to, the value of min.
3. ?event 1 before[\$,max] ?event 2  
is satisfied if the value of t2 - t1 is less than, or equal to, the value of max.
4. ?event 1 before ?event 2  
is satisfied if the value of t2 - t1 is positive or null.

## Example

This example assumes that a User instance is in the working memory and that an Alarm instance is inserted. A partial instance of the ReportAlarmPairsToUsers rule is created and maintained for four ticks. If the User instance is then retracted, the partial instance of the rule is immediately deleted. An instance of the ReportAlarmsToUsers rule is created on each instance of the User class in the working memory when ?a1 is no greater than 4 ticks before a second alarm event ?a2 is inserted.

```
rule ReportAlarmPairsToUsers {
 when {
 ?u: User();
 ?a1: event Alarm();
 ?a2: event Alarm(?a1 before[1, 4] ?this);
 } then {
 ?u.report(?a1, ?a2);
 }
};
```

**Parent topic:** [IRL keywords](#)

### Related reference:

[after](#)  
[event \(in rule conditions\)](#)  
[event \(in rule actions\)](#)  
[occursin](#)  
[timeof](#)

## body (in flowtask)

The body keyword defines the body of the ruleflow.

### Purpose

This keyword is used to define the body of a flow task in flowtask statements.

### Context

Flow task definitions

### Syntax

```
flowtask flowTaskName
{
 body {ruleflow}
};
```

### Description

The flow task body defines a ruleflow. It is composed of task calls that are chained together through control statements. The control statements are:

sequence, if, switch, while (break and continue), fork, goto.

### Example

```
flowtask main
{
 body =
 {
 while(!ending)
 {
 if (turn == Constants.Player1) ChooseMovePlayer1;
 else ChooseMovePlayer2;

 CheckMove;
 if (ending) break;

 UpdateDistance;
 ExpandObjects;
 DetectConnect4;
 if (ending) break;

 DetectGridFull;
 if (ending) break;
 ChangeTurn;
 }
 EndOfGame;
 }
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[goto](#)

[flowtask](#)

[fork](#)

[if \(in ruleflow\)](#)

[switch](#)

[while/break/continue \(in ruleflow\)](#)

## body (in action task)

The body keyword defines the body of an action task.

### Purpose

This keyword is used to define the body of an action task in `functiontask` statements.

### Context

Function task definitions

### Syntax

```
functiontask functionTaskName
{
 body
 {
 action1;
 ...
 actionn;
 }
};
```

### Description

The action task body is equivalent to an IRL function with no arguments and with the `void` return type. The task execution consists in executing the statements that compose its body after its initial actions and before its final actions if they are defined.

### Example

```
functiontask UpdateDistance
{
 body =
 {
 int x = move.x;
 int y = move.y;
 for (int i = y + 1; i < 10; i++)
 {
 int p = grid.position(x,i);
 if (p == null) break;
 p.distance--;
 update(p);
 }
 }
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[evaluate](#)

[for](#)

[functiontask](#)

[if \(in ruleflow\)](#)

[modify](#)

[retract](#)

[return](#)

[try](#)

[update](#)

[while](#)

## body (in ruletask)

The body keyword defines the body of the rule task.

### Purpose

This keyword is used to define a rule task body in ruletask statements.

### Context

Rule task definitions

### Syntax

```
(1) ruletask ruleTaskName
{
 body {ruleName1, ..., ruleNamen}
};

(2) ruletask ruleTaskName
{
 body = select([variableName])
 {
 action1;
 ...
 actionn;
 }
};

(3) ruletask ruleTaskName
{
 body = dynamicselect([variableName])
 {
 action1;
 ...
 actionn;
 }
};

(4) ruletask ruleTaskName
{
 body = dynamicselect([variableName])
 {
 action1;
 ...
 actionn;
 } in Expression;
};
```

### Description

You can write the body of a rule task in different syntaxes.

#### Syntax (1)

The rule task body is composed of the rules that are explicitly listed by their names. These rules must belong to the same ruleset as the rule task.

#### Syntax (2)

The rule task body has been specified by comprehension: the rules that compose the task body are not explicitly listed. The contents of the task body is computed when the code given as body for each rule in the ruleset is executed.

This body definition can be seen as either of the following IRL functions:

- As an IRL function with a Boolean return type and an argument of type [IIRule](#). In this case, a variable name is specified. The code is executed for each rule of the ruleset. When the execution returns true, the rule becomes part of the rule task. Otherwise it does not.

- As an IRL function with no argument and with `ilog.rules.engine.IlrRule[]` rule array as its return type. In this case, the rule task body contains the rules returned by the code.

In all cases for syntax (2), the code is executed only once for the task, even if the task is executed several times. It is evaluated the first time it is needed.

### Syntax (3)

The difference with Syntax (2) is the time at which the body is computed before each task is executed. This is useful when the body content depends on variables whose values change during the ruleflow execution.

#### Important:

Use a [scope](#) keyword in conjunction with syntax (2) or (3) because it limits the scope of the execution of the [select](#) or [dynamicselect](#).

### Example

```
ruletask DetectGridFull
{
 ordering = literal;
 firinglimit = 1;
 body = { DetectGridFull }
};

ruletask Expand0bjects
{
 ordering = dynamic;
 firing = allrules;

 body = select(?rule)
 {
 String ?task = ?rule.getProperties().getString("task","");
 return ?task.equals("Expand0bjects");
 }
};
```

**Parent topic:** [IRL keywords](#)

#### Related reference:

[algorithm](#)

[ruletask](#)

[rule \(in ruletask\)](#)

[scope](#)

# break

The break keyword exits a loop.

## Purpose

A side statement for exiting a loop.

## Context

Functions or technical rule actions

## Syntax

```
break;
```

## Description

The break statement is used in the action part of a rule or in functions. Use it to exit a while loop or a for loop. This statement forces the rule engine to go to the end of the containing statement block and to continue to the next instruction.

## Example

The AlarmSurveillance rule tests for a new Alarm object and loops while the state is equal to ON unless the level is equal to 3. The single rule condition matches an object Alarm if the field state equals the static value NEW. If such a Alarm object is matched, the engine can execute the action part. The modify statement is used to change the field state from NEW to ON. The while statement is used to loop as long as the field state has the value ON. An if statement tests the field level. If the level is equal to 3, a break statement is executed and the program exits the while loop.

```
rule AlarmSurveillance {
 when {
 ?a: Alarm(state == NEW);
 }
 then {
 modify ?a {
 state = ON;
 }
 while (?a.state == ON) {
 ...
 if (?a.level == 3)
 break;
 }
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[continue](#)

[for](#)

[foreach](#)

[while](#)

## case

The case keyword identifies a statement block in a switch statement.

### Purpose

This keyword is used to identify a statement block in a switch ruleflow statement.

### Context

switch statements

### Syntax

```
switch (expression)
{
 case value1:
 {ruleflowStatement}
 ...
 case valuen:
 {ruleflowStatement}
 default:
 {ruleflowStatement}
}
```

### Description

The switch statement is a conditional ruleflow statement. The integer expression is evaluated. If the value of a case block is equal to the evaluated expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

**Parent topic:** [JRL keywords](#)

**Related reference:**  
[switch](#)



# catch

The catch keyword designates exceptions that are caught if thrown.

## Purpose

This keyword is used within a try statement to designate exceptions that are caught if thrown.

## Context

Functions or rule actions

## Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

## Description

The try-catch-finally statements establish a block of code for exception handling. Such blocks all begin and end with curly braces.

The catch statements are designed to handle a specific type of exception. The code within a catch statement is executed if an exception is caught. A catch statement is declared with an `exceptionType` that specifies the type of exception that the statement can handle. It also provides an identifier that the statement can use to refer to the exception object that it is currently handling. The identifier must be of type `Throwable` or one of its subclasses.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[finally](#)

[try](#)

## collect

The collect keyword constructs a collection object.

### Purpose

This statement is used to construct a collection object.

### Context

Rule conditions

### Syntax

```
[?variable:] collect [(expression)] collectionTarget
[where (collectionTest1 ... collectionTestn)];
```

### Description

Use the collect statement in the condition part of a rule to create a collection object. The collection object stores instances of the class `collectionTarget` that match the condition. This condition can contain tests on the class fields. The collection object can be bound to a variable for the scope of the rule. The expression is optional, but if provided, it must return a collector object that implements the [IlrCollection](#) interface, as shown in the example below. The collect statement can contain a list of tests on the collection object in the where part of the statement.

If a rule contains some conditions that are matched and a collect condition, the eligibility of the rule depends on the existence of a where statement, as follows:

- Without a where statement, the rule is eligible to be executed, even if no object has been matched by `collectionTarget`. In other words, the rule is eligible and the collection object is empty.
- With a where statement, the rule is eligible only if the where statement is evaluated to `true`.

The collector object, declared by an expression, must implement the `IlrCollection` interface and its public methods: [addElement](#), [updateElement](#), and [removeElement](#). The `IlrCollection` interface is defined as follows:

```
public interface IlrCollection {
 public void addElement(java.lang.Object element);
 public void updateElement(java.lang.Object element);
 public void removeElement(java.lang.Object element);
}
```

If the *expression* argument is left empty, a [IlrDefaultCollector](#) object is used.

The where part of the statement can contain tests that the collection object must fulfill. Or it can be left empty. The [IlrCollection](#) interface and [IlrDefaultCollector](#) class provide methods for all collections, such as `size`, `isEmpty`, `contains`, and `elements`.

### Example

The `MusicCollector` rule collects `Music` objects and stores them in an `ItemCollector` object. The first condition returns in variable `?s` the `Store` objects that have the field `domain` equal to `MUSIC`. The variable `?c` refers to the `ItemCollector` object. The constructor for the `ItemCollector` object takes a `Store` object as an argument.

```
rule MusicCollector {
 when {
 ?s: Store(domain==MUSIC);
 ?c: collect(new ItemCollector(?s))
 Music(store==?s; category==JAZZ;
 artist equals "Miles Davis";
 production > 1956 & < 1965)
 where (size() > 5 && ItemCollector.averagePrice()
 < 15.0);
 }
 then {
 Enumeration ?enum = ?c.elements();
 while (?enum.hasMoreElements()) {
```

```

 Music ?cd = (Music)enum.nextElement();
 System.out.println("At " + ?s.name
 "you can find the title: " + ?cd.title);
 }
}
}

```

The collectionTarget is the class Music with the following field values:

- store equal to the result in ?s
- category equal to JAZZ
- artist equal to the string Miles Davis
- production between the dates 1956 and 1965

Any object that match this condition is inserted into the ItemCollector object. For the collect statement to be true, two tests are defined in the where part of the statement. The default method size tests that the number of items in the collector is greater than 5 and the user-defined method averagePrice tests that the average price of the items is lower than 15.

If the when part of the rule is true, the then part is executed. The variable ?c refers to the ItemCollector object. The first statement creates a variable ?enum which refers to an enumeration of the elements of the ItemCollector object. The while statement prints the name field of the Store object and the corresponding title field of each Music object.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[exists](#)  
[from](#)  
[in](#)  
[not](#)

# continue

The continue keyword skips an iteration.

## Purpose

A side statement for skipping an iteration.

## Context

Rule actions or functions

## Syntax

```
continue;
```

## Description

Use the continue statement in the action part of a rule and in a function to skip an iteration of a while or for loop. This statement forces the rule engine to skip to the next test of the loop statement.

## Example

The rule StockDropWarning checks the percent change of a customer's stocks and issues a warning if a stock price dropped by more than a certain percentage. The condition Client returns an account number in variable ?a and a decimal value percentWarningMark in variable ?p. The second condition, ClientStockList, returns a stock list in variable ?s corresponding to the account number equal to ?a. In the action part of the rule, the while statement iterates over every stock in the stock list ?s. If a stock type is equal to LONG, the continue statement cause the iteration to skip the rest of the statements in the while block and proceed to the while test. If a stock type is not equal to LONG, the difference between the purchase price and the current price is calculated. If the stock price decreased in ?p percent below the purchasePrice, a message is printed.

```
rule StockDropWarning {
 when {
 ?c:Client(?a:account_number;?p:percentWarningMark);
 ClientStockList(account_number == ?a; ?s:stockList);
 }
 then {
 Enumeration ?enum = ?s.elements();
 while (?enum.hasMoreElements()) {
 Stock ?x = (Stock)enum.nextElement();
 if (?x.type == Stock.LONG)
 continue;
 if (((1.-?p)*?x.purchasePrice) > ?x.currentPrice) {
 System.out.println("Dear "+ ?c.name + " Your stock "
 + ?x.ticker + " has dropped " +
 ((?x.purchasePrice - ?x.currentPrice) /
 ?x.purchasePrice) + " percent.");
 }
 }
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[break](#)

[for](#)

[foreach](#)

[while](#)

## default (in ruletask)

This default keyword selects the RetePlus execution mode for a rule task.

### Purpose

In the context of execution modes, the default keyword is used to select the RetePlus execution mode for a rule task.

### Context

Rule task definitions

### Syntax

```
ruletask ruleTaskName
{
 algorithm = default|sequential;
};
```

### Description

The value that follows the algorithm keyword indicates which execution mode is used to execute the rules:

- RetePlus if the algorithm value is default, because RetePlus was the default execution mode in versions earlier than Operational Decision Manager V8.6.
- The sequential mode if the algorithm value is sequential.

Fastpath is an option within the sequential mode.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[algorithm](#)

[ruletask](#)

## default (in switch)

This default keyword executes the else condition statement.

### Purpose

This keyword is used in the switch statement to execute the else condition.

### Context

switch statements

### Syntax

```
switch (expression)
 case value1:
 {ruleflowStatement}
 ...
 default:
 {ruleflowStatement}
```

### Description

The switch statement is a conditional ruleflow statement. The integer expression is evaluated. If a value of the case block is equal to the evaluated expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

**Parent topic:** [JRL keywords](#)

**Related reference:**  
[switch](#)

# dynamic

The `dynamic` keyword specifies the dynamic ordering of rules.

## Purpose

This keyword is used to specify the dynamic ordering of the rules that are executed in a rule task.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 [ordering = dynamic|sorted|literal;]
};
```

## Description

Rule tasks execute rules. You can use parameters to specify how the rules are ordered and how many rules are executed.

The keyword `ordering` specifies how the rules are sorted.

- `dynamic`: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- `sorted`: The rules are sorted in decreasing order of static priority.
- `literal`: The rules are listed in the same order as they have been listed in the rule task body.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[ordering](#)  
[ruletask](#)

# dynamicselect

The `dynamicselect` keyword specifies the dynamic selection of a rule.

## Purpose

This keyword is used to specify the dynamic selection of a rule in a rule task body.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 body = dynamicselect(?rule) {action1 ... actionr}
};
```

## Description

The body of a rule task contains a list of rules. You can either specify each rule name explicitly (extension) or have the list of rules computed from the code, which uses the `select` or `dynamicselect` keywords (comprehension). The code must return a value of type `boolean`. The selection method changes the way the filter is applied to the rules. With dynamic rule selection, the filter is evaluated each time the rule task is invoked.

There is no difference between `dynamicselect` and `select`. They behave the same way in that the statement is called each time the task is executed.

To optimize performance, the decision engine does not execute the code statements the same number of times depending on whether they are before the use of the `(?rule)` variable or after. Statements that come before the statement in which the `(?rule)` variable is used are calculated only once per ruletask execution, when the task body is evaluated. Statements including and after the first statement in which the `(?rule)` variable is used are calculated for every rule under review in the task scope. For example, `?contract` is a ruleset variable with a `country` attribute and an `effectiveDate` attribute, and the task applies only for a selection of countries and only to the rules whose `effectiveDate` covers the one of the contract:

```
body = dynamicselect(?rule) {
return ?contract.country in { "France", "Spain", "Italy"} &&
 ?contract.effectiveDate.after((java.util.Date)?
rule.getPropertyValue("effectiveDate"));
}
```

In the previous example, `?contract.country in { "France", "Spain", "Italy"}` is evaluated once per task execution and `?contract.effectiveDate.after((java.util.Date)?rule.getPropertyValue("effectiveDate"))` is calculated once per rule for each task execution.

However, if you write the following code, then both conditions are calculated once per rule for each task execution:

```
body = dynamicselect(?rule) {
return ?contract.effectiveDate.after((java.util.Date)?
rule.getPropertyValue("effectiveDate")) &&
 ?contract.country in { "France", "Spain", "Italy" };
}
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in ruletask\)](#)  
[ruletask](#)  
[select](#)



## else (in if)

In functions or rule actions, the `else` keyword executes an alternate statement block.

### Purpose

A side statement within an `if` statement to execute an alternate statement block depending on a Boolean expression value.

### Context

Functions or rule actions

### Syntax

```
if (test) {statement}
else {statement}
```

### Description

Use the `if` statement in the action part of a rule, a function, or a ruleflow. The test can be any legal test as in the Java™ programming language. If the test returns `true`, the first statement block is executed and the `else` block is skipped over. If the test returns `false`, the first block is skipped over and the statement block that follows the `else` clause is executed.

In a rule or function, any IRL statement, arithmetic expressions, and method calls can be executed within the statement block. In a ruleflow, any ruleflow statement can be executed within the statement block. A statement block can contain one or more statements. If it contains only one statement, the braces (`{}`) are not required. The `else` keyword and its statement block is optional.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[if](#)

## else (in rule)

In rule definitions, the `else` keyword executes an alternate action block depending on the value of an `evaluate` statement.

### Purpose

In the condition part of a rule, this keyword is used to execute an alternate action block depending on the value of an `evaluate` statement.

### Context

Rule definitions

### Syntax

```
rule ruleName {
 when {condition evaluate (expression)}
 then {[action1 ... actionm]}
 else {[action1 ... actionp]}
};
```

### Description

When the condition part of the rule ends with an `evaluate` statement, the action part can have an `else` part, which is executed if the last `evaluate` statement returns `false`. If it returns `true`, the `then` part is executed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[then](#)  
[evaluate](#)  
[rule](#)

## evaluate

The evaluate keyword specifies tests on objects.

### Purpose

This evaluate statement is used in the condition part of a rule to test objects in the working memory.

### Context

Rule conditions

### Syntax

```
evaluate (expression);
```

### Description

Before the evaluate statement, you must write a simple condition that binds a variable to an object or a value. The engine can test any such variable bound to an object or a value. Note that the statements not, exists, and collect are not simple conditions. An evaluate statement is true if all the tests carried out in the expression are true. The expression can be made of multiple tests enclosed in braces ({}).

When the condition part of a rule ends with an evaluate statement, the action part can have an else part, which is executed if the last evaluate statement returns false. If it returns true, the then part is executed.

The evaluate statement differs from the not and exists statements. The not and exists statements apply tests only within the scope of their statement. The evaluate statement is more general in that it specifies tests that must be satisfied by the objects that exist in the working memory.

### Example

The following two CloseOrder rules show equivalent ways of testing objects of the working memory. The first rule has two conditions followed by an evaluate statement. In the second rule, the test of the evaluate statement is incorporated in the second condition.

#### Rule 1

```
rule CloseOrder {
 when {
 ?s: CustomerShipment(?id:CustomerId; ?sl:shipmentList);
 ?o: CustomerOrder(?id==CustomerId; ?ol:orderList);
 evaluate (?sl.equals(?ol));
 }
 then {
 retract(?s);
 insert Shipped(?o,currentDate());
 }
};
```

#### Rule 2

```
rule CloseOrder {
 when {
 ?s: CustomerShipment(?id:CustomerId; ?sl:shipmentList);
 ?o: CustomerOrder(?id==CustomerId; ?ol:orderList;
 ?sl.equals(?ol))
 }
 then {
 retract(?s);
 insert Shipped(?o,currentDate());
 }
};
```

For both rules, variable ?s refers to a CustomerShipment object, variable ?id returns the CustomerId field, and variable ?sl returns the shipmentList field. In the second condition, variable ?o refers to a CustomerOrder object where the CustomerId field matches ?id and variable ?ol contains the orderList field. The evaluate statement calls a test, the equals method, which returns true if the two lists contain the same elements. If the conditions are true, the action part is executed. The CustomerShipment object is deleted from the working memory by the retract statement, and a Shipped object is inserted into the working

memory by the insert statement.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[else \(in if\)](#)

[exists](#)

[instanceof](#)

[isknown](#)

[isunknown](#)

[not](#)

[rule](#)

## event (in rule conditions)

The event keyword declares an event condition or an event object.

### Purpose

This keyword is used for declaring an event condition or an event object in rule conditions.

### Context

Rule conditions

### Syntax

```
[?var:] event className (test1;...;testn);
where a temporal test test i has the form:
{after|before|occursin} [interval]
```

### Description

#### Deprecated as of V7.5.

On the side of a rule, an event condition can be bound to a variable. The event class name is defined in the statement together with tests.

Tests can be temporal or nontemporal.

- In a temporal test, you can use the following keywords: The interval, when provided, restricts the allowed delay between the events.
  - To express a temporal constraint, use the test operators after, before, or occursin.
  - To express temporal constraints between two events, use the after and before operators.
  - To express a temporal constraint on a single event, use the occursin operator. An occursin test is satisfied if the timestamp of the event is included in the interval.

Do not write references to bound variables or in other event conditions in the expression. The expression is evaluated once, when the events occurring as operands of the temporal constraint operator are inserted.

- You can combine nontemporal tests and temporal constraints as shown in the following code lines:

```
?a: event Alarm();
?b: event Alarm(sev >= HIGH || ?this after[1,5] ?a);
```

### Example

The alarmManager rule prints a message when two alarms occur on the same managed object with a time difference between 1 and 5 clock ticks.

```
rule alarmManager {
 when {
 ?mo: ManagedObject();
 ?operator: Operator() in ?mo.subscribers();
 ?alarm1: event Alarm(managedObject == ?mo);
 ?alarm2: event Alarm(managedObject == ?mo; ?this after[1, 5]
 ?alarm1);
 }
 then {
 System.out.println("Seen two alarms on " + ?mo.name);
 ?operator.report(?alarm1, ?alarm2);
 }
}
```

The when keyword introduces four conditions:

1. In the first condition, a variable ?mo is matched by a ManagedObject object.

2. The second condition returns the subscribed Operator objects in the variable ?operator.
3. The third condition is an event statement. The variable ?alarm1 points to an event object named Alarm with a single constraint: the value of the field managedObject is equal to the variable ?mo.
4. The fourth condition is also an event statement, where the variable ?alarm2 points to an event object named Alarm with the following constraints: the value of the field managedObject is equal to the variable ?mo and the difference between the first event object timestamp and the second event object timestamp is greater than, or equal to, 1 and less than, or equal to, 5 clock ticks.

The then keyword introduces two actions:

1. The first action prints a message that names a managed object.
2. The second action executes an ?operator.report method using the event object variables ?alarm1 and ?alarm2.

Any alarm is automatically retracted from the working memory 6 clock ticks after its insertion.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[after](#)  
[before](#)  
[occursin](#)  
[timeof](#)

## event (in rule actions)

The event keyword declares an event condition or an event object.

### Purpose

This keyword is used for declaring an event condition or an event object in rule actions.

### Context

Functions or rule actions

### Syntax

```
insert event [(timeExpression)] object [{statement1;...;statementn}];
```

### Description

#### Deprecated as of V7.5.

Next to a rule, when an object is inserted into the working memory as an event, a timestamp is automatically associated with it. The `timeExpression` argument defines an integer value which is used as the timestamp when an event is inserted. The term `object` must be a constructor for the class. For the constructor to apply, you must declare it as `public` in your Java™ code.

Following the optional `timeExpression`, the `insert event` statement can either terminate with a semicolon (;) or specify a block of executable statements. The statements are executed on the object before this object is inserted into the working memory.

You can remove an event object explicitly from the working memory by using the `retract` statement or the `IlrContext.retract` method.

### Example

The `alarmManager` rule prints a message when two alarms occur on the same managed object with a time difference between 1 and 5 clock ticks.

```
rule alarmManager {
 when {
 ?mo: ManagedObject();
 ?operator: Operator() in ?mo.subscribers();
 ?alarm1: event Alarm(managedObject == ?mo);
 ?alarm2: event Alarm(managedObject == ?mo; ?this after[1, 5]
 ?alarm1);
 }
 then {
 System.out.println("Seen two alarms on " + ?mo.name);
 ?operator.report(?alarm1, ?alarm2);
 }
}
```

The `when` keyword introduces four conditions:

1. In the first condition, a variable `?mo` is matched by a `ManagedObject` object.
2. The second condition returns the subscribed `Operator` objects in the variable `?operator`.
3. The third condition is an event statement. The variable `?alarm1` points to an event object named `Alarm` with a single constraint: the value of the field `managedObject` is equal to the variable `?mo`.
4. The fourth condition is also an event statement, where the variable `?alarm2` points to an event object named `Alarm` with the following constraints: the value of the field `managedObject` is equal to the variable `?mo`, and the difference between the first event object timestamp and the second event object timestamp is greater than or equal to 1 and less than or equal to 5 clock ticks.

The `then` keyword introduces two actions:

1. The first action prints a message naming a managed object.
2. The second action executes an `?operator.report` method that uses the event object variables `?alarm1` and `?alarm2`.

Any alarm is automatically retracted from the working memory 6 clock ticks after its insertion.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[after](#)  
[before](#)  
[occursin](#)  
[timeof](#)



## exists

The exists keyword tests whether a class condition is true.

### Purpose

This keyword is used in the condition part of a rule to test whether the condition is true.

### Context

Rule conditions

### Syntax

```
exists condition;
```

### Description

Use the exists statement in the condition part of a rule, on a simple condition, an in statement, or a from statement. The exists statement tests whether the condition is true. The contrary of exists is not (avoid not exists).

The exists statement returns true when any object in the working memory matches the condition.

- It returns true for a simple condition if the working memory contains at least one object that can match the condition.
- It also returns true with an in or from statement when the in or from statement returns true.

Because the condition does not discriminate which object was matched, you cannot bind an exists statement to an external variable. Any variable bound within the exists statement is local to the statement. No statement of the rule can contain any reference to the bound variable.

You can use the special variable ?this within the exists statement to designate the current working memory object being tested.

### Example

The GeneralCustomerRequest rule uses the exists statement to verify that an item requested by a customer exists. The exists statement returns true when a single instance of the object is found, which is faster than providing all the items that match the customer's request.

```
rule GeneralCustomerRequest {
 when {
 CustomerRequest(?item:request);
 exists Item(?this == ?item);
 }
 then {
 System.out.println(?item + " are in stock.");
 }
};
```

In the CustomerRequest condition, the variable ?item is matched with the field request. In the exists statement, the Item object is represented by the variable ?this. The statement tests whether the Item object is equal to the variable ?item. If an Item object is matched, the println command in the action part prints that the items are in stock.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[collect](#)  
[fork](#)  
[from](#)  
[in](#)  
[not](#)

# filter

The filter keyword defines an agenda filter.

## Purpose

This keyword is used to define the filter of an `agendafilter` in a rule task.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 agendafilter = filter (variableName)
 {
 action1
 ...
 actionn
 }
};
```

## Description

This format presents a way to define an agenda filter in the rule task. In this case, you write the code of the agenda filter within the task definition. Such code is similar to an IRL function with an [IlrRuleInstance](#) parameter and a Boolean return type. The *(variableName)* placeholder represents this `IlrRuleInstance` object. The inline code of the agenda filter can hold a reference to it. Ruleset variables are accessible from the agenda filter code. This agenda filter is applied to each instance of the rules that compose the task body. Only the instance for which the agenda filter returns `true` is executed. Others are not.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[agendafilter](#)  
[ruletask](#)

# finalaction

The finalaction keyword declares a final action.

## Purpose

This keyword is used to declare a final action in a ruleflow task.

## Context

Rule task definitions

## Syntax

```
finalaction {action1 ... actionn}
```

## Description

A subflow task can have an initial action or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as functions. They are similar to an IRL function with the void return type and no arguments.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[flowtask](#)

[functiontask](#)

[ruletask](#)

# finally

The `finally` keyword introduces a control block that is executed after a `try` block.

## Purpose

This keyword is used within a `try` statement that specifies a control block to be executed after the `try` block has been executed, including its exceptions.

## Context

Functions or rule actions

## Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

## Description

The `try-catch-finally` statements establish a block of code for exception handling. The blocks all begin and end with curly braces.

The `finally` statement is executed if any portion of the `try` statement is executed, regardless of how the code in the `try` and `catch` statements completes.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[catch](#)

[try](#)

# firing

The `firing` keyword determines whether all the rules are executed.

## Purpose

This keyword is used to determine whether all the rules of a rule task are executed or only one of them.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 [firing = allrules|rule;]
};
```

## Description

The value following the `firing` keyword indicates whether all rules are executed (`allrules`, the default) or whether only one rule is executed (`rule`).

### Note:

All instances of one rule are executed. If only one rule is executed, use the keyword [firinglimit](#). See also [ruletask](#) for more information.

**Parent topic:** [IRL keywords](#)

### Related reference:

[allrules](#)  
[rule](#)  
[ruletask](#)

## firinglimit

The `firinglimit` keyword specifies the number of rules to be executed.

### Purpose

This keyword is used to specify the maximum number of rules to be executed in a rule task.

### Context

Rule task definitions

### Syntax

```
ruletask ruleTaskName
{
 [firinglimit = integer value;]
};
```

### Description

The purpose of a rule task is to execute rules. You can modify the execution order and the number of rules that are executed.

The keyword `firinglimit` specifies how many rules are executed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in ruletask\)](#)  
[ruletask](#)

# flowtask

The flowtask keyword declares a subflow task.

## Purpose

This keyword is used to declare a flow task.

## Context

At the top level of rulesets

## Syntax

```
flowtask flowTaskName
{
 [property propertyName = value;]
 [initialaction {action1 ... actionm}]
 [finalaction {action1 ... actionn}]
 [completionflag = value;]
 body {ruleflow}
};
```

## Description

A flow task is one of the three kinds of tasks available in a ruleflow. A flow task describes how task executions are chained together and under which conditions.

### Note:

1. The ruleflow syntax is described in [Grammar specification](#).
2. If a formal comment (/\*\*...\*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

A subflow task can have the following attributes:

### property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value later using the API.

### initialaction, finalaction

A subflow task can have an initial or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as IRL functions. They are similar to an IRL function with the void return type and no arguments.

### body

The flow task body defines a ruleflow. It consists of task calls that are chained together through control statements. The control statements are: fork, goto, if, sequence, switch, while (break and continue)

## Example

This example illustrates a flow task definition.

```
flowtask main
{
 initialaction =
 {
 turn = Constants.Player2;
 saved = new SavedGame(grid);
 for (var i = 0; i < 7; i++) for (var j = 0; j < 6; j++)
 insert(grid.array[i][j]);
 };
 finalaction =
 {
 grid = null;
 move = null;
 winner = Constants.None;
 };
}
```

```

 connect4 = null;
 ending = false;
 message = null;
};
body =
{
 while(!ending)
 {
 if (turn == Constants.Player1) ChooseMovePlayer1;
 else ChooseMovePlayer2;

 CheckMove;
 if (ending) break;

 UpdateDistance;
 ExpandObjects;
 DetectConnect4;
 if (ending) break;

 DetectGridFull;
 if (ending) break;
 ChangeTurn;
 }
 EndOfGame;
}
};

```

This task is executed in this order:

1. Its initial actions are executed first.
2. The body is executed.

The body consists of a while loop. Each loop corresponds to a move from a player in the Connect4 game. If the move causes a Connect4 object to be accomplished or the grid to be full, the game is finished and the while loop can be interrupted by a break statement. Here the ruleflow resumes after the while loop. In this particular example, the EndOfGame task is executed.

3. When the flow finishes execution, the final actions of the task are executed.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[body \(in flowtask\)](#)  
[functiontask](#)  
[ruletask](#)



## for

The for keyword introduces an execution loop.

### Purpose

A statement for executing a statement block as many times as matching objects are found to start the action.

### Context

Functions or rule actions

### Syntax

```
for (initialize; test; increment) statement;
```

### Description

Use the for statement in the action part of rules and in functions to execute multiple times the statement or a block of statements enclosed in braces ({}). The `initialize`, `test`, and `increment` arguments can be any valid expression, as in the Java™ programming language. The statement block can execute any IRL statement and any arithmetic expressions and method calls.

### Example

The `ConnectivityUpdate` rule tests whether a new node exists in the network and adds a link to it from each of its neighbors.

```
rule ConnectivityUpdate{
 when {
 ?n: Node(state == NEW; ?neighbors: neighbors());
 }
 then {
 for (int ?i = 0; ?i < ?neighbors().size(); ?i++) {
 ((Node)?neighbors().elementAt(i)).addLink(?n);
 }
 update (?n);
 }
};
```

This rule is processed as follows:

1. The rule condition matches an object `Node` when the field `state` equals the static value `NEW` and it binds the variable `?neighbors` with the vector field `neighbors()`.
2. If such a `Node` object is found, the action part can be executed. The variable `?i` is initialized to 0. The for statement loops for each neighbor and the `addLink(?n)` method updates the links from the neighbors to the new node.
3. The last action uses the update command to update the agenda with the new node.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[break](#)  
[continue](#)  
[foreach](#)  
[while](#)

## foreach

The foreach keyword executes a statement block several times.

### Purpose

The foreach statement is used in the action part of a rule for executing a statement block numerous times.

### Context

Functions or rule actions

### Syntax

```
foreach (type variable in expression) statement;
```

### Description

Use the foreach statement in the action part of rule and in functions to execute the statement, or a block of statements enclosed in braces ({}), on each element of a collection or array. The foreach statement introduces a new variable which you can use in the remainder of the execution block.

The variable type can be any valid type, as in the Java™ programming language. The expression can be any legal expression that returns a Collection or Array value type, as in the Java programming language. The statement block can execute any IRL statement and any arithmetic expressions and method calls.

This foreach statement slightly differs from the corresponding statement in Java in that it automatically filters out the objects that are not of the expected type. See the second example below.

### Example

Here are two examples of foreach statements.

#### Example 1

The first example shows the basic behavior of the foreach statement:

```
rule ConnectivityUpdate{
 when {
 ?n: Node(state == NEW; ?neighbors: neighbors());
 }
 then {
 foreach (Node node in ?neighbors) {
 node.addLink(?n);
 }
 update (?n);
 }
};
```

The ConnectivityUpdate rule tests whether a new node exists in the network and adds a link to it from each of its neighbors. This rule is processed as follows:

1. The rule condition matches an object Node when the field state equals the static value NEW, and it binds the variable ?neighbors with the vector field neighbors().
2. If such a Node object is found, the action part can be executed. The foreach statement loops for each neighbor. A new variable node is defined and the method addLink(?n) updates the links of the neighbors to the new nodes.
3. The last action uses the update command to update the agenda concerning the new node.

#### Example 2

The second example shows the filtering capability of the foreach statement<

```
List l = new ArrayList();
l.add(1);
l.add(null);
l.add("a string");

foreach (String s in l) {
```

```
System.out.println(s);
}
```

This statement block produces a `string` because the instances that are not of the expected type (here `String`) are filtered out automatically.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[break](#)  
[continue](#)  
[for](#)  
[in](#)  
[while](#)

# fork

The fork keyword executes several statement blocks sequentially.

## Purpose

The fork statement is a ruleflow statement used to execute several statement blocks in sequence.

## Context

Body of flowtask blocks

## Syntax

```
fork
{ruleflowStatement} &&
{ruleflowStatement}
[&& {ruleflowStatement}] *
```

## Description

Use this statement to define several ruleflow statement blocks that execute sequentially. The flow continues after the fork statement when all the statement blocks of the fork statement have been executed.

## Example

In the following fork statement, the first block is executes T1, then T2. When T2 finishes execution, the second block executes T3, then T4. When T4 finished, all the fork blocks are finished, T5 can be executed.

```
flowtask main
{
 body =
 {
 fork
 { T1; T2; }
 &&
 { T3; T4; }
 }
 T5;
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in flowtask\)](#)  
[flowtask](#)

## from

The `from` keyword supports relations between objects.

### Purpose

The `from` statement is used in the condition part of a rule to express relations between objects.

### Context

Rule conditions

### Syntax

```
condition from expression;
```

### Description

This statement returns `true` when the `from` condition matches an object that is returned by the `from` expression.

Using the `from` statement, you can directly access objects in the working memory and objects that are linked to other objects by fields or by method calls. Methods that are attached directly to the context object can be called in the expression part of the statement.

The `from` statement fosters performance because pattern matching is done on a reduced number of objects and not on all the objects in the working memory (see the first example below).

You can also use the `from` statement to access objects in a database. Using SQL queries in the expression part of the statement, you can retrieve information from a relational database (see the second example below).

### Example

The first example uses the `from` keyword to extract a subset of objects while the second example uses the `from` keyword to query a relational database.

#### Example 1

The rule `FindBookStore` returns bookstores or stores with a book department in the city of London.

```
rule FindBookStore {
 when {
 ?s: Store(city == London);
 ?d: Department(type == Books) from ?s.department;
 }
 then {
 System.out.println(
 "The following book store/department was found: "
 + ?s.address);
 }
}
```

This rule is processed as follows:

1. The first condition provides the stores in London in variable `?s`.
2. The second condition uses this reduced set of stores to find which store has a book department.
3. If the `when` part is successful, the action part prints the stores found.

#### Example 2

The rule `FindCarOwner` finds the owner of a car by means of an SQL query to a relational database where employee information resides. The first condition matches a `CarToMove` object and returns a license number in variable `?l`. The second condition returns, in variable `?c`, a `Car` object found using `?l`, the license number. The `from` statement uses the `Person` object to return a name in variable `?n`, which matches the name of the car owner. An SQL query launched by the `getOwner()` method retrieves the car owner. The action part of the rule prints the person's name and telephone number, the car model, and the driver license number.

```
rule FindCarOwner {
 when {
 CarToMove(?l:license);
```

```
 ?c: Car(license == ?l);
 ?p: Person(?n:name) from ?c.getOwner();
}
then {
 System.out.println("Contact " + ?n + " at telephone "
 ?p.telephone + " to move a " + ?c.model +
 " with license: " + ?l);
}
}
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[collect](#)

[exists](#)

[in](#)

[not](#)

## function

The function keyword defines a function to be called.

### Purpose

The function keyword is used to declare a function that can be invoked in the action part of a rule.

### Context

At the top level of rulesets

### Syntax

```
function returnType functionName (argList) {statement}
```

### Description

You can declare functions in a rule file before rule declarations. The function declaration consists of a header part and the statement block. The header part contains the return type of the function, the function name, and a list of arguments. The arguments are listed in pairs: the argument type and the argument name, separated by a comma. The statement block can contain any ILOG® Rule Language side statement, arithmetic expressions, and method calls. A statement block contains one or more statements. Use the keyword return to declare the value returned by the function, of the type returnType.

### Example

```
function String buildMessage(Client ?c, ShoppingCart ?s)
{
 int ?a = ?s.getAmount();
 int ?t = ?c.getTotalAmount();
 if (?a > 100.0 && ?t > 1000.0)
 return("Dear "+ ?c.getName() + ", you are a GOLD
 customer and will receive a 10% discount today!!!");
 else if (?a > 100.0)
 return("Dear "+ ?c.getName() + ", you are a SILVER
 customer and will receive a 5% discount today!!!");
 else if (?t > 1000.0)
 return("Dear "+ ?c.getName() + " , we can offer you a 10%
 discount on a shopping cart valued at over $100 today.");
 else
 return("Dear "+ ?c.getName() + " , we can offer you a 5%
 discount on a shopping cart valued at over $100 today.");
}
rule Promotion
{
 when {
 ?s:ShoppingCart(?a:amount;?id:clientNumber);
 ?c:Client(clientNumber == ?id; ?t:totalPurchaseToDate);
 }
 then {
 System.out.println(buildMessage(?c,?s));
 }
}
};
```

The function buildMessage takes a Client and a ShoppingCart as arguments and returns a message with the possible discount. The variable ?a is the current shopping cart amount and the variable ?t is the total purchased to date for the client. Three if statements identify which discount corresponds to the client.

Four cases are treated:

- The current shopping cart is valued at over \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at over \$100 and past purchases have not exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have not exceeded \$1000.

The Promotion rule provides a client with a promotion depending on the value of the shopping cart and past purchases. The condition ShoppingCart returns an amount in variable ?a and a client number in variable ?c . The second condition, Client, returns a total amount spent in variable ?t corresponding to the client number equal to ?c. In the action part of the rule, the function buildMessage is called and returns the appropriate message, which is displayed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[evaluate](#)

[for](#)

[if](#)

[modify](#)

[retract](#)

[return](#)

[throw](#)

[try](#)

[update](#)

[while](#)



# functiontask

The functiontask keyword declares an action task.

## Purpose

This keyword is used to declare an action task that defines a ruleflow function.

### Note:

In product versions earlier than 7.1.x, action tasks were called function tasks.

## Context

At the top level of rulesets

## Syntax

```
functiontask functionTaskName
{
 [property propertyName = value;]
 [initialaction {action1 ... actionm}]
 [finalaction {action1 ... actionn}]
 [completionflag = value;]
 body {ruleflow}
};
```

## Description

An action task is one of the three kinds of tasks available in a ruleflow.

### Note:

1. The ruleflow syntax is described in [Grammar specification](#).
2. If a formal comment (/\*\*...\*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

An action task can have the following attributes:

### property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value later using the API.

### initialaction, finalaction

An action task can have an initial or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. Initial and final actions for action tasks are composed of inline IRL code, such as IRL functions. They are similar to IRL functions with a void return type and no arguments.

### body

An action task is composed of inline IRL code, such as IRL functions. The body is similar to an IRL function with a void return type and no arguments.

## Example

This example defines an action task whose body is the code given between the braced brackets.

```
functiontask UpdateDistance
{
 body =
 {
 int x = move.x;
 int y = move.y;
 for (var i = y + 1; i < 10; i++)
 {
 int p = grid.position(x,i);
 if (p == null) break;
 p.distance--;
 }
 }
}
```

```
 update(p);
 }
}
};
```

This code is executed when the task is called to be executed in a ruleflow.

If initial actions are provided, they are executed before the task body. If final actions are defined, they are executed after the task body.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in action task\)](#)

[flowtask](#)

[ruletask](#)

## goto

The goto keyword jumps to another ruleflow statement.

### Purpose

The goto keyword is a ruleflow statement used to jump to another statement in the ruleflow.

### Context

Body of flowtask blocks

### Syntax

```
goto label;
```

### Description

The target statement must be labeled and this label is the one referenced in the goto statement.

You can use a goto statement in a fork or while statement if the following rules are respected:

- In a fork statement, the goto statement must not jump to a statement that does not belong to one of the fork blocks.
- In a while statement, the goto statement must not jump to a statement outside the while loop.

### Example

```
flowtask main
{
 body =
 {
 start: if (turn == Constants.Player1) ChooseMovePlayer1;
 else ChooseMovePlayer2;
 CheckMove;
 if (ending) goto end;
 UpdateDistance;
 ExpandObjects;
 DetectConnect4;
 if (ending) goto end;
 DetectGridFull;
 if (ending) goto end;
 ChangeTurn;
 goto start;
 end: EndOfGame;
 }
};
```

The ruleflow consists of a loop implemented with the goto statement goto start, which jumps to the beginning of the flow. The loop can be interrupted under some conditions and this interruption is implemented with the goto statements goto end. The start and end parts are the labels of the first and last ruleflow statements, respectively.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in flowtask\)](#)  
[flowtask](#)

# hasher

The hasher keyword defines a hashing expression.

## Purpose

This keyword is used in a ruleset declaration to define a hashing expression.

## Context

At the top level of rulesets

## Syntax

```
ruleset rulesetName
{
 hasher (typeName variableName) = value;
};
```

## Description

With the hasher keyword, you can define a hashing expression to optimize rule execution.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruleset](#)

# hierarchy

The hierarchy keyword defines a hierarchical property.

## Purpose

This keyword is used to define a hierarchy that defines values and gives a partial order between those values.

## Context

At the top level of rulesets

## Syntax

```
hierarchy hierarchyName
{ "rootName"
 { "nodeName1" { "nodeName11" "nodeName12" ... }
 "nodeName2"
 "nodeName3" { "nodeName31" { "nodeName311" } }
 }
}
```

## Description

IRL rules can have hierarchical properties. Specific predicates are defined to deal with hierarchical properties and get the rules that have a specific property value.

When a rule hierarchical property is accessed, its value is considered to be a string.

## Example

This example defines three rules r1, r2, and r3. All define the location property which is a hierarchical property.

```
hierarchy Geography
{ "North"
 { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } }
 "Europe" { "France" { } "Germany" }
 }
}
propertydefinition { Geography location; }
rule r1 { property location = "North/USA/Texas/Paris" when { ... } then { ... }
}
rule r2 { property location = "France" when { ... } then { ... } }
rule r3 { property location = "North/Europe/France/Paris" when { ... } then { ... } }
```

In r1 and r3, the location value is given as a path from the hierarchy root to the final leaf, so as to resolve the ambiguity between the two nodes named Paris. The r2 location property value is France which refers to a unique node in the hierarchy.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[match](#)  
[propertydefinition](#)  
[rule](#)

## if

The `if` keyword executes one of two statement blocks.

### Purpose

This statement is used for executing one of two statement blocks in the action part of rules or in functions.

### Context

Functions or rule actions

### Syntax

```
if (test) {statement}
[else {statement}]
```

### Description

The test can be any valid test, as in the Java™ programming language. If the test returns `true`, the first statement block is executed and the `else` block is skipped over. If the test returns `false`, the first block is skipped over and the statement block following the `else` keyword is executed. Any IRL statement can be executed within the statement block, as well as arithmetic expressions and method calls. A statement block consists of one or more statements. A single statement does not require the braces (`{}`). The `else` keyword and its statement block are optional.

### Example

In this example, the rule `PromotionLevel` provides a client with a promotion depending on the value of the shopping cart and past purchases. The rule has a priority value of `1,000,000`. The condition `ShoppingCart` returns an amount in variable `?a` and a client number in variable `?c`. The second condition, `Client`, returns a total amount spent in variable `?t` corresponding to the client number equal to `?c`. In the action part of the rule, `if` statements are used to identify which promotion, if any, corresponds to the client.

```
rule PromotionLevel {
 priority = 1,000,000;
 when {
 ?s:ShoppingCart(?a:amount;?id:clientNumber);
 ?c:Client(clientNumber == ?id; ?t:totalPurchaseToDate);
 }
 then {
 if (?a > 100.0 && ?t > 1000.0)
 insert Promotion() {level = 1;}
 else { if (?a > 100.0)
 insert Promotion() {level = 2;}
 else if (?t > 1000.0)
 System.out.println("Dear "+ ?c.name +
 ", we can offer you a 10% discount on a shopping
 cart valued at over $100 today.");
 }
 }
};
```

Three cases are treated:

- The current shopping cart is valued at over \$100 and past purchases have exceeded \$1000.
- The current shopping cart is valued at over \$100 and past purchases have not exceeded \$1000.
- The current shopping cart is valued at below \$100 and past purchases have exceeded \$1000.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[else \(in if\)  
rule](#)

## if (in ruleflow)

The `if` keyword executes one statement block or another, depending on a Boolean expression value.

### Purpose

This statement is a conditional ruleflow statement that executes one statement block or another depending on the return value of a Boolean expression.

### Context

In a ruleflow body

### Syntax

```
if (test)
 {ruleflowStatement}
[else {ruleflowStatement}]
```

### Description

The test can be any valid test, as in the Java™ programming language. If the Boolean expression returns `true`, the ruleflow statement block that follows the `if` is executed. If the expression returns `false`, the ruleflow statement block corresponding to the `else` part is executed. The `else` part is not mandatory.

### Example

```
flowtask main
{
 body =
 {
 while(!ending)
 {
 if (turn == Constants.Player1) ChooseMovePlayer1;
 else ChooseMovePlayer2;
 CheckMove;
 if (ending) break;

 UpdateDistance;
 ExpandObjects;
 DetectConnect4;
 if (ending) break;
 DetectGridFull;
 if (ending) break;
 ChangeTurn;
 }
 EndOfGame;
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[else \(in rule\)](#)

[body \(in flowtask\)](#)

[flowtask](#)

# import

The `import` keyword imports a Java™ class or package into a rule file.

## Purpose

This statement is used to specify which Java classes a ruleset uses.

## Context

At the top level of rulesets

## Syntax

```
import [packageName
.] className |packageName
.*;
```

## Description

The `import` declarations must be the first statements in a rule file. Import statements are not mandatory, but when present, they are always placed at the beginning of the rule file, before the ruleset header. After it is imported, the class or package is accessible from the rules. The scope of the `import` declarations is the ruleset file.

It is possible to import a specific class or an entire package.

For some Java packages, the `import` declaration is implicit.

```
import java.lang.*;
import ilog.rules.engine.*;
```

The rule interpreter adds these packages automatically, therefore you do not need to import any classes from these packages explicitly.

## Example

If you declare the following imports:

```
import java.util.*;
import userPackage.UserClass;
```

the rule interpreter reads:

```
import java.lang.*;
import ilog.rules.engine.*;
import java.util.*;
import userPackage.UserClass;
```

Here are some examples of class names that might be used as a result of the two imports above:

Class Name	Qualified Class Name
Vector	java.util.Vector
java.util.Vector	java.util.Vector
UserClass	userPackage.UseClass
Number	java.lang.Number
IlrRule	ilog.rules.engine.IlrRule

Parent topic: [IRL keywords](#)



# in

The `in` keyword supports relations between object values.

## Purpose

This statement is used in the condition part of a rule to express the relations between values.

## Context

Rule conditions

## Syntax

```
condition in expression;
```

## Description

Use this statement to test whether the value of the condition is a member of the set of values returned by the `in` expression. The expression can be an array, a `java.util.Vector`, `java.util.Enumeration`, or a `java.util.Collection` instance.

Using the `in` statement, you can retrieve not only objects in the working memory but also objects that are linked to other objects by fields or by method calls.

In combination with the `collect` statement, the `in` statement expresses logical statements such as `all`, `at least`, or `none` (see the examples below).

The `in` statement provides performance advantages since the pattern matching is done on a reduced number of objects and not on all the objects in the working memory.

## Example

The first example retrieves objects even if they are not in the working memory. The other examples combine `in` and `collect` statements.

### Example 1

This example uses the `in` statement to retrieve the items bought by the customer even if they are not inserted in the working memory. The variable `?d` returns the items bought. The method `boughtItems()` can return a user-defined array, a `java.util.Vector` value, or a `java.util.Enumeration` value.

```
?c: Customer();
?d: Item() in ?c.boughtItems();
```

### Example 2

This example uses the `in` statement within a `collect` statement to retrieve all the items bought by the customer with a price higher than 10. The `where` part uses the method `size()` to test that the collection has at least one element.

```
?c: Customer();
collect Item(price > 10) in ?c.boughtItems() where (size() > 0);
```

### Example 3

This example is similar to the previous example except that the `where` part now tests that all the items bought had a price higher than 10, by testing the equality of the results returned by the `size()` and `length()` methods.

```
?c: Customer();
collect Item(price > 10) in ?c.boughtItems() where (size() == ?
c.boughtItems().length());
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[collect](#)  
[exists](#)  
[from](#)  
[not](#)

## in — !in (predicates)

In any expression, the `in` keyword tests whether a value belongs to a collection and or the `!in` keyword tests whether a value does not belong to a collection.

### Purpose

A predicate to test whether a value belongs or does not belong to a collection.

### Context

Any expression

### Syntax

```
(1) [?var:] className (?var1 in|!in ?var2);
(2) [?var:] className (?var1 in|!in {? var1, var2, ..., varn});
```

### Description

Use the `in` keyword as a binary predicate in a condition to test whether an operand is part of another operand. The `!in` keyword does the negation of this test.

The operand can be expressed in two ways:

- In syntax (1), the operand type must be an array or implement the `java.util.Collection` package.
- In syntax (2), the values that compose the enumeration must be homogeneous. The enumeration contains either primitive types only or objects only.

The tests are done with equals, as in Java™ collections.

### Example

```
rule InTestEnum
{
 when {
 ?i: Integer(intValue() in {1, 2, 3});
 }
 then {
 out.println("InTestEnum rule");
 }
}

rule InTestCollection
{
 when {
 ?i: Integer(intValue() in getValues());
 }
 then {
 out.println("InTest rule");
 }
}
```

with:

```
function int[] getValues()
{
 int values = new int[2];
 values[0] = 1;
 values[1] = 2;
 return values;
}
```

**Parent topic:** [JRL keywords](#)

## in (ruleset parameter)

The `in` keyword defines a ruleset parameter.

### Purpose

This keyword is used in a ruleset declaration to define an `in` ruleset parameter.

### Context

Ruleset declarations

### Syntax

```
ruleset rulesetName
{
 [in typeName variableName;]
 [inout typeName variableName;]
 [out typeName variableName [= value];]
};
```

### Description

Ruleset parameters are a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: `in`, `inout`, and `out`.

You cannot specify an initial value in the ruleset for the `in` and `inout` parameters because they are initialized by the [setParameters](#) method. However, you can initialize the `out` parameter.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruleset](#)

# initialaction

The `initialaction` keyword declares an initial action.

## Purpose

This keyword is used to declare an initial action in a ruleflow task.

## Context

Rule task definitions

## Syntax

```
initialaction {action1 ... actionnn}
```

## Description

A subflow task can have an initial action or a final action, or both. Initial actions are executed before the task body. Final actions are executed after the task body. They consist of inline IRL code, such as IRL functions. They are similar to an IRL function with the void return type and no arguments.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[flowtask](#)

[functiontask](#)

[ruletask](#)

# inout

The `inout` keyword defines a ruleset parameter that is both an input parameter and an output parameter.

## Purpose

The keyword in a ruleset declaration to define a parameter value that is provided as input to a ruleset when it is executed and can be modified by the execution process to produce an output value when the execution is completed.

## Context

At the top level of rulesets

## Syntax

```
ruleset rulesetName
{
 [in typeName variableName;]
 [inout typeName variableName;]
 [out typeName variableName [= value];]
};
```

## Description

Ruleset parameters are a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: `in`, `inout`, and `out`.

You cannot specify an initial value in the ruleset for the `in` and `inout` parameters because they are initialized by the [setParameters](#) method. However, you can initialize the `out` parameter.

**Parent topic:** [JRL keywords](#)

**Related reference:**  
[ruleset](#)

# insert

The insert keyword inserts an object into the working memory.

## Purpose

This statement is used in the action part of rules or in functions to create an object and insert it into the working memory.

## Context

Functions or rule actions

## Syntax

```
insert [event[(timeExpression)] | logical] object | typeName[arguments]
[{statement1 ... statementn}] ;
```

## Description

The insert statement creates a new object with the *typeName* [arguments], executes statements on the scope of the object, and inserts the object into the working memory. When you specify *typeName*, you must specify the type of the created object and *arguments* are the arguments that you pass to the constructor. If you do not pass any arguments, the object is created with the default constructor. To allow the rule engine to apply the constructor, you must declare it as public in your Java™ code.

Optionally, statements can follow the object to be inserted into the working memory. Such statements operate implicitly on the inserted object. They are executed before the object is inserted into the working memory. If the statement block contains only one statement, the braces ({} ) are not required.

When an object is inserted into the working memory as an event, a timestamp is automatically associated with it. The *timeExpression* expression defines an integer value that is used as the timestamp when an event is inserted.

You can remove an event object explicitly from the working memory by using the retract statement or the `IlrContext.retract` method.

**Note: The usage of the event (in rule actions) keyword is deprecated as of V7.5.**

To specify the uniqueness of the object, you must redefine the `java.lang.Object.equals` method in your Java class (see the example below). The process goes on as follows:

1. An object is created normally from the specified constructor.
2. This object is then tested by the `equals` method to determine whether an object that equals this object already exists in the working memory.
  - If an existing object in the working memory equals the object, the new object is not inserted into the working memory. The existing object is set to have a new justification that corresponds to the condition part of the rule that has just been executed.
  - If no existing object in the working memory equals the object, the new object is inserted into the working memory. This object is then maintained by the condition part of the rule that inserted it.

Maintenance of a logical object means that, as long as the condition part of a rule that justified the object remains true, the object is kept in the working memory. If the condition part becomes false, the object loses a justification. A logical object that loses its last justification is automatically retracted from the working memory.

To benefit of the Truth Maintenance System, you must redefine the `java.lang.Object.equals` method and define a `hashCode` method in your Java class, as follows:

```
public boolean equals (Object obj)
{
 if (obj == null || !(obj instanceof className))
 return(false);
 className myObj = (className)obj;
 return (fieldName.equals(myObj
j.fieldName));
}
...
```

```
public int hashCode()
```

```
{
 return (fieldName.hashCode());
}
```

**Note:**

If more than one field discriminates the object, the equals and hashCode methods must apply to each field.

Optionally, after an object has been inserted into the working memory, you can have a daemon run on them. To execute a daemon, the class must implement the interface [IlrAssertDemon](#) and define the method [asserted](#). For example, these code lines print a message every time a `className` object is inserted into the working memory:

```
public void asserted(IlrContext context)
{
 System.out.println("Asserted className
object in memory
 with fieldName: " + fieldName
);
}
```

**Example**

Here are three different ways of using the insert keyword.

**Example 1**

This example uses an insert statement to insert an object into the working memory and another to pass an object value.

```
integer v = new Integer(2);
insert v;

insert Integer(13)
{
 System.out.println("Inserting the value: " + intValue());
}
```

This example runs as follows:

1. The first line creates an Integer object.
2. The first insert statement inserts this object into the working memory.
3. The second insert statement passes the value 13 as a parameter.
4. The engine interprets the block of executable statements as applying to the current object. When the method `intValue` is encountered, it is considered as the call to the method `intValue` on the current object, which returns 13 as a result.
5. The print statement prints out:

```
Inserting the value: 13
```

and the object is then inserted into the working memory.

**Example 2**

This example calls a constructor with no parameters, then executes the statement block. The code assigns values to the fields `color` and `shape` of the class `Form` *before* the object is inserted into the working memory.

```
insert Form()
{
 color=Red;
 shape=Circle;
}
```

**Example 3**

This example uses logical objects to manage alarms in a chemical plant.

```
rule CheckTemperature {
 when {
 Sensor(type==Temperature; value>150);
 }
 then {
 insert logical (new Alarm());
 }
};
rule CheckPressure {
 when {
 Sensor(type==Pressure; value>2);
 }
 then {
 insert logical Alarm();
 }
};
```

The CheckTemperature rule issues an alarm if the temperature is greater than 150. The CheckPressure rule issues the same alarm if the pressure is greater than 2. Here, the purpose is to have a single alarm if both temperature and pressure are exceeded.

If the rule CheckTemperature is executed, an alarm is created. Because the object is logical, it is maintained by the condition of the rule. If the temperature changes and is no longer greater than 150, the alarm is automatically removed from the working memory. However, if the temperature does not change and the rule CheckPressure is executed, instead of inserting a new alarm, the code justifies the existing alarm a second time, based on the condition part of the rule.

In an insert statement, you can write the object to be inserted in either of the following ways;

- Write the keyword `new` before the object and use parentheses around the object, as in the first example.
- Use neither parentheses nor the `new` keyword, as in the second example.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[event \(in rule conditions\)](#)  
[retract](#)



# instanceof

The instanceof keyword tests whether an expression can be cast into a type.

## Purpose

This keyword is used in the condition part or in the action part of rules and in functions to test whether an expression can be cast into a referenced type.

## Context

Any expression

## Syntax

```
[?var:] className (expression instanceof type)
```

## Description

Use the instanceof operator in the condition part or in the action part of rules and in functions to test whether the passed expression (an operand of instanceof) can be cast into the passed type (another operand of instanceof) without raising a `ClassCastException` exception. If the expression is passed successfully, this test returns `true`. It returns `false` if the test is unsuccessful.

## Example

```
rule InstanceofTest {
 when {
 ?a: Alarm(equipment instanceof Multiplex);
 }
 then {
 System.out.println("Alarm occurs on Multiplex");
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[as](#)

# instances

The instances keyword declares class instances.

## Purpose

This keyword is used in a ruleset declaration to declare class instances.

## Context

At the top level of rulesets

## Syntax

```
ruleset rulesetName
{
 instances (className) = value|{value1,...,valuen};
};
```

## Description

Use the instances keyword to declare on which instances of a specified class the rule engine works, instead of looking for objects of this class in the working memory.

There are two ways to define class instances:

- Give a value whose type implements the `java.util.Collection` package. This collection contains the objects of the specified class on which the rule engine works.
- Explicitly specify the instances used by the engine. Values are given to compute these instances. The value type is derived from the specified class.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[ruleset](#)

# isknown

The `isknown` keyword tests whether a class attribute has an initialized value.

## Purpose

This keyword is used in the condition part or in the action part of a rule or in a function to test whether the attribute has a value.

## Context

Any expression

## Syntax

```
[?var:] className (?attribute isknown);
```

## Description

In the case of XML binding, an attribute has no value in the following cases:

- When the value is not given in the XML file, or
- When the value is given in the XML file and set to `xsi:nil`.

In the case of Java™ classes, the test always returns `true` because a Java attribute of a Java class is always considered initialized.

## Example

```
rule TestknownValue {
 when {
 ?u: User(address isknown);
 }
 then {
 sendToAddress(?u.address);
 }
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[isunknown](#)

## isunknown

The `isunknown` keyword tests whether a class attribute has an initialized value.

### Purpose

This keyword is used in the condition part or in the action part of a rule or in a function to test whether the attribute has a value.

### Context

Any expression

### Syntax

```
[?var:] className (?attribute isunknown);
```

### Description

In the case of XML binding, an attribute has no value in the following cases:

- When the value is not given in the XML file, or
- When the value is given in the XML file and set to `xsi:nil`.

In the case of Java™ classes, the test always returns `false` because a Java attribute of a Java class is never considered as not initialized.

### Example

```
rule TestUnknownValue {
 when {
 ?u: User(address isunknown);
 }
 then {
 askForAddress(?u);
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**  
[isknown](#)

## logical (in wait)

In a `wait` statement, the `logical` keyword designates that conditions must remain true for a `wait` statement to become true.

### Purpose

This keyword is used in the `wait` statement to test whether conditions become valid or remain true during a designated waiting period.

### Context

Rule conditions

### Syntax

```
(1) wait logical [[until] expression] [{condition}];
(2) ?variable
: wait [logical] [until] expression {condition}
 ...
 then ...
 timeout ?variable {action}
```

### Description

In `wait` statements, use the keyword `logical` to indicate that the previously satisfied conditions must remain true for the `wait` statement to become true. If you do not use the `logical` keyword, all the previously satisfied conditions are ignored: they might become false during the waiting period and the `wait` statement might succeed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[wait](#)

## match

The match keyword deals with hierarchical property values.

### Purpose

The match predicates are used to handle hierarchical property values.

### Context

Any expression involving objects

### Syntax

```
rule property access value
match
string value;
rule property access value
 match up
string value;
rule property access value match updown
string value;
rule property access value match down
string value;
```

### Description

The match predicates have two arguments: the first argument is an access to a rule hierarchical property. To make this access possible, you must define the rule property in a propertydefinition section. The second argument is a string.

### Example

This example uses the following element:

- The Geography property, a hierarchical property that has been defined as follows:

```
hierarchy Geography { "North"
 { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } }
 "Europe" { "France" { } "Germany" } } }
```

- The location property, a rule property of type Geography,
- The ?rule variable

You can use the following keywords:

#### match

?rule.?location match "California" returns true if the location value is equal to California.

#### match up

?rule.?location match up "California" returns true if the location value is one of the values from the hierarchy root to California node.

#### match down

?rule.?location match down "California" returns true if the location value is one of the values from the "California" node to the hierarchy leaves (children of California node).

#### match updown

?rule.?location match updown "California" returns true if the location value either matches up or matches down California.

In this example, the rule task selects all the rules except the rule France.

```
hierarchy Geography { "North"
 { "USA" { "California" { "San Francisco" } "Texas" { "Houston" "Paris" } } }
```

```
 "Europe" { "France" { } "Germany" } }
}
propertydefinition { Geography location; }
rule usa { property location = "USA"; when { ... } then { ... } }
rule california { property location = "California"; when { ... } then { ... }
}
rule france { property location = "France"; when { ... } then { ... } }
ruletask main
{
 body = dynamicselect()
 {
 Collection rules = collect IlrRule(?this.?location match up "California")
 in getRuleset().getAllRules();
 return rules;
 }
}
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[hierarchy](#)  
[rule](#)

## modify

The modify keyword modifies objects and updates the agenda.

### Purpose

This statement is used in the action part of rules or in functions to modify objects and updates the agenda accordingly.

### Context

Functions or rule actions

### Syntax

```
modify [refresh] object {statement1 ... statementn};
```

### Description

The statement block after the object argument can modify the state of an object of the working memory. Those statements can be arithmetic expressions or method calls. When an object of the working memory is modified, the agenda is also updated. If the block contains only one statement, the braces ({} ) are not required.

#### Note:

In a modify statement, the statements that modify the object are specified directly in the statement block that follows the object, as shown in the syntax above. This is different from the update statement, where the modifications are specified before the update keyword, independently of the update statement.

If you apply the refresh keyword, the rules that remain true or become true after the modification of the object are reinserted into the agenda. Without the refresh keyword, only the rules that become true as a result of the modification are inserted into the agenda.

After the object has been updated in the working memory, the update can optionally launch a daemon. For this purpose, the class must implement the [IlrUpdateDemon](#) interface and must define the updated method (see the [update](#) keyword for details).

### Example

```
rule InventoryUpdate {
 priority = 1,000,000;
 when {
 Sold(item == book; ?ISBN:ISBN);
 ?b:Book(?ISBN == ISBN; currentStock > 0);
 }
 then {
 modify ?b {
 currentStock-=1;
 }
 }
};
```

The InventoryUpdate rule has a 1,000,000 priority. The first condition matches an object Sold with field item equal to book and instantiates the variable ?ISBN with the value of the field ISBN. The second condition uses the variable ?ISBN to match an object Book and tests that the field current\_stock is greater than 0. The condition is true only if the current stock is one or more. When both conditions are fulfilled, the action part can be executed. The object referenced by ?b is updated in the working memory by the modify statement; the field currentStock is decremented by one.

```
import tmp.*;

rule IncrementClock {
 when {
 ?t:Time();
 }
 then {
 modify refresh ?t {
 ?t.setSeconds(?t.getSeconds()+1);
 }
 }
};
```



```
};
 }
}
```

If the refresh keyword were not used, the rule would be executed only once. The refresh keyword is used to reinsert the rules matching a Time object into the agenda. In the example, the rule is executed forever.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[refresh](#)

[update](#)

## new

The new keyword creates a new object or array.

### Purpose

This allocation expression is used in the action part of a rule or function to create a new object.

### Context

Functions or rule actions

### Syntax

```
new className
(
arguments
) ;
new className
[
dimension
] [
dimension
]* { initialValues
} ;
```

### Description

In the case of arrays, you can specify how to initialize it with the `initialValues` method.

### Example

```
function Integer getAnInt(int value)
{
 return new Integer(value);
}
function Integer[][] getAnArray(int dim1, int dim2)
{
 return new Integer[dim1][dim2];
}
function Integer[][] getAnInitializedArray()
{
 int[][] array2 = {{1,2}, {3,4}, {5,6,7}};
 return array2;
}
```

This example consists of three functions:

- `getAnInt` returns an `Integer` object.
- `getAnArray` returns a two-dimensional array with lengths equal to `dim1` and `dim2`.
- `getanInitializedArray` returns a two-dimensional array. The first dimension is itself an array of length 3, that is, the number of braced elements in the `initialValues` list.

Each array consists of an `Integer` array:

1. The first one is an array of length 2, initialized with the `Integer` values 1 and 2.
2. The second one is an array of length 2, initialized with the values 3 and 4.
3. The last one is an array of length 3, initialized with the values 5, 6, and 7.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[function](#)  
[return](#)

## not

The not keyword tests the negation of a rule condition.

### Purpose

This statement is used in the condition part of rules to test the negation of a rule condition.

### Context

Rule conditions

### Syntax

```
not condition;
```

### Description

You use the not statement in the condition part of rules, to a simple condition, an in statement, or a from statement. The not statement returns true if the specified condition is false. A not condition returns true for a simple condition if the working memory does not contain any object that can match the condition. A not statement returns true with an in or from statement when the in or from statement returns false. Use not for the contrary of both exists *class* or just *class*.

A not condition cannot be bound to an external variable. The not condition returns true when no object in the working memory matches the condition. Hence it is invalid to bind a false condition to a variable.

Any variable bound within the not condition is local to the condition. The remainder of the rule cannot contain any reference to it.

You can use the special variable ?this within the not statement to designate the current working memory object being tested.

### Example

```
rule GuestPromotion {
 when {
 ?g:Guest(?i:id);
 not GroupId(this.contains(?i));
 }
 then {
 ?g.promotions();
 }
}
```

The GuestPromotion rule verifies that a guest is not part of a group, in order to provide a promotion. The first condition uses the variable ?g to reference a Guest object and the variable ?i to return the field value id. The not statement is used to test whether this id is contained in the object GroupId. The method contains is applied to the variable ?this. The then part launches the method promotions().

```
rule LargestPercentStockChange {
 when {
 ?s: Stock(?c:currentValue; ?l:lastClosing; ?p:(?c-?l)/?l);
 not Stock(?c1:currentValue; ?l1:lastClosing; ?p < (?c1-?l1)/?l1);
 }
 then {
 System.out.println(?s.ticker + "has the largest change of: "+ ?p +"%");
 }
};
```

The purpose of the LargestPercentStockChange rule is to print out the stock with the largest percent change.

- The first condition, in the line starting with ?s, specifies a stock, and the rule variables named ?c and ?l are bound to the current stock value and the last closing value, respectively. The stock percent change is calculated and saved as ?p.
- In the second condition, the line contains the not keyword. As in the previous condition, a percent change is calculated with rule variables ?c1 and ?l1. The condition then compares the result with the previous calculated change ?p. The not keyword causes the condition to fail for any percentage change except the largest.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[collect](#)  
[exists](#)  
[from](#)  
[fork](#)  
[in](#)

## occursin

The occursin keyword defines a unary temporal constraint in an event condition.

### Purpose

This keyword is used in the condition part of rules in an event condition to express a temporal constraint on a single event.

### Context

Rule conditions

### Syntax

```
[?var:] event className (?eventVar occursin interval);
```

### Description

#### Deprecated as of V7.5.

An occursin test is satisfied if the timestamp of the event is included in the interval.

An interval is of the form [*lowerBound*, *upperBound*L], where a bound is either an expression evaluating to an integer, or the \$ sign which denotes infinity.

### Example

```
rule CheckErrorReport {
 when {
 ?u: User();
 ?a1: not event Check(?this occursin [1, 60]);
 } then {
 ?u.report(?a1);
 }
};
```

An instance of the CheckErrorReport rule is created on each instance of the User class in the working memory when the timestamp of the Check event ?a1 does not occur between 1 and 60.

**Parent topic:** [IRL keywords](#)

#### Related reference:

[after](#)  
[before](#)  
[event \(in rule conditions\)](#)  
[timeof](#)

# ordering

The ordering keyword sets the order of the rules executed in a rule task.

## Purpose

This keyword is used to specify how the rules are ordered and how many rules are executed in a rule task.

## Context

Rule task definitions

## Syntax

```
ruletask ruleTaskName
{
 [ordering = dynamic|sorted|literal;]
};
```

## Description

The keyword ordering specifies how the rules are sorted. It takes these values:

- **dynamic**: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- **sorted**: The rules are sorted in decreasing order of static priority.
- **literal**: The rules are listed in the same order as they have been listed in the rule task body.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[dynamic](#)  
[ruletask](#)

# out

The out keyword defines a ruleset output parameter.

## Purpose

This keyword is used in a ruleset declaration to define a ruleset output parameter to exchange data between the application and the ruleset.

## Context

At the top level of rulesets

## Syntax

```
ruleset rulesetName
{
 [in typeName variableName;]
 [inout typeName variableName;]
 [out typeName variableName [= value];]
};
```

## Description

Ruleset parameters are accessible from a rule, a function, or a task definition in the ruleset.

Rulesets support three types of parameters: in, inout, and out.

You cannot specify an initial value in the ruleset for the in and inout parameters because they are initialized by the [setParameters](#) method. However, you can initialize the out parameter.

The out ruleset variables can be initialized in the ruleset.

You access the in, inout, and out parameters through the [EngineInput](#) and [EngineOutput](#) objects that are created when you execute your ruleset. Use the [setParameters](#) method to initialize the in and inout parameters.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruleset](#)

## overriding, overrides

The overriding and overrides keywords declare overriding relations.

### Purpose

The overriding and overrides keywords are used to declare overriding relations.

### Context

At the top level of rulesets

### Syntax

```
overriding { "groupName1" overrides "groupeName2";
 "groupName3" overrides "groupName4", "groupName5"; }
```

### Description

You can organize rules in groups by using an `ilog.rules.group` rule property. The overriding keyword introduces overriding declarations which apply to the rules that compose a rule task and take groups as parameters. In the parameter block, the `overrides` keyword indicates which rule group is executed instead of which other rule groups: a declaration in which `group1` overrides `group2` means that when rules of `group1` are together with rules of `group2` in a rule task, the rules of `group2` are removed from the task and therefore not executed. Rule overriding is the last rule selection done before the task is executed.

If a rule does not define an `ilog.rules.group` property, a default one is created. Its value is the fully qualified name of the rule.

### Example

```
import java.util.Collection;

hierarchy Geography
{
 "North" {
 "USA" {
 "California"
 ...
 }
 ...
 }
}

propertydefinition
{
 Geography location;
}

overriding
{
 "california" overrides "usa";
 "blacklisted" overrides "california";
}

rule usa
{
 property location = "USA";
 when {
 ...
 }
 then {
 ...
 }
}
rule california
{
```



```

 property location = "California";
 when {
 ...
 }
 then {
 ...
 }
}
rule blacklisted
{
 property ilog.rules.group = "blacklisted";
 when {
 ...
 }
 then {
 ...
 }
}

ruletask main
{
 body = dynamicselect(){
 Collection rules = collect IlrRule(?this.?location match up "California")
in getRuleset().getAllRules();
 return rules;
 }
}

```

This example defines a rule task that selects all the rules in which the location property matches up the value California. The resulting selection contains the rules in which the value of the location property is one of the values in the Geography hierarchy from the root (North) to the California node.

The collected rules are usa and california. These rules have not defined an `ilog.rules.group` property. Therefore, the property value for these rules is their fully qualified names, here `usa` and `california`. The overriding declaration removes the `usa` rule from the selection and keeps only the `california` rule.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruletask](#)

# package

The package keyword declares a package.

## Purpose

This keyword is used to declare rule packages that you can use to better manage IRL artifacts such as variables, functions, rules, and tasks.

## Context

At the top level of rulesets

## Syntax

```
package packageName {variables, functions, rules, tasks}
```

## Description

Rule packages are held by the ruleset. A ruleset contains at least one package, the default package. The default package name is the empty string. IRL artifacts defined outside a package definition are considered to be part of the default package. An IRL artifact has a short name, which is the name used for its definition, and a fully qualified name, which is its short name prefixed with its package name. You can use IRL artifacts defined in a package A in another package B by either referencing the artifacts by their fully qualified name or if their package is imported into the package B, by using the use keyword. You cannot import the default package, but you can import the artifacts of the default package.

## Example

```
function void displayPrice(double price)
{
 ...
}
package pricing
{
 variables {
 Customer customer;
 }
 rule isEligible
 {
 when {
 ...
 }
 then {
 ...
 }
 }
}

package europe.pricing
{
 use pricing.*;
 use displayPrice(double);
 ruletask main
 {
 body {
 isEligible
 }
 finalaction {
 displayPrice(3.2d);
 }
 }
}
rule FlightDisplayExpired {
};
```

This example defines two packages, pricing and europe.pricing. The displayPrice function belongs to

the default package. The `europa.pricing` package uses the `pricing` package and the `displayPrice` function.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[flowtask](#)

[function](#)

[functiontask](#)

[rule](#)

[ruletask](#)

[use](#)

# property

The `property` keyword declares a rule, ruleset, or ruleflow property.

## Purpose

This keyword is used to declare specific properties for a rule, a ruleset, or a ruleflow in a rule file.

## Context

Rulesets, ruleflows, and rules

## Syntax

```
property propertyName = value;
```

## Description

The *propertyName* argument is any identifier and value is of type `String`.

You can use the `property` keyword at two levels: the ruleset level and the individual rule level.

- At ruleset level, the property declaration must be shown after the `import` statement.
- At rule level, the property declaration is part of the rule header.

Users can declare properties at both the ruleset level and the rule level.

As a ruleflow property, this property is used to define a user property on the task. The engine does not interpret this property. You can retrieve its value later by using the API.

## Example

The first example shows a ruleset property. The second example shows a rule property.

### Example 1

When used at the ruleset level, as in this example, the `property` keyword is enclosed by a declaration that begins with the word `ruleset`, followed by any identifier. Two user-defined properties store the first and last names of the ruleset author.

```
ruleset Vacation {
 property authorlastname = "Brans";
 property authorfirstname = "Sue";
}
```

### Example 2

When used at the rule level, the `property` keyword is part of the header declarations. A user-defined property declares the date at which the rule was last changed. Note that the value is of type `String`.

```
Rule CdUpdate {
 priority = 100;
 property lastChangeDate = "22 02 06";
 when { ...
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[flowtask](#)

[functiontask](#)

[rule](#)

[ruleset](#)

[ruletask](#)

## propertydefinition

The `propertydefinition` keyword predeclares the types of rule properties.

### Purpose

This keyword is used to predeclare the type of rule properties to make rule properties accessible as regular members of the rule and to make type checking possible.

### Context

At the top level of rulesets

### Syntax

```
propertydefinition { Type propertyName; }
```

### Description

You must define rule properties in the `propertydefinition` section. For example, if a `location` property is defined in the `propertydefinition` section, you can write in a rule selection body:

```
body = select(?rule)
{
 String location = ?rule.location;
 return location.equals("London");
}
```

For hierarchical properties, use the hierarchy name as the property type. The property value is type-checked and considered as a string.

### Example

```
hierarchy Geography
{
 "North America" {
 "USA" "Canada"
 }
}

propertydefinition
{
 Geography location;
 java.util.Date effectiveDate;
}

rule usa
{
 property location = "USA"
 property effectiveDate = java.util.Date(120000);
 when {
 ...
 }
 then {
 ...
 }
}

function void displayRuleProperties()
{
 IlrRule[] rules = getRulesetI().getAllRules();
 for(int i=0; i<rules.length; i++)
 {
 out.println("Location of " + rules[i].getName() + " is " +
rules[i].?location);
 out.println("Date of " + rules[i].getName() + " is " +
```

```
rules[i].?effectiveDate);
 }
}
```

A Geography hierarchy is defined. The propertydefinition section predeclares two properties, location and effectiveDate. The usa rule has two properties: location and effectiveDate. Their values are compatible with the type declared in the propertydefinition statement. The displayRuleProperties function displays the properties of the rules (here only usa), made easily accessible by the propertydefinition statement.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[hierarchy](#)  
[rule](#)

# refresh

The refresh keyword requests a refresh of the agenda.

## Purpose

This keyword is used in the modify or update statement to request a refresh of the agenda after an update.

## Context

Functions or rule actions

## Syntax

```
modify [refresh] object {statement1 ... statementn};
update [refresh] object;
```

## Description

If you specify the refresh keyword, the rules that remain true or become true after the modification of the object are reinserted into the agenda. If you do not specify the refresh keyword, only the rules that become true as a result of the modification are inserted into the agenda.

**Note:** In the decision engine you must add the **com.ibm.rules.engine.repeatabl**e property, see [Repeatabl](#)e rules.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[modify](#)

## retract

The retract keyword removes an object from the working memory.

### Purpose

This keyword is used in the action part of rules or in functions to remove an object from the working memory.

### Context

Functions or rule actions

### Syntax

```
retract object;
```

### Description

The object can be defined by a variable or by an expression that denotes an object associated with the current rule engine or the engine that you specify (context in the code sample below).

After the object has been retracted from the working memory, the removal can optionally execute a daemon. To execute a daemon, the object class must implement the interface [IlrRetractDemon](#) and define the method `retracted`.

### Example

The first example shows a basic retract statement while the second example shows the implementation of a daemon after the object is removed.

#### Example 1

```
rule FlightDisplayExpired {
 when {
 ?f: Flight(status==ARRIVED;
 landingTime + 30 < currentTime());
 }
 then {
 retract ?f;
 }
};
```

The `FlightDisplayExpired` rule removes a `Flight` object after the arrival time has passed 30 minutes. The first condition returns a `Flight` object in the variable `?f`. The object field `status` must be equivalent to the static constant `ARRIVED`. The field `landingTime + 30` must be less than the time returned by the method `currentTime()`. If the condition is true, the object `?f` is removed from the working memory by the `retract` statement.

#### Example 2

```
public void retracted(IlrContext context)
{
 System.out.println("Retracted a className object from
 memory with fieldName: " + fieldName);
}
```

This example prints a message each time a `className` object is retracted from the working memory.

**Parent topic:** [IRL keywords](#)



# return

The return keyword returns to the caller.

## Purpose

The return statement returns to the caller of the code that contains this statement.

## Context

Only in IRL functions or inline IRL code in a task definition, such as initial actions, final actions, the action task body, an agenda filter, or a rule task body defined with the select or dynamicselect keyword.

## Syntax

```
return [expression];
```

## Description

A return statement with no expression is allowed in a function with the return type void. In the task definition, such a statement is authorized in initial actions, final actions, and the action task body.

A return statement with an expression is allowed in a function with any return type other than void. The type of the returned expression must be assignable to the function return type.

A return statement is not allowed in a rule on either side. It causes a parsing error.

## Example

### Example 1

This example illustrates a return statement with an expression. A function returning an int is declared. The return statements in this function are followed by int expressions.

```
function int getOtherPlayer(int player)
{
 if (player == Constants.Player1) return Constants.Player2;
 else return Constants.Player1;
}
```

### Example 2

This second example illustrates a return statement with an expression. The return statement is shown in a rule task body declared with the select keyword. The returned value is of boolean type.

```
ruletask ExpandObjects
{
 ordering = dynamic;
 firing = allrules;
 body = select(?rule)
 {
 String ?task = ?rule.getProperties().getString("task","");
 return ?task.equals("ExpandObjects");
 }
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[function](#)

## rule

The rule keyword declares a rule.

### Purpose

This keyword is used to declare a rule.

### Context

At the top level of rulesets

### Syntax

```
rule ruleName {[priority = value;]
 [property propertyName = value;]
when {condition1 ... conditionn [evaluate (expression)]}
then {[action1 ... actionm]}
[else {[action1 ... actionp]}]
};
```

### Description

A rule in IRL is composed of three parts:

- A header part, which defines the name of the rule, its priority, and properties.
- The condition part, which begins with the keyword when, also referred to by its side of the rule.
- The action part, which begins with the keyword then, also referred to by its side of the rule.

When the condition part ends with an evaluate statement, the action part can have an else part, executed if the evaluate statement returns false. If it returns true, the then part is executed.

If a formal comment (/\*\*...\*/) precedes the rule definition, the comment is saved so that it can be retrieved later using the API.

### Example

```
rule VideoCallBilling {
 when {
 ?c:Customer(?p:phoneNo);
 ?v:collect (new videoCallCollection())
 Usage(phoneNo == ?p; type == videoCall)
 where (size() > 0);
 }
 then {
 float ?t = 0.;
 Enumeration ?enum = ?v.elements();
 while (?enum.hasMoreElements()) {
 Usage ?x = (Usage)enum.nextElement();
 ?t += ?x.charge();
 }
 System.out.println("Dear "+ ?c.name + "Your bill for
 Video conference calls is : "+ ?t);
 }
};
```

The VideoCallBilling rule collects appropriate Usage objects, sums the charge, and prints the result with the client's name. The first condition returns the Customer object in variable ?c, with the field phoneNo stored in variable ?p. The second condition is a collect statement. This statement returns a videoCallCollection object in variable ?v. This collection object contains Usage objects with the field phoneNo equal to the variable ?p, phoneNo from the previous condition, and field type equal to videoCall. The where part of the collect statement verifies that one or more Usage objects have been collected. The then part of the rule declares two variables ?t, initialized to 0, and variable ?enum, set to the elements of the collection using the default collection method elements(). Using the while statement, the elements of the collection are enumerated and the Usage objects are accessed and referenced using the variable ?x. Variable ?t and the method charge() are used to sum the charge for each Usage object. The println statement prints the client's name and the total charge.

---

```
rule CheckTemperature {
 priority = 1,000,000 + 1;
 when {
 Sensor(type==Temperature; value>150);
 }
 then {
 insert Alarm();
 }
};
```

The CheckTemperature rule has a priority of 1,000,001. If an object Sensor with a field type equal to Temperature has a value greater than 150, the condition part is true and the action part is executed. An object Alarm is inserted into the working memory using the command insert.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[evaluate](#)

[property](#)

[ruleset](#)

[ruletask](#)

[then](#)

[when](#)

## rule (in ruletask)

The rule keyword sets a single rule as eligible for execution.

### Purpose

This keyword is used in rule tasks to set a single rule as eligible to be executed.

### Context

Rule tasks

### Syntax

```
ruletask ruleTaskName
{
 firing = allrules|rule;
};
```

### Description

The value that follows the firing keyword indicates whether all rules are executed (allrules, the default) or whether only one rule is executed (rule). When the firing value is set to rule, it means that all instances of the first eligible rules are executed.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruletask](#)

## ruleset

The ruleset keyword declares a ruleset.

### Purpose

This keyword is used to declare a ruleset.

### Context

At the top level of rulesets

### Syntax

```
ruleset rulesetName
{
 [property propertyName = value;]
 [instances (className) = value|{value1,...,valuen};]
 [hasher (typeName variableName) = value;]
 [in typeName variableName [= value];]
 [inout typeName variableName [= value];]
 [out typeName variableName [= value];]
};
```

### Description

It is not mandatory to declare rulesets. If you choose to do so, you must specify at least the ruleset name.

In addition to the ruleset name, a ruleset declaration can also contain the following elements:

- **property**

You can define properties on the ruleset.

- **instances**

You can specify class instances to declare on which instances for a specified class the rule engine works. If you do not specify class instances, the engine looks for objects of this class in the working memory.

There are two ways to define class instances:

- Give a value whose type implements `java.util.Collection`. Such a collection contains the objects of the specified class on which the rule engine works.
- Explicitly specify the instances used by the engine. Values are given to compute these instances. The value type is derived from the specified class.

- **hasher**

You can define a hashing expression to optimize rule execution.

- **in, out, inout**

You can define ruleset parameters as a simple way to exchange data between the application and the ruleset. Ruleset parameters are accessible from a rule, function, or task definition in the ruleset.

### Example

```
ruleset Connect4
{
 // Provides the grid, the player and the adversary
 in Grid grid;
 out SavedGame saved;

 // The latest move
 int turn;
 Position move;

 // Who's the winner, what is the connect4, and any other reason
 // to end the game.
 int winner = Constants.None;
```

```
Connect4 connect4;
boolean ending = false;
String message;

// Where to find the position objects.
instances(Position) = ?grid.getAllPositions();
};
```

This example illustrates a ruleset declaration. The ruleset name is Connect4. Ruleset parameters are declared. One of them, named grid, is an input parameter. Its initial value is set by a call to `IlrContext.setParameters`. The parameter saved is an output parameter. Its value can be obtained from the application with the different API calls listed above. The ruleset variables turn, move, winner, connect4, ending, and message are local.

Instances are defined for the class Position. Each object of type Position is bound in a rule condition part. The engine does not look for those objects in the working memory but in the specified list obtained with `?grid.getAllPositions()`.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[flowtask](#)

[function](#)

[functiontask](#)

[import](#)

[property](#)

[rule](#)

[ruletask](#)

# ruletask

The ruletask keyword declares a rule task.

## Purpose

The keyword used to declare a rule task.

## Context

At the top level of rulesets

## Syntax

```
ruletask ruleTaskName
{
 [property propertyName = value;]
 [algorithm = default|sequential;]
 [matchedclasses = matchedClasses]
 [iterator = value;]
 [ordering = dynamic|sorted|literal;]
 [firing = allrules|rule;]
 [firinglimit = integer value;]
 [agendafilter = agendaFilter]
 [initialaction {action1 ... actionm}]
 [finalaction {action1 ... actionn}]
 [completionflag = value;] [scope = scope]
 [body body]
};
matchedClasses is given a value in one of two ways:
matchedclasses = {class1, class2, ..., classn}
matchedclasses = value;
agendaFilter is defined in one of two ways:
agendafilter = filter(?instance) {action1 ... actions}
agendafilter = value;
body is defined in one of three ways:
body {ruleName1, ruleName2,..., ruleNamep}
body = select(?rule) {action1 ... actionq}
body = dynamicselect(?rule) {action1 ... actionr}
```

## Description

A rule task is one of the three kinds of tasks available in a ruleflow. A rule task lists the rule or rules that define the rule task.

Note:

1. The ruleflow syntax is described in [Grammar specification](#).

2. If a formal comment (/\*...\*/) precedes the task definition, it is saved so that you can retrieve it later using the API.

A rule task can have the following attributes:

- property

Use this attribute to define a user property on the task. The engine does not interpret this property. You can retrieve its value by defining API class and methods.
- algorithm

The value that follows this keyword indicates which execution mode is used to execute the rules: the RetePlus mode (the algorithm value is then default, which is its default value) or the sequential mode (the algorithm value is sequential).
- ordering, firing, firinglimit

A rule task is designed to execute rules. You can set up some rule execution parameters, such as how the rules are ordered and how many rules are executed.

The keyword `ordering` specifies how the rules are sorted.

You can have rules sorted according to the following values:

- **dynamic**: The rule instances are sorted in the agenda according to the traditional rules presented for the agenda.
- `agendafilter`

Use the `agendafilter` keyword in the rule task definition to filter the rules that are actually executed, according to specified criteria.
- `initialaction`, `finalaction`

You can define an initial action or a final action, or both, for a rule task. Initial actions are executed before the task body. Final actions are executed after the task body. They are composed of inline IRL code, such as IRL functions. They are similar to an IRL function with the `void` return type and with no arguments.
- `scope`

A scope defines the rules that can be used in the rule task, before the optional selection defined in the body is applied. The use of a scope is optional.
- `body`

A rule task consists of a list of rules that compose its body. You can specify the rule list either by passing the rule names explicitly (extension), or by using `select` or `dynamicselect` keywords so that the list is computed from the code (comprehension). The given code must return a `boolean` value.

The use of a task body is optional, but there must be at least a scope or a body.

## Example

### Example 1

In this rule task, the body consists of the rules `DetectConnect4` and `DetectGridFull`. The rule ordering is set to `literal`: the order in which the rules are listed in the body is kept for the execution.

```
ruletask DetectConnect4
{
 ordering = literal;
 body = { DetectConnect4, DetectGridFull }
};
```

### Example 2

In this rule task, the body is defined by comprehension. To compute the body, the code is executed for each rule in the ruleset. The rules for which the return value is `true` are part of the rule task body. The others are not part of the body.

The ordering here is set to `sorted`: the rules are sorted in decreasing order of static priority before being executed.

The firing value is set to `rule`: only one rule is executed among the rules that compose the body, even if more than one are eligible to be executed.

The initial actions are executed before the task body. If final actions are provided, they are executed after the task body.

```
ruletask ChooseMovePlayer1
{
 initialaction =
 {
 move = null;
 };

 body = select(?rule)
 {
 String ?task = ?rule.getProperties().getString("task","");
 return ?task.equals("ChooseMovePlayer1");
 }
 ordering = sorted;
 firing = rule;
};
```



**Parent topic:** [IRL keywords](#)

**Related reference:**

[agendafilter](#)

[body \(in ruletask\)](#)

[flowtask](#)

[functiontask](#)

## scope

The `scope` keyword defines a scope in a rule task.

### Purpose

This keyword is used to define a scope in a rule task, that is, a superset of the rules that compose the rule task.

### Context

Rule tasks

### Syntax

```
ruletask main { body = select(?rule) { ... }
 scope = {packageName1.*, ..., packageNameM.*,
 group(groupName1), ..., group(groupNameP),
 ruleName1, ..., ruleNameN} }
ruletask main { body = select(?rule) { ... } scope = all; }
```

### Description

You can define a scope with packages or rules. If you specify a package, all the rules of the package are part of the scope. When you choose to define a scope, the final task body cannot contain rules that are not part of the scope. Domains are converted into a scope. You cannot use domains and scope together; otherwise a parsing error is raised.

The `all` value for a scope means that the scope includes all the rules of the ruleset. This is equivalent to not defining a scope.

If the rule task runs in sequential mode and you have defined a scope, the bytecode generation is done at parsing time.

### Example

In this example, the rule task keeps all the rules defined in the scope (always returns true): `pricing.r1` and `europe.pricing.r2`.

```
package pricing
{
 rule r1
 {
 when {
 ...
 }
 then {
 ...
 }
 }
}
package europe.pricing
{
 rule r1
 {
 when {
 ...
 }
 then {
 ...
 }
 }
 rule r2
 {
 when {
 ...
 }
 then {
 ...
 }
 }
}
```

```
 }
 }
}

ruletask main
{
 scope = { pricing.*, europe.pricing.r2}
 body = dynamicselect(?rule) {
 return true;
 }
}
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in flowtask\)](#)

[body \(in ruletask\)](#)

[ruletask](#)

# select

The select keyword selects a rule in a rule task.

## Purpose

This keyword is used to select a rule in the list of rules that compose the rule task body.

## Context

Rule tasks

## Syntax

```
ruletask ruleTaskName
{
 body = select(?rule) {action1 ... actionq}
};
```

## Description

You can either specify each rule name explicitly (extension) or have the list of rules computed from the code, which uses the select or dynamicselect keywords (comprehension). The given code must return a value of the type boolean. The selection method changes the way the filter is applied to the rules.

**Note:** There is no difference between dynamicselect and select. They behave the same way in that the statement is called each time the task is executed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in ruletask\)](#)  
[dynamicselect](#)  
[ruletask](#)

# sequential

The `sequential` keyword specifies the execution mode of a rule task.

## Purpose

This keyword is used to select the sequential execution mode for a rule task.

## Context

Rule tasks

## Syntax

```
ruletask ruleTaskName
{
 algorithm = default|sequential;
};
```

## Description

The value that follows the `algorithm` keyword indicates which execution mode is used to execute the rules: the RetePlus mode (the `algorithm` value is then `default`, which is its default value) or the sequential mode (the `algorithm` value is `sequential`).

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[ruletask](#)

## switch

The switch keyword executes one statement block or another according to an integer expression value.

### Purpose

This ruleflow conditional statement is used to execute one statement block or another according to an integer expression value.

### Context

Ruleflow body

### Syntax

```
switch (expression)
{
 case value1:
 {ruleflowStatement}
 ...
 case valuen:
 {ruleflowStatement}
 default:
 {ruleflowStatement}
}
```

### Description

The integer expression *expression* is evaluated. If a case block evaluates the specified expression, the corresponding statement block is executed. If no block with the requested value is found, the default block is executed.

### Example

```
ruleset r
{
 int count;
}

flowtask main
{
 body =
 {
 switch(count)
 {
 case 1:
 {
 T1;
 }
 case 3:
 {
 T2;
 }
 default:
 {
 T3;
 }
 }
 }
};
```

This example illustrates a switch statement. The count variable on which the switch is executed is a ruleset variable in this example. When the switch is executed, the value of the count variable is evaluated. Depending on its value, one of the switch statement blocks is executed. If the count equals 1, the first case block (case 1) is executed. If it equals 3, the second case block (case 3) is executed. Otherwise, the default block is executed.

**Parent topic:** [JRL keywords](#)

**Related reference:**

[flowtask](#)

[body \(in flowtask\)](#)

# then

The then keyword declares the action part of a rule.

## Purpose

This keyword is used to specify the action part of a rule, also known as the right-hand side.

## Context

Rule actions

## Syntax

```
then {
 [insert|if|modify|update|retract|timeout|while]
 action1 ...
 [insert|if|modify|update|retract|timeout|while]
 actionn
}
```

## Description

It can contain one or more action statements to be done when the rule is executed, or it can be empty. The variables defined in the condition part of a rule can be used in the action part of the rule, except the variables defined within not and exists statements. The timeout action must be associated with a wait condition.

## Example

In this example, the ConnectivityUpdate rule tests whether a new node exists in the network and adds a link to such new node from each of its neighbors. The single rule condition matches an object Node when the field state equals the static value NEW and it binds the variable ?neighbors with the vector field neighbors(). If such a Node object is found, the action part can be executed. The variable ?i is initialized to 0. The for statement loops for each neighbor, updating the links of the neighbors with the new node by calling the method addLink(?n). The last action uses the update command to update the agenda concerning the new node.

```
rule ConnectivityUpdate{
 when {
 ?n: Node(state == NEW; ?neighbors: neighbors());
 }
 then {
 for (int ?i = 0; ?i < ?neighbors().size(); ?i++) {
 ((Node)?neighbors().elementAt(i)).addLink(?n);
 }
 update (?n);
 }
};
```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[else \(in rule\)](#)

[for](#)

[if](#)

[modify](#)

[retract](#)

[throw](#)

[try](#)

[update](#)

[wait](#)

[while](#)



# throw

The throw keyword throws an exception.

## Purpose

This statement is used in the action part of rules or in functions to throw an exception that makes control flow immediately to an exception handler.

## Context

Functions or rule actions

## Syntax

```
throw
expression
```

## Description

The throw statement requires a single expression that is a throwable object. Throwable objects are instances of any subclass of the Java™ Throwable class.

The exception in the throw statement is not the exception actually thrown. The thrown exception is an instance of the [IlrUserRuntimeException](#) class, a subtype of [IlrRuntimeException](#), which encapsulates the exception thrown in the throw statement. The inherited [getTargetException](#) accessor retrieves the exception thrown in the throw statement. This method has the following signature: public Throwable getTargetException()

## Example

```
function void replaceValue(String name, Object newValue)
{
 String ?attr = find(name);
 if (?attr == null)
 throw new NoSuchAttributeException(name,this);
 ?attr.valueOf(newValue);
}
```

This example shows a function that replaces the attribute valueOf by an object called newValue. The function assigns the name string to an attr object and then tests whether the attr object can be found. If the attr object does not exist, the NoSuchAttributeException is thrown. The thrown exception can be caught either in Java code by catching an IlrUserRuntimeException or superclass object, or in an IRL rule or function by catching a NoSuchAttributeException or superclass object.

Note that, unlike the Java code, the declaration of the replaceValue function does not contain a throws statement specifying that the function can throw a NoSuchAttributeException instance.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[try](#)

# timeofday

The `timeofday` keyword accesses an event timestamp.

## Purpose

This keyword is used to access an event timestamp.

## Context

Functions or rule actions

## Syntax

```
timeofday (?eventVar)
?time
```

## Description

### Deprecated as of V7.5.

The value of the `timeofday` operator applied to an event is the event timestamp returned as a long number. The variable `?time` is synonymous with `timeofday(?this)`.

## Example

The following rule prints a timestamp each time an `Alarm` object is inserted in the working memory.

```
rule TraceAlarms {
 when {
 ?a: event Alarm();
 } then {
 System.out.println(timeof(?a));
 }
};
```

**Parent topic:** [IRL keywords](#)

### Related reference:

[after](#)  
[before](#)  
[event \(in rule conditions\)](#)  
[occursin](#)

# timeout

The `timeout` keyword is associated with a `wait` statement.

## Purpose

This keyword is used in the action part of a rule and is executed if the associated `wait` statement fails.

## Context

Functions or rule actions

## Syntax

```
?variable
: wait [logical] [until] expression {condition}
 ...
 then ...
 timeout ?variable {action}
```

## Description

### Deprecated as of V7.5.

Use the `wait` keyword in the condition part of a rule to test whether conditions become valid or remain true during a specified waiting period. The `wait` statement creates a timer that delays the execution of a rule. You can include multiple timers in a rule and you can also specify an optional `timeout` statement for each timer. The `timeout` statement is part of the action part of a rule. If the `wait` statement fails, the `timeout` statement is executed.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[then](#)  
[wait](#)

## try

The `try` keyword executes control statements.

### Purpose

The `try` keyword is used to execute control statements with the ability to predeclare named exception handlers. A `catch` statement specifies exceptions that are caught if thrown within a `try` block. A `finally` statement specifies a control block that is executed after a `try` block is finished processing, including its exceptions, if any.

### Context

Functions or rule actions

### Syntax

```
try {statements}
catch (exceptionType1 identifier) {statements}
[catch (exceptionType2 identifier) {statements}
...]
finally {statements}
```

### Description

Use `try-catch-finally` statements in the action part of rules or in functions. Such statements define a block of code for exception handling. The `try`, `catch`, and `finally` blocks all begin and end with curly braces.

The `try` block must be followed by at least one `catch` or a `finally` statement, or both. The `try` statement governs the statements enclosed within it and defines the scope of any exception handlers associated with it.

The `catch` statements are designed to handle a specific type of exception. The code within a `catch` statement is executed if an exception is caught. The *exceptionType* parameters declared in the `catch` statement specify the type of exception that the statement can handle and also provide identifiers that the `catch` statement can use to refer to the exception object that it is handling. The identifier must be of type `Throwable` or one of its subclasses.

The `finally` statement is guaranteed to be executed if any portion of the `try` statement is executed, regardless of how the code in the `try` and `catch` statements completes.

### Example

#### Example 1

In this example, the `division` rule includes a `try` statement in its action part. When the `?x` variable is equal to zero, the division throws an `ArithmeticException`. The exception is caught by the `catch` clause which prints a message.

```
rule division {
 when {
 ?x = Integer() ;
 ?y = Integer() ;
 }
 then {
 try {
 int ?z = ?y/?x;
 }
 catch (ArithmeticException e) {
 System.out.println("Exception message : " + e) ;
 }
 }
}
```

#### Example 2

This example describes two `catch` statements, with one handler for each of the two types of exceptions that can be thrown within the `try` block: `ArrayIndexOutOfBoundsException` and `IOException`. When an exception occurs, the first `catch` statement that has an exception type compatible with the thrown exception is executed.

```
try {
```

```
 ...
}
catch (ArrayIndexOutOfBoundsException e) {
 System.err.println("Caught ArrayIndexOutOfBoundsException: "
 + e);
}
catch (IOException e) {
 System.err.println("Caught IOException: " + e);
}
```

### Example 3

This example shows a finally statement.

```
try{
 ...
}
finally {
 note("in finally");
}
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[catch](#)  
[finally](#)  
[throw](#)

# until

The `until` keyword specifies an absolute time.

## Purpose

This keyword is used in the `wait` statement to specify an absolute time period.

## Context

Rule conditions

## Syntax

```
(1) wait [[until] expression] {condition};
(2) wait logical [[until] expression] [{condition}];
(3) ?variable
: wait [logical] [until] expression {condition}
 ...
 then ...
 timeout ?variable {action}
```

## Description

### Deprecated as of V7.5.

Use the `wait` keyword to test whether conditions become valid or remain true during a specified waiting period. The `until` keyword indicates that the expression specifies an absolute time by which the `wait` statement must succeed. If you do not use the `until` keyword, the expression specifies a relative time, that is, a specific duration as the waiting period. The expression must return an integer value.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[wait](#)

# update

The update keyword updates an object.

## Purpose

This keyword is used in the action part of rules or in functions to update a modified object in the working memory. You can use it in RetePlus mode to notify the rule engine of an object state change. The engine then matches the rules against the new object state, which can result in new rule instances being added to the agenda.

## Context

Functions or rule actions

## Syntax

```
update [refresh] object;
```

## Description

You can define the *object* parameter as a variable or an expression that denotes an object associated with the current context or a specified context.

When an object is modified (for example, in a function or in Java™ code), the rules of the agenda might not be in a consistent state with respect to the new contents of the object. In such cases, you must use the update statement to notify the rule engine of the modification.

If the refresh keyword applies, rules that remain true or become true after the modification of the object are reinserted into the agenda. Without this keyword, only the rules that become true as a result of the modification are inserted into the agenda.

**Note:** In the decision engine you must add the **com.ibm.rules.engine.repeatabable** property, see [Repeatabable rules](#).

After an object has been updated in the working memory, the update statement can optionally execute a daemon. For this purpose, write a class that implements the interface [IlrUpdateDemon](#) and defines the method `updated`. For example, this method prints a message each time a *className* object is updated in the working memory:

```
public void updated(IlRuleContext context)
{
 System.out.println("Updated a className object in
 memory with fieldName: " + fieldName);
}
```

## Example

```
rule JobProcessing
{
 priority = -?a;
 when {
 ?n:NewJob(?s:size;?p:priority);
 ?proc:Server(?a:activity; ?id:identifier);
 }
 then {
 retract(?n);
 insert(new Job(?s,?p,?id));
 ?a = ?a + ?proc.updateActivity(?n);
 update refresh ?proc;
 }
};
```

The JobProcessing rule assigns new jobs to servers based on the server activity level. The rule uses a dynamic priority that is equal to the negation of the activity level. The lower the activity level, expressed by the variable ?a, the higher the priority. Hence, the server with the lowest activity level executes first.

The rule has two conditions. The first condition matches an object NewJob which is returned by the variable ?n. This condition contains variable ?s, which returns the value of field size, and variable ?p, which returns the value of field priority. The second condition matches an object Server, referenced by the variable ?proc.

This condition contains variable ?a, which returns the value of field activity, and variable ?id, which returns the value of field identifier.

The action part of the rule follows the keyword then and works as follows.

1. The first action applies the retract statement to the NewJob object, pointed to by variable ?n.
2. The second action applies an insert statement to a Job object with three fields, size, priority, and identifier, represented by the three variables ?s, ?p, and ?id. A new activity value is calculated by incrementing the activity ?a with the result of the method updateActivity. This method is called on the Server object pointed to by the variable ?proc.
3. Finally, the Server object is updated in the working memory using the update statement with the refresh keyword. Any rule matching a Server object in the condition part can be reinserted into the working memory.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[modify](#)



## use

The use keyword imports an IRL package or artifact.

### Purpose

This keyword is used to import an IRL package or artifact.

### Context

Packages

### Syntax

```
use [packageName.] artifactName | packageName.*;
```

### Description

The effect of referencing artifacts is different depending on whether they belong to the default package or not.

#### Using an artifact from another package

To use an artifact from another package, write this statement if the artifact is a variable, rule, or task:

```
use artifactName;
```

For functions, the syntax is more complex because you must uniquely identify the artifact that you want to import. To do so, include the function signature in the use statement:

```
use displayPrice(double);
```

#### Referencing artifacts in the default package

To reference artifacts from the default package in package B, you can import the artifacts one by one (not necessarily all of them) from the default package, using the use keyword. Only these artifacts are visible in package B.

When artifact names conflict, the following rule applies: If the default package defines an artifact named `artifactName` and package B defines an artifact also named `artifactName` and uses package A, all references to `artifactName` in B are considered to be references to the B artifact and priority is given to the local artifact. There is no way to reference the artifact in the default package.

#### Referencing another package that is not the default

To reference artifacts from package A (A not being the default package) in package B, you can do one of the following:

- You can write a use statement to import package A into package B. In this case, all its artifacts are visible in package B.
- If you can import artifacts from package A one by one (not necessarily all of them) using the use keyword. In this case, only the imported artifacts are visible in package B.

When artifact names conflict, the following rules apply:

- If package A defines an artifact named `artifactName` and package B defines an artifact also named `artifactName` and uses package A, all references to `artifactName` in package B are considered to be references to the B artifact and priority is given to the local artifact. To reference the A artifact, you must use its fully qualified name.
- If two packages A and B define an artifact named `artifactName`, and if package C uses packages A and B and references the `artifactName` artifact, an error is raised because it is impossible to give priority to a package against another. To avoid any ambiguity, use the fully qualified name.

### Example

This example defines two packages, `pricing` and `europa.pricing`. The `displayPrice` function belongs to the default package. The `europa.pricing` package uses the `pricing` package and the `displayPrice` function.

```
function void displayPrice(double price)
{
```

```

 ...
}
package pricing
{
 variables {
 Customer customer;
 }
 rule isEligible
 {
 when {
 ...
 }
 then {
 ...
 }
 }
}

package europe.pricing
{
 use pricing.*; All the pricing package is imported. All its artifacts are
visible in this package
 use displayPrice(double); The function displayPrice from the default package
is imported
 ruletask main
 {
 body {
 isEligible
 }
 finalaction {
 displayPrice(3.2d);
 }
 }
}

```

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[package](#)

## variables

The variables keyword declares variables.

### Purpose

This keyword is used to declare the variables that are associated with a ruleset.

### Context

Packages

### Syntax

```
variables {
 typeName variableName [= value];
 ...
}
```

### Description

You can declare the variables that are attached to a package. You can then reference these variables in rules, tasks, and functions, and in other variable initialization. You can use variables defined in a package from another package as soon as these variables are referenced with their fully qualified name or imported with the use keyword. The default package can also use other packages.

#### Note:

A fully qualified name is the short name of the variable prefixed with its package name.

use statements are at the same place as import statement in IRL code.

### Example

```
package pricing
{
 variables {
 Customer customer;
 }
 rule isEligible
 {
 when {
 ...
 }
 then {
 out.println("Customer is " + customer.name);
 }
 }
}

package europe.pricing
{
 use pricing.*;

 ruletask main
 {
 initialaction {
 out.println("Customer is " + customer.name); customer is the variable of
the pricing package.
 }
 body {
 isEligible
 }
 }
}
```

The pricing package has defined a customer variable. This variable is used in a rule that belongs to the

pricing package. The europe.pricing package uses the pricing package. Therefore, all pricing artifacts are visible in the europe.pricing package, (for example, the customer variable).

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[package](#)

# wait

The wait keyword incorporates time as a rule parameter.

## Purpose

This keyword is used in the condition part of rules to add time as a rule parameter.

## Context

Rule conditions

## Syntax

```
(1) wait [[until] expression] {condition};
(2) wait logical [[until] expression] [{condition}];
(3) ?variable
: wait [logical] [until] expression {condition}
 ...
 then ...
 timeout ?variable {action}
```

## Description

### Deprecated as of V7.5.

Use the wait statement in the condition part of rules to test whether conditions become valid during a specified waiting period or whether conditions remain true for a waiting period. The waiting period or time frame can be an absolute time or a specific duration relative to the current time. If you provide no *expression* parameter, the wait is indefinite.

The wait statement creates a timer that delays the execution of a rule. You can include multiple timers in a rule and, optionally, a timeout statement for each timer.

In syntaxes (2) and (3), the logical keyword is used to specify that the previously realized conditions must remain true for the wait statement to become true. If the logical keyword is not used, all the previously realized conditions are ignored, that is, they might become false during the waiting period and the wait statement might succeed.

The keyword until is used to designate that the expression specifies an absolute time by which the wait statement must succeed. If until is not used, the expression specifies a relative time, that is, a specific duration as the waiting period. The expression must return an integer value.

The wait statement (3) might include a timeout statement that is part of the action part of a rule. If the wait statement fails, the timeout statement is executed.

The rule engine has a time counter that is updated by the application according to the application requirements. The initial value of the time counter is 0.

You can synchronize the time counter of the rule engine with an external clock by using the following methods of [llrContext](#):

- The [time](#) method returns the current time.
- The [nextTime](#) method increments time by 1.
- The [nextTime](#) method increments time by the specified increment.
- The [setTime](#) method sets time to a specified time.

### Note:

It is the developer's responsibility to synchronize the time counter with any external clock according to the application requirements.

## Example

### Example 1

In this example, the DisplayScreen rule displays either a user-defined screen or a default screen.

The rule contains two condition statements and two action statements.

1. The first condition matches a Display object.

2. The second condition is a wait statement which specifies a wait duration of 5 time units. This condition tests whether a User object is not found in the working memory within the 5 time units.
3. The first action launches a displayScreen method applied to the user's response, represented by the variable ?r. This statement is executed if the wait statement succeeds.
4. If the wait statement fails, that is, a User object is not found in the working memory within the 5-unit time duration, the second action executes the timeout statement which calls the defaultScreen method.

```
rule DisplayScreen {
 when {
 Display();
 ?to: wait 5 {User(?r:response);}
 }
 then {
 displayScreen(?r);
 timeout ?to { defaultScreen(); }
 }
};
```

## Example 2

In this example, the NetworkMonitoring rule checks the traffic load of network elements and inserts an alarm if an element load passes 95% of its maximum load value.

This rule contains six condition statements and two action statements.

1. The first condition matches a Network object with variable ?e returning the value of field element, the field status being equal to alive and the variable ?m returning the value of the field maxLoad.
2. The second condition tests that an Alarm object for the element referenced by variable ?e does not exist in the working memory.
3. The third condition returns the Traffic object in variable ?t, where the field element is equal to variable ?e and the variable ?l returns the value of the field load.
4. The first wait logical statement simply causes a wait of one time unit.
5. The second wait logical statement matches the object Traffic within one time unit, as in the third condition, and the current value of the field load, returned in variable ?l2, is verified to be less than 95% of the maxLoad, using variable ?m.
6. If the load is below this threshold, the first action statement applies a modify statement on object Traffic with the calculated derivative of the load change.
7. If the wait statement fails, that is, the load is above 95% for over one time unit, the timeout statement inserts an Alarm object with the element pointed to by the variable ?e, using the insert statement.

```
rule NetworkMonitoring {
 when {
 Network(?e:element; status == alive; ?m:maxLoad);
 not Alarm(?e);
 ?t:Traffic(element == ?e; ?l:load);
 wait logical 1;
 ?a:wait logical 1 {Traffic(element == ?e; ?l2:
 load && ?l2 < 0.95*?m);}
 }
 then {
 modify ?t {?t.derivative = ?l2-?l}
 timeout ?a {insert Alarm(?e);}
 }
}
```

## Example 3

The WatchDogSample rule checks whether an Alarm exists for a network element and tests whether the problem is regulated within 10 time units.

This rule contains two condition statements and two action statements.

1. In the first condition, the variable ?a points to an Alarm object with the following constraints: variable ?id contains the value of the field networkElement, the value of the field severity is either LOW or MEDIUM, and variable ?t contains the value of the field time.
2. The second condition is a wait statement. This statement does not use the keyword logical. As a consequence, the previous conditions might turn false and the rule might still succeed. The until keyword means that the expression value is an absolute time and not a relative time. Therefore, the wait condition must become valid by time ?t+10, 10 time units past the value of ?t. The wait condition tests for an Alarm object with the value of the field networkElement equal to ?id, the field severity equal to CLEAR, and field time greater than ?t. If within 10 time units an Alarm object is found that fulfills these conditions, the retract action statement is executed.
3. The retract statement removes the Alarm object pointed to by ?a from the working memory. If the wait statement fails, the timeout statement is executed. The variable ?to designates the corresponding timeout statement for the wait statement.
4. The timeout statement prints a message We have a problem with ?id at time: ?t.

```
rule WatchDogSample {
 when {
 ?a: Alarm(?id:networkElement; severity == LOW ||
 severity == MEDIUM; ?t:time;)
 ?to: wait until ?t+10 {Alarm(networkElement == ?id;
 severity == CLEAR; time > ?t);}
 }
 then {
 retract ?a;
 timeout ?to {
 System.out.println("We have a problem with "
 + ?id " at time: " +?t);}
 }
};
```

**Parent topic:** [JRL keywords](#)

**Related reference:**

[timeout](#)

[logical \(in wait\)](#)

[until](#)

## when

The when keyword declares the condition part of a rule.

### Purpose

This keyword is used to introduce the condition part of a rule which consists of one or more conditions that must be satisfied for the rule to be executed.

### Context

Rule conditions

### Syntax

```
when {
 [collect|evaluate|exists|not|wait] condition1
 ...
 [collect|evaluate|exists|not|wait] condition2
}
```

### Description

A condition can be instantiated with objects in the working memory. Using a class name, the rule engine matches instances of that class, or instances of any derived class. The keyword not, exists, or evaluate can precede a condition.

Use conditions for the following purposes:

- Match patterns with objects of the working memory to test their validity. For example, `Car(color == blue)` tests whether a Car object has a color field equal to the value blue.
- Instantiate a rule variable with a field of an object in the working memory. For example, given a Car object in the working memory with a field color, the condition `Car(?c:color)` instantiates the variable ?c with the color of the car.
- Bind a rule variable with an object of the working memory that fulfills the constraints of the condition. For example, `?c:Car(color == blue)` return a Car object with the field color equal to blue to variable ?c. If no such object exists in the working memory, the condition fails.

For each rule, the cardinality of rule instances in the agenda equals the product of the number of matched instances for each condition of the rule. For example, suppose we have two classes, Depart and Destination, and for each, three city instances: Paris, New York, and Tokyo.

```
rule example
{
 when {
 Depart(?c1:city);
 Destination(?c2:city);
 }
 then {
 System.out.println("Possible city pairs are: "
 + ?c1 + ":" + ?c2);
 }
}
```

This rule generates nine instances of the rule in the agenda, including the three naive results such as Tokyo:Tokyo.

### Example

#### Example 1

In this example, the first condition of the CarColor rule is true if a Car object exists with the field color equal to red. The second condition tests that no Car object exists with the field color equal to green. If both conditions are true, the action is done, printing the message There is a red car but no green car.

```
rule CarColor
{
 when {
```



```

 Car(color == red);
 not Car(color == green);
 }
 then {
 System.out.println("There is a red car but no green car.")
 }
}

```

## Example 2

In this example, the first condition matches a Request object, binding the three variables ?m, ?h, and ?l with the fields mark, highPrice, and lowPrice, respectively.

The rule CarsAvailableAtPriceRange searches for a car from the information supplied by the user. The rule has two condition statements and one action statement. To execute the action statement, the two condition statements must be fulfilled.

The second condition matches a UsedCar object, with the following constraints: the field mark equals ?m and the field price has a value that is between ?h and ?l. If a UsedCar object is found that fulfills these constraints, it is bound to the variable ?car. The action part of the rule prints out any car found, for example, A 1996 BMW 573i is available for you. The action part might print multiple cars because each UsedCar object fulfilling the constraints generates a separate rule instance. For more information on rule instances of the agenda, see [Agenda](#)).

```

rule CarsAvailableAtPriceRange {
 when {
 Request(?m: mark; ?h: highPrice; ?l: lowPrice);
 ?car: UsedCar(mark equals ?m ; price < ?h & > ?l)
 }
 then {
 System.out.println("A " + ?car.year + " " + ?car.mark
 + " " + ?car.model + " is available for you.");
 }
}

```

**Parent topic:** [IRL keywords](#)

**Related reference:**

[collect](#)  
[evaluate](#)  
[exists](#)  
[not](#)  
[then](#)  
[wait](#)

## where

The where keyword defines a constraint in a collect statement.

### Purpose

This keyword is used in a collect statement in the condition part of a rule to contain tests that the collection object must fulfill.

### Context

Rule conditions

### Syntax

```
[?variable:] collect [(expression)] collectionTarget
[where (collectionTest1 ... collectionTestn)];
```

### Description

You can leave this statement empty.

**Parent topic:** [IRL keywords](#)

**Related reference:**  
[collect](#)

## while

The `while` keyword executes a statement block.

### Purpose

This statement is used in the action part of rules or in functions to execute a statement block numerous times.

### Context

Functions or rule actions

### Syntax

```
while (test) [{statement1; ... statementn;}]
```

### Description

The *test* argument can be any legal test, as in the Java™ programming language. The *statement* block can execute any IRL statement, arithmetic expressions, and method calls. Empty statement blocks are valid. If you specify only one *statement* block, the braces `{}` are not mandatory.

### Example

```
rule StockDropWarning {
 when {
 ?c:Client(?a:account_number;?p:percentWarningMark);
 ClientStockList(account_number == ?a; ?s:stockList);
 }
 then {
 Enumeration ?enum = ?s.elements();
 while (?enum.hasMoreElements()) {
 Stock ?x = (Stock)enum.nextElement();
 if (((1.-?p)*?x.purchasePrice) > ?x.currentPrice) {
 System.out.println("Dear "+ ?c.name + "Your stock "
 + ?x.ticker + " has dropped " + ((?x.purchasePrice
 - ?x.currentPrice) / ?x.purchasePrice) +
 " percent.");
 }
 }
 }
};
```

The rule `StockDropWarning` checks the percent change of a client's stocks and warns if a stock price is below its purchase price by a certain percentage. The condition `Client` returns an account number in variable `?a` and a decimal value `percentWarningMark` in variable `?p`. The second condition, `ClientStockList`, returns a stock list in variable `?s` corresponding to the account number equal to `?a`. In the action part of the rule, the `while` statement iterates on every stock in the stock list `?s` and check the difference between the purchase price and the current price. If the stock price decreased in `?p` percent below the purchase price, the `println` function in the action statement prints a message.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[break](#)  
[continue](#)  
[for](#)  
[foreach](#)

## while/break/continue (in ruleflow)

The while keyword executes a loop.

### Purpose

This ruleflow conditional statement is used to execute a loop while a Boolean expression value remains true.

### Context

Ruleflow body

### Syntax

```
while (test)
 {ruleflowStatement}
```

### Description

The *test* argument can be any legal test, as in the Java™ programming language. As long as this value remains true, the statements inside the while block are executed. Any ruleflow statement can be in a while block. When the value becomes false, the engine stops executing the while block and resumes ruleflow execution after the while statement.

A break statement can interrupt a while loop. When the engine encounters a break statement, the while loop is interrupted and ruleflow execution continues after the while statement.

You can include a continue statement in a while loop. When the engine encounters a continue statement, ruleflow execution returns to the while statement again, reevaluates the *test*, and continues ruleflow execution according to this value.

### Example

```
ruleset Connect4
{
 // The latest move
 int turn;
 // Who's the winner, what is the connect4, and any other reason
 // to end the game.
 int winner = Constants.None;
 Connect4 connect4;
 boolean ending = false;
};
flowtask main
{
 body =
 {
 while(!ending)
 {
 if (turn == Constants.Player1) ChooseMovePlayer1;
 else ChooseMovePlayer2;
 CheckMove;
 if (ending) break;
 UpdateDistance;
 ExpandObjects;
 DetectConnect4;
 if (ending) break;
 DetectGridFull;
 if (ending) break;
 ChangeTurn;
 }
 EndOfGame;
 }
}
```

This example illustrates a while statement. The while block is executed while the ending variable value remains false (because then !ending remains true).

If the ending Boolean value becomes true, a break statement can interrupt the loop.

**Parent topic:** [IRL keywords](#)

**Related reference:**

[body \(in flowtask\)](#)

[break](#)

[continue](#)

[flowtask](#)

# IRL grammar

The ILOG® Rule Language (IRL) grammar conforms to a specific notation and has naming restrictions. It is composed of several operators that are a subset of the operators provided in the Java™ programming language.

## Operator summary

The operators of the ILOG Rule Language are a subset of the operators provided in the Java programming language.

## Naming restrictions

A number of keywords are reserved and naming restrictions apply to package names.

## Grammar notation

The IRL grammar notation follows the notation presented in *The Java Language Specification* by James Gosling, Bill Joy and Guy Steele (Addison Wesley, 2nd Edition, 2000).

## Grammar specification

The ILOG Rule Language (IRL) defines a lexical grammar.

**Parent topic:** [ILOG Rule Language \(IRL\)](#)

# Operator summary

The operators of the ILOG® Rule Language are a subset of the operators provided in the Java™ programming language.

The ILOG Rule Language operators are summarized in the table below.

The column marked P presents the precedence of the operator: the larger the value the higher the precedence. The column marked A represents the Associativity of the operator. Given an expression with several operators of the same precedence, the associativity determines the priority.

Table 1. ILOG Rule Language Operators

Operator	Operand Type(s)	Operation Performed	P	A
.	object, member	object member access	10	L
[]	array, int	array element access	10	L
(args)	method, arglist	method invocation	10	I
+	String, String	String concatenation	10	L
++, --	variable	post-increment, decrement	10	L
++, --	variable	pre-increment, decrement	9	R
+, -	number	unary plus, unary minus	9	R
!	boolean	boolean NOT	9	R
new	class, arglist	object creation	8	R
(type)	type, any	cast (type conversion)	8	R
*, /, %	number, number	multiplication, division, remainder	7	L
+, -	number, number	addition, subtraction	6	L
+	string, any	string concatenation	6	L
<, <=	number, number	less than, less than or equal	5	L
>, >=	number, number	greater than, greater than or equal	5	L
instanceof	reference, type	type comparison	5	L
as	reference, type	type conversion	5	L
==	primitive, primitive	equal (have identical values)	4	L
!=	primitive, primitive	not equal (have different values)	4	L
==	reference, reference	equal (refer to same object)	4	L
!=	reference, reference	not equal (refer to different objects)	4	L
&&	boolean, boolean	conditional AND	3	L
	boolean, boolean	conditional OR	2	L
=	variable, any	assignment	1	R
*, /=, %=, +=, -=,	variable, any	assignment with operation	1	R

Parent topic: [IRL grammar](#)

## Naming restrictions

A number of keywords are reserved and naming restrictions apply to package names.

### Deprecated as of V7.5.

You must adhere to the naming restrictions.

The reserved keywords are:

- break
- catch
- collect
- continue
- else
- evaluate
- event
- exists
- finally
- hasher
- if
- instanceof
- instances
- isknown
- isunknown
- logical
- new
- not
- refresh
- return
- throw
- timeof
- timeout
- try
- until

**Important:**

Reserved keywords cannot be used as identifiers, unless otherwise indicated in the grammar.

## Package Names

The reserved keyword event cannot be used as an identifier but can be used as a package name.

### @ Operator

The @ character can be used to prefix an identifier that might otherwise be interpreted as an IRL keyword. An identifier can be a variable name, a field, a method, or a class name.

**Parent topic:** [IRL grammar](#)



# Grammar notation

The IRL grammar notation follows the notation presented in *The Java™ Language Specification* by James Gosling, Bill Joy and Guy Steele (Addison Wesley, 2nd Edition, 2000).

Terminal symbols, that is, the language keywords, are shown in **Courier bold font** in the syntactic and lexical rules presented in this chapter.

Nonterminal symbols are shown in *italic font* in the syntactic rules. A definition of a nonterminal symbol starts with the symbol followed by a colon. For example:

```
PropertyEntry:
 property
 PropertyName = PropertyValue;
```

states that the nonterminal symbol *PropertyEntry* has a single definition, which comprises the terminal token `property` followed by a *PropertyName*, the token equals sign, followed by an *Identifier*, followed by the token semicolon.

A definition might have several rules. Each rule is described on a separate line. The following example shows a nonterminal symbol with two rule definitions:

```
TestAssignmentList:
 TestAssignment
 TestAssignmentList; TestAssignment
```

The syntactic definition of the nonterminal symbol *TestAssignmentList* can be either *TestAssignment* or *TestAssignmentList*, followed by the token semicolon, followed by *TestAssignment*.

The subscripted suffix “*opt*” indicates an optional symbol. It might be shown after a terminal or nonterminal symbol. For example:

```
WaitStatement :
 wait untilopt Expression
```

is equivalent to:

```
WaitStatement :
 wait Expression
 wait until Expression
```

When the words “one of” follow the colon in a grammar definition, they signify that each of the symbols on the following line or lines is an alternative definition. For example, the following lexical grammar rule:

```
AssignmentOperator: one of
 = *= /= %= += -=
```

is a convenient abbreviation for:

```
AssignmentOperator:
 =
 *=
 /=
 %=
 +=
 -=
```

The words “but not” are used to indicate symbols that are excluded from the rule definition. For example:

```
ReducedAssignmentOperator:
 AssignmentOperator but not -=
```

is equivalent to:

*ReducedAssignmentOperator*: one of  
=    \*=    /=    %=    +=

An explanatory phrase in parentheses might be shown after some expressions.

**Parent topic:** [IRL grammar](#)

# Grammar specification

The ILOG® Rule Language (IRL) defines a lexical grammar.

**Note:**  
A bullet (•) before a grammar rule indicates that the terminal symbol in the rule is detailed in [IRL keywords](#).

Table 1. General form of an IRL program. Table shows the general form of an IRL program.

General form of an IRL program	
	<i>RuleSetDefinition</i> :  <i>ImportOrUseDeclaration</i> opt <i>RulesetPropertyDefinition</i> opt <i>HierarchicalPropertyDeclaration</i> opt <i>PropertyTypingDeclaration</i> opt <i>RuleOverridingDeclaration</i> opt <i>VariableDeclaration</i> opt <i>PackageList</i> opt
Package definition	
	<i>PackageList</i> :  <i>Package</i> <i>PackageList</i> opt
	<i>Package</i> :  <i>PackageDefinition</i>  <i>PackageContent</i>
•	<i>PackageDefinition</i> :  package <i>Identifier</i>  { <i>ImportOrUseDeclaration</i> opt <i>VariableDeclaration</i> opt  <i>PackageContentList</i> }
	<i>PackageContentList</i> :  <i>PackageContent</i> <i>PackageContentList</i> opt
•	<i>PackageContent</i> :  <i>FunctionDefinitionList</i> opt  <i>RuleDefinitionList</i> opt  <i>TaskDefinitionList</i> opt
Import or use declaration	
	<i>ImportOrUseDeclaration</i> :  <i>SingleImportOrUseDeclaration</i>  <i>ImportOrUseOnDemandDeclaration</i>
•	<i>SingleImportOrUseDeclaration</i> :  ImportOrUseKeyword <i>PackageName</i> . *;
•	<i>ImportOrUseOnDemandDeclaration</i> :  ImportOrUseKeyword <i>TypeName</i> ;
	<i>ImportOrUseKeyword</i> : one of

	<div>import</div> <div>use</div>
Ruleset property definition	
	<div>RulesetPropertyDefinition :</div> <div>ruleset identifier</div> <div>{ PropertyListopt RulesetParameterListopt HasherDefinitionListopt InstanceDefinitionListopt }</div>
	<div>PropertyList:</div> <div>PropertyEntry PropertyListopt</div>
<div>•</div>	<div>PropertyEntry:</div> <div>property PropertyName = PropertyValue;</div>
	<div>RulesetParameterList:</div> <div>RulesetParameter RulesetParameterListopt</div>
	<div>RulesetParameter:</div> <div>InRulesetParameter InOutRulesetParameter OutRulesetParameter</div> <div>LocalRulesetParameter</div>
<div>•</div>	<div>InRulesetParameter:</div> <div>in Type Variable;</div> <div>in Type Variable = Expression;</div>
<div>•</div>	<div>InoutRulesetParameter:</div> <div>inout Type Variable;</div>
<div>•</div>	<div>OutRulesetParameter:</div> <div>out Type Variable;</div> <div>out Type Variable = Expression;</div>
<div>•</div>	<div>LocalRulesetParameter:</div> <div>Type Variable;</div> <div>Type Variable = Expression;</div>
<div>•</div>	<div>HasherDefinitionList:</div> <div>hasher (ReturnType Variable) = Expression;</div>
<div>•</div>	<div>InstanceDefinitionList:</div> <div>instances (ReturnType) = Expression;</div> <div>instances (ReturnType) = {InstancesList};</div>
	<div>InstancesList:</div>

	<i>Expression</i> <i>Expression, InstanceList</i>
	<i>Type</i> : one of  boolean char byte short int long float double String  <i>ClassName</i>
	<i>ReturnType</i> :  void <i>Type</i>
<b>Variable declaration</b>	
•	<i>VariableDeclaration</i> :  variables { <i>LocalRulesetParameterList</i> }
	<i>LocalRulesetParameterList</i> :  <i>LocalRulesetParameter</i> <i>LocalRulesetParameterList</i>
<b>Hierarchical property declaration</b>	
	<i>HierarchicalPropertyDeclaration</i> :  hierarchy <i>identifier</i>  { <i>HierarchicalNodeDefinition</i> }
•	<i>HierarchicalChildren</i> :  { <i>HierarchyDefinitionList</i> }
	<i>HierarchyDefinitionList</i> :  <i>HierarchicalNodeDefinition</i> <i>HierarchyDefinitionList</i> opt
	<i>HierarchicalNodeDefinition</i>  <i>STRING_LITERAL</i> <i>HierarchicalChildren</i> opt
<b>Property typing declaration</b>	
•	<i>PropertyTypingDeclaration</i> :  propertydefinition  { <i>PropertyTypingList</i> }
	<i>PropertyTypingList</i> :  PropertyTyping <i>PropertyTypingList</i> opt
	<i>PropertyTyping</i> :  <i>Type</i> <i>Variable</i> ;
<b>Rule overriding declaration</b>	
•	<i>RuleOverridingDeclaration</i> :  overriding  { <i>RuleOverridingList</i> }

	<i>RuleOverridingList :</i> <i>RuleOverriding RuleOverridingListopt</i>
	<i>RuleOverriding:</i> <i>STRING_LITERAL overrides OverriddenList</i>
	<i>OverriddenList:</i> <i>STRING_LITERAL</i> <i>STRING_LITERAL , OverriddenList</i>
<b>Function definition</b>	
	<i>FunctionDefinitionList :</i> <i>FunctionDefinition FunctionDefinitionList</i>
•	<i>FunctionDefinition :</i> <i>function ReturnType FunctionName (</i> <i>FunctionArgumentList )</i> <i>{ FunctionActionList }</i>
	<i>FunctionArgumentList:</i> <i>FunctionArgument</i> <i>FunctionArgument, FunctionArgumentList</i>
	<i>FunctionArgument:</i> <i>ReturnType Variable</i>
	<i>FunctionActionList:</i> <i>FunctionAction FunctionActionListopt</i>
	<i>FunctionAction: one of</i> <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i>  <i>ForFunctionAction IfElseFunctionAction</i> <i>ModifyAction RetractAction</i>  <i>ReturnStatement ThrowAction</i> <i>TryFunctionAction UpdateAction</i>  <i>WhileFunctionAction</i>

Table 2. Syntax of a rule

Rule header	
	<i>RuleDefinitionList :</i> <i>RuleDefinition RuleDefinitionList</i>
•	<i>RuleDefinition :</i> <i>rule Identifier { RuleParametersopt</i> <i>RuleConditions RuleActions } ;</i>
	<i>RuleParameters:</i>

	<i>RulePriority</i> opt <i>RuleProperty</i> opt
•	<i>RulePriority:</i> priority = <i>PriorityExpression</i> ;
•	<i>RuleProperty:</i> <i>ActivationProperty</i> opt <i>PropertyList</i>
	<i>ActivationProperty:</i> property activation = true;
<b>Rule conditions</b>	
•	<i>RuleConditions :</i> when { <i>FirstRuleCondition</i> <i>RuleConditionList</i> opt }
	<i>RuleConditionList:</i> <i>RuleCondition</i> <i>RuleConditionList</i> <i>RuleCondition</i>
	<i>FirstRuleCondition:</i> <i>RuleCondition</i> except <i>WaitStatementList</i> , <i>EvaluateCondition</i>
	<i>RuleCondition:</i> one of <i>SimpleCondition</i> <i>NotCondition</i> <i>ExistsCondition</i> <i>EvaluateCondition</i> <i>FromExpression</i> <i>InExpression</i> <i>CollectCondition</i> <i>WaitStatementList</i>
•	<i>SimpleCondition :</i> <i>Variable</i> opt:opt eventopt <i>ClassCondition</i>
	<i>Variable:</i> <b>?</b> opt <i>Identifier</i>
	<i>ClassCondition:</i> <i>ClassName</i> ( <i>TestAssignmentList</i> );
•	<i>NotCondition:</i> not <i>ClassCondition</i> ;
•	<i>ExistsCondition:</i> exists <i>ClassCondition</i> ;
•	<i>EvaluateCondition:</i> evaluate ( <i>TestAssignmentList</i> );
•	<i>FromExpression:</i>

	<i>Variable</i> <i>opt SimpleCondition</i> from <i>PrimaryExpression</i> ;
•	<i>InExpression</i> :  <i>Variable</i> <i>opt SimpleCondition</i> in <i>PrimaryExpression</i> ;
•	<i>CollectCondition</i> :  <i>Variable</i> <i>opt collect (Expressionopt) SimpleCondition</i> where  <i>TestAssignmentSet</i> ;
	<i>WaitStatementList</i> :  <i>WaitLogicalStatement</i>  <i>WaitStatement</i>
•	<i>WaitLogicalStatement</i> :  wait logical until <i>opt Expression WaitCondition</i> <i>opt</i>  <i>Variable</i> wait logical until <i>opt Expression WaitConditionopt</i>  <i>TimeoutAction</i>
•	<i>WaitStatement</i> :  wait <i>Expressionopt WaitConditions</i>  wait until <i>Expression WaitConditions</i>  <i>Variable</i> wait <i>Expressionopt WaitConditions</i> <i>TimeoutAction</i>  <i>Variable</i> wait until <i>Expression WaitConditions</i> <i>TimeoutAction</i>
	<i>WaitConditions</i> :  { <i>WaitConditionList</i> }
	<i>WaitConditionList</i> :  <i>WaitCondition</i>  <i>WaitConditionList</i>
	<i>WaitCondition</i> : one of  <i>SimpleCondition NotCondition ExistsCondition</i>  <i>CollectCondition</i>
<b>Tests and variable declarations</b>	
	<i>TestAssignmentList</i> :  <i>TestAssignment</i>  <i>TestAssignmentList; TestAssignment</i>
	<i>TestAssignment</i> : one of  <i>AndExpresssionList OrExpressionList</i>



	<i>VariableAssignment TestAssignmentSet</i> <i>BooleanExpression NotAssignment</i> <i>TemporalConstraint</i>
	<i>AndExpressionList:</i> <i>TestAssignment &amp;&amp; TestAssignment</i>
	<i>OrExpressionList:</i> <i>TestAssignment    TestAssignment</i>
	<i>VariableAssignment :</i> <i>Variable Expression BooleanAssignmentListopt</i>
	<i>BooleanAssignmentList:</i> <i>BooleanAssignment</i> <i>BooleanAssignmentList BooleanAssignment</i>
	<i>BooleanAssignment:</i> <i>&amp; BooleanArgument</i>
	<i>TestAssignmentSet :</i> <i>(TestAssignment)</i>
	<i>NotAssignment :</i> <i>! TestAssignment</i>
	<i>TemporalConstraint:</i> <i>Expressionopt.opt TemporalOperator</i> <i>Arguments</i>
•	<i>TemporalOperator:</i> one of after before occursin
	<i>BooleanExpression :</i> <i>Expression</i> <i>BooleanArgumentList</i>
	<i>BooleanArgumentList :</i> <i>BooleanArgument</i> <i>BooleanArgumentList &amp; BooleanArgument</i>
•	<i>BooleanArgument:</i> <i>BooleanArgument</i> instanceof <i>Identifier</i> isknown isunknown InTest

	<i>BooleanTerm</i> : one of <i>PredefinedPredicate Expression</i> Identifier Expression (use of custom predicate)
	<i>PredefinedPredicate</i> : one of  == != < > >= <=
	InTest: one of in <i>Expression</i> in { InList } !in Expression !in { InList }
	<i>InList</i> : one of <i>Expression</i> <i>InList, Expression</i>
<b>Expressions</b>	
	<i>Expression</i> : <i>AdditiveExpression</i>
	<i>AdditiveExpression</i> : <i>MultiplicativeExpression</i> <i>MultiplicativeExpressionList</i>
	<i>MultiplicativeExpressionList</i> : + <i>MultiplicativeExpression</i> - <i>MultiplicativeExpression</i> + <i>MultiplicativeExpressionList</i> - <i>MultiplicativeExpressionList</i>
	<i>MultiplicativeExpression</i> : <i>UnaryExpression</i> <i>UnaryExpressionList</i>
	<i>UnaryExpressionList</i> : * <i>UnaryExpression</i> / <i>UnaryExpression</i> % <i>UnaryExpression</i> * <i>UnaryExpressionList</i> / <i>UnaryExpressionList</i> % <i>UnaryExpressionList</i>
	<i>UnaryExpression</i> : + <i>UnaryExpression</i>

	<div>- <i>UnaryExpression</i></div> <div><i>++ UnaryExpression</i></div> <div><i>-- UnaryExpression</i></div> <div><i>UnaryExpression ++</i></div> <div><i>UnaryExpression - -</i></div> <div><i>CastExpression</i></div> <div><i>TimeOfExpression</i></div> <div><i>PrimaryExpression</i></div> <div><i>AsExpression</i></div>
	<div><i>CastExpression:</i></div> <div><i>(ClassName) UnaryExpression</i></div>
<div>•</div>	<div><i>AsExpression:</i></div> <div><i>UnaryExpression <b>as</b> (ClassName)</i></div>
	<div><i>TimeOfExpression:</i></div> <div><i>timeofday (Expression);</i></div>
<b>Primary Expressions</b>	
<div>•</div>	<div><i>PrimaryExpression:</i></div> <div><i>PrimaryPrefix</i></div> <div><i>PrimaryPrefix PrimarySuffixList</i></div>
	<div><i>PrimaryPrefix:</i> one of</div> <div><i>Literal Name AllocationExpression</i></div> <div><i>(Expression)</i></div>
	<div><i>Literal:</i> one of</div> <div><i>INTEGER_LITERAL FLOATING_POINT_LITERAL</i></div> <div><i>CHARACTER_LITERAL STRING_LITERAL true</i></div> <div><i>false null</i></div>
	<div><i>Name:</i></div> <div><i>Variable IdentifierListopt</i></div>
	<div><i>IdentifierList:</i></div> <div><i>DotIdentifier</i></div> <div><i>DotIdentifier IdentifierList</i></div>
	<div><i>DotIdentifier:</i></div> <div><i>. Identifier</i></div>
	<div><i>AllocationExpression:</i></div> <div><i>new ClassName Arguments</i></div> <div><i>new Type DimensionList</i></div> <div><i>new Type OptDimensionList</i></div>

	<i>BracedInitializationList</i> <i>BracedInitializationList</i>
	<i>Arguments:</i> ( <i>ExpressionList</i> )
	<i>ExpressionList:</i> <i>Expression</i> <i>ExpressionList, Expression</i>
	<i>DimensionList:</i> <i>Dimension</i> <i>DimensionList Dimension</i>
	<i>Dimension:</i> [ <i>Expression</i> ]
	<i>OptDimensionList:</i> <i>OptDimension</i> <i>OptDimensionList OptDimension</i>
	<i>OptDimension:</i> [ <i>Expression</i> opt]
	<i>BracedInitializationList:</i> { <i>InitializationList</i> }
	<i>InitializationList:</i> <i>Initialization</i> <i>InitializationList, Initialization</i>
	<i>Initialization :</i> <i>Expression</i> <i>BracedInitializationList</i> { <i>ExpressionList</i> }
	<i>PrimarySuffixList:</i> <i>PrimarySuffix</i> <i>PrimarySuffixList PrimarySuffix</i>
	<i>PrimarySuffix: one of</i> <i>. Variable</i> (access to variable or class field) <i>Arguments</i> (access to IRL function or class method) <i>DimensionList</i> (access to array element)
Rule actions	

•	<i>RuleActions :</i> <i>then {RuleActionListopt}</i>
	<i>RuleActionList:</i> <i>RuleAction</i> <i>RuleActionList RuleAction</i>
	<i>RuleAction:</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i>  <i>ForAction IfElseAction ModifyAction</i> <i>RetractAction ThrowAction</i>  <i>TimeoutAction TryAction UpdateAction</i> <i>WhileAction</i>
•	<i>InsertAction :</i>  <i>insert logicalopt eventopt ClassName</i> <i>Argumentsopt</i>  <i>ExecutableActionStatementsopt</i>
•	<i>VarDeclAction:</i>  <i>Type Variable = Expression;</i>
•	<i>ForAction :</i>  <i>for(ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i>  <i>LoopRuleAction</i>  <i>for(ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i>  <i>{LoopRuleActionList}</i>
•	<i>ForFunctionAction :</i>  <i>for (ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i>  <i>LoopFunctionAction</i>  <i>for (ForInitopt; TestAssignmentopt;</i> <i>ForUpdateopt)</i>  <i>{LoopFunctionActionList}</i>
	<i>ForInit:</i>  <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i>
	<i>ForUpdate:</i>  <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i>
•	<i>IfElseAction :</i>  <i>if TestAssignmentSet RuleAction</i> <i>ElseStatement opt</i>

	<i>if TestAssignmentSet {RuleActionList} ElseStatementopt</i>
•	<i>ElseStatement:</i> <i>else RuleAction</i> <i>else {RuleActionList}</i>
•	<i>IfElseFunctionAction :</i> <i>if TestAssignmentSet FunctionAction</i> <i>ElseFunctionActionopt</i> <i>if TestAssignmentSet {FunctionActionList}</i> <i>ElseFunctionActionopt</i>
•	<i>ElseFunctionAction:</i> <i>else FunctionAction</i> <i>else {FunctionActionList}</i>
•	<i>ModifyAction :</i> <i>modify refreshopt PrimaryExpression</i> <i>ExecutableActionStatements;opt</i>
•	<i>RetractAction :</i> <i>retract PrimaryExpression;</i>
•	<i>ThrowAction:</i> <i>throw Expression;</i>
•	<i>TimeoutAction :</i> <i>timeout Variable {NonTimeoutRuleActionList}</i>
	<i>NonTimeoutRuleActionList:</i> <i>NonTimeoutRuleAction</i> <i>NonTimeoutRuleActionList</i> <i>NonTimeoutRuleAction</i>
	<i>NonTimeoutRuleAction:</i> <i>RuleAction</i> but not <i>TimeoutAction</i>
	<i>RuleAction :</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i> <i>ForAction IfElseAction ModifyAction</i> <i>RetractAction</i> <i>UpdateAction WhileAction</i> break; continue;
	<i>FunctionAction:</i> one of <i>InsertAction VarDeclAction</i> <i>ExecutableStatement</i>

	<i>ForFunctionAction IfElseFunctionAction ModifyAction</i>  <i>RetractAction UpdateAction WhileFunctionAction</i>  <i>break; continue; ReturnStatement</i>
•	<i>ReturnStatement:</i>  <i>return ( expression )</i>
	<i>RuleActionList:</i>  <i>RuleAction RuleActionListopt</i>
	<i>FunctionActionList:</i>  <i>FunctionAction FunctionActionListopt</i>
•	<i>TryAction:</i> <i>try {RuleActionListopt}</i>  <i>Catches</i>  <i>or</i>  <i>try {RuleActionListopt}</i>  <i>Catchesopt</i>  <i>Finally</i>
	<i>Catches:</i>  <i>CatchClause</i>  <i>Catches CatchClause</i>
•	<i>CatchClause:</i>  <i>catch (ExceptionType Identifier)</i> <i>{RuleActionListopt}</i>
•	<i>Finally:</i>  <i>finally {RuleActionListopt}</i>
•	<i>TryFunctionAction:</i> <i>try {FunctionActionListopt}</i>  <i>FunctionCatches</i>  <i>or</i>  <i>try {FunctionActionListopt}</i>  <i>FunctionCatchesopt</i>  <i>Finally</i>
	<i>FunctionCatches:</i>  <i>FunctionCatchClause</i>  <i>FunctionCatches FunctionCatchClause</i>
•	<i>FunctionCatchClause:</i>

	catch ( <i>ExceptionType Identifier</i> ) { <i>FunctionActionList</i> opt}
•	<i>FunctionFinally:</i> finally { <i>FunctionActionList</i> opt}
•	<i>UpdateAction :</i> update refreshopt <i>PrimaryExpression</i> ;
•	<i>WhileAction :</i> while <i>TestAssignmentSet LoopRuleAction</i> while <i>TestAssignmentSet</i> { <i>LoopRuleActionList</i> }
•	<i>WhileFunctionAction :</i> while <i>TestAssignmentSet LoopFunctionAction</i> while <i>TestAssignmentSet</i> { <i>LoopFunctionActionList</i> }
<b>Executable statements</b>	
	<i>ExecutableStatementList:</i> <i>ExecutableStatement</i> <i>ExecutableStatementList</i> <i>ExecutableStatement</i>
	<i>ExecutableActionStatements:</i> { <i>ExecutableStatementList</i> }
	<i>ExecutableStatement:</i> <i>PrimaryExpression</i> <i>AssignmentOperatorExpression</i> opt;
	<i>AssignmentOperatorExpression:</i> <i>AssignmentOperator Expression</i>
	<i>AssignmentOperator : one of</i> = *= /= %= += -=

Table 3. Syntax of a ruleflow

Ruleflow tasks	
	<i>TaskDefinitionList :</i> <i>TaskDefinition TaskDefinitionList</i>
	<i>TaskDefinition :</i> <i>RuleTaskDefinition</i> <i>FunctionTaskDefinition</i> <i>FlowTaskDefinition</i>
•	<i>RuleTaskDefinition :</i> ruletask identifier {



	<i>CommonTaskParameters</i> <i>RuleTaskBody</i> <i>RuleTaskScope</i> <i>RuleTaskParametersopt</i> };opt
•	<i>FunctionTaskDefinition :</i> functiontask <i>identifier</i> { <i>CommonTaskParameters</i> <i>FunctionTaskBody</i> };opt
•	<i>FlowTaskDefinition :</i> flowtask <i>identifier</i> { <i>CommonTaskParameters</i> <i>FlowTaskBody</i> };opt
•	<i>CompletionFlagParameter:</i> completionflag = <i>Expression</i>
•	<i>InitialActions:</i> initialaction { <i>FunctionActionList</i> }
•	<i>FinalActions:</i> finalaction { <i>FunctionActionList</i> }
	<i>RuleTaskBody:</i> <i>ExtendedBodyDefinition</i> <i>ComprehensiveBodyDefinition</i> <i>DynamicComprehensiveBodyDefinition</i>
•	<i>RuleTaskScope:</i> scope { <i>ScopeDefinition</i> }
	<i>ScopeDefinition:</i> <i>RuleName</i> <i>PackageName.*</i> <i>ScopeDefinitionopt</i>
•	<i>ExtendedBodyDefinition:</i> body { <i>RulesList</i> };opt
	<i>RulesList:</i>

	<i>identifier</i> <i>identifier, RulesList</i>
•	<i>ComprehensiveBodyDefinition:</i>  body = select ( <i>Variable</i> ) { <i>FunctionActionList</i> };opt  body = select () { <i>FunctionActionList</i> };opt
•	<i>DynamicComprehensiveBodyDefinition:</i>  body = dynamicselect( <i>Variable</i> ) { <i>FunctionActionList</i> }  EndDynamicComprehensiveBodyDefinition  or  body = dynamicselect () { <i>FunctionActionList</i> }  EndDynamicComprehensiveBodyDefinition
	EndDynamicComprehensiveBodyDefinition:  ;opt  or  <i>DomainValueDefinition</i>
•	<i>DomainValueDefinition:</i>  in Expression;
	<i>RuleTaskParameters:</i>  <i>AgendaFilterParameter</i> <i>AlgorithmParameters</i> <i>FiringParameter</i> <i>FiringLimitParameter</i> <i>OrderingParameter</i>
•	AgendaFilterParameter:  agendafilter = <i>Expression</i> ;  agendafilter = filter ( <i>Variable</i> ) { <i>FunctionActionList</i> };opt
•	<i>AlgorithmParameters:</i>  algorithm = <i>AlgorithmValues</i> ;  iterator = <i>Expression</i> ;  <i>MatchedClasses</i>
	<i>AlgorithmValues:</i>  default sequential
•	<i>MatchedClasses:</i>  matchedclasses = <i>Expression</i> ;  matchedclasses = { <i>MatchedClassesList</i> }; opt
	<i>MatchedClassesList:</i>  <i>ReturnType</i>

	<i>ReturnType, MatchedClassesList</i>
•	<i>FiringParameter:</i> firing = <i>FiringValues</i> ;
	<i>FiringValues:</i> allrules rule
•	<i>FiringLimitParameter:</i> firinglimit = <i>Expression</i> ;
•	<i>OrderingParameter:</i> ordering = <i>OrderingValues</i> ;
	<i>OrderingValues:</i> dynamic sorted literal
•	<i>FunctionTaskBody:</i> body { <i>FunctionActionList</i> }; opt
•	<i>FlowTaskBody:</i> body { <i>FlowActionList</i> }; opt
	<i>FlowActionList:</i> <i>FlowAction FlowActionList</i> opt
	<i>FlowAction:</i> one of <i>FlowTaskInvocationAction FlowForkAction</i> <i>FlowGotoAction</i> <i>FlowIfElseAction FlowSwitchAction</i> <i>FlowWhileAction</i>
	<i>FlowTaskInvocationAction :</i> <i>Label</i> opt <i>identifier</i> ;
	<i>Label:</i> <i>identifier:</i>
•	<i>FlowForkAction :</i> <i>Label</i> opt fork <i>FlowForkActionList</i> ;
•	<i>FlowForkActionList:</i> { <i>FlowActionList</i> }; opt { <i>FlowActionList</i> } && <i>FlowForkActionList</i>
•	<i>FlowGotoAction :</i> goto <i>identifier</i> ;
•	

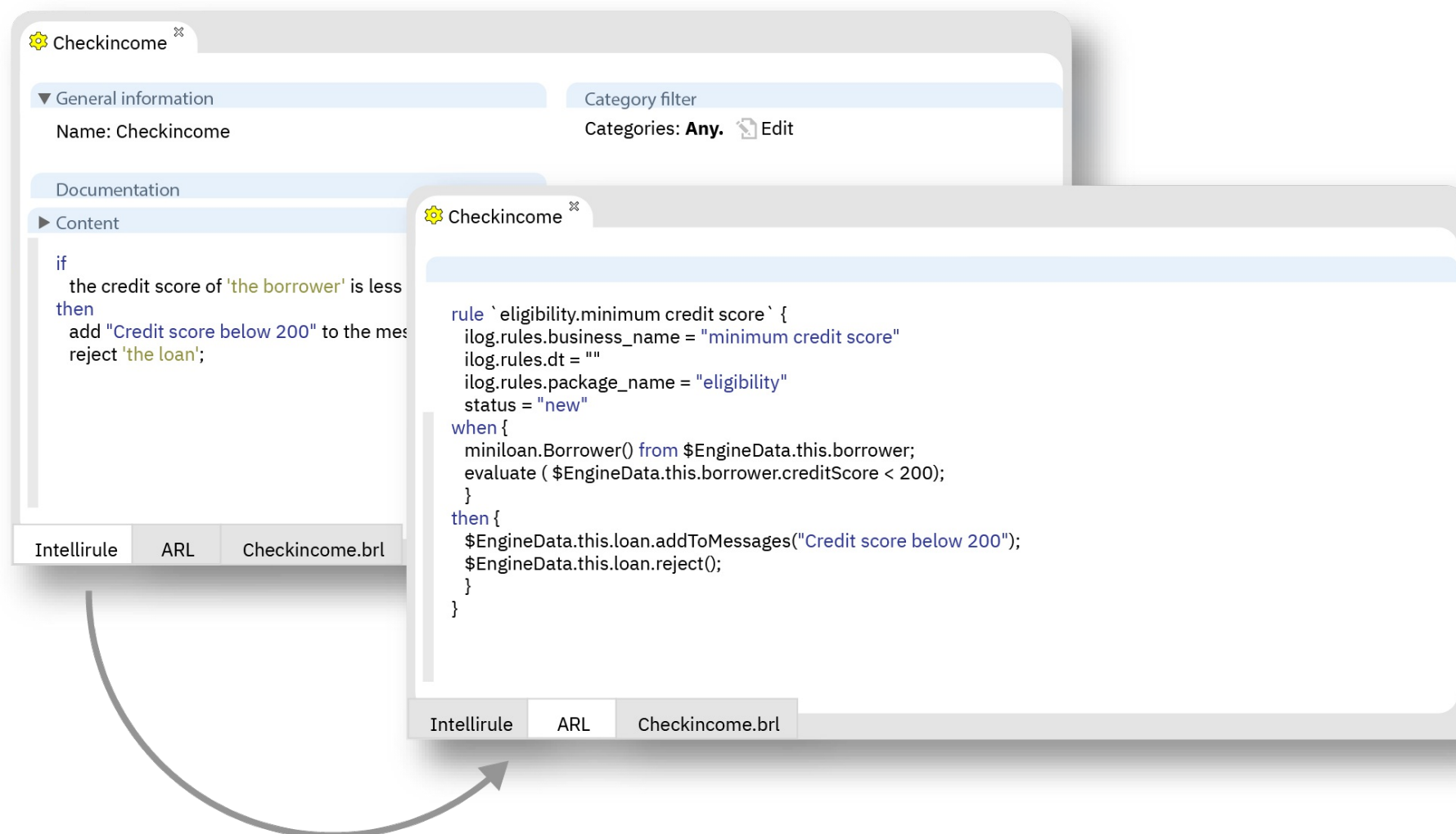
	<i>FlowIfElseAction :</i>  <i>Label</i> opt if <i>TestAssignmentSet</i> <i>FlowAction</i> <i>FlowElseAction</i> opt ;  <i>Label</i> opt if <i>TestAssignmentSet</i> { <i>FlowActionList</i> }  <i>FlowElseAction</i> opt;
•	<i>FlowElseAction :</i>  else <i>FlowAction</i>  else { <i>FlowActionList</i> }
•	<i>FlowSwitchAction :</i>  <i>Label</i> opt switch ( <i>Expression</i> )  { <i>Cases</i> opt <i>Default</i> opt }
•	<i>Cases:</i>  <i>CaseClause</i>  <i>Cases CaseClause</i>
•	<i>CaseClause :</i>  case <i>INTEGER_LITERAL:</i>  <i>FlowAction</i>  case <i>INTEGER_LITERAL:</i>  { <i>FlowActionList</i> }
•	<i>Default :</i>  default:  <i>FlowAction</i>  default:  { <i>FlowActionList</i> }
•	<i>FlowWhileAction :</i>  <i>Label</i> opt while <i>TestAssignmentSet</i> <i>LoopFlowAction</i>  while <i>TestAssignmentSet</i> { <i>LoopFlowActionList</i> }
•	<i>LoopFlowAction :</i>  <i>FlowAction</i>  break;  continue;
	<i>LoopFlowActionList:</i>  <i>LoopFlowAction LoopFlowActionList</i> opt

Parent topic: [IRL grammar](#)

## Advanced Rule Language (ARL)

For each Business Action Language (BAL) expression that you create for BOM-to-XOM mapping in the decision engine, you can review its equivalent Advanced Rule Language (ARL) expression. ARL is a read-only rule language that is similar in syntax to Java 7.

If you are familiar with Java 7, you can review how your BAL expressions are interpreted by the decision engine by reviewing the content in the ARL tab in Rule Designer.



BAL and ARL are equivalent in meaning, as you can see in the following two examples. In the first rule condition that is written in BAL, the credit score of a borrower is examined to check whether the borrower has a credit score less than 200:

```
if
 the credit score of 'the borrower' is less than 200
then
 add "Credit score below 200" to the messages of 'the loan' ;
 reject 'the loan';
```

The decision engine interprets the previous expression and produces its equivalent in ARL, readable from the ARL tab in Rule Designer. The following ARL expression describes the exact same rule condition as its counterpart in BAL:

```
rule `eligibility.minimum credit score` {
 ilog.rules.business_name = "minimum credit score"
 ilog.rules.dt = ""
 ilog.rules.package_name = "eligibility"
 status = "new"
when {
 miniloan.Borrower() from $EngineData.this.borrower;
 evaluate ($EngineData.this.borrower.creditScore < 200);
}
then {
 $EngineData.this.loan.addToMessages("Credit score below 200");
 $EngineData.this.loan.reject();
}
}
```

### [Differences between ARL and Java 7](#)

ARL is a read-only rule language that is similar in syntax to Java 7. To understand the difference between ARL and Java 7, review this list of ARL keywords and operators.

### [ARL rules examples](#)

The ARL we are going to describe in this section is only used to describe rules to help users to

disambiguate BAL.

**Parent topic:** [Rule languages](#)

# Differences between ARL and Java 7

ARL is a read-only rule language that is similar in syntax to Java 7. To understand the difference between ARL and Java 7, review this list of ARL keywords and operators.

## Aggregates

Unlike Java, ARL has aggregate operators that are used in rule descriptions.

## Anonymous classes

In Java, you can create anonymous class directly in the body of a method or constructor. In ARL, you cannot create anonymous classes.

## Back-quotes

Unlike Java, ARL uses back-quotes (``) to name variables that are not parsed as Java identifies. See the following example:

```
String `the lazy dog` = "pluto";
```

## Cast Operators

Much like in C#, you can use the cast operator "as" to do certain types of conversions between compatible reference types or nullable types. See the following example:

```
AClass anObject = o as AClass;
```

## Goto

ARL does not support goto.

## Inner classes

You can refer to the fields of an enclosing class by using the syntax 'A.this', where A represents the enclosing class. This mechanism is extended to refer to certain classes of the engine. For example, if 'ruleName' is ambiguous, you can print a rule name by writing the following code:

```
if (RuleInstance.this != null) note("ARL mapping of method M being called by rule "+ RuleInstance.this.ruleName);
```

## ARL Keywords

The following list is the complete set of keywords in the ARL rule language:

abstract	continue	final	in	not	queryTemplate	select	throw	refresh
aggregate	default	finally	insert	null	query	signature	throws	retract
as	do	for	instanceof	operator	once	static	transient	void
assert	else	from	interface	out	over	stipulation	true	volatile
break	enum	goto	match	package	repeatable	strictfp	try	when
case	evaluate	groupby	method	priority	restricts	super	typedef	where
catch	exists	if	modal	private	return	switch	update	while
class	explicit	implements	modify	property	rule	synchronized	updateEngineData	xpath
conditionTemplate	extends	implicit	native	protected	ruleset	then	updateGenerator	xs
const	false	import	new	public	ruleTemplate	this	updateGenerator	

**Loops**

foreach loops have a different syntax.

**Switch**

It is not necessary to use the keyword "break" at the end of each "case" section.

**Parent topic:** [Advanced Rule Language \(ARL\)](#)



## ARL rules examples

The ARL we are going to describe in this section is only used to describe rules to help users to disambiguate BAL.

We describe the most common statements corresponding to the BAL constructs in [BAL constructs](#).

### From parameters

BAL and ARL are equivalent in meaning, as you can see in the following comparison. The same rule condition is presented side by side:

BAL	ARL
<pre>definitions   set b to 'the borrower'; if   the credit score of b is less than 200 then   add "Credit score below 200" to the messages of 'the loan' ;   reject 'the loan';</pre>	<pre>rule `eligibility.minimum credit score` {   ilog.rules.business_name = "minimum credit score"   ilog.rules.dt = ""   ilog.rules.package_name = "eligibility"   status = "new"   when {     miniloan.Borrower() from \$EngineData.this.borrower;     b : miniloan.Borrower() from \$EngineData.this.borrower;     evaluate ( b.creditScore &lt; 200);   }   then {  \$EngineData.this.loan.addToMess ages("Credit score below 200");  \$EngineData.this.loan.reject();   } }</pre>

In the previous BAL example, a new variable (b) is defined by writing the following expression: ‘set b to ‘the borrower’;’. In the Miniloan ServiceParameters, one can see that ‘the borrower’ is the verbalization of the variable ‘borrower’, which is an instance of the BOM class Borrower. Notice also that ‘the loan’ is a verbalization of an instance of Loan.

If you are familiar with Java, you can see from the previous example that ARL has some particularities:

- The identifiers are written in free text between back-quotes: as `eligibility.minimum credit score`.
- The keywords rule and when are unique to ARL. Other examples are: ruleset, signature, evaluate, from, ruleset, match, many, in, flowtask, ruletask, and aggregate. For a complete list of keywords, see [ARL keywords](#).
- The syntactic construction \$EngineData.this allows BOM objects and ruleset parameters to be referenced. See [Execution class mapping](#).
- The declaration b : miniloan.Borrower() from \$EngineData.this.borrower; is called a binding. It means b is a new variable whose value is the ruleset parameter ‘borrower’ referenced by \$EngineData.this.borrower.

The ‘if’ expression in BAL corresponds to the ‘when’ expression in the ARL translation. Both languages have a then expression:

BAL	ARL
<pre>the credit score of b is less than 200</pre>	<pre>miniloan.Borrower() from \$EngineData.this.borrower;  b : miniloan.Borrower() from \$EngineData.this.borrower;</pre>

	<pre>evaluate ( b.creditScore &lt; 200);</pre>
--	------------------------------------------------

ARL is designed to express pattern matching, which is the operating principle of the underlying rule engine. The expression `miniloan.Borrower()` from `$EngineData.this.borrower`; means that the ruleset parameter 'borrower' must be an instance of the class `Borrower`, the binding: `'b : miniloan.Borrower()'` means that if an instance of `Borrower` is matched, it is referred to as 'b', finally the expression: `'evaluate ( b.creditScore < 200);'` returns the value of the expression: `'b.creditScore < 200'`. Considering that the semicolon behaves like a logical and, end-to-end these expressions mean the same as 'the credit score of b is less than 200' in a java-like dialect.

Let's go back to the notation `'$EngineData.this'`. It's necessary to use such a notation because there are many structures used below the keyword 'rule', in particular:

- `EngineData` to give access to BOM and ruleset parameters object
- `RunningEngine` to give access to methods of the underlying engine

For this reason, the keyword 'this' becomes ambiguous. A simple way to lift this ambiguity is to use the prefix 'this'. See [Execution class mapping](#).

The action of the conditions in both languages (introduced by `then`) are self-explanatory: If the condition, introduced by `when` or `if`, is true, then the corresponding actions are taken, in the ARL version, a series of calls to methods of the 'Loan' instance 'loan' which is, as told above, a ruleset parameter.

Match object in working memory

What happens when you modify the definitions section? Notice that the `from` keyword, previously used to reference a parameter, vanishes. Additionally, the binding `b : miniloan.Borrower()`; acts on each instance of `Borrower` found in the working memory. (The working memory is the set of objects to which the rule engine is applied.)

BAL	ARL
<pre>definitions   set b to 'the borrower'; if   the credit score of b is less than 200 then   add "Credit score below 200" to the messages of 'the loan' ;   reject 'the loan';</pre>	<pre>rule `eligibility.minimum credit score` {   ilog.rules.business_name = "minimum credit score"   ilog.rules.dt = ""   ilog.rules.package_name = "eligibility"   status = "new"   when {     b : miniloan.Borrower();     evaluate ( b.creditScore &lt; 200);   }   then {  \$EngineData.this.loan.addToMess ages("Credit score below 200");  \$EngineData.this.loan.reject();   } }</pre>

Collections

Imagine a scenario in which a single loan has several co-borrowers. The loan could be refused if one of the co-borrowers has a low credit score.

BAL	ARL
<pre>definitions   set b to a borrower in 'several co-borrowers' ; if   the credit score of b is less than 200</pre>	<pre>rule `eligibility.minimum credit score` {   ilog.rules.business_name = "minimum credit score"   ilog.rules.dt = ""   ilog.rules.package_name =</pre>

<pre>then   add "Credit score below 200" to the messages of 'the loan' ; reject 'the loan';</pre>	<pre>"eligibility"   status = "new"   when {     b : miniloan.Borrower() in \$EngineData.this.borrowers;     evaluate ( b.creditScore &lt; 200);   }   then {  \$EngineData.this.loan.addToMess ages("Credit score below 200");  \$EngineData.this.loan.reject();   } }</pre>
---------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The variable ‘several co-borrowers’ is the verbalization of the variable ‘borrowers’ defined as a collection of borrowers. Noticed that as borrowers is a collection the binding, ‘b : miniloan.Borrower() in \$EngineData.this.borrowers;’ is written with the keyword in instead of from because in this case it is an inclusion.

Binding and loops

The following rule condition defines the variable risky borrowers, which corresponds to borrowers whose yearly income is below 1000. If this list is not empty it prints all its elements. In the following ARL example, the definition of this variable is made by using a binding. The variable name followed by a colon (:), and an expression:

```
`risky borrowers`:aggregate {
 collect_class_1 : miniloan.Borrower($EngineData.this.borrower.yearlyIncome
< 1000);
}
```

The expression used to define `risky borrowers` is an *aggregate* (see below). In the action part of the expression, there is a loop iterating over the elements of the just created variable: it has the same syntax as in Java.

Similar to the example above, in the following example, the variable b is defined in ARL by using the binding: b : miniloan.Borrower() in \$EngineData.this.borrowers;;

BAL	ARL
<pre>definitions   set 'risky borrowers' to all borrowers where   the yearly income of 'the borrower' is less than 1000; if   the number of objects in 'risky borrowers' is more than 0 then   for each borrower called b, in 'risky borrowers':   - print  the name of b;</pre>	<pre>rule `doc.called` {   ilog.rules.business_name = "called"   ilog.rules.dt = ""   ilog.rules.package_name = "doc"   status = "new"   when {     miniloan.Borrower() from \$EngineData.this.borrower;     `risky borrowers`:aggregate {   collect_class_1 : miniloan.Borrower(\$EngineData.t his.borrower.yearlyIncome &lt; 1000);   }   do {  java.util.ArrayList&lt;miniloan.Bo rrower&gt;{collect_class_1};   }   evaluate (</pre>

	<pre>\$EngineData.this.borrower.name.equals("Jack"));     }     then {     {     for (miniloan.Borrower b : `risky borrowers`){     {  ilog.rules.brl.System.printMess age((b.name);     } } }</pre>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Evaluate

The constraints expressed in BAL on a variable are often translated in ARL by using the operator evaluate::

```
definitions
 set b to a borrower in 'several co-borrowers' ;
if
 the credit score of b is less than 200
then
 add "Credit score below 200" to the messages of 'the loan' ;
 reject 'the loan';
```

The constraint on the credit score b is expressed in ARL as evaluate ( b.creditScore < 200);.

The same constraint could be done directly in the binding: b : miniloan.Borrower(b.creditScore < 200) in \$EngineData.this.borrowers; This way of expressing constraints is often used in aggregates.

Aggregates

As in database aggregations, aggregates allow to group multiple values together to form a single value. There are various types of aggregates depending on the way values are grouped. The following aggregates are often used in BAL to ARL translations: collections, count, exists, match any.

Collections

Values can be accumulated in collections. For example, in the following rule we want to create a collection regrouping all borrowers whose yearly income is below a given threshold of 1000. To create this collection in ARL, we create a collection aggregate: collect\_class\_1 : miniloan.Borrower(\$EngineData.this.borrower.yearlyIncome < 1000); and add it to the ArrayList:

```
do {
 java.util.ArrayList<miniloan.Borrower>{collect_class_1};
}
```

The result returned by the previous do statement is bound to the variable of the binding.

BAL	ARL
<pre>definitions   set 'risky borrowers' to all   borrowers   where     the yearly income of 'the   borrower' is less than 1000; if   there is at most one   borrower in 'risky borrowers' then   print "there are risky   borrowers";</pre>	<pre>rule `doc.Collect` {   ilog.rules.business_name =   "Collect"   ilog.rules.dt = ""   ilog.rules.package_name =   "doc"   status = "new"   when {     miniloan.Borrower() from   \$EngineData.this.borrower;     risky borrowers:aggregate   {     collect_class_1 :   miniloan.Borrower(\$EngineData.t</pre>

	<pre>his.borrower.yearlyIncome &lt; 1000);     }     do {  java.util.ArrayList&lt;miniloan.Bor rower&gt;{collect_class_1};     }     evaluate ( `risky borrowers`.size() &lt;= 1);     }     then {  ilog.rules.brl.System.printMess age("there are risky borrowers");     } }</pre>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Count

Aggregation functions such as the count function can be used, for example, in the following rule condition expression to determine whether there are 8 borrowers with a yearly income superior to 1200. The aggregate count function counts the number of borrowers whose yearly income is more than 12000. The where clause attached to the aggregate is similar to the evaluate operator already defined. The boolean results of the where clause is bound to the variable of the binding:

BAL	ARL
<pre>if     there are 8 borrowers     where the yearly income of each borrower is more than 12000, then     print "there are at least 8 of them";</pre>	<pre>rule `doc.There are more` {     ilog.rules.business_name = "There are more"     ilog.rules.dt = ""     ilog.rules.package_name = "doc"     status = "new"     when {         aggregate_1:aggregate {             collect_1 : miniloan.Borrower(this.yearlyIn come &gt; 12000);         }         do {             count{collect_1};         }         where (aggregate_1 == 8);     }     then {  ilog.rules.brl.System.printMess age("there are at least 8 of them");     } }</pre>

Exists

The operator exists tests that there is *at least one* item in a collection of items. So, in the following rule testing there is at least one borrower is translated by an 'exists' clause:

BAL	ARL
<pre>if     there are 8 borrowers</pre>	<pre>rule `doc.There is at least one` {</pre>

where the yearly income of each borrower is more than 12000, then  
print "there are at least 8 of them";

```
ilog.rules.business_name =
"There is at least one"
ilog.rules.dt = ""
ilog.rules.package_name =
"doc"
status = "new"
when {
exists {

miniloan.Borrower(this.borrower
.yearlyIncome > 12000 &&
this.borrower.yearlyIncome <
1000000);
}
}
then {

ilog.rules.brl.System.printMess
age("it exists");
}
}
```

Match many

The match many instruction is used to translate decision tables. In the decision table below, the match many instruction follows its semantic. The first case in the first match many corresponds to the first condition debt to income of the first two lines. If the condition is satisfied, an embedded match many evaluates the corresponding conditions on the credit score lines. If the first condition on the credit score line is satisfied (a credit score between 0 and 200), then a then clause describes the action to be taken: the loan is refused.

The other match many evaluates the other conditions of the table and takes the corresponding actions. The conditions corresponding to disabled actions in the decision table do not have corresponding cases in the match many instruction:

the yearly repayment of 'the loan' \* 100 / the yearly income of 'the borrower' is at least <a number> and less than <a number>

	debt to income		credit score		message	rejected
	min	max	min	max		
1			0	200	debt-to-income too high c...	-
2	0 %	30 %	200	800	⊗	-
3			0	400	debt-to-income too high c...	-
4	30 %	45 %	400	800	⊗	-
5			0	600	debt-to-income too high c...	-
6	45 %	50 %	600	800	⊗	-
7	≥ 50 %		0	800	debt-to-income too high c...	-
8						-
9						-
10						-
11						-
12						-
13						-
14						-
15						-
16						-
17						-

GeneralDecision TableARLrepayment and score.dta

ARL translation of the decision table:

```
rule `eligibility.repayment and score` {
 ilog.rules.business_name = "repayment and score"
 ilog.rules.dt = "eligibility.repayment and score"
 ilog.rules.package_name = "eligibility"
 status = "new"
 when {
 miniloan.Borrower() from $EngineData.this.borrower;
```



```

 evaluate ($EngineData.this.borrower.yearlyIncome > 0);
 }
 match many {
 case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [0,30[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
 case ($EngineData.this.borrower != null &&
[0,200[.contains($EngineData.this.borrower.creditScore)) : then 1{
 $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
 $EngineData.this.loan.reject();
 }
 }
 case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [30,45[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
 case ($EngineData.this.borrower != null &&
[0,400[.contains($EngineData.this.borrower.creditScore)) : then 3{
 $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
 $EngineData.this.loan.reject();
 }
 }
 case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && [45,50[.contains(($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome))) : match many {
 case ($EngineData.this.borrower != null &&
[0,600[.contains($EngineData.this.borrower.creditScore)) : then 5{
 $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
 $EngineData.this.loan.reject();
 }
 }
 case ($EngineData.this.loan != null && ($EngineData.this.borrower !=
null && ($EngineData.this.loan.yearlyRepayment * 100) /
$EngineData.this.borrower.yearlyIncome >= 50)) : if ($EngineData.this.borrower
!= null && [0,800[.contains($EngineData.this.borrower.creditScore)){
 then 7{
 $EngineData.this.loan.addToMessages("debt-to-income too high
compared to credit score");
 $EngineData.this.loan.reject();
 }
 }
 }
}

```

**Parent topic:** [Advanced Rule Language \(ARL\)](#)

# Rule Designer Messages

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRA</a>	Messages pertaining to rule authoring
<a href="#">GBRE</a>	Messages pertaining to the rule engine
<a href="#">GBRM</a>	Messages pertaining to the model and the business object model (BOM)
<a href="#">GBRSB</a>	Messages pertaining to the business object model
<a href="#">GBRSD</a>	Messages pertaining to debug
<a href="#">GBRSE</a>	Messages pertaining to rule execution
<a href="#">GBRSM00</a>	Messages pertaining to the model (00 - 99)
<a href="#">GBRSM01</a>	Messages pertaining to the model (100-199)
<a href="#">GBRSM02</a>	Messages pertaining to the model (200-299)
<a href="#">GBRST</a>	Messages pertaining to tools
<a href="#">GBRSU</a>	Messages pertaining to core user interface
<a href="#">GBRT</a>	Messages pertaining to testing and simulation



# Rule Designer Messages - Messages pertaining to rule authoring

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRAB0001E</a>	Bad index for token: {0} (line: {2}, column: {3}).
<a href="#">GBRAB0002E</a>	Cannot add a child to token {0} (line: {2}, column: {3}).
<a href="#">GBRAB0003E</a>	Reference to unknown operator "{0}" in grammar (type: {1}) at: {2}.
<a href="#">GBRAB0004E</a>	Reference to unknown terminal "{0}" in grammar (type: {1}) at: {2}.
<a href="#">GBRAB0005E</a>	Invalid numeric value.
<a href="#">GBRAB0006E</a>	Empty numeric value.
<a href="#">GBRAB0007E</a>	Empty value.
<a href="#">GBRAB0008E</a>	The schema has no element.
<a href="#">GBRAB0009E</a>	Element type not found: {0}.
<a href="#">GBRAB0010E</a>	Grammar not set on this syntax tree.
<a href="#">GBRAB0011E</a>	There is no root named "{0}" in grammar.
<a href="#">GBRAB0012E</a>	Current grammar node is not a list: "{0}" (type: {1}) in path: {2}
<a href="#">GBRAB0013E</a>	The grammar node "{0}" has not been found in path: {1}
<a href="#">GBRAB0014E</a>	Bad index for syntax tree node {0} : {1}.
<a href="#">GBRAB0015E</a>	Node not found in the abstract syntax tree: {0}.
<a href="#">GBRAD0001E</a>	Exception raised while loading "{0}". See log file.
<a href="#">GBRAD0002E</a>	Exception raised while checking "{0}". See log file.
<a href="#">GBRAL0000W</a>	The search server cannot be started because it is not configured correctly. ({0})
<a href="#">GBRAL0001W</a>	Could not connect to the search server. ({0})
<a href="#">GBRAL0002W</a>	The search query failed to run. ({0})
<a href="#">GBRAL0003W</a>	The search request cannot be processed. Ensure that the server is configured correctly. ({0})
<a href="#">GBRAL0500E</a>	Bad configuration. Variable {0} is not set. Search service disabled.
<a href="#">GBRAL0501E</a>	Bad configuration. Invalid value for {0}. Supported search service providers are: {1}.
<a href="#">GBRAL0502E</a>	Bad configuration. Variable {0} must be set when search service {1} is enabled.
<a href="#">GBRAL0503E</a>	Bad configuration. Invalid value {0} for variable {1}.
<a href="#">GBRAL0504E</a>	Failed to start search service.
<a href="#">GBRAL0505I</a>	Search service is currently unavailable due to bad configuration.

# Rule Designer Messages - Messages pertaining to the rule engine

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBREA0001E</a>	The syntactic node of class "{0}" is not well formed
<a href="#">GBREA0002E</a>	The checking of a syntactic node of class "{0}" is not implemented
<a href="#">GBREA0003E</a>	The type "{0}" is undefined
<a href="#">GBREA0004E</a>	The primitive type "{0}" is undefined
<a href="#">GBREA0005E</a>	The type "{0}" is not a valid annotation type
<a href="#">GBREA0006E</a>	The type "{0}" is ambiguous, the possible meanings are "{1}"
<a href="#">GBREA0007E</a>	The type "{0}" is not a valid component type
<a href="#">GBREA0008E</a>	The type "{0}" is not a valid attribute type
<a href="#">GBREA0009E</a>	The type "{0}" is not a valid indexer type
<a href="#">GBREA0010E</a>	The type "{0}" is not a valid variable type
<a href="#">GBREA0011E</a>	Only a single extension is supported
<a href="#">GBREA0012E</a>	The type "{0}" is not a valid extension type
<a href="#">GBREA0013E</a>	Interface extension is not supported
<a href="#">GBREA0014E</a>	The type "{0}" is not a valid interface type
<a href="#">GBREA0015E</a>	The declaration of type "{0}" is recursive. Check the declaration of type "{1}"
<a href="#">GBREA0016E</a>	The type "{0}" cannot be used as the bound of a type parameter
<a href="#">GBREA0017E</a>	The type "{0}" is not a valid type argument
<a href="#">GBREA0018E</a>	The type "{0}" cannot be used as the bound of a type argument
<a href="#">GBREA0019E</a>	The type "{0}" cannot be used as the bound of a type argument
<a href="#">GBREA0020E</a>	The type "{0}" is not a valid exception type
<a href="#">GBREA0021E</a>	A type variable named "{0}" already exists
<a href="#">GBREA0</a>	A type named "{0}" already exists

<a href="#">022E</a>	
<a href="#">GBREA0023E</a>	The modifier "{0}" is unknown
<a href="#">GBREA0024E</a>	The modifier "{0}" has already been specified
<a href="#">GBREA0025E</a>	The modifier "{0}" cannot be used here
<a href="#">GBREA0026E</a>	The modifier "{0}" is conflicting with the other modifiers
<a href="#">GBREA0027E</a>	"{0}" cannot be assigned
<a href="#">GBREA0030E</a>	An attribute named "{0}" already exists
<a href="#">GBREA0031E</a>	Unable to find an attribute named "{0}" in type "{1}"
<a href="#">GBREA0032E</a>	Attribute "{0}" is write-only
<a href="#">GBREA0033E</a>	Attribute "{0}" is read-only
<a href="#">GBREA0034E</a>	Attribute "{0}" is not static
<a href="#">GBREA0035E</a>	Attribute "{0}" has "{1}" access
<a href="#">GBREA0036E</a>	All the branches of the getter of the attribute named "{0}" should return an instance of "{1}"
<a href="#">GBREA0037E</a>	The attribute "{0}" of type "{1}" cannot be initialized with a value of type "{2}"
<a href="#">GBREA0040E</a>	The name of the constructor should be "{0}"
<a href="#">GBREA0041E</a>	A constructor with a similar signature already exists
<a href="#">GBREA0042E</a>	Unable to find a constructor of signature "{0}" in type "{1}"
<a href="#">GBREA0043E</a>	Constructor "{0}" has "{1}" access
<a href="#">GBREA0044E</a>	Recursive constructor call
<a href="#">GBREA0045E</a>	The constructor named "{0}" should catch or declare exceptions of type "{1}"
<a href="#">GBREA0050E</a>	A method with a similar signature already exists
<a href="#">GBREA0051E</a>	This operator cannot be defined
<a href="#">GBREA0052E</a>	Cannot find a method "{1}" in type "{0}"
<a href="#">GBREA0053E</a>	Method "{0}" has "{1}" access
<a href="#">GBREA0054E</a>	Method "{0}" is not static
<a href="#">GBREA0055E</a>	All the branches of the body of the method named "{0}" should return an instance of "{1}"
<a href="#">GBREA0056E</a>	The method named "{0}" should catch or declare exceptions of type "{1}"
<a href="#">GBREA0060E</a>	An indexer with a similar signature already exists
<a href="#">GBREA0061E</a>	An indexer must declare at least one parameter
<a href="#">GBREA0</a>	Unable to find an indexer of signature "{0}" in type "{1}"

<a href="#">062E</a>	
<a href="#">GBREA0063E</a>	This indexer is not visible from here
<a href="#">GBREA0064E</a>	Indexer "{0}" is not static
<a href="#">GBREA0065E</a>	All the branches of the getter of the indexer named "{0}" should return an instance of "{1}"
<a href="#">GBREA0070E</a>	A variable named "{0}" already exists
<a href="#">GBREA0071E</a>	A variable of type "{0}" cannot be initialized with a value of type "{1}"
<a href="#">GBREA0072E</a>	Unable to find a variable named "{0}"
<a href="#">GBREA0073E</a>	Variable "{0}" has not been initialized
<a href="#">GBREA0074E</a>	Variable "{0}" might not have been initialized
<a href="#">GBREA0080E</a>	This value could not be checked
<a href="#">GBREA0081E</a>	This value is ambiguous
<a href="#">GBREA0082E</a>	A value of type "{0}" cannot be initialized with a value of type "{1}"
<a href="#">GBREA0083E</a>	This literal value is undefined
<a href="#">GBREA0084E</a>	The parsing of literal value "{0}" has failed
<a href="#">GBREA0085E</a>	Cannot use a value of type "{0}" when type "{1}" is expected
<a href="#">GBREA0086E</a>	A constant cannot be assigned a value
<a href="#">GBREA0090E</a>	The unary operator "{0}" is undefined
<a href="#">GBREA0091E</a>	Cannot find the operator "{0}" to apply to "{1}"
<a href="#">GBREA0092E</a>	The binary operator "{0}" is undefined
<a href="#">GBREA0093E</a>	Cannot find an operator "{0}" to apply to "{1}", "{2}"
<a href="#">GBREA0094E</a>	Post operator "{0}" is not supported
<a href="#">GBREA0095E</a>	The ternary operator "{0}" is undefined
<a href="#">GBREA0096E</a>	Unable to find a "{0}" ternary operator that takes ("{1}", "{2}", "{3}")
<a href="#">GBREA0097E</a>	The cast operation from type "{0}" to type "{1}" is undefined
<a href="#">GBREA0098E</a>	The "as" cast operation from type "{0}" to type "{1}" is undefined because it is restricted to reference types
<a href="#">GBREA0099E</a>	Literal "{0}" is unexpected here
<a href="#">GBREA0100E</a>	Division by zero is not allowed "{0}"
<a href="#">GBREA0105E</a>	An array dimension or an array initializer is expected
<a href="#">GBREA0106E</a>	An array initializer may not be specified when an array dimension exists
<a href="#">GBREA0</a>	Unable to find a constructor of signature "{0}" in array type "{1}"

<a href="#">107E</a>	
<a href="#">GBREA0110E</a>	An aggregate is unexpected here
<a href="#">GBREA0111E</a>	At least one generator is required in an "aggregate"
<a href="#">GBREA0112E</a>	At least one argument is required in an "aggregate"
<a href="#">GBREA0113E</a>	Cannot find aggregation function called "{0}"
<a href="#">GBREA0114E</a>	The type "{0}" of the value is not a valid collection type
<a href="#">GBREA0120E</a>	A constant value is expected
<a href="#">GBREA0121E</a>	A "default" already exists for this "switch"
<a href="#">GBREA0122E</a>	A case with the same value already exists for this "switch"
<a href="#">GBREA0123E</a>	A value is expected for this "switch" case
<a href="#">GBREA0124E</a>	The type "{0}" of this "case" is incompatible with the expected type "{1}"
<a href="#">GBREA0125E</a>	A default result is expected for this "switch"
<a href="#">GBREA0126E</a>	A value of type "{0}" cannot be passed to a "switch"
<a href="#">GBREA0127E</a>	A "default" already exists for this "switch"
<a href="#">GBREA0128E</a>	A case of type "{0}" is not compatible with a switch on type "{1}"
<a href="#">GBREA0129E</a>	A case with the same value already exists for this "switch"
<a href="#">GBREA0130E</a>	A body is expected for this "switch" case
<a href="#">GBREA0140E</a>	"return" is unexpected here
<a href="#">GBREA0141E</a>	The type "{0}" of the value is incompatible with the specified return type "{1}"
<a href="#">GBREA0142E</a>	"break" is unexpected here
<a href="#">GBREA0143E</a>	"continue" is unexpected here
<a href="#">GBREA0144E</a>	The type "{0}" is already caught
<a href="#">GBREA0145E</a>	A value of type "{0}" cannot be thrown
<a href="#">GBREA0146E</a>	The exception type "{0}" should be caught or declared as thrown
<a href="#">GBREA0147E</a>	Unreachable statement "{0}"
<a href="#">GBREA0148E</a>	Missing "return" statement
<a href="#">GBREA0149E</a>	This value is not a statement
<a href="#">GBREA0150E</a>	This value is not a test, its type is not "boolean"
<a href="#">GBREA0160E</a>	The argument "{0}" of the annotation of class "{1}" is invalid
<a href="#">GBREA0</a>	The type "{0}" can't be used as a base type in a restriction

<a href="#">161E</a>	
<a href="#">GBREA0162E</a>	A type named "{0}" already exists
<a href="#">GBREA0163E</a>	Wrong number of type arguments for generic type "{0}". Expected "{1}", got "{2}"
<a href="#">GBREA0170E</a>	"this" cannot be used in this scope
<a href="#">GBREA0171E</a>	"super" cannot be used in this scope
<a href="#">GBREB0001E</a>	The "{0}" exception has been thrown:
<a href="#">GBREB0002I</a>	Location stack trace:
<a href="#">GBREB0003I</a>	Caused by: {0}
<a href="#">GBREB0004I</a>	at line {0}, column {1}
<a href="#">GBREB0005I</a>	in file "{0}", at line {1}, column {2}
<a href="#">GBREB0006I</a>	at offset {0}, with length {1}
<a href="#">GBREB0007E</a>	Version read {0} expected version {1}
<a href="#">GBREB0008E</a>	Not a valid decision engine jar file or stream
<a href="#">GBREB0009W</a>	Generated with Java specification version {0}, used in version {1}
<a href="#">GBRED0001I</a>	in the conditions of rule "{0}", between line {1}, column {2} and line {3}, column {4}
<a href="#">GBRED0002I</a>	in the conditions of rule "{0}"
<a href="#">GBRED0003I</a>	in the action "{1}" of rule "{0}", between line {2}, column {3} and line {4}, column {5}
<a href="#">GBRED0004I</a>	in the action "{1}" of rule "{0}"
<a href="#">GBRED0005I</a>	in rule "{0}", between line {1}, column {2} and line {3}, column {4}
<a href="#">GBRED0006I</a>	in the action of the rule "{0}"
<a href="#">GBRED0007E</a>	A ruleset named "{0}" already exists
<a href="#">GBRED0007I</a>	in the conditions of rule "{0}", at offset {1}, and length {2}
<a href="#">GBRED0008E</a>	The namespace "{0}" does not contain any rule
<a href="#">GBRED0008I</a>	in rule "{0}", at offset {1}, and length {2}
<a href="#">GBRED0009E</a>	Unable to find the rule named "{0}"
<a href="#">GBRED0010E</a>	Could not load custom exception handler class "{0}"
<a href="#">GBRED0011E</a>	The rule named "{0}" requires an environment of type "{1}" which is incompatible with an environment of type "{2}"
<a href="#">GBRED0011I</a>	While integrating the custom exception handler "{0}"
<a href="#">GBRED0012E</a>	Declared custom exception handler class "{0}" has to implement "{1}"
<a href="#">GBRED0</a>	The custom exception handler class "{0}" has no public default constructor



<a href="#">013E</a>	
<a href="#">GBRED0014E</a>	A rule named "{0}" already exists
<a href="#">GBRED0015E</a>	The return type "{0}" is not compliant with the expected return type "{1}"
<a href="#">GBRED0016E</a>	A product condition is expected
<a href="#">GBRED0017E</a>	A case with the same value already exists
<a href="#">GBRED0018E</a>	A variable named "{0}" already exists
<a href="#">GBRED0019E</a>	The type "{0}" is not a valid condition type
<a href="#">GBRED0020E</a>	All the branches of this rule content should return an instance of "{0}"
<a href="#">GBRED0021E</a>	The type "{0}" is not a valid engine data type
<a href="#">GBRED0022E</a>	An object of the type "{0}" cannot be used as an argument to "stop"
<a href="#">GBRED0023E</a>	A value of type "{0}" cannot be used as an "in" generator
<a href="#">GBRED0024E</a>	Unexpected element "{1}" in the "{0}" stipulation
<a href="#">GBRED0025E</a>	A ruleset property stipulation must be located in the body of the ruleset
<a href="#">GBRED0026E</a>	The parameter "{0}" of the instruction "{1}" of the stipulation "{2}" is not correct
<a href="#">GBRED0027E</a>	No matching condition template of namespace "{0}" and of name "{1}" declared in the ruleset
<a href="#">GBRED0028E</a>	Duplicate declaration of the "{0}" condition template
<a href="#">GBRED0029E</a>	Recursive declaration in the "{0}" condition template
<a href="#">GBRED0030E</a>	No matching rule template of namespace "{0}" and of name "{1}" declared in the ruleset
<a href="#">GBRED0031E</a>	Duplicate declaration of the "{0}" rule template
<a href="#">GBRED0032E</a>	A rule property stipulation must be located in the body of the ruleset
<a href="#">GBRED0033E</a>	Plug-in uniqueness error: two declared compilation plug-ins of class "{0}" and "{1}" share the same id "{2}"
<a href="#">GBRED0034E</a>	Plug-in dependency cycle found involving the plug-in of class "{0}"
<a href="#">GBRED0035E</a>	You declared too many references to the current enclosing class instance in a finder of the "{0}" stipulation
<a href="#">GBRED0036E</a>	Invalid key in a finder of the "{0}" stipulation
<a href="#">GBRED0037E</a>	No key found in a finder declaration of the "{0}" stipulation
<a href="#">GBRED0038E</a>	Arity impossible to infer from signature in a finder declaration of the "{0}" stipulation
<a href="#">GBRED0039E</a>	Must be an Iterable or an Array
<a href="#">GBREF1001E</a>	Cannot find a main task named "{0}"
<a href="#">GBREF1002E</a>	A task named "{0}" already exists
<a href="#">GBREF2</a>	task not found

<a href="#">001E</a>	
<a href="#">GBREF2002E</a>	The rule {1} in task {0} has a dynamic priority, which is not allowed for SORTED tasks
<a href="#">GBREF3001W</a>	A ruleset containing homogeneous rules is recommended for sequential task "{0}"
<a href="#">GBREF3002W</a>	A ruleset with at least one rule is recommended for rule task "{0}"
<a href="#">GBREF3003W</a>	A ruleset with no object modification in the action part is recommended for Fastpath and Sequential algorithms in rule task "{0}"
<a href="#">GBREF4001E</a>	Engine not running
<a href="#">GBREF4002E</a>	No main task set
<a href="#">GBREF4003E</a>	Stopped by installed controller
<a href="#">GBREF4004E</a>	Cannot use dynamic ordering with FIRINGKIND set to FIRST_ELIGIBLE_RULE
<a href="#">GBREF4005E</a>	does not work on this kind of engine
<a href="#">GBREF4006E</a>	The specified task "{0}" has not been defined
<a href="#">GBREF4007I</a>	in task "{0}"
<a href="#">GBREF4008I</a>	in task "{0}", between line {1}, column {2} and line {3}, column {4}
<a href="#">GBREM0008E</a>	Cannot migrate postfix operation used as a value
<a href="#">GBREM0009E</a>	Cannot migrate context value
<a href="#">GBREM0010E</a>	Cannot migrate invocation of method {0}, no corresponding method found in type {1}
<a href="#">GBREM0011E</a>	Cannot migrate invocation of instance method {0} as the instance could not be migrated
<a href="#">GBREM0012E</a>	Cannot migrate access to the value of attribute {0}
<a href="#">GBREM0013E</a>	Cannot migrate invocation of function {0}
<a href="#">GBREM0014E</a>	Cannot migrate binding of null
<a href="#">GBREM0015E</a>	Cannot migrate operator unknown
<a href="#">GBREM0016E</a>	Cannot migrate usage of binary operator {0}
<a href="#">GBREM0017E</a>	Cannot migrate usage of unary operator {0}
<a href="#">GBREM0018E</a>	Cannot migrate assignment
<a href="#">GBREM0019E</a>	Cannot migrate parameter/package variable {0}. The variable is of type {1} whereas the default value is of type {2}
<a href="#">GBREM0020E</a>	Cannot migrate call to IlrContext.invokeFunction, function {0} cannot be found
<a href="#">GBREM0021E</a>	Cannot migrate call to IlrContext.invokeFunction, function {0} has not been migrated
<a href="#">GBREM0022E</a>	Cannot migrate call to IlrContext.invokeFunction with a non inline creation of an arguments array
<a href="#">GBREM0023E</a>	Cannot migrate call to IlrContext.invokeFunction without an array as a parameter
<a href="#">GBREM0</a>	Cannot migrate call IlrContext.invokeFunction when the function name is not constant



<a href="#">024E</a>	
<a href="#">GBREM0025E</a>	Cannot migrate call <code>llrContext.invokeFunction({0},...)</code> . Internal error the engine value is missing.
<a href="#">GBREM0026E</a>	Cannot migrate access to package name.
<a href="#">GBREM0027E</a>	Cannot migrate deprecated <code>{0}</code> statement
<a href="#">GBREM0028E</a>	Rule instance access cannot be migrated in this scope
<a href="#">GBREM0029E</a>	Events and time are not supported. Cannot migrate <code>{0}</code>
<a href="#">GBREM0030E</a>	"collect *from* <source>" expression is not supported
<a href="#">GBREM0031E</a>	"collect in <source> *where*" expression is not supported
<a href="#">GBREM0032E</a>	An update refresh action has been detected. It is forbidden because the update refresh is not migrated.
<a href="#">GBREM0033E</a>	Could not migrate rule property <code>{0}</code>
<a href="#">GBREM0034E</a>	Could not migrate access to rule property <code>{0}</code>
<a href="#">GBREM0035E</a>	Cannot migrate the hasher <code>{0}</code>
<a href="#">GBREM0036E</a>	Events and time are not supported. Cannot migrate temporal conditions beginning with "event"
<a href="#">GBREM0037E</a>	Only <code>llrDefaultCollector</code> is supported when migrating a collect condition
<a href="#">GBREM0038E</a>	Cannot translate <code>llrContext.getCurrentTask</code> outside of task actions
<a href="#">GBREM0039E</a>	Cannot translate <code>llrContext.currentTask</code> outside of task actions
<a href="#">GBREM0040E</a>	Cannot translate dynamic select using rule array selections for task <code>{0}</code> .
<a href="#">GBREM0041E</a>	Cannot migrate access to package name in dynamic select.
<a href="#">GBREM0042E</a>	Cannot migrate access to rule name in dynamic select.
<a href="#">GBREM0043E</a>	Internal error cannot migrate condition variable of type <code>{0}</code>
<a href="#">GBREM0044E</a>	Internal error cannot migrate negation of <code>{0}</code>
<a href="#">GBREM0045E</a>	Cannot migrate instanceof <code>{0}</code>
<a href="#">GBREM0046E</a>	Cannot migrate match updown
<a href="#">GBREM0047E</a>	stop task does not exist.
<a href="#">GBREM0048E</a>	Internal migration error when trying to migrate <code>{0}</code>
<a href="#">GBREM0048W</a>	Property " <code>{0}</code> " is not used by the decision engine
<a href="#">GBREM0049W</a>	"ilrmain" function is not migrated
<a href="#">GBREM0050E</a>	Internal error migration when trying migrate B2X. Got execution factory parser errors <code>{0}</code>
<a href="#">GBREM0051E</a>	Cannot migrate variable " <code>{0}</code> "
<a href="#">GBREM0</a>	Missing parameter to call super constructor " <code>{0}</code> " from constructor " <code>{1}</code> "

<a href="#">052E</a>	
<a href="#">GBREM0053W</a>	Don't know what to do with parameter "{0}" : there is no attribute with such a name in class "{1}", a throw statement has been generated in constructor
<a href="#">GBREM0054E</a>	Cannot find execution class "{0}" for translating business class "{1}"
<a href="#">GBREM0055E</a>	Error when parsing B2X IRL body : {0}
<a href="#">GBREM0056E</a>	Error in B2X : {0}
<a href="#">GBREM0057W</a>	Warning in B2X : {0}
<a href="#">GBREP1001E</a>	Must be an Iterable, an Array, or an Enumeration
<a href="#">GBREP1002E</a>	"exists" not supported in sequential mode without a generator
<a href="#">GBREP1003E</a>	"not" not supported in sequential mode without a generator
<a href="#">GBREP1004E</a>	"aggregate" not supported in sequential mode without a generator
<a href="#">GBREP1005E</a>	"or" not supported in sequential mode
<a href="#">GBREP2005E</a>	does not work on this kind of engine
<a href="#">GBRETO001E</a>	Failed to transform method "{0}"
<a href="#">GBRETO002E</a>	Failed to transform type "{0}"
<a href="#">GBRETO003E</a>	Cannot find an operator "{0}" to apply to "{1}","{2}"
<a href="#">GBRETO004E</a>	Failed to transform value of attribute "{0}"
<a href="#">GBRETO005E</a>	Failed to transform assignment of attribute "{0}"
<a href="#">GBRETO006E</a>	Failed to transform value of indexer "{0}"
<a href="#">GBRETO007E</a>	Failed to transform assignment of indexer "{0}"
<a href="#">GBRETO008E</a>	Failed to transform invocation of method "{0}"
<a href="#">GBRETO009E</a>	Failed to transform usage of constructor "{0}"
<a href="#">GBRETO010E</a>	Failed to retrieve access to type "{0}" in a body of type "{1}"
<a href="#">GBRETO011E</a>	Cannot convert "{0}" to type "{1}"
<a href="#">GBRETO012E</a>	Failed to create instance of abstract class "{0}"
<a href="#">GBRETO013E</a>	Failed to transform indexer "{0}"
<a href="#">GBRETO030I</a>	While transforming the implementation of method "{0}"
<a href="#">GBRETO031I</a>	While transforming the implementations of the members of class "{0}"
<a href="#">GBRETO032I</a>	While transforming the statement "{0}"
<a href="#">GBRETO033I</a>	While transforming the implementation of constructor "{0}"
<a href="#">GBRETO</a>	While transforming the implementation of the getter of attribute "{0}"

<a href="#">034I</a>	
<a href="#">GBRETO035I</a>	While transforming the implementation of the setter of attribute "{0}"
<a href="#">GBRETO036I</a>	While transforming the value "{0}"
<a href="#">GBRETO037I</a>	While transforming the initial value of attribute "{0}"
<a href="#">GBRETO038I</a>	While applying model rewriter "{0}"
<a href="#">GBRETO039I</a>	While transforming the declaration of constructor "{0}"
<a href="#">GBRETO040I</a>	While transforming the declaration of method "{0}"
<a href="#">GBRETO041I</a>	While transforming the declaration of attribute "{0}"
<a href="#">GBRETO042I</a>	While transforming the declaration of indexer "{0}"
<a href="#">GBRETO043I</a>	While transforming the declaration of the members of class "{0}"
<a href="#">GBRETO050E</a>	Duplicated service installers for the service class "{0}"
<a href="#">GBRETO051E</a>	The engine service "{0}" cannot be instantiated because it depends on another service "{1}" that is missing
<a href="#">GBRETO052E</a>	Found a dependency loop related to the installer of the service class "{0}"
<a href="#">GBRETO053E</a>	The engine service "{0}" is mandatory and was neither provided nor created
<a href="#">GBREW0027I</a>	While applying business to execution (B2X) model mapping
<a href="#">GBREX0001E</a>	Cannot find execution class "{0}" for translating business class "{1}"
<a href="#">GBREX0002E</a>	Cannot find execution generic class "{0}" for translating business generic class "{1}"
<a href="#">GBREX0003E</a>	Cannot map business class "{0}" to an execution class
<a href="#">GBREX0004E</a>	Cannot find extender "{0}" for business class "{1}"
<a href="#">GBREX0005E</a>	Extender method "{0}" must be static
<a href="#">GBREX0006E</a>	Cannot find class "{0}" in business object model
<a href="#">GBREX0007E</a>	Cannot find generic class "{0}" in business object model
<a href="#">GBREX0008E</a>	Mismatch of static modifier between "{0}" and "{1}"
<a href="#">GBREX0009E</a>	Mismatch of member type, type "{0}" is returned by member "{1}", whereas the type "{2}" or a subclass is expected.
<a href="#">GBREX0010W</a>	Mismatch of member type, type "{0}" is returned by member "{1}", whereas type "{2}" or a subclass is expected. It might come from the use of covariance.
<a href="#">GBREX0011E</a>	Cannot find method "{0}" in execution class "{1}"
<a href="#">GBREX0012I</a>	In the B2X getter for attribute "{0}" of class "{1}"
<a href="#">GBREX0013I</a>	In the B2X setter for attribute "{0}" of class "{1}"
<a href="#">GBREX0014I</a>	In the B2X body for method "{0}" of class "{1}"
<a href="#">GBREX0</a>	In the B2X body for constructor "{0}" of class "{1}"

<a href="#">015I</a>	
<a href="#">GBREX0016I</a>	In the B2X tester of class "{1}"
<a href="#">GBREX0017E</a>	Business class "{0}" is marked as not translated
<a href="#">GBREX0018E</a>	Business attribute "{0}" is marked as not translated
<a href="#">GBREX0019E</a>	Business method "{0}" is marked as not translated
<a href="#">GBREX0020E</a>	Business constructor "{0}" is marked as not translated
<a href="#">GBREX0021E</a>	Cannot find attribute "{0}" in execution class "{1}"
<a href="#">GBREX0022E</a>	Cannot find getter for attribute "{0}"
<a href="#">GBREX0023E</a>	Cannot find setter for attribute "{0}"
<a href="#">GBREX0024E</a>	Syntax error in pattern "{0}" : "{1}"
<a href="#">GBREX0025E</a>	B2X cannot work without an engine data class concrete implementation
<a href="#">GBREX0026W</a>	The attributes "{0}" and "{1}" in business enum class "{2}" are both mapped to the same value
<a href="#">GBREX0028E</a>	Cannot generate the implementation of method "{0}" for type "{1}"
<a href="#">GBREX0029E</a>	Attribute "{1}" of class "{0}" is a collection of type "{2}", which is not supported
<a href="#">GBREX0030E</a>	Error "{0}" about "{1}" may be related to the usage of Classic Rule Engine API in the BOM
<a href="#">GBREX0031E</a>	Attribute "{1}" of business class "{0}" is mapped to a readonly attribute
<a href="#">GBREX0032E</a>	Attribute "{1}" of business class "{0}" is mapped to a writeonly attribute
<a href="#">GBREX0033E</a>	Cannot load execution class "{0}" for translating business class "{1}", got "{2}"
<a href="#">GBREX0034E</a>	Cannot load execution generic class "{0}" for translating business generic class "{1}", got "{2}"
<a href="#">GBREX0035E</a>	Return found in setter for attribute "{0}"
<a href="#">GBREX0200E</a>	in B2X body of {0}, at line {1}, column {2}
<a href="#">GBREX1000E</a>	No importer registered for node "{0}"
<a href="#">GBREX1001E</a>	Exception "{0}" occurred: "{1}" when importing the node "{2}"
<a href="#">GBREX1002E</a>	Unexpected empty node "{0}"
<a href="#">GBREX1003E</a>	Parse exception "{0}" occurred when reading "{1}"
<a href="#">GBREX1004E</a>	Missing node "{0}" in the node "{1}"
<a href="#">GBREX1005E</a>	Cannot load class "{0}" when reading the node "{1}"
<a href="#">GBREX1006E</a>	Class "{0}" must have a default constructor or a constructor marked with "dataio.default"
<a href="#">GBREX1007W</a>	Unknown attribute "{0}" when reading node "{1}"
<a href="#">GBREX1</a>	Cannot set the value of read-only attribute "{0}"

<a href="#">008E</a>	
<a href="#">GBREX1009E</a>	Missing reference in enum value node "{0}"
<a href="#">GBREX1010E</a>	Cannot find static attribute named "{0}" when reading "{1}" node
<a href="#">GBREX1011E</a>	Cannot find component type "{0}"
<a href="#">GBREX1012E</a>	Parameter "{0}" is missing for constructing an instance of class "{1}"
<a href="#">GBREX1013E</a>	Component type is missing in array node
<a href="#">GBREX1022E</a>	Object "{0}" of class "{1}" must be of expected type "{2}"
<a href="#">GBREX1023E</a>	Method not accessible while writing or reading objects
<a href="#">GBREX1024W</a>	Cannot instantiate collection class "{0}", the default "java.util.ArrayList" will be used
<a href="#">GBREX1025W</a>	Cannot instantiate map class "{0}", the default "java.util.HashMap" will be used
<a href="#">GBREX1026I</a>	While importing the value of attribute "{0}" of expected type "{1}"
<a href="#">GBREX1027I</a>	While importing the value of a collection item at index "{0}"
<a href="#">GBREX1028I</a>	While importing the content of an object of type "{0}"
<a href="#">GBREX1050E</a>	No exporter registered for the class "{0}"
<a href="#">GBREX1051E</a>	Cannot export attribute "{0}" with value of class "{1}"
<a href="#">GBREX1052E</a>	Exception {0}:{1} occurred when exporting value of attribute "{2}"
<a href="#">GBREX1053E</a>	Cannot find static attribute corresponding to value "{0}" when exporting node "{1}"
<a href="#">GBREX1080W</a>	Cannot use constructor "{0}" for creating instance of class "{1}". No usable importer will be created for this class.
<a href="#">GBREX1081W</a>	Cannot use constructor "{0}" for creating instance of class "{1}". No usable converter will be created for this class.
<a href="#">GBREX1082W</a>	In the constructor tagged "dataio.default" of class "{0}", the parameter "{1}" of type "{2}" is not compatible with attribute "{1}" of type "{3}"
<a href="#">GBREX1083W</a>	In the constructor tagged "dataio.default" of class "{0}", there is no attribute named as the parameter "{1}"
<a href="#">GBREX1084W</a>	Business class "{0}" is mapped to "{1}" without a tester. All instances of "{1}" will be considered as being of business class "{0}" if property dataio.ignore is not true
<a href="#">GBREX1100E</a>	Expected "{0}", got "{1}"
<a href="#">GBREX1101E</a>	Unexpected end of document
<a href="#">GBREX1102I</a>	At line {0} column {1}
<a href="#">GBREX1103E</a>	Expected digit, got "{0}"
<a href="#">GBREX1104E</a>	Expected hexadecimal number, got "{0}"
<a href="#">GBREX1105E</a>	While reading unfinished string "{0}", got unexpected character "{1}" (in hexadecimal)
<a href="#">GBREX1106E</a>	Invalid escape sequence, got "{0}"
<a href="#">GBREX1</a>	Invalid beginning of JSON value, got unexpected character "{0}"

<a href="#">107E</a>	
<a href="#">GBREX1 108E</a>	Cannot retrieve value using JSONPATH "{0}"
<a href="#">GBREX1 109E</a>	Type "{0}" should have a "value" element
<a href="#">GBREX1 110E</a>	Unexpected element "{0}" in type "{0}"
<a href="#">GBREX1 111E</a>	Enumerated type "{0}" should have a "\$enumref" element
<a href="#">GBREX1 112W</a>	Ignoring unexpected element "{0}"
<a href="#">GBREX1 113E</a>	Unexpected string "{0}" as a value of type "{1}"
<a href="#">GBREX1 114E</a>	Expected end of document, got "{0}"
<a href="#">GBREX1 115E</a>	Unexpected end of document while reading unfinished string "{0}"



# Rule Designer Messages - Messages pertaining to the model and the business object model (BOM)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRMI0001E</a>	Unknown attribute name
<a href="#">GBRMI0002E</a>	unknown component property name
<a href="#">GBRMI0003E</a>	Unknown type when creating the property {0}
<a href="#">GBRMI0004E</a>	Parameter declaration is too late
<a href="#">GBRMI0005E</a>	unknown method name
<a href="#">GBRMI0006E</a>	Wrong domain definition
<a href="#">GBRMI0007E</a>	Unexpected domain outside domain intersection or collection domain
<a href="#">GBRMI0008E</a>	Unexpected modifier {0}
<a href="#">GBRMI0009E</a>	invalid super class {0}
<a href="#">GBRMI0010E</a>	invalid exception class {0}
<a href="#">GBRMI0011E</a>	Type {0} cannot be changed: the metaclass is not an IlrMutableClass
<a href="#">GBRMI0012E</a>	The type of attribute {0} is unknown when trying to read the default value
<a href="#">GBRMI0013E</a>	Unknown type when creating the attribute {0}
<a href="#">GBRMI0014E</a>	Unknown return type when creating the indexed property {0}
<a href="#">GBRMI0015E</a>	Unknown return type when creating the method {0}
<a href="#">GBRMI0016E</a>	Cannot change the type of the member {0} to {1}
<a href="#">GBRMI0017E</a>	{0} errors found while parsing object model
<a href="#">GBRMI0018E</a>	Method {0} not found for indexed component property {1}
<a href="#">GBRMI0019E</a>	Method {0} not found for component property {1}
<a href="#">GBRMI0020W</a>	Cannot find formatter {0}
<a href="#">GBRMI0021W</a>	Cannot instantiate formatter {0}
<a href="#">GBRMI0022W</a>	Class {0} already exists. The new declaration is ignored.
<a href="#">GBRMI0023W</a>	Modifier {0} specified more than once
<a href="#">GBRMI0024E</a>	Cannot not bind non generic type {0}
<a href="#">GBRMI0025E</a>	Cannot not bind generic type {0} with parameters {1}
<a href="#">GBRMI0026E</a>	Cannot bind {0} which is not a class
<a href="#">GBRMI0027E</a>	Cannot include {0}: file not found
<a href="#">GBRMI0028E</a>	Cannot include {0}: I/O Exception {1}
<a href="#">GBRMI0029E</a>	Mismatch in properties
<a href="#">GBRMI0030E</a>	Unresolved type {0}
<a href="#">GBRMI0031E</a>	identifier expected, found "{0}"
<a href="#">GBRMI0032E</a>	String literal expected, found "{0}"

<a href="#">GBRMI0033E</a>	Word expected, found "{0}"
<a href="#">GBRMI0034E</a>	End of file expected, found "{0}"
<a href="#">GBRMI0035E</a>	integer value expected, found "{0}"
<a href="#">GBRMI0036E</a>	Symbol "{0}" expected, found "{1}"
<a href="#">GBRMI0037E</a>	Keyword "{0}" expected, found "{1}"
<a href="#">GBRMI0038E</a>	Word "{0}" expected, found {1}
<a href="#">GBRMI0039E</a>	Expected "class" or "interface" expected, found "{0}"
<a href="#">GBRMI0040E</a>	Class name expected, found primitive type {0}
<a href="#">GBRMI0041E</a>	Minimal cardinality has to be more than 0. It cannot be {0}
<a href="#">GBRMI0042E</a>	invalid destructor declaration
<a href="#">GBRMI0043E</a>	An operator has to be a method
<a href="#">GBRMM0001E</a>	No BOM found
<a href="#">GBRMM0002E</a>	Cannot migrate type {0}
<a href="#">GBRMM0003E</a>	Cannot migrate attribute {0}
<a href="#">GBRMM0004E</a>	Cannot migrate constructor {0}
<a href="#">GBRMM0005E</a>	Cannot migrate method {0}
<a href="#">GBRMM0006E</a>	Cannot migrate static reference "{0}": cannot find the corresponding attribute
<a href="#">GBRMM0007W</a>	Enum "{0}" must not contain a public constructor
<a href="#">GBRMM0008W</a>	Initial value "{0}" of type "{1}" is ignored as its type is not compatible with attribute "{2}"
<a href="#">GBRMO0001J</a>	Object model
<a href="#">GBRMO0002E</a>	Loop in inheritance tree
<a href="#">GBRMO0003E</a>	LinkageError {1} while trying to load the class {0}
<a href="#">GBRMO0004E</a>	ClassNotFoundException while trying to load the class {0}
<a href="#">GBRMO0005E</a>	Cannot find attribute {0} referenced in domain
<a href="#">GBRMO0006E</a>	Referenced attribute {0} must be static
<a href="#">GBRMO0007E</a>	Referenced attribute is of type {0}, whereas type {1} is expected
<a href="#">GBRMO0008E</a>	Actual value {0} is of type {1}, whereas type {2} is expected
<a href="#">GBRMO0009E</a>	Actual value {0} is not well formatted: reading it back produces {1}
<a href="#">GBRMO0010W</a>	Class {0} is deprecated. {1}
<a href="#">GBRMO0011W</a>	Member {0} is deprecated. {1}
<a href="#">GBRMO0012W</a>	Referenced type {0} is not defined
<a href="#">GBRMO0013E</a>	Type is not known
<a href="#">GBRMO0014E</a>	Error {1} when loading members of class {0}: {2}



<a href="#">GBRMO0015</a> E	Error {1} when loading members of class {0} : {2}
<a href="#">GBRMO0016</a> E	Error {1} when loading members of class {0} : {2}
<a href="#">GBRMO0017</a> E	There are two namespaces (probably a class and a package) with the same name {0}
<a href="#">GBRMO0018</a> E	Cannot merge class "{0}"
<a href="#">GBRMO0019</a> E	Cannot find referenced attribute ""{0}" from restricted attribute "{1}"
<a href="#">GBRMO0020</a> E	Cannot create package "{0}"
<a href="#">GBRMO0021</a> E	Member "{0}" has two parameters with the same name "{1}"
<a href="#">GBRMO0023</a> E	Enum class {0} cannot inherit from class {1} which has an incompatible domain
<a href="#">GBRMO0024</a> E	The domain of enum class {0} cannot contain the attribute {1} which is not in the domain of enum class {2}
<a href="#">GBRMO0025</a> E	The BOM element "{0}" has a legacy translation property "{1}". Translation properties are not supported anymore.

# Rule Designer Messages - Messages pertaining to the business object model

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRS B000 1E</a>	Failed to load XOM class: "{0}" for XOM path entry: "{1}"
<a href="#">GBRS B000 2E</a>	Error processing "{0}" for element "{1}".
<a href="#">GBRS B000 3E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 4E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 5E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 6E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 7E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 8E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B000 9E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 0E</a>	Unexpected exception in BOM plugins
<a href="#">GBRS B001 1E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 2E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 3E</a>	Unexpected exception in BOM plugins
<a href="#">GBRS B001 4E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 5E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS</a>	Unexpected exception in BOM plugins.

<a href="#">B001 6E</a>	
<a href="#">GBRS B001 7E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 8E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B001 9E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 0E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 1E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 2E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 3E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 4E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 5E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 6E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 7E</a>	Unexpected exception in BOM plugins.
<a href="#">GBRS B002 8E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B002 9E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 0E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 1E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 2E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 3E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 4E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 5E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B003 6E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS</a>	Unexpected exception in BOM wizards plugins

<a href="#">B0037E</a>	
<a href="#">GBRS B0038E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0039E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0040E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0041E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0042E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0043E</a>	Unexpected exception in BOM wizards plugins
<a href="#">GBRS B0044E</a>	NoClassDefFoundError: "{0}"
<a href="#">GBRS B0045E</a>	Could not load all the selected classes. These classes may be private or protected, or other classes that are required to load these classes may be missing. Please verify the classes can be loaded by the JDK using Class.forName(...).
<a href="#">GBRS B0046E</a>	Errors raised loading classes:
<a href="#">GBRS B0047I</a>	Warnings raised loading classes:
<a href="#">GBRS B0048E</a>	Impossible to read "{0}" : "{1}"
<a href="#">GBRS B0049E</a>	NoClassDefFoundError: "{0}"
<a href="#">GBRS B0050E</a>	The BOM entry file contains invalid information. Fix the problems and reopen the editor. See the log file for details.
<a href="#">GBRS B0052E</a>	Unexpected exception. See log file for details.
<a href="#">GBRS B0053E</a>	Failed to find type in temporary object model: "{0}"
<a href="#">GBRS B0055E</a>	Failed to create a renderer for a domain used in the BOM.
<a href="#">GBRS B0056E</a>	The domain on the element "{0}" contains an invalid entry: "{1}".

# Rule Designer Messages - Messages pertaining to debug

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSD0001E</a>	Exception while retrieving underlying stack frame
<a href="#">GBRSD0002E</a>	Exception while computing source support
<a href="#">GBRSD0003E</a>	Exception while invoking method : "{0}"
<a href="#">GBRSD0004E</a>	Exception while stepping out of rule stack frame
<a href="#">GBRSD0005E</a>	Exception while stepping out of rule stack frame
<a href="#">GBRSD0006E</a>	Exception while stepping over rule stack frame
<a href="#">GBRSD0007E</a>	Exception while stepping over rule stack frame
<a href="#">GBRSD0008E</a>	Exception while stepping into rule stack frame
<a href="#">GBRSD0009E</a>	Exception while stepping into rule stack frame
<a href="#">GBRSD0010E</a>	Exception while stepping into rule stack frame
<a href="#">GBRSD0011E</a>	Exception while stepping into rule stack frame
<a href="#">GBRSD0012E</a>	Exception while stepping into rule stack frame
<a href="#">GBRSD0013E</a>	Exception while launching debug target
<a href="#">GBRSD0014E</a>	Exception while removing breakpoint
<a href="#">GBRSD0015E</a>	Exception while removing breakpoint (bad location)
<a href="#">GBRSD0016E</a>	Exception while retrieving breakpoint markers
<a href="#">GBRSD0017E</a>	Exception while adding object breakpoint
<a href="#">GBRSD0018E</a>	Exception while computing source support
<a href="#">GBRSD0019E</a>	Exception while computing source support
<a href="#">GBRSD0020E</a>	Exception while initializing data connector tab
<a href="#">GBRSD0021E</a>	Exception while updating extractor from configuration
<a href="#">GBRSD0022E</a>	Exception while loading Java project

<a href="#">GBRSD0023E</a>	Exception while updating project from configuration
<a href="#">GBRSD0024E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0025E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0026E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0027E</a>	Exception while initializing name
<a href="#">GBRSD0028E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0029E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0030E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0031E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0032E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0033E</a>	Exception while loading Java project
<a href="#">GBRSD0034E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0035E</a>	Exception while retrieving classpath from configuration
<a href="#">GBRSD0036E</a>	Exception while retrieving classpath from configuration
<a href="#">GBRSD0037E</a>	Exception while displaying default classpath
<a href="#">GBRSD0038E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0039E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0040E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0041E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0042E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0043E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0044E</a>	Exception while initializing source lookup table
<a href="#">GBRSD0045E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0046E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0047E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0048E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0049E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0050E</a>	Exception while retrieving attribute from configuration
<a href="#">GBRSD0051E</a>	Exception while initializing image path

<a href="#">GBRSD005 2E</a>	Exception while creating image descriptor
<a href="#">GBRSD005 3E</a>	Exception while creating image descriptor
<a href="#">GBRSD005 4E</a>	Exception while creating image descriptor
<a href="#">GBRSD005 5E</a>	Exception while creating image descriptor
<a href="#">GBRSD005 6E</a>	Exception while retrieving model presentation
<a href="#">GBRSD005 7E</a>	Exception while saving configuration
<a href="#">GBRSD005 8E</a>	Exception while looking for source element
<a href="#">GBRSD005 9E</a>	Exception while showing source page
<a href="#">GBRSD006 0E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 1E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 2E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 3E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 4E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 5E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 6E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 7E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 8E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD006 9E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 0E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 1E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 2E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 3E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 4E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 5E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 6E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 7E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 8E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD007 9E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD008 0E</a>	Unexpected exception in launching plugins

<a href="#">GBRSD0081E</a>	Unexpected exception in launching plugins
<a href="#">GBRSD0082E</a>	VM not fully specified in launch configuration {0} - missing VM name. Reverting to default VM.
<a href="#">GBRSD0084E</a>	Invalid launchBuildModeSupport declaration from plugin {0}.
<a href="#">GBRSD0085E</a>	Failed to instantiate an IlrProjectLaunchBuildModeSupport for the {0} build mode. Was the declaring plugin unloaded?
<a href="#">GBRSD0086E</a>	No registered launchBuildModeSupport extension for build mode {0}.
<a href="#">GBRSD0087E</a>	An unexpected error occurred while retrieving the location of the jrules-engine.jar
<a href="#">GBRSD0088E</a>	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".
<a href="#">GBRSD0089E</a>	An unexpected error occurred while generating ruleset archive "{0}" with extractor "{1}".
<a href="#">GBRSD0090E</a>	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".



# Rule Designer Messages - Messages pertaining to rule execution

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSE0001E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0002E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0003E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0004E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0005E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0006E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0007E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0008E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0009E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0010E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0011E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0012E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0013E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0014E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0015E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0016E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0017E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0018E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0019E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0020E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0021E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0022E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0023E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0024E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0025E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0026E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0027E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0028E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0029E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0030E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0031E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0032E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0033E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0034E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0035E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0036E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0037E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0038E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0039E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0040E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0041E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0042E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0043E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0044E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0045E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0046E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0047E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0048E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0049E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0050E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0051E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0052E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0053E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0054E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0055E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0056E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0057E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0058E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0061E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0064E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0065E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0066E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0067E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0068E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0069E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0070E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0071E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0072E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0073E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0074E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0075E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0076E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0077E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0078E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0079E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0080E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0081E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0082E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0083E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0084E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0085E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0086E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0087E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0088E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0089E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0090E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0091E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0092E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0093E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0094E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0095E</a>	An unexpected error occurred while loading the RuleApp XML descriptor "{0}".
<a href="#">GBRSE0102E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0103E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0104E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0105E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0106E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0107E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0108E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0109E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0110E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0111E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0112E</a>	Unexpected exception while retrieving the engine type for ruleset archive "{0}".
<a href="#">GBRSE0117E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0118E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0119E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0120E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0121E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0122E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0123E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE 0124E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0125E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0126E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0127E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0128E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0129E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0130E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0131E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0132E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0134E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0135E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0136E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0137E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0138E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0139E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0140E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0145E</a>	Failed to instantiate a password encryption manager. Will fall back to a non-encryption policy.
<a href="#">GBRSE 0146E</a>	Failed to encrypt password. Returning the password not encrypted.
<a href="#">GBRSE 0147E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0148E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0149E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0150E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0153E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0154E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0155E</a>	The file which stored the login and password information could not be found. Please re-enter the login and password values for your Rule Execution Server Configurations.
<a href="#">GBRSE 0156E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0157E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0158E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0159E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0160E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0161E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0162E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0163E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0164E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0165E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0166E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0167E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0168E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0169E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0170E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0171E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0172E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0173E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0174E</a>	"{0}" : "{1}"
<a href="#">GBRSE0175E</a>	The ruleset archive cannot be found.
<a href="#">GBRSE0176E</a>	Not a valid ruleset archive.
<a href="#">GBRSE0177E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0178E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0179E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0180E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0181E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0182E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0183E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0184E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0185E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0186E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0187E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0188E</a>	Unexpected exception in Rule Execution Server UI plug-ins.



<a href="#">GBRSE0189E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0190E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0191E</a>	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
<a href="#">GBRSE0192E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0193E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0194E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0195E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0196E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0197E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0198E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0199E</a>	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
<a href="#">GBRSE0200E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0201E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0202E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0203E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0204E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0205E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0206E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0207E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0208E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0209E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0210E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0211E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0212E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0213E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0214E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0215E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE0216E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0217E</a>	Unexpected exception in Rule Execution Server UI plug-ins.

<a href="#">GBRSE 0218E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0219E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0220E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0221E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0222E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0223E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0224E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0225E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0226E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0227E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0228E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0229E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0230E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0231E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0232E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0233E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0234E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0235E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0236E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0237E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0240E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0241E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0242E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0243E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0244E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0245E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0262E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0263E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0264E</a>	Unexpected exception in DVS plugins



<a href="#">GBRSE 0265E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0266E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0267E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0268E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0269E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0270E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0271E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0272E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0273E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0274E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0275E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0276E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0277E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0278E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0279E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0280E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0281E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0282E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0283E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0284E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0285E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0286E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0287E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0288E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0289E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0290E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0291E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0292E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0293E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0294E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0295E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0296E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0297E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0298E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0299E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0300E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0301E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0302E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0303E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0304E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0305E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0306E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0307E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0308E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0309E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0310E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0311E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0312E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0313E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0314E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0315E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0316E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0317E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0318E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0319E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0320E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0321E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0322E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0323E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0324E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0325E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0326E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0327E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0328E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0329E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0330E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0331E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0332E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0333E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0334E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0335E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0336E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0337E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0338E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0339E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0340E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0341E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0342E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0343E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0344E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0345E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0346E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0347E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0348E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0349E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0350E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0351E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0352E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0353E</a>	Ruleset cannot be generated
<a href="#">GBRSE0354E</a>	Directory cannot be created: "{0}"
<a href="#">GBRSE0355E</a>	The ruleset archive cannot be parsed, see following errors.
<a href="#">GBRSE0358E</a>	The ruleset archive cannot be parsed, verify the XOM path
<a href="#">GBRSE0359E</a>	The referenced rule project seems to be closed.
<a href="#">GBRSE0360E</a>	The file cannot be written: it is read-only.
<a href="#">GBRSE0365E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0366E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0367E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0368E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0369E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0370E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0371E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0372E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0373E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0374E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0375E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0376E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0377E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0378E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0379E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0380E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0381E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0382E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0383E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0384E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0385E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0386E</a>	Unexpected exception in DVS UI plug-ins

<a href="#">GBRSE0387E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0388E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0389E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0390E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0391E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0392E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0393E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0394E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0395E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0396E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0397E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0398E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0399E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0400E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0401E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0402E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0403E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0404E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0405E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0406E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0407E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0408E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0409E</a>	An unexpected exception occurred while accessing the project {0}.
<a href="#">GBRSE0410E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0411E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0412E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0413E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0414E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0415E</a>	An unexpected error occurred while accessing the DVS project {0}.

<a href="#">GBRSE 0416E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0417E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0418E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0419E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0420E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0422E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0423E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0424E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0425E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0426E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0427E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0428E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0429E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0430E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0431E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0432E</a>	An exception occurred while checking decision operation "{0}" for compatibility with testing and simulation.
<a href="#">GBRSE 0434E</a>	Failed to run command
<a href="#">GBRSE 0435E</a>	Failed to run command
<a href="#">GBRSE 0436E</a>	An error occurs during the repackaging. See the log file for more information.
<a href="#">GBRSE 0437E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0438E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0439E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0440E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0441E</a>	Exception while generating deployment.xml file.
<a href="#">GBRSE 0442E</a>	Unexpected exception while reading launch configuration
<a href="#">GBRSE 0443E</a>	There is no entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0444E</a>	An unexpected error occurred while accessing flags for the IType element {0} in DVS project {1}.
<a href="#">GBRSE 0445E</a>	An unexpected error occurred while retrieving the hierarchy of IType element {0} in DVS project {1}.
<a href="#">GBRSE 0446E</a>	An unexpected error occurred while retrieving the IType element {0} in DVS project {1}.



<a href="#">GBRSE 0447E</a>	An unexpected error occurred while retrieving the KPI Result classes compatible with the Decision Center renderer {0} in DVS project {1}.
<a href="#">GBRSE 0448E</a>	Exception while making .zip file for XOM deployment.
<a href="#">GBRSE 0449E</a>	The generation of the DVS KPI implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0450E</a>	An unexpected error occurred during the generation of the DVS KPI implementation classes.
<a href="#">GBRSE 0451E</a>	The generation of the DVS KPI Result Aggregator implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0452E</a>	An unexpected error occurred during the generation of the DVS KPI Result Aggregator implementation classes.
<a href="#">GBRSE 0453E</a>	The generation of the DVS Scenario Provider implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0454E</a>	An unexpected error occurred during the generation of the Scenario Provider implementation classes.
<a href="#">GBRSE 0455E</a>	An unexpected error occurred while applying the changes brought about by renaming the DVS Format file "{0}".
<a href="#">GBRSE 0457E</a>	An unexpected error occurred while synchronizing the editor of the DVS Customization "{0}".
<a href="#">GBRSE 0458E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Format file "{0}".
<a href="#">GBRSE 0459E</a>	An unexpected error occurred while applying the changes brought about by the DVS Configuration "{0}".
<a href="#">GBRSE 0460E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Configuration "{0}".
<a href="#">GBRSE 0461E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the Rule project "{0}" to "{1}".
<a href="#">GBRSE 0462E</a>	An unexpected error occurred while checking the nature of the project "{0}".
<a href="#">GBRSE 0467E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE 0471E</a>	An unexpected exception occurred while checking rule project "{0}" for compatibility with testing and simulation.
<a href="#">GBRSE 0477E</a>	Errors when trying to detect the engine type of rulesets declared in RuleApp {0}.
<a href="#">GBRSE 0479E</a>	There is no deployment.xml file for rule project "{0}". To generate this file, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0480E</a>	There is no deployed XOM entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0481E</a>	The rule project that is passed to the validator for XOM deployment descriptor files is undefined (null value).
<a href="#">GBRSE 0482E</a>	An unexpected error occurred while setting up the test report data for DVS launch configuration "{0}".
<a href="#">GBRSE 0483E</a>	An unexpected error occurred upon a workspace refresh during the execution of DVS launch configuration "{0}".
<a href="#">GBRSE 0484E</a>	An unexpected error occurred while opening the DVS execution report "{0}".
<a href="#">GBRSE 0485E</a>	An unexpected error occurred while refreshing the workspace resource "{0}".
<a href="#">GBRSE 0486E</a>	An unexpected exception occurred while retrieving the signature of the ruleset archive that is located in "{0}". Please check the Error View for more information about the cause of this issue.
<a href="#">GBRSE 0487E</a>	An unexpected exception occurred while looking for decision operations in rule project "{0}".
<a href="#">GBRSE 0488E</a>	An exception occurred while checking rule project "{0}" for compatibility with testing and simulation.
<a href="#">GBRSE 0489E</a>	The check for compatibility with testing and simulation was interrupted for rule project "{0}".

<a href="#">GBRSE0490E</a>	The variable "{0}" was not found in rule package "{1}".
<a href="#">GBRSE0491E</a>	The variable "{0}" that is referenced by decision operation "{1}" was not found in the default package.
<a href="#">GBRSE0492E</a>	The variable set "{0}" that is referenced by variable "{1}" in decision operation "{2}" was not found.
<a href="#">GBRSE0493E</a>	An unexpected error occurred while retrieving the JRE upon generation of Java project "{0}".
<a href="#">GBRSE0494E</a>	An unexpected exception occurred while retrieving the Decision Service property for project "{0}".
<a href="#">GBRSE0495E</a>	An unexpected exception occurred while retrieving attribute "{0}" of type "java.lang.String" for launch configuration "{0}".
<a href="#">GBRSE0496E</a>	An unexpected exception occurred while retrieving attribute "{0}" of type "boolean" for launch configuration "{0}".
<a href="#">GBRSE0497E</a>	An unexpected error occurred while checking file location "{0}", where the Excel file with DVS output values is stored, for write access.
<a href="#">GBRSE0498E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0499E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0500E</a>	Unexpected exception in SDsExcelRunner.
<a href="#">GBRSE0501E</a>	An unexpected exception occurred when retrieving DVS input and output parameters values.
<a href="#">GBRSE0502E</a>	An unexpected exception occurred when generating the DVS output values Excel file.
<a href="#">GBRSE0503E</a>	Unexpected exception in SDsExcelRunner.
<a href="#">GBRSE0504E</a>	Unexpected exception in IlrExcelRunner.
<a href="#">GBRSE0505E</a>	Unexpected exception in IlrExcelRunner.
<a href="#">GBRSE0506E</a>	Unexpected exception in IlrDVSArchiveRunner.
<a href="#">GBRSE0507E</a>	An unexpected error occurred while exporting the RuleApp "{0}".
<a href="#">GBRSE0508E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE0509E</a>	An unexpected error occurred during XOM deployment.



# Rule Designer Messages - Messages pertaining to the model (00 - 99)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSM0001E</a>	Unexpected exception in model
<a href="#">GBRSM0002E</a>	Unexpected exception in model
<a href="#">GBRSM0003E</a>	Unexpected exception in model
<a href="#">GBRSM0004E</a>	Unexpected exception in model
<a href="#">GBRSM0005E</a>	Unexpected exception in model
<a href="#">GBRSM0006E</a>	Unexpected exception in model
<a href="#">GBRSM0007E</a>	Unexpected exception in model
<a href="#">GBRSM0008E</a>	Unexpected exception in model
<a href="#">GBRSM0009E</a>	Unexpected exception in model
<a href="#">GBRSM0010E</a>	Unexpected exception in model
<a href="#">GBRSM0011E</a>	Unexpected exception in model
<a href="#">GBRSM0012E</a>	Unexpected exception in model
<a href="#">GBRSM0013E</a>	Unexpected exception in model
<a href="#">GBRSM0014E</a>	Unexpected exception in model
<a href="#">GBRSM0015E</a>	Unexpected exception in model
<a href="#">GBRSM0016E</a>	Unexpected exception in model
<a href="#">GBRSM0017E</a>	Unexpected exception in model
<a href="#">GBRSM0018E</a>	Unexpected exception in model
<a href="#">GBRSM0019E</a>	Unexpected exception in model
<a href="#">GBRSM0020E</a>	Unexpected exception in model
<a href="#">GBRSM0021E</a>	Unexpected exception in model
<a href="#">GBRSM0022E</a>	Unexpected exception in model
<a href="#">GBRSM0023E</a>	Unexpected exception in model
<a href="#">GBRSM0024E</a>	Unexpected exception in model
<a href="#">GBRSM0025E</a>	Unexpected exception in model
<a href="#">GBRSM0026E</a>	Unexpected exception in model
<a href="#">GBRSM0027E</a>	Unexpected exception in model
<a href="#">GBRSM0028E</a>	Unexpected exception in model
<a href="#">GBRSM0029E</a>	Unexpected exception in model
<a href="#">GBRSM0030E</a>	Unexpected exception in model
<a href="#">GBRSM0031E</a>	Unexpected exception in model
<a href="#">GBRSM0032E</a>	Unexpected exception in model
<a href="#">GBRSM0033E</a>	Unexpected exception in model
<a href="#">GBRSM0034E</a>	Unexpected exception in model

<a href="#">GBRSM0035E</a>	Unexpected exception in model
<a href="#">GBRSM0036E</a>	Unexpected exception in model
<a href="#">GBRSM0037E</a>	Unexpected exception in model
<a href="#">GBRSM0038E</a>	Unexpected exception in model
<a href="#">GBRSM0039E</a>	Unexpected exception in model
<a href="#">GBRSM0040E</a>	Unexpected exception in model
<a href="#">GBRSM0041E</a>	Unexpected exception in model
<a href="#">GBRSM0042E</a>	Unexpected exception in model
<a href="#">GBRSM0043E</a>	Unexpected exception in model
<a href="#">GBRSM0044E</a>	Unexpected exception in model
<a href="#">GBRSM0045E</a>	Unexpected exception in model
<a href="#">GBRSM0046E</a>	Exception raised while compiling "{0}"
<a href="#">GBRSM0047E</a>	Unexpected exception in model
<a href="#">GBRSM0048E</a>	Unexpected exception in model
<a href="#">GBRSM0049E</a>	Unexpected exception in model
<a href="#">GBRSM0050E</a>	Unexpected exception in model
<a href="#">GBRSM0051E</a>	Unexpected exception in model
<a href="#">GBRSM0052E</a>	Unexpected exception in model
<a href="#">GBRSM0053E</a>	Unexpected exception in model
<a href="#">GBRSM0054E</a>	Unexpected exception in model
<a href="#">GBRSM0055E</a>	Unexpected exception in model
<a href="#">GBRSM0056E</a>	Unexpected exception in model
<a href="#">GBRSM0057E</a>	Unexpected exception in model
<a href="#">GBRSM0058E</a>	Unexpected exception in model
<a href="#">GBRSM0059E</a>	Unexpected exception in model
<a href="#">GBRSM0060E</a>	Unexpected exception in model
<a href="#">GBRSM0061E</a>	Unexpected exception in model
<a href="#">GBRSM0062E</a>	Unexpected exception in model
<a href="#">GBRSM0063E</a>	Unexpected exception in model
<a href="#">GBRSM0064E</a>	Unexpected exception in model
<a href="#">GBRSM0065E</a>	Unexpected exception in model
<a href="#">GBRSM0066E</a>	Unexpected exception in model
<a href="#">GBRSM0067E</a>	Unexpected exception in model
<a href="#">GBRSM0068E</a>	Unexpected exception in model
<a href="#">GBRSM0069E</a>	Unexpected exception in model
<a href="#">GBRSM0070E</a>	Unexpected exception in model
<a href="#">GBRSM0071E</a>	Unexpected exception in model
<a href="#">GBRSM0072E</a>	Unexpected exception in model
<a href="#">GBRSM0073E</a>	Unexpected exception in model
<a href="#">GBRSM0074E</a>	Unexpected exception in model
<a href="#">GBRSM0075E</a>	Unexpected exception in model
<a href="#">GBRSM0076E</a>	Unexpected exception in model
<a href="#">GBRSM0077E</a>	Unexpected exception in model
<a href="#">GBRSM0078E</a>	Unexpected exception in model
<a href="#">GBRSM0079E</a>	Unexpected exception in model
<a href="#">GBRSM0080E</a>	Unexpected exception in model
<a href="#">GBRSM0081E</a>	Unexpected exception in model

<a href="#">GBRSM0082E</a>	Unexpected exception in model
<a href="#">GBRSM0083E</a>	Unexpected exception in model
<a href="#">GBRSM0084E</a>	Unexpected exception in model
<a href="#">GBRSM0085E</a>	Unexpected exception in model
<a href="#">GBRSM0086E</a>	Unexpected exception in model
<a href="#">GBRSM0087E</a>	Unexpected exception in model
<a href="#">GBRSM0088E</a>	Unexpected exception in model
<a href="#">GBRSM0089E</a>	Unexpected exception in model
<a href="#">GBRSM0090E</a>	Unexpected exception in model
<a href="#">GBRSM0091E</a>	Unexpected exception in model
<a href="#">GBRSM0092E</a>	Unexpected exception in model
<a href="#">GBRSM0093E</a>	Unexpected exception in model
<a href="#">GBRSM0094E</a>	Unexpected exception in model
<a href="#">GBRSM0095E</a>	Unexpected exception in model
<a href="#">GBRSM0096E</a>	Unexpected exception in model
<a href="#">GBRSM0097E</a>	Unexpected exception in model
<a href="#">GBRSM0098E</a>	Unexpected exception in model
<a href="#">GBRSM0099E</a>	Unexpected exception in model

# Rule Designer Messages - Messages pertaining to the model (100-199)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSM0100E</a>	Unexpected exception in model
<a href="#">GBRSM0101E</a>	Unexpected exception in model
<a href="#">GBRSM0102E</a>	Unexpected exception in model
<a href="#">GBRSM0103E</a>	Unexpected exception in model
<a href="#">GBRSM0104E</a>	Unexpected exception in model
<a href="#">GBRSM0105E</a>	Unexpected exception in model
<a href="#">GBRSM0106E</a>	Unexpected exception in model
<a href="#">GBRSM0107E</a>	Unexpected exception in model
<a href="#">GBRSM0108E</a>	Unexpected exception in model
<a href="#">GBRSM0109E</a>	Unexpected exception in model
<a href="#">GBRSM0110E</a>	Unexpected exception in model
<a href="#">GBRSM0111E</a>	Unexpected exception in model
<a href="#">GBRSM0112E</a>	Unexpected exception in model
<a href="#">GBRSM0113E</a>	Unexpected exception in model
<a href="#">GBRSM0114E</a>	Unexpected exception in model
<a href="#">GBRSM0115E</a>	Unexpected exception in model
<a href="#">GBRSM0116E</a>	Unexpected exception in model
<a href="#">GBRSM0117E</a>	Unexpected exception in model
<a href="#">GBRSM0118E</a>	Unexpected exception in model
<a href="#">GBRSM0119E</a>	Unexpected exception in model
<a href="#">GBRSM0120E</a>	Unexpected exception in model

<a href="#">GBRSM0121E</a>	Unexpected exception in model
<a href="#">GBRSM0122E</a>	Failed to create error marker on .project for rule project "{0}"
<a href="#">GBRSM0124E</a>	Unexpected exception in model
<a href="#">GBRSM0125E</a>	Unexpected exception in model
<a href="#">GBRSM0126E</a>	Unexpected exception in model
<a href="#">GBRSM0127E</a>	Unexpected exception in model
<a href="#">GBRSM0128E</a>	Unexpected exception in model
<a href="#">GBRSM0129E</a>	Unexpected exception in model
<a href="#">GBRSM0130E</a>	Unexpected exception in model
<a href="#">GBRSM0131E</a>	Unexpected exception in model
<a href="#">GBRSM0132E</a>	Unexpected exception in model
<a href="#">GBRSM0133E</a>	Failed to initialize variable {0}.
<a href="#">GBRSM0134E</a>	Unexpected exception in model
<a href="#">GBRSM0135E</a>	Failed to save the build mode property of rule project {0} on disk.
<a href="#">GBRSM0136E</a>	Unexpected exception in model
<a href="#">GBRSM0137E</a>	Unexpected exception in model
<a href="#">GBRSM0138E</a>	Unexpected exception in model
<a href="#">GBRSM0139E</a>	Unexpected exception in model
<a href="#">GBRSM0140E</a>	Unexpected exception in model
<a href="#">GBRSM0141E</a>	Unexpected exception in model
<a href="#">GBRSM0142E</a>	Unexpected exception in model
<a href="#">GBRSM0143E</a>	Unexpected exception in model
<a href="#">GBRSM0144E</a>	Unexpected exception in model
<a href="#">GBRSM0145E</a>	Unexpected exception in model
<a href="#">GBRSM0146E</a>	Cannot create impact file: "{0}"
<a href="#">GBRSM0147E</a>	Error while registering impact for file: "{0}"
<a href="#">GBRSM0148E</a>	Error while registering impact for file: "{0}"
<a href="#">GBRSM0149E</a>	Error while retrieving impact for file: "{0}"
<a href="#">GBRSM0150E</a>	Error while retrieving impact for file: "{0}"

<a href="#">GBRSM0151E</a>	Error while resetting impact for file: "{0}"
<a href="#">GBRSM0152E</a>	Error while retrieving impact for file: "{0}"
<a href="#">GBRSM0153E</a>	An unexpected error occurred while retrieving the model elements impacted by {0} changes on rule project {1}.
<a href="#">GBRSM0154E</a>	An unexpected error occurred while checking whether a {0} change took place on rule project: {1}.
<a href="#">GBRSM0155E</a>	Error getting impacts for element: "{0}"
<a href="#">GBRSM0156E</a>	Exception raised while compiling "{0}"
<a href="#">GBRSM0157E</a>	Error while retrieving rule elements for group: {0}
<a href="#">GBRSM0158E</a>	Error creating refactoring change for "{0}"
<a href="#">GBRSM0159E</a>	Unexpected error while retrieving content of the folder "{0}".
<a href="#">GBRSM0160E</a>	Failed to move element: "{0}" to index: "{1}". Current folder elements: "{2}"
<a href="#">GBRSM0161E</a>	Failed to move element: "{0}" to index: "{1}". Current package elements: "{2}"
<a href="#">GBRSM0162E</a>	Error during dependency cycle scan
<a href="#">GBRSM0163E</a>	Failed to save project references: "{0}"
<a href="#">GBRSM0164E</a>	Failed to remove error marker on .project.
<a href="#">GBRSM0165E</a>	WARNING: Duplicate ID for extension point "{0}"."{1}"
<a href="#">GBRSM0166E</a>	Unexpected model structure : the BOM Entry "{0}" has a vocabulary element that is null or that is a proxy for the locale "{1}"
<a href="#">GBRSM0167E</a>	Impossible to read "{0}" : "{1}"
<a href="#">GBRSM0168E</a>	Rule Project without BOM Path
<a href="#">GBRSM0169E</a>	Project not associated with a rule project
<a href="#">GBRSM0170E</a>	Could not flush plugin preferences on disk after modification; platform may have more chance later.
<a href="#">GBRSM0171E</a>	Attempt made to register a Refactoring Participant which does not implement IlrRefactoringParticipant: "{0}"
<a href="#">GBRSM0172E</a>	Failed to load BOM from: "{0}"
<a href="#">GBRSM0173E</a>	Class "{0}" not found
<a href="#">GBRSM0174E</a>	Class "{0}" not found
<a href="#">GBRSM0175E</a>	Cannot read engine configuration file
<a href="#">GBRSM0176E</a>	No model folder type is associated with the element: {0}.
<a href="#">GBRSM0177E</a>	Failed to resolve rule project for URL: "{0}"
<a href="#">GBRSM0178E</a>	Failed to resolve rule project for URL: "{0}"
<a href="#">GBRSM0179E</a>	Failed to resolve XOM Path entry for URL: "{0}"



<a href="#">GBRSM0180E</a>	Failed to resolve XOM Path entry for URL: "{0}"
<a href="#">GBRSM0181E</a>	Cannot read engine configuration file
<a href="#">GBRSM0182E</a>	The engine.conf file "{0}" has not been found.
<a href="#">GBRSM0183E</a>	Null value for "{0}" attribute in extension "{1}". Extension ignored.
<a href="#">GBRSM0184E</a>	BOMDomainValueProvider must specify a value for classname or property file
<a href="#">GBRSM0185E</a>	Failed to instantiate BOMDomainValueProvider "{0}"
<a href="#">GBRSM0186E</a>	Registered BOMDomainValueProvider "{0}" must implement IlrBOMDomainValueProvider
<a href="#">GBRSM0187I</a>	Rule project reference cache cleared for project: "{0}"
<a href="#">GBRSM0188I</a>	Referenced rule projects for project: "{0}" are: "{1}"
<a href="#">GBRSM0189I</a>	Referencing projects for: "{0}" : "{1}"
<a href="#">GBRSM0190I</a>	Failed to load build options settings. Using default values: "{0}"
<a href="#">GBRSM0191I</a>	Full build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0192I</a>	Full build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0193I</a>	Full build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0194I</a>	Incremental build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0195I</a>	Incremental build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0196I</a>	Incremental build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0197I</a>	Auto build for Rule Project "{0}" was interrupted after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0198I</a>	Auto build for Rule Project "{0}" was canceled after: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.
<a href="#">GBRSM0199I</a>	Auto build for Rule Project "{0}" completed in: {1} ms. Number of built artifacts: {2}. Total artifacts: {3}.

# Rule Designer Messages - Messages pertaining to the model (200-299)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSM0200I</a>	Clean build for Rule Project "{0}" was interrupted after: {1} ms.
<a href="#">GBRSM0201I</a>	Clean build for Rule Project "{0}" was canceled after: {1} ms.
<a href="#">GBRSM0202I</a>	Clean build for Rule Project "{0}" completed in: {1} ms.
<a href="#">GBRSM0203I</a>	Loading file : "{0}"
<a href="#">GBRSM0204I</a>	Loading file : "{0}"
<a href="#">GBRSM0205I</a>	Unloading resource : "{0}"
<a href="#">GBRSM0206E</a>	Uncaught static analysis exception
<a href="#">GBRSM0208E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0209E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0210E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0211E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0212E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0213E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0214E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0215E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0216E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0217E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0218E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0219E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0220E</a>	Unexpected exception in shared plugin
<a href="#">GBRSM0221E</a>	Invalid namespace for package name conversion: {0} Using default package name instead.



<a href="#">GBRSM0222E</a>	Failed loading L&F:
<a href="#">GBRSM0226E</a>	An unexpected error occurred. See log file for details.
<a href="#">GBRSM0227E</a>	An error occurred while loading the XOM file {0}. See log file for details.
<a href="#">GBRSM0228E</a>	Could not load RulePackage "{0}". The associated .rulepackage file might be corrupted.
<a href="#">GBRSM0229E</a>	Failed to initialize classpath variable {0} from {1}, which is null.
<a href="#">GBRSM0230E</a>	An error occurred while loading the XML schema {0}. See log file for details.
<a href="#">GBRSM0231E</a>	Unexpected exception when trying to build a URL for the location {0}
<a href="#">GBRSM0232E</a>	Importing a XOM from a WSDL file was deprecated in JRules 6 and removed in JRules 7.
<a href="#">GBRSM0233E</a>	Unmanaged XOM type error for dynamic XOM entry: {0}.
<a href="#">GBRSM0234E</a>	Unable to find resource file: "{0}"
<a href="#">GBRSM0235E</a>	Failed to locate entry "{0}" in bundle {1}
<a href="#">GBRSM0236E</a>	Failed to convert URL for entry "{0}" in bundle {1} into a "file:" URL
<a href="#">GBRSM0237E</a>	A file was deleted before the classloader could open it or a file could not be read.
<a href="#">GBRSM0238E</a>	
<a href="#">GBRSM0239E</a>	An unexpected error occurred while saving the Ruleflow "{0}". Please consult the Error Log for more details.
<a href="#">GBRSM0240E</a>	An error occurred when saving the Ruleflow impacted by moving the Rule package "{0}" to "{1}".
<a href="#">GBRSM0241E</a>	Multiple errors occurred when saving the Ruleflows impacted by moving the Rule package "{0}" to "{1}".
<a href="#">GBRSM0242E</a>	An unexpected error occurred when moving the Rule package "{0}" to "{1}". Please consult the Error Log for more details.
<a href="#">GBRSM0243E</a>	An unexpected error occurred while initializing the class "{0}" for the "ilog.rules.studio.model.rulePackageMovedListeners" extension point.
<a href="#">GBRSM0244E</a>	Failed to read a local copy of the output file for resource "{0}".
<a href="#">GBRSM0245E</a>	Illegal "archiveKind" value "{0}" in declaration of a rulesetArchiveExporter in plugin {1}. Ignoring this configuration element.
<a href="#">GBRSM0246E</a>	Failed to convert project element "{0}" into IRL.
<a href="#">GBRSM0247E</a>	Encountered multiple main ruleflows while extracting a ruleset archive: "{0}", "{1}". Check the "main ruleflow" property of the ruleflows or the extractor used for this ruleset archive export.
<a href="#">GBRSM0248E</a>	Unexpected error while compiling ruleflow {0}.
<a href="#">GBRSM0249E</a>	Unexpected error: the decision engine migration did not produce a BuildContext but did not report any error either.
<a href="#">GBRSM0250E</a>	Unexpected error: expected error objects of type "{0}" but found a "{1}"
<a href="#">GBRSM0251E</a>	Code converter raised some errors for rule "{0}"
<a href="#">GBRSM0252E</a>	Unknown hierarchical property name "{0}"
<a href="#">GBRSM0253E</a>	Unknown element "{0}" in hierarchical property "{1}"

<a href="#">GBRSM 0254E</a>	Code converter raised some errors for function "{0}"
<a href="#">GBRSM 0255E</a>	Unsupported direction kind "{0}" for ruleset parameter "{1}"
<a href="#">GBRSM 0256E</a>	Found multiple rules with the same fully-qualified name while browsing project "{0}" : "{1}"
<a href="#">GBRSM 0257E</a>	Found multiple declarations of hierarchical property "{0}". one with root "{1}" another with root "{2}"
<a href="#">GBRSM 0258E</a>	Found multiple declarations of hierarchical rule property type for property "{0}": "{1}" and "{2}"
<a href="#">GBRSM 0259E</a>	Found multiple declarations of rule property type for property "{0}": "{1}" and "{2}"
<a href="#">GBRSM 0260E</a>	Found multiple variables with same name and package in project "{0}" : "{1}" in package "{2}"
<a href="#">GBRSM 0261E</a>	An unexpected error occurred while retrieving the local file for the URI "{0}".
<a href="#">GBRSM 0262E</a>	Unexpected ruletask algorithm: {0}
<a href="#">GBRSM 0263E</a>	Failed to get an XML serialization of ruleflow element.
<a href="#">GBRSM 0264E</a>	Failed to read the .xom file "{0}" for XSD XOMs in the output directory of the rule project.
<a href="#">GBRSM 0265E</a>	Error while saving B2X migration file.
<a href="#">GBRSM 0266E</a>	Project references cycle for the element "{0}"
<a href="#">GBRSM 0267E</a>	Failed to read rule compiled unit from "{0}". It was probably compiled by a previous version of the product. Please force a rebuild of your projects before retrying.
<a href="#">GBRSM 0268E</a>	No decision engine rule compiler found for rule type {0}.
<a href="#">GBRSM 0269E</a>	Extension {0} : Unknown type of rule artifact "{1}". Ignoring it.
<a href="#">GBRSM 0270E</a>	Extension {0} : Missing name of rule artifact type. Ignoring it.
<a href="#">GBRSM 0271E</a>	Extension {0} : Missing class name of rule compiler. Ignoring it.
<a href="#">GBRSM 0272E</a>	Extension {0} : Error when trying to instantiate rule compiler. Has this plugin been unregistered ? Discarding this extension.
<a href="#">GBRSM 0273E</a>	No decision engine rule compiler found for artifact type {0}
<a href="#">GBRSM 0274E</a>	Clean request failed for some projects.
<a href="#">GBRSM 0275E</a>	Unexpected exception in model
<a href="#">GBRSM 0276E</a>	An unexpected error occurred while generating ruleset archive "{0}" for rule project "{1}".
<a href="#">GBRSM 0277E</a>	Unknown ruleset extractor "{0}" on project "{1}"
<a href="#">GBRSM 0278E</a>	Found multiple ruleEnginePolicy plug-in extension. Expected at most one. Reverting to the default built-in one.
<a href="#">GBRSM 0279E</a>	Failed to instantiate ruleEnginePolicy extension. Skipping.
<a href="#">GBRSM 0280E</a>	Workspace error when creating error marker on project "{0}"
<a href="#">GBRSM 0281E</a>	Failed to create the temporary Ant Build file "{0}".
<a href="#">GBRSM 0282E</a>	Failed to execute the temporary Ant Build file "{0}".

<a href="#">GBRSM0283E</a>	Failed to delete temporary file "{0}".
<a href="#">GBRSM0284E</a>	Failed to parse the file "{0}" containing DVS classpath of required archives.
<a href="#">GBRSM0285E</a>	An unexpected error occurred while compiling technical rule "{0}". Compilation failed with no error message.

# Rule Designer Messages - Messages pertaining to tools

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRST0001E</a>	Completeness analysis HTML rendering failed
<a href="#">GBRST0002E</a>	Rule Analysis: Could not set job icon
<a href="#">GBRST0003E</a>	Rule Analysis Error
<a href="#">GBRST0004E</a>	Rule Analysis View: cannot set source folder for rule wizard using rule project : "{0}"
<a href="#">GBRST0005E</a>	Rule Analysis View: Could not render body section
<a href="#">GBRST0006E</a>	Data space coverage item rendering failed: no valid result data
<a href="#">GBRST0007E</a>	Consistency checking item rendering failed: no valid result data
<a href="#">GBRST0008E</a>	Error while rendering SA View results:
<a href="#">GBRST0009E</a>	Rule Analysis Error: add view action failed
<a href="#">GBRST0010E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0011E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0012E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0013E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0014E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0015E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0016E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0017E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0018E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0019E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0020E</a>	Unexpected exception in BOM update plugins
<a href="#">GBRST0021E</a>	Failed to get initial value for classpath variable {0} from the Java system property {1}.
<a href="#">GBRST0022E</a>	Failed to initialize classpath variable {0}

<a href="#">GBRST0024</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0025</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0026</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0027</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0028</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0029</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0030</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0031</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0032</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0033</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0034</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0035</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0036</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0037</a> <a href="#">E</a>	Unexpected exception in studio editor plugins
<a href="#">GBRST0038</a> <a href="#">E</a>	Exception in "{0}".getNewFile(): "{1}"
<a href="#">GBRST0039</a> <a href="#">E</a>	Exception in "{0}".getNewFile(): "{1}"
<a href="#">GBRST0040</a> <a href="#">E</a>	Unexpected exception in function editor plugins
<a href="#">GBRST0041</a> <a href="#">E</a>	Unexpected exception in function editor plugins
<a href="#">GBRST0042</a> <a href="#">E</a>	Unexpected exception in function editor plugins
<a href="#">GBRST0043</a> <a href="#">E</a>	Unexpected exception in function editor plugins
<a href="#">GBRST0044</a> <a href="#">E</a>	Unexpected exception in rule editor plugins
<a href="#">GBRST0045</a> <a href="#">E</a>	Unexpected exception in technical editor plugins
<a href="#">GBRST0046</a> <a href="#">E</a>	Unexpected exception in technical editor plugins
<a href="#">GBRST0047</a> <a href="#">E</a>	Unexpected exception in technical editor plugins
<a href="#">GBRST0048</a> <a href="#">E</a>	Unexpected exception in technical editor plugins
<a href="#">GBRST0049</a> <a href="#">E</a>	Unexpected exception in irl editor plugins
<a href="#">GBRST0050</a> <a href="#">E</a>	Unexpected exception in irl editor plugins
<a href="#">GBRST0051</a> <a href="#">E</a>	Unexpected exception in irl editor plugins
<a href="#">GBRST0052</a> <a href="#">E</a>	Unexpected exception in irl editor plugins

<a href="#">GBRST0053</a> E	Unexpected exception in irl editor plugins
<a href="#">GBRST0054</a> E	Unexpected exception in irl editor plugins
<a href="#">GBRST0055</a> E	Unexpected exception in irl editor plugins
<a href="#">GBRST0056</a> E	Unexpected exception in irl editor plugins
<a href="#">GBRST0103</a> E	Caught exception launching BIRT Web Viewer:
<a href="#">GBRST0104</a> E	Exception in "{0}".getNewFile(): "{1}"
<a href="#">GBRST0106</a> E	Background task searching for the "IRL keyword" topic in documentation failed.
<a href="#">GBRST0154</a> E	An exception has occurred during migration prerequisites check: {0}
<a href="#">GBRST0155</a> E	An exception has occurred during migration action creation: {0}
<a href="#">GBRST0159</a> E	An exception has occurred during action {0}: {1}
<a href="#">GBRST0165</a> E	An exception has occurred during migration perform action: {0}
<a href="#">GBRST0166</a> E	An exception has occurred while persisting project "{0}"
<a href="#">GBRST0167</a> E	The element is not an instance of IlrResourceElement
<a href="#">GBRST0168</a> E	The project is not an instance of IlrRuleProject
<a href="#">GBRST0169</a> E	An exception has occurred while persisting element "{0}"
<a href="#">GBRST0176</a> E	The project "{0}" referenced by "{1}" does not exist or is closed.
<a href="#">GBRST0177</a> E	The project "{0}" referenced by "{1}" is not in sync with the file system.
<a href="#">GBRST0178</a> E	The project "{0}" is in read-only mode.
<a href="#">GBRST0179</a> E	The project "{0}" has no source folder.
<a href="#">GBRST0180</a> E	The project "{0}" source folder does not exist.
<a href="#">GBRST0181</a> E	The project "{0}" source folder is in read-only mode.
<a href="#">GBRST0184</a> E	An exception "{0}" occurred during ruleset archive refresh.
<a href="#">GBRST0191</a> E	An exception occurred during RuleApp archive creation.
<a href="#">GBRST0193</a> E	An exception "{0}" occurred during RuleApp archive refresh.
<a href="#">GBRST0200</a> E	An exception has occurred during the Deployment Configuration checking.
<a href="#">GBRST0203</a> E	An exception has occurred during the build of a RuleApp.
<a href="#">GBRST0220</a> E	Exception while retrieving attribute from launch configuration
<a href="#">GBRST0221</a> E	A value must be specified
<a href="#">GBRST0222</a> E	Exception while retrieving mapped resources from launch configuration



<a href="#">GBRST0230E</a>	No server definition found in the imported file: "{0}"
<a href="#">GBRST0231E</a>	An unexpected error occurred while importing the server list from file "{0}".
<a href="#">GBRST0232E</a>	An unexpected error occurred while exporting the server list to the file "{0}".
<a href="#">GBRST0240E</a>	An unexpected error occurred while retrieving the credentials for target "{0}".
<a href="#">GBRST0241E</a>	An unexpected error occurred while retrieving the credentials for target "{0}".
<a href="#">GBRST0242E</a>	An unexpected error occurred while setting the credentials for target "{0}".
<a href="#">GBRST0243E</a>	An unexpected error occurred while checking the connections to the selected targets
<a href="#">GBRST0250E</a>	An unexpected error occurred during the inspection of the class path of Java project "{0}".
<a href="#">GBRST0251E</a>	An unexpected error occurred while adding the entry "{0}" to the exported JAR file "{1}".
<a href="#">GBRST0252E</a>	An unexpected error occurred while adding the output directory of the Java project "{0}" to its exported JAR file.
<a href="#">GBRST0253E</a>	An unexpected error occurred while adding the entry "{0}" to the root of its exported ZIP file.
<a href="#">GBRST0254E</a>	An unexpected error occurred while adding the entry "{0}" to the path "{1}" of its exported ZIP file.
<a href="#">GBRST0255E</a>	An unexpected error occurred when creating the exported ZIP file "{0}" for the class folder "{1}".
<a href="#">GBRST0256E</a>	An unexpected error occurred when creating an instance of class for the extension point "{0}".
<a href="#">GBRST0257E</a>	The JAR export operation was interrupted.
<a href="#">GBRST0258E</a>	An unexpected error occurred when running the JAR export operation.
<a href="#">GBRST0259E</a>	An unexpected error occurred while updating the XOM URIs of rule project "{0}".
<a href="#">GBRST0260E</a>	An unexpected error occurred while retrieving the first existing directory for the location "{0}".
<a href="#">GBRST0261E</a>	An unexpected error occurred while checking the location "{0}" for write access.
<a href="#">GBRST0262E</a>	An unexpected error occurred while checking the location "{0}": The value does not actually refer to an existing file.
<a href="#">GBRST0270E</a>	An unexpected error occurred while creating the launch configuration for the decision operation "{0}" ({1}).
<a href="#">GBRST0271E</a>	An unexpected error occurred while retrieving existing decision operation launch configurations.
<a href="#">GBRST0272E</a>	Classic rule project "{0}" cannot reference decision service rule project "{1}".
<a href="#">GBRST0273E</a>	Decision service rule project "{0}" cannot reference classic rule project "{1}".
<a href="#">GBRST0274E</a>	A standard rule project cannot reference the main rule project "{0}".
<a href="#">GBRST0275E</a>	An unexpected exception occurred while retrieving the message associated with the key "{0}" in bundle "{1}".
<a href="#">GBRST0276E</a>	An unexpected exception occurred while initializing the feature "{0}".
<a href="#">GBRST0277E</a>	An unexpected exception occurred while loading the server list from the preferences of the plug-in "{0}".
<a href="#">GBRST0278E</a>	An unexpected exception occurred while saving the server list to the preferences of the plug-in "{0}".

<a href="#">GBRST0279E</a>	An unexpected exception occurred while saving the server list to the preferences of the plug-in "{0}".
<a href="#">GBRST0280E</a>	An unexpected exception occurred while retrieving the message associated with the key "{0}" in bundle "{1}".
<a href="#">GBRST0281E</a>	An unexpected error occurred while validating rule project "{0}" that is referenced by decision operation "{1}".
<a href="#">GBRST0282E</a>	An unexpected exception occurred while generating a RuleApp client ({0})
<a href="#">GBRST0283E</a>	An unexpected error occurred while setting the encoding type "{0}" to an instance of the class "{1}".
<a href="#">GBRST0284E</a>	An unexpected error occurred while parsing the command line.
<a href="#">GBRST0285E</a>	The following Rule Execution Server definitions were removed or overwritten: "{0}".
<a href="#">GBRST0286E</a>	An exception has occurred while retrieving the Resource URIs defined in Rule project "{0}" for target "{1}": {2}.
<a href="#">GBRST0287E</a>	An unexpected error occurred while running the Rule Designer headless deployment tool.
<a href="#">GBRST0288E</a>	An unexpected exception occurred while loading server definition file specified by the "{0}" option.
<a href="#">GBRST0289E</a>	Initialization of plug-in extension failed: {0}.
<a href="#">GBRST0290E</a>	An unexpected exception occurred while retrieving the rulesets for RuleApp {0} version {1}: {2}
<a href="#">GBRST0291E</a>	An unexpected exception occurred while retrieving the versions for RuleApp {0}: {1}
<a href="#">GBRST0292E</a>	An exception occurred while connecting to a remote server "{0}"
<a href="#">GBRST0293E</a>	ARL view : failed to get local modification timestamp for element "{0}". Resource exists: "{1}"
<a href="#">GBRST0294E</a>	Exception raised while deserializing SemRule for element "{0}"
<a href="#">GBRST0295E</a>	Server list file contains multiple servers with the same name



# Rule Designer Messages - Messages pertaining to core user interface

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSU0001E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0002E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0003E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0004E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0005E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0006E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0007E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0008E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0009E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0010E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0011E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0012E</a>	Fail to load template "{0}"
<a href="#">GBRSU0013E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0014E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0015E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0016E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0017E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0018E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0019E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0020E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0021E</a>	Unexpected exception in studio ui plugins

<a href="#">GBRSU0022E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0023E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0024E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0025E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0026E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0027E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0028E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0029E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0030E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0031E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0032E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0033E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0034E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0035E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0036E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0037E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0038E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0039E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0040E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0041E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0042E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0043E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0044E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0045E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0046E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0047E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0048E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0049E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0050E</a>	Unexpected exception in studio ui plugins

<a href="#">GBRSU0051</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0052</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0053</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0054</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0055</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0056</a> <a href="#">E</a>	Error while creating parameters viewer
<a href="#">GBRSU0057</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0058</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0059</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0060</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0062</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0063</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0064</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0065</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0066</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0067</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0068</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0069</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0070</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0071</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0072</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0073</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0074</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0075</a> <a href="#">E</a>	Unexpected exception in studio ui plugins
<a href="#">GBRSU0076</a> <a href="#">E</a>	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0077</a> <a href="#">E</a>	The default classpath cannot be computed. Check the Rule Designer variable.
<a href="#">GBRSU0078</a> <a href="#">E</a>	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0079</a> <a href="#">E</a>	WARNING: Duplicate ID for extension point {0}.{1}.
<a href="#">GBRSU0082</a> <a href="#">E</a>	WARNING: Duplicate ID for extension point {0}.{1}.

<a href="#">GBRSU0083</a> E	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0084</a> E	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0085</a> E	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0086</a> E	Cannot read engine configuration file
<a href="#">GBRSU0087</a> E	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0088</a> E	WARNING: Duplicate id for extension-point :
<a href="#">GBRSU0089</a> E	Error committing refactoring for "{0}"
<a href="#">GBRSU0090</a> E	Couldn't flush plugin preferences on disk after modification. Platform may have more chance later.
<a href="#">GBRSU0091</a> E	Exception in "{0}".performCopy(): "{1}"
<a href="#">GBRSU0092</a> E	Unexpected exception in studio ui opensource plugins
<a href="#">GBRSU0093</a> E	Unexpected exception in studio ui opensource plugins
<a href="#">GBRSU0094</a> E	WARNING: Duplicate id for extension-point org.eclipse.jdt.ui.javaElementFilters
<a href="#">GBRSU0100</a> E	Exception in "{0}".getNewFile(): "{1}"
<a href="#">GBRSU0101</a> E	An internal error has occured. See log file for details.
<a href="#">GBRSU0102</a> E	A problem occured while opening "{0}".
<a href="#">GBRSU0103</a> E	An error has occurred during the execution of the command {0}. See log file for details.
<a href="#">GBRSU0104</a> E	An error occured while creating an URL for "{0}". See log file for details.
<a href="#">GBRSU0105</a> E	An error occured while attaching the source {0}. See log file for details.
<a href="#">GBRSU0106</a> E	An unexpected error occurred while persisting the remote help preferences.
<a href="#">GBRSU0107</a> E	An unexpected error occurred while accessing the remote help properties file.
<a href="#">GBRSU0108</a> E	The IBM_DS_HOME classpath variable is not defined. ODM Remote Help infocenter preferences will not be defined.
<a href="#">GBRSU0109</a> E	The infocenter.properties file was not found at the expected location {0}.
<a href="#">GBRSU0110</a> E	An exception occured while trying to access members of an IContainer {0}
<a href="#">GBRSU0111</a> E	An unexpected error occurred while updating UUIDs.
<a href="#">GBRSU0112</a> E	The UUIDs update operation was interrupted by the user.
<a href="#">GBRSU0113</a> E	An unexpected error occurred while loading the dynamic XOM path entry {0} or one of its dependencies.

# Rule Designer Messages - Messages pertaining to testing and simulation

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	An error has been detected. Contact your Administrator as this error typically occurs when trying to retrieve the signature of the ruleset. The XOM might not be correctly deployed to the SSP or to Rule Execution Server. Check the error trace for more detailed information.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	The simulation part definition contains an error. A scenario suite part is defined with a first index {0} and last index {1}. The first index must be a positive value and the last index must be greater than or equal to the first index. Contact your Administrator to solve the issue with the scenario provider implementation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">03</a> <a href="#">E</a>	The simulation part definition contains an error. A balanced list of scenario part is defined using a total number of scenario of {0} and a number of parts of {1}. The total number of scenarios must be greater than 0 and the number of parts must be less than or equal to the total number of scenarios. Contact your Administrator to solve the issue with the scenario provider implementation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">04</a> <a href="#">E</a>	SSP has stopped the execution of the scenario suite because no resources were available on the server. Contact your Administrator to add more resources to the work managers used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">05</a> <a href="#">E</a>	An error has been detected in one of the work managers used by the SSP. Contact your Administrator to fix the problem.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">06</a> <a href="#">E</a>	The execution of the scenario suite has been stopped by the SSP, because there were no resources available on the server. Contact your Administrator to add more resources to the thread pool used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">07</a> <a href="#">I</a>	The resource allocation policy manager has rejected the execution request.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">08</a> <a href="#">I</a>	The execution request has been accepted by the resource allocation policy manager. The resource allocation policy for this job is: Execution Priority {0}, Maximum number of threads to use for parallel execution {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a>	Decision Engine is not supported for the supplied scenario provider "{0}"

<a href="#">0009E</a>	
<a href="#">G BR TE 02 01 E</a>	Cannot convert string value {0} to an instance of {1}
<a href="#">G BR TE 02 02 E</a>	Cannot parse XML string {0}
<a href="#">G BR TE 02 03 E</a>	Cannot compute equality. There are too many possible matching values because too many expected objects have no value provided.
<a href="#">G BR TE 02 04 E</a>	Cannot cast object {0} to an instance of {1}
<a href="#">G BR TE 02 05 E</a>	Cannot parse the Gregorian calendar {0}. The <time> XML element is missing.
<a href="#">G BR TE 02 06 E</a>	Cannot compare complex object {0} directly. This object should be compared field by field.
<a href="#">G BR TE 02 07 E</a>	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
<a href="#">G BR TE 02 08 W</a>	No converter defined for type: "{0}". The toString() method is used to compare the expected value "{1}" with the observed value "{2}".
<a href="#">G BR TE 02 09 W</a>	The equals method is used to compare object {0}.
<a href="#">G BR TE 02 10 W</a>	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
<a href="#">G BR TE 02 11 W</a>	Ensure that the equals method is correctly implemented for this object.
<a href="#">G</a>	Cannot compare object {0} using BOM serialization because all the corresponding expected fields are empty in the

<a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">12</a> <a href="#">E</a>	Excel scenario file.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">13</a> <a href="#">W</a>	Object {0} is not compared because all the corresponding expected fields are empty in the Excel scenario file.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">14</a> <a href="#">E</a>	Cannot serialize object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">01</a> <a href="#">I</a>	GET request handled...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">02</a> <a href="#">I</a>	Logout requested, invalidating the session
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">03</a> <a href="#">I</a>	PING requested
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">04</a> <a href="#">I</a>	POST request handled, deserializing the request object...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">05</a> <a href="#">I</a>	ServiceCall deserialized from input stream, starting the service...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">06</a> <a href="#">I</a>	Service method called
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">07</a> <a href="#">I</a>	Writing result {0} to the response stream
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">08</a> <a href="#">I</a>	OK, result sent, response stream flushed
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">09</a>	SSP: No value defined in web.xml for parameter {0}: using default value {1}



<a href="#">W</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">10</a> <a href="#">I</a>	SSP: Using {0} rule sessions
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">11</a> <a href="#">E</a>	SSP: Unable to initialize Trace DAO
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">12</a> <a href="#">I</a>	SSP: Successfully initialized the trace DAO factory: {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">13</a> <a href="#">E</a>	SSP: Unable to instantiate job results store from class: {0}. The in-memory store is used instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">14</a> <a href="#">I</a>	SSP: Using the job results store: {0} instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">15</a> <a href="#">W</a>	Value of parameter {0} in web.xml is not a valid integer. Using default value {1}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">16</a> <a href="#">I</a>	SSP: Using {0} as the custom policy manager for allocation of resources
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">17</a> <a href="#">W</a>	SSP: An exception occurred while creating a new instance of the custom policy manager {0} for allocation of scenario suite resources. The default allocation policy manager is used instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">18</a> <a href="#">I</a>	SSP: The default policy manager for resource allocation is used with a setting of {0} for the maximum number of threads per parallel simulation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">19</a> <a href="#">I</a>	SSP pool size: {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">20</a> <a href="#">I</a>	SSP: High-priority work manager name : {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a>	SSP: Normal-priority work manager name : {0}



<a href="#">06 21 J</a>	
<a href="#">G BR TE 06 22 J</a>	SSP: Low-priority work manager name : {0}
<a href="#">G BR TE 06 23 E</a>	SSP: Cannot retrieve the high-priority work manager {0} in the JNDI directory
<a href="#">G BR TE 06 24 E</a>	SSP: Cannot retrieve the normal-priority work manager {0} in the JNDI directory
<a href="#">G BR TE 06 25 E</a>	SSP: Cannot retrieve the low-priority work manager {0} in the JNDI directory
<a href="#">G BR TG 00 01 E</a>	Cannot find sheet {0} in Excel scenario file.
<a href="#">G BR TG 00 02 E</a>	Index out of bounds: scenario {0} requested, while valid index are from {1} to {2} for the scenario file.
<a href="#">G BR TG 00 03 E</a>	Class {0}, referenced in the headers of sheet {1}, is not defined in the business object model.
<a href="#">G BR TG 00 04 E</a>	Missing property {0} that defines the name of the sheet for input objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G BR TG 00 05 E</a>	Object "{0}" referenced in cell {1} of sheet {2} cannot be found in sheet {3}.
<a href="#">G BR TG 00 06 E</a>	Missing property {0} that defines the name of the sheet for scenarios. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G BR TG 00 07 E</a>	Missing property {0} that defines the name of the sheet for output objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G</a>	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results

<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">08</a> <a href="#">E</a>	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">09</a> <a href="#">E</a>	Incorrect direction ({0}) defined for parameter {1} in property {2} of the Excel sheet {3}. The list of correct directions is: {4}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">10</a> <a href="#">E</a>	The Excel scenario provider does not support rulesets that define no input parameters.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">11</a> <a href="#">E</a>	Missing property {0} that defines the locale of the scenario file. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">12</a> <a href="#">E</a>	{0} is not a valid name for the scenario description column: there is an input parameter defined with the same name. Please choose a different name for the scenario description column.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">13</a> <a href="#">E</a>	Header rows are missing in the sheet {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">14</a> <a href="#">E</a>	Invalid value {0} for property {1} in the Excel sheet {2} to define the type of parameter {3}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">15</a> <a href="#">E</a>	Invalid value {0} for property {1} in the Custom tab of the Excel file properties dialog to define the ruleset output (direction OUT) parameters. Valid values are comma separated lists of PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION items.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">16</a> <a href="#">E</a>	Invalid value {0} in the headers of column {1} in the Excel sheet {2}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">17</a> <a href="#">E</a>	Mandatory value missing in cell {1} of sheet {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">18</a> <a href="#">E</a>	The BOM of the ruleset to test is not suitable for generating an Excel Scenario File Template. Please check additional messages.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">19</a>	Ruleset input parameter {0}, which is an array, is not supported in flat mode.

<a href="#">E</a>	
<a href="#">G BR TG 00 20 E</a>	The data for your scenarios is not suitable for generating a flat Excel Scenario File Template. However, you may use it to generate a Tabbed Excel Scenario File Template. Please talk to your developer for assistance if this format is not currently available.
<a href="#">G BR TG 00 21 E</a>	The business object model of the ruleset to test is not suitable for generating a flat Excel Scenario File Template. However, you can use it to generate a Tabbed Excel Scenario File Template. Please check additional messages.
<a href="#">G BR TG 00 22 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}.
<a href="#">G BR TG 00 23 E</a>	Unable to create XML reader.
<a href="#">G BR TG 00 24 E</a>	The provided data must be a String representation of the XML when using BOM model type.
<a href="#">G BR TG 00 25 E</a>	The provided data must be a String representation of the XML when using XML model type.
<a href="#">G BR TG 00 26 E</a>	Unable to translate XML into BOM XML.
<a href="#">G BR TG 00 27 E</a>	Unable to find parameter in the signature.
<a href="#">G BR TG 00 28 E</a>	Ruleset Path not found.
<a href="#">G BR TG 00 29 E</a>	Unexpected exception during the BOM serialization.
<a href="#">G BR TG 00 30 E</a>	Unable to read XML.
<a href="#">G BR TG</a>	BOM type not found. Create a special converter for this type: "{0}".

<a href="#">0031E</a>	
<a href="#">G BR TG 0032E</a>	Unknown property: "{0}".
<a href="#">G BR TG 0033E</a>	Unable to find conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0034E</a>	SecurityException on the constructor for conversion class for type "{0}": "{1}"
<a href="#">G BR TG 0035E</a>	No default public constructor for conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0036E</a>	Exception during the construction of conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0037E</a>	The conversion class for type "{0}" does not implement IlrBOM2DVSCConverter interface: "{1}"
<a href="#">G BR TG 0038I</a>	Additional converter loaded for type: "{0}".
<a href="#">G BR TG 0039E</a>	Unable to read inline BOM type "{0}".
<a href="#">G BR TG 0040E</a>	Missing default constructor for type "{0}".
<a href="#">G BR TG 0041E</a>	Unsupported BOM type "{0}".
<a href="#">G BR TG 0042E</a>	Field "{0}" in class "{1}" not found.

<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">43</a> <a href="#">E</a>	Ruleset input parameter {0}, which is a collection, is not supported in flat mode.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">44</a> <a href="#">E</a>	It is impossible to create scenario data because of cross references between the following objects defined in the scenario: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">45</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a byte (integer between -128 and 127) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">46</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a short (integer between -32,768 and 32,767) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">47</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: an integer (between -2,147,483,648 and 2,147,483,647) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">48</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a long (integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">49</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a single character was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">50</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a boolean value (TRUE or FALSE) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">51</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a double (decimal) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">52</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a float (decimal) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">53</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a date was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a>	Too many domains are defined for the input data, they cannot all be stored in the scenario suite.

<a href="#">54 E</a>	
<a href="#">G BR TG 00 55 E</a>	Only domains with less than {0} entries can be stored in the Excel file.
<a href="#">G BR TG 00 56 W</a>	BOM type not found. Create a special converter for this type: "{0}".
<a href="#">G BR TG 00 57 W</a>	Unsupported business object model type "{0}".
<a href="#">G BR TG 00 58 E</a>	Cannot read the Excel file: Try to open the file in Microsoft Excel. If you can open the file in Microsoft Excel but you still receive this error, please contact your Administrator.
<a href="#">G BR TG 00 59 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigDecimal (decimal) was expected.
<a href="#">G BR TG 00 60 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigInteger (integer) was expected.
<a href="#">G BR TG 00 61 E</a>	The default Excel format is not compatible with the ruleset parameter "{1}" of type {2}: it cannot be instantiated because of a cross-reference issue that involves an attribute of type {0}.
<a href="#">G BR TG 00 62 E</a>	An unexpected error occurred while checking whether the Excel Flat format is compatible with the ruleset parameter "{0}" of type {1}.
<a href="#">G BR TG 00 63 E</a>	An unexpected error occurred while invoking method "{0}" on instance "{1}".
<a href="#">G BR TG 02 01 E</a>	Exception {0} raised during the generation of a test rule: {1}.
<a href="#">G BR TG 02 02 W</a>	It was not possible to find a navigation phrase in the vocabulary for member {0} in BOM class {1}: a default verbalization is used for the Excel Scenario File.
<a href="#">G</a>	You have reached the maximum number of columns allowed in an Excel sheet.

<a href="#">BR</a> <a href="#">TR</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TR</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	Exception thrown during the business object model serialization process of the execution trace: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	{0}: the construction of {1} instances is not supported by the Excel Scenario Provider for testing and simulation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	{0}: cannot find the type {1} in the business object model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">03</a> <a href="#">E</a>	{0}: cannot find a testing and simulation constructor for the business object model class. If there are no constructors, create one. If there are several constructors in the business object model class, you must specify a constructor by selecting the "Testing and simulation constructor" option in the business object model editor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">04</a> <a href="#">E</a>	{0}: It is not possible to create instances of the business object model class. There is a cross-reference problem with the testing and simulation constructor, because at least one argument references the business object model class directly or indirectly. Check the arguments of the testing and simulation constructor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">05</a> <a href="#">E</a>	{0}: there are no constructor arguments or attributes that enable the creation of instances of the business object model class. Select a testing and simulation constructor with arguments, or create attributes that are nonstatic and writable.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">06</a> <a href="#">W</a>	{0}: the arguments of the constructor have generic names (arg1, arg2, etc.). Rename the arguments by reusing the names of the corresponding attributes or with your own meaningful names.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">07</a> <a href="#">E</a>	{0}: the Excel Format does not support multidimensional arrays.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">08</a> <a href="#">E</a>	{0}: cannot define the contents of the collection. Add a domain to the collection.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">09</a> <a href="#">W</a>	{0}: cannot use the default Excel format because of a cross-reference problem with an attribute. You can use the Excel Tabbed Format.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">10</a>	{0}: input parameters that are collections are not supported because the domain of the collection cannot be defined. Replace the input parameter with JavaBeans that hold the collection in a single attribute, and define the collection domain of this attribute.

<a href="#">E</a>	
<a href="#">G BR TV 00 11 E</a>	There are no input parameters. Add at least one ruleset parameter of direction IN or IN_OUT.
<a href="#">G BR TV 00 12 W</a>	Generic collections as output ruleset parameters are not supported.
<a href="#">G BR TV 00 13 W</a>	Generic collections as attributes of a BOM class are not supported.
<a href="#">G BR TV 00 14 W</a>	No supported child attribute found for this ruleset parameter.
<a href="#">G BR TV 00 15 W</a>	No supported child attribute found for this attribute.
<a href="#">G BR TV 00 16 W</a>	Multidimensional arrays as ruleset parameters are not supported.
<a href="#">G BR TV 00 17 W</a>	Multidimensionals arrays ruleset parameter attributes are not supported.
<a href="#">G BR TV 00 18 W</a>	Ruleset parameters of type "{0}" are not supported.
<a href="#">G BR TV 00 19 W</a>	Ruleset parameter attributes of type "{0}" are not supported.
<a href="#">G BR TV 00 21 E</a>	{0}: argument "{1}" of the constructor does not correspond to any attribute in the {2} class or its superclasses. Rename the arguments by reusing the names of the corresponding attributes or create virtual attributes that correspond to the arguments in the business object model.
<a href="#">G BR TV 00 22 W</a>	{0}: is not ignored because it is not optional.
<a href="#">G BR TV</a>	Engine type "{0}" is not supported.



<a href="#">0023E</a>	
<a href="#">G BR TV 00 24 E</a>	{0}: cannot create an instance for BOM class {1} because it is an abstract class. Use a concrete class instead of an abstract class.
<a href="#">G BR TX 00 01 E</a>	Column index is out of bounds. The allowable column range is [0..{0}].
<a href="#">G BR TX 00 02 E</a>	Row index is out of bounds. The allowable row range is [0..{0}].
<a href="#">G BR TX 00 03 E</a>	Could not create a Sheet in a readable Excel file.
<a href="#">G BR TX 00 04 E</a>	Could not retrieve the drawing patriarch for the Excel Sheet.
<a href="#">G BR TX 00 05 E</a>	Unable to deserialize this BOM type descriptor: {0}.
<a href="#">G BR TX 00 05 W</a>	Cell at [{0},{1}] is unexpectedly out of range.
<a href="#">G BR TX 00 06 E</a>	Unable to create a new XOM array instance for this BOM type: {0}.
<a href="#">G BR TX 00 06 W</a>	Cell at [{0},{1}] is unexpectedly within range.
<a href="#">G BR TX 00 07 E</a>	Unable to retrieve the XOM class mapping for this BOM type: {0}.
<a href="#">G BR TX 00 08 E</a>	Unable to create the object instance for this BOM type: {0}.
<a href="#">G</a>	Unable to cast the value at index {0} in the Input Record into {1}.

<a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">09</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">10</a> <a href="#">E</a>	The Input Record does not contain a value at index: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">11</a> <a href="#">E</a>	BOM support has not been enabled or the BOM is missing for the ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">12</a> <a href="#">E</a>	Unable to retrieve the type of the array elements for this BOM type: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">13</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is an array.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">14</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is a collection.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">15</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is a collection with a single factory parameter.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">16</a> <a href="#">E</a>	Unable to retrieve the BOM class that declares the field {0} of the BOM type {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">17</a> <a href="#">E</a>	Unable to retrieve the ruleset archive from Rule Execution Server.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">18</a> <a href="#">E</a>	Unable to parse the XML descriptor of the ruleset signature.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">19</a> <a href="#">E</a>	Unable to extract the XML descriptor of the ruleset signature from the ruleset archive.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">20</a>	Unable to serialize the ruleset parameter {0}.

<a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">21</a> <a href="#">E</a>	Unable to extract the BOM from the ruleset archive.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">22</a> <a href="#">E</a>	The BOM type for the ruleset parameter {0} is missing.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">23</a> <a href="#">E</a>	Unable to retrieve the BusinessDataIEService instance for ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">24</a> <a href="#">E</a>	Unable to convert BOM instance into XOM instance.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">25</a> <a href="#">E</a>	The BOM is missing for ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">26</a> <a href="#">E</a>	Unable to transform the input map into a compliant map.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">01</a> <a href="#">E</a>	Cannot create a ruleset parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE];PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">02</a> <a href="#">E</a>	Cannot extract the ruleset parameter name from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE];PARAMETER_DIRECTION with a string that is not empty after it has been trimmed.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">03</a> <a href="#">E</a>	Cannot extract the ruleset parameter type from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE];PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">04</a> <a href="#">E</a>	Cannot extract the ruleset parameter direction from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE];PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">05</a> <a href="#">E</a>	Cannot create an object factory parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE];PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a>	Cannot extract the name of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE];PARAMETER_IS_OPTIONAL with a string that is not empty after it has been

<a href="#">TX</a> <a href="#">02</a> <a href="#">06</a> <a href="#">E</a>	trimmed.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">07</a> <a href="#">E</a>	Cannot extract the type of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">08</a> <a href="#">E</a>	Cannot extract the direction of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">09</a> <a href="#">E</a>	Cannot find BOM file <{0}> in the class path.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">10</a> <a href="#">E</a>	The created object instance for class {0} must not be null.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">11</a> <a href="#">E</a>	Value "{0}" is not valid for class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">12</a> <a href="#">E</a>	Mandatory parameter "{0}" is missing for building an instance of {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">13</a> <a href="#">E</a>	There is no attribute "{0}" in the BOM class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">14</a> <a href="#">E</a>	Maps are not supported.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">15</a> <a href="#">E</a>	Value of mandatory parameter {0} for class {1} must be a string.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">16</a> <a href="#">E</a>	The object factory signature for class {0} already contains a parameter that is named {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">17</a> <a href="#">E</a>	Class {0} is not considered a simple type.

<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">18</a> <a href="#">E</a>	The node kind for class {0} must not be null.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">19</a> <a href="#">E</a>	Cannot convert date {0} to a string format for class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">20</a> <a href="#">E</a>	The object factory signature for class {0} must contain only one parameter.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">21</a> <a href="#">E</a>	Cannot convert an instance of BOM class {0} into a XOM instance.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">22</a> <a href="#">E</a>	Cannot create a node instance for the XMLGregorianCalendar class.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">23</a> <a href="#">E</a>	The argument "{0}" of the constructor does not correspond to any attribute in the {1} class or its super classes. Rename the arguments by reusing the names of the corresponding attributes, or create virtual attributes corresponding to the arguments in the Business Object Model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">24</a> <a href="#">E</a>	Cannot find class {0} in the business object model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">25</a> <a href="#">E</a>	Cannot find the default constructor for class {0} in the business object model. If there are no constructors, create one. If there are several constructors in the class, you must specify a constructor by selecting the "DVS constructor" option in the business object model editor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">26</a> <a href="#">E</a>	Cannot get metadata for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">27</a> <a href="#">E</a>	Cannot get array type for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">28</a> <a href="#">E</a>	Cannot get object value for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a>	Cannot create a node instance for the Calendar class using value {0}.

<a href="#">02</a> <a href="#">29</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">30</a> <a href="#">E</a>	{0} does not represent a valid Calendar value.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">31</a> <a href="#">E</a>	Cannot create an instance for BOM class {0} because it is an abstract class. Use a concrete class instead of an abstract class.

## Decision Center reference

Refer to these topics for public API and messages.

### [Decision Center REST API](#)

You can use the Decision Center REST API to build, test, and deploy decision services.

### [Decision Center modeling language reference](#)

This section lists the business rule language elements and syntax that are available for decision modeling.

### [Messages](#)

Each listed message provides a detailed description about the error message and some steps to try and resolve the error.

# Decision Center REST API

You can use the Decision Center REST API to build, test, and deploy decision services.

## [Overview](#)

Decision Center exposes a REST API that you can use to build, test, and deploy decision services. With this REST API, you can easily set up and enforce a continuous deployment process by using the programming language of your choice.

## [REST API](#)

Explore, build, test and deploy decision services stored in Decision Center.

**Parent topic:** [Decision Center reference](#)



## Overview

Decision Center exposes a REST API that you can use to build, test, and deploy decision services. With this REST API, you can easily set up and enforce a continuous deployment process by using the programming language of your choice.

Any user with the relevant permissions can use the end points provided by the REST API to do the following actions:

- Retrieve the decision services of your repository, their branches, deployment configurations, and test suites.
- Retrieve the list of servers available.
- Build, download, or deploy a RuleApp for a deployment configuration.
- Run a test suite.
- Import and export decision services.

For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api/v1/decisionservices --user username@company_name.com:password
```

This command returns a JSON object such as:

```
{
 "elements": [
 {
 "id": "77072920-2ec3-11db-bb3d-a34db576b043",
 "internalId": "brm.RuleProject:1:1",
 "name": "miniloan-rules",
 "buildMode": "DecisionEngine"
 },
 {
 "id": "4d9b5dc7-8a7e-404d-a05d-59023707c7d9",
 "internalId": "brm.RuleProject:2:2",
 "name": "AutoQuote",
 "buildMode": "DecisionEngine"
 }
],
 "totalCount": 2,
 "number": 0,
 "size": 2
}
```


## Accessing the REST API tool

You can access the Swagger user interface at `https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api`. This interface exposes a view of the available endpoints, their documentation, and a **Try it out** button to test each endpoint.

## Authentication

All endpoints, except `/about`, require authentication. You can authenticate by using basic authentication, with your Decision Center user name and password. For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-api/v1/decisionservices --user username@company_name.com:password
```

**Note:** In the Swagger user interface, you might see a  warning icon in some of the REST API methods. When you click this warning, a window opens asking for your login credentials. If you are authenticated to Operational Decision Manager on Cloud portal, you do not need to authenticate again, and you can disregard this warning.

As an alternative, you can use service credentials to authenticate to the application. For more information, see [Service credentials for client applications](#).

**Attention:** If you change your Operational Decision Manager on Cloud user role, or create a new set of service credentials, the Decision Center REST API is temporarily unavailable, and becomes available again after a maximum of 5 minutes.

## Errors

Conventional HTTP response codes are used to indicate the success or failure of an API request. When an error occurs, the body of the HTTP response always contains a JSON error object with an error code, a reason, and a reference. For example:

```
{
 "error": "IlrConnectException",
 "reason": "Could not look up datasource named 'mydatasource'",
 "status": "BAD_REQUEST",
 "details": [
 "Could not look up datasource named 'mydatasource'",
 "Name [mydatasource] is not bound in this Context. Unable to find [mydatasource].",
]
}
```

## Filtering and pagination

Explore methods that return collections of objects, such as /decisioncenter-api/v1/decisionservices, can be paginated. By default, all elements of the collection are returned, but you can specify a page size and a page number to retrieve only a subset of the collection. For example:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-
api/v1/decisionservices?page=2&size=20 --user
username@company_name.com:password
```

You can also filter returned objects by their property, by using the q parameter:

```
curl https://<vhostname>.bpm.ibmcloud.com/odm/dev/decisioncenter-
api/v1/decisionservices?q=name:AutoQuote --user
username@company_name.com:password
```

**Parent topic:** [Decision Center REST API](#)

## Decision Center modeling language reference

This section lists the business rule language elements and syntax that are available for decision modeling.

You use constructs, operators, and literals to write rules. You use different language constructs depending on whether you are writing the definitions, conditions, or actions part of a rule. You use punctuation to structure rules, avoid ambiguity, and provide clarity.

### **Definition part**

The definition part starts with the `definitions` keyword and introduces local variables that you can use in the rest of the rule.

### **Condition part**

The condition part of the rule starts with the `if` keyword.

### **Action part**

The action part describes the actions that the rule completes when the conditions are met. This part might start with the `then` keyword and can contain an `else` construct.

### **Operators**

You use operators in rule statements to perform arithmetic operations, associate or negate conditions, and compare expressions.

### **Literals**

The business rule language supports number, string, date, and time data types.

### **Punctuation in rules**

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules, mandatory punctuation is usually predicted in the completion menu.

**Parent topic:** [Decision Center reference](#)

## Definition part

The definition part starts with the `definitions` keyword and introduces local variables that you can use in the rest of the rule.

### definitions

This construct introduces the part of the rule where you define variables.

### from (object)

This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.

### in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

### (attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

### set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

### where (test)

This construct matches objects against tests.

**Parent topic:** [Decision Center modeling language reference](#)

## definitions

This construct introduces the part of the rule where you define variables.

### Purpose

You use this construct to define local variables that you can use elsewhere in the same rule. Use local variables to produce more concise and readable rules.

### Syntax

```
definitions
 set <variable> to <definition> in <list> | from <object> [where <test>*]
;*
```

### Description

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules. You define local variables using the [set \(variable\) to \(definition\)](#) construct.

### Example

The following example declares the local variable 'preferred customer' in the definitions part of the rule and uses it in the if and then parts of the rule.

```
definitions
 set 'preferred customer' to a customer from 'the customer';
if
 the age of 'preferred customer' is less than 18
then
 print "apply a 10% discount to the shopping cart of this customer";
```

**Parent topic:** [Definition part](#)

## from (object)

This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.

### Purpose

You use this construct to expand the set of data that can be used by a rule.

### Syntax

```
from <object>
```

### Description

The `from` construct is used to retrieve one distinct object from another one, and creates a variable that you can use in the rule to refer to this data. You cannot use it to retrieve a collection of objects. To access a collection of objects, use the [in \(list\)](#) construct.

### Example

The following rule declares a variable based on the relationship between a customer and the customer's preferred item.

```
definitions
 set 'preferred CD' to a CD
 from the preferred item of the customer;
if
 the price of 'preferred CD' is more than 25
then...
```

Here the decision node does not have access to "CD" objects from its input data. In this case, to write a rule that uses a CD object, you must first declare a variable of type CD that pulls in data from the preferred item object of the customer. This is possible only if the customer object (and therefore the customer's preferred item) is already known to the decision node.

**Parent topic:** [Definition part](#)

## in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

### Purpose

You use this construct in the definition or condition part of a rule to access objects from collections.

### Syntax

```
in <list>
```

### Description

The `in` construct is used to retrieve data from a collection of objects, and creates a variable that you can use in the rule to refer to this data. The `<list>` parameter of the `in` construct must be an expression that denotes a collection.

The `in` construct can be used in the definitions part of a rule and in count conditions specified in the `if` part of a rule.

### Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
 set 'item' to a CD
 in the items of the shopping cart of the customer ;
if
 there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
 there is an item
 in the items of the shopping cart of the customer ;
then...
```

**Parent topic:** [Definition part](#)

**Parent topic:** [Condition part](#)

## (attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

### Purpose

You use this construct in any part of the rule to access a list of attribute values.

### Syntax

```
<attribute> of <list>
```

### Description

You use this construct with other constructs that work with lists. For example, you can use it with the `in <list>` construct in a condition, or with the `set` construct in an action. The list can be an expression or an embedded list.

### Example

The following action sets the value of the variable to the list of the names of the banks of all the customers.

```
set decision to the bank accounts of 'the customers';
```

The following definition creates a variable that represents the list of all the cities from the addresses of all customers.

```
definitions
 set 'registered city' to the cities of the addresses of 'the customers';
```

In the following rule, the action executes only if there is one or more customers who live in France.

```
if
 the countries of the addresses of 'the customers' contain "France"
then
 set decision to "include EU countries";
```

**Parent topic:** [Definition part](#)

**Parent topic:** [Condition part](#)

**Parent topic:** [Action part](#)



## set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

### Purpose

You use this construct in the definitions part of a rule to declare a local variable.

### Syntax

```
set <variable> to <definition> [in <list> | from <object>] [where <test>*] ;
```

### Description

Variables produce more concise rules by replacing a value or the result of an expression with a short, convenient identifier. A local variable can be used anywhere within the rule in which it is defined, but is not available in other rules.

Enclose the name of each variable in single quotes. This is compulsory for variable names that contain spaces. While it is not essential to enclose one-word variable names in single quotes, it makes them easier to identify and reduces the risk of confusion.

### Example

The following examples show how to define a variable as an object.

```
definitions
 set 'i' to an item;
 set 'house' to a house
 where the price of this house is more than 1000;
```

The following examples show how to define a variable as a literal, string, or collection.

```
definitions
 set 'category' to Gold ;
 set 's' to "a string";
 set 'expensive items' to all items
 in the items of the shopping cart of the customer
 where the price of each item is more than 200;
```

The following example shows how to define a variable as the result of an expression.

```
definitions
 set 'expr' to the price of the car of the customer + 100;
 set 'h2' to the house of the customer
 where the price of this house is more than 1000;
```

The following example shows how to define a variable as another variable.

```
definitions
 set 'expr2' to 'expr';
```

**Parent topic:** [Definition part](#)

## where (test)

This construct matches objects against tests.

### Purpose

Matches objects against tests.

### Syntax

```
where <test>[,]*
```

### Description

In the `definitions` part of a rule, you can use one or more `where` clauses with the `set` construct to test that an object meets one or more conditions in order to be a valid value for the variable being declared. In the `definitions` part of the rule, the `where` statement does not end with a comma. Instead, each `set` construct should end with a semi-colon (;).

In the `if` part of a rule, you can use one or more `where` clauses in count conditions (`there is more than`, `there is less than`, and so on) to test that an object meets one or more conditions before being counted.

In a `where` construct, an implicit variable that refers to the current instance of the object being tested is automatically declared. In the test, you can thus refer to potential instances of the variable as `this <object type>` (or `each <object type>` if they are part of a collection.)

### Example

The following example shows the `definitions` part of a rule that defines the local variable `'expensive items'` as a collection of items whose price is greater than 100. The implicit variable `each item` is used in the `where` clause to refer to each item object in the collection.

```
definitions
 set 'expensive items' to all items
 where the price of each item is more than 100;
```

The following example shows the `where` clause used together with the `there is one` construct in the `if` part of a rule to test if there is a customer older than 100. The implicit variable `this customer` is used in the `where` clause to refer to the customer object being tested.

```
if
 there is one customer
 where the age of this customer is more than 100,
```

**Parent topic:** [Definition part](#)

**Parent topic:** [Condition part](#)

## Condition part

The condition part of the rule starts with the `if` keyword.

### **all of the following conditions are true**

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

### **any of the following conditions is true**

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

### **if**

This construct introduces the part of the rule in which you define conditions.

### **in (list)**

This construct defines a local variable in the rule that you can use to access objects from a collection.

### **it is not true that (a condition)**

This construct negates a condition statement.

### **(attribute) of (list)**

You use this construct to access the list of values of a given attribute for a list of objects.

### **none of the following conditions are true**

This construct negates a group of conditions.

### **there are (number) (objects)**

This construct tests whether there is a specific number of objects of a given type in the specified collection.

### **there are at least (number) (objects)**

This construct tests whether there is at least a number of objects of a given type in the specified collection.

### **there are at most (number) (objects)**

This construct tests whether there is no more than a specified number of objects of a given type in a collection.

### **there are less than (number) (objects)**

This construct tests whether there are fewer than a specified number of objects of a given type.

### **there are more than (number) (objects)**

This construct tests whether there are more than a specified number of objects of a given type.

### **there is at least one (object)**

This construct tests whether there is at least one object of a given type in a collection.

### **there is at most one (object)**

This construct tests whether there is at most one object of a given type in a collection.

### **there is no (object)**

This construct tests whether a data set contains no objects of the given type.

### **there is one (object)**

This construct tests whether a collection contains one and only one object of a given type.

### **where (test)**

This construct matches objects against tests.

**Parent topic:** [Decision Center modeling language reference](#)

## all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

### Purpose

The construct `all of the following conditions are true` provides a more compact and convenient alternative to the logical operator `and` when you want to combine multiple conditions. The resulting statement is considered true only if each of the listed conditions is true.

### Syntax

```
all of the following conditions are true:
 - <condition>* [,]
```

### Description

This is equivalent to writing `if <condition 1> and <condition 2> and ... <condition 2n>`. However, if you have a large number of conditions, using the `all of the following conditions are true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This separates those conditions that form part of the construct from any additional condition statements that might follow.

### Example

The following group of conditions tests that a customer is Gold junior member.

```
if
 all the following conditions are true:
 - the category of the customer is Gold
 - the age of the customer is at most 15
then...
```

**Parent topic:** [Condition part](#)

## any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

### Purpose

The construct `any of the following conditions are true` provides a more compact and convenient alternative to the logical operator `or` when you want to combine multiple conditions. The resulting statement is considered true if at least one of the listed conditions is true.

### Syntax

```
any of the following conditions is true:
 - <condition>* [,]
```

### Description

This is equivalent to writing `if <condition 1> or <condition 2> or ... <condition n>`. However, if you have a large number of conditions, using the `any of the following conditions is true` construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

#### Note:

Indentation in rules is for readability only; it does not influence the way the rule is processed.

### Example

This example shows how to test if a customer is a Gold member or a senior customer.

```
if
 any of the following conditions is true:
 - the category of the customer is Gold
 - the age of the customer is more than 60
then...
```

The following more complex example shows the use of a comma to mark the end of a group of conditions. In this example, the order must be over \$50, the customer must be either a Gold customer or a senior customer, and the order must be shipping to NJ.

```
if
 all of the following conditions are true:
 - the order total is greater than $50
 - any of the following conditions is true:
 - the category of the customer is Gold
 - the age of the customer is more than 60,
 - the shipping address is in NJ
then...
```

Without the comma to mark the end of the `any of the following conditions are true` block, the final condition (the shipping address is in NJ) would be considered part of the previous group, and the rule would be interpreted to mean that the order must be over \$50 and (either the customer is a Gold customer or a senior customer or the shipping address is in NJ).

**Parent topic:** [Condition part](#)

## if

This construct introduces the part of the rule in which you define conditions.

### Purpose

You use this construct in a rule to define one or more condition statements.

### Syntax

```
if
 <condition>*
```

### Description

If the conditions in the `if` part are met, the actions in the `then` part of the rule are executed. The `if` part of a rule is optional. If there is no `if` part, all actions in the `then` part of the rule are performed each time the rule is executed.

If the conditions in the `if` part of the rule are not met, the actions in the `else` part of the rule are executed. If the conditions are not met and the rule does not have an `else` part, the rule does nothing.

### Example

The following rule shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if
 the age of the customer is less than 18
then
 print "apply a 10% discount to the shopping cart of this customer";
```

**Parent topic:** [Condition part](#)

## it is not true that (a condition)

This construct negates a condition statement.

### Purpose

You use this construct when there is no operator to express the opposite of a condition.

### Syntax

```
it is not true that <condition>
```

### Description

When there is no operator to express the opposite of a condition, you can insert the `it is not true that` construct in front of an existing condition to negate it.

This can be useful in cases where a boolean (true/false) attribute defined in your model is verbalized as an expression such as 'customer is a loyalty member', but where you want to test for the opposite of this expression ('customer is not a loyalty member'). You can write 'it is not true that customer is a loyalty member' to achieve the required result.

In general, the logic of a negative expression is more difficult to interpret, so it is advisable to avoid them where possible.

### Example

The following example uses the `it is not true that` construct to check the age of the customer, assuming that the boolean property 'is a minor' has been verbalized in the vocabulary, but that the opposite of the expression 'is a major' has not been defined.

```
if
 it is not true that the Customer is a minor
```

The following example shows how to negate a set of conditions that have been grouped using the `all of the following conditions are true` construct. The resulting expression will return True except if the customer is both a gold member and over 60.

```
if
 it is not true that all of the following conditions are true:
 - the category of the Customer is gold
 - the age of the customer is more than 60
```

**Parent topic:** [Condition part](#)

## none of the following conditions are true

This construct negates a group of conditions.

### Purpose

You use this construct to group together a series of condition statements that you want to negate.

### Syntax

```
none of the following conditions are true:
 - <condition>* [,]
```

### Description

The group evaluates to true only if none of the conditions listed are true.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

### Example

The following group of conditions tests that a customer is not a Gold senior member and not under 60.

```
if
 none of the following conditions are true:
 - the category of the customer is gold
 - the age of the customer is least 60
then...
```

**Parent topic:** [Condition part](#)



## there are (number) (objects)

This construct tests whether there is a specific number of objects of a given type in the specified collection.

### Purpose

You use this construct to determine whether the collection contains exactly the specified number of occurrences of a particular object.

### Syntax

```
there are <number> <objects> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests if the number of Gold customers is equal to 8.

```
if
 there are 8 Customers in the customers of 'the bank'
 where the category of each customer is Gold
then...
```

**Parent topic:** [Condition part](#)

## there are at least (number) (objects)

This construct tests whether there is at least a number of objects of a given type in the specified collection.

### Purpose

You use this construct to determine whether the collection contains at least the specified number of occurrences of a particular object.

### Syntax

```
there are at least <number> <objects> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if there are at least 10 Gold customers.

```
if
 there are at least 10 Customers in the customers of 'the bank'
 where the category of each customer is Gold
then...
```

**Parent topic:** [Condition part](#)

## there are at most (number) (objects)

This construct tests whether there is no more than a specified number of objects of a given type in a collection.

### Purpose

You use this construct to determine whether the specified collection contains no more than the specified number of occurrences of a particular object.

### Syntax

```
there are at most <number> <objects> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if the number of Gold customers is lower than or equal to 3.

```
if
 there are at most 3 Customers in the customers of 'the bank'
 where the category of each customer is Gold
then...
```

**Parent topic:** [Condition part](#)

## there are less than (number) (objects)

This construct tests whether there are fewer than a specified number of objects of a given type.

### Purpose

You use this construct to determine whether the specified collection contains fewer than the specified number of occurrences of a particular object.

### Syntax

```
there are less than <number> <objects> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The *<number>* argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if the number of Gold customers is lower than 3.

```
if
 there are less than 3 Customers in the customers of 'the bank'
 where the category of each customer is Gold
then...
```

**Parent topic:** [Condition part](#)

## there are more than (number) (objects)

This construct tests whether there are more than a specified number of objects of a given type.

### Purpose

You use this construct to determine whether the specified collection contains more than the specified number of occurrences of a particular object.

### Syntax

```
there are more than <number> <objects> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

#### Note:

The `<number>` argument cannot be a `BigDecimal` or a `BigInteger`. Although the language parser does not generate an error, an error may be generated.

### Example

The following condition tests if there are more than 10 customers with the Gold category.

```
if
 there are more than Customers in the customers of 'the bank'
 where the category of each customer is Gold,
then...
```

**Parent topic:** [Condition part](#)

## there is at least one (object)

This construct tests whether there is at least one object of a given type in a collection.

### Purpose

You use this construct to determine whether the specified collection contains at least one occurrence of a particular object.

### Syntax

```
there is at least one <object> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests whether or not a Customer object exists.

```
if
 there is at least one customer
 where...
then...
```

The following condition tests if a senior Gold customer exists.

```
definitions
 set 'gold customers' to all Customers in the customers of 'the bank'
 where the category of each customer is Gold;
if
 there is at least one customer in 'gold customers'
 where the age of this customer is at least 60,
then...
```

**Parent topic:** [Condition part](#)

## there is at most one (object)

This construct tests whether there is at most one object of a given type in a collection.

### Purpose

You use this construct to determine whether the specified collection contains zero or one occurrence of a particular object.

### Syntax

```
there is at most one <object> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the `if` part of a rule.

### Example

The following condition tests if there is at most one Gold customer in the list of customers.

```
if
 there is at most one Customers in the customers of 'the bank'
 where the category of this customer is Gold,
then...
```

**Parent topic:** [Condition part](#)

## there is no (object)

This construct tests whether a data set contains no objects of the given type.

### Purpose

You use this construct to determine whether the specified collection contains no occurrences of a particular object.

### Syntax

```
there is no <object> in <list> [where <tests>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the `if` part of a rule.

### Example

The following condition tests that there is no object of type `Customer` with an age over 60 in the list of customers.

```
if
 there is no Customer in the customers of 'the bank' where the age of this
 customer is at least 60
then...
```

**Parent topic:** [Condition part](#)



## there is one (object)

This construct tests whether a collection contains one and only one object of a given type.

### Purpose

You use this construct to determine whether the specified collection contains one and only one occurrence of a particular object.

### Syntax

```
there is one <object> in <list> [where <test>,*]
```

### Description

Use the `in` clause to apply the test to a specific collection of objects. Use one or more optional `where` clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the `if` part of a rule.

### Example

The following condition tests if there is one Gold customer in the list of customers.

```
if
 there is one Customer in the customers of 'the bank'
 where the category of this customer is Gold
then...
```

**Parent topic:** [Condition part](#)

## Action part

The action part describes the actions that the rule completes when the conditions are met. This part might start with the then keyword and can contain an else construct.

### [add \(object\) to decision](#)

The add construct adds an item to a list of objects.

### [else](#)

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

### [for each \(object\) in \(list\)](#)

This construct specifies the actions that must be performed to every element of a collection.

### [for each \(object\) called \(variable\), in \(list\)](#)

This construct specifies that the rest of the rule must be applied to each element of a collection.

### [\(attribute\) of \(list\)](#)

You use this construct to access the list of values of a given attribute for a list of objects.

### [print \(string\)](#)

This construct prints a string to the standard output.

### [remove \(object\) from decision](#)

The remove construct removes an item from a list.

### [set decision to \(value\)](#)

This construct specifies a value for the decision to make.

### [then](#)

This construct introduces the part of the rule that defines the actions that are executed if the condition part of a rule is satisfied.

**Parent topic:** [Decision Center modeling language reference](#)

## add (object) to decision

The add construct adds an item to a list of objects.

### Purpose

You use this construct in the action part of a rule to add an object to a list of objects.

### Syntax

```
add <object> to decision;
```

### Description

The <object> specifies the object that you want to add. The list in which you want to include this object corresponds to the type of the decision node output, which must be a list.

### Example

The following action adds a string with the name of a race to the decision, which is a list of strings.

```
add "New matching race: " + the name of 'the race' to decision;
```

**Parent topic:** [Action part](#)

## else

This construct introduces actions that are executed if the conditions of a rule are not satisfied.

### Purpose

You use this construct to declare actions that are executed if the `if` part of a rule has not been satisfied. These actions are ignored if all local variables are not correctly initialized due to preconditions in the definitions part failing to be satisfied.

### Syntax

```
else
 <action>;*
```

### Description

The `else` part of a rule is optional and allows you to specify one or more actions to perform if the conditions in the condition part of the rule are not met.

The actions in the `else` part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

If one or more variables with preconditions are defined in the definition part of a rule, the rule is only processed if these preconditions are satisfied and all local variables are correctly initialized. This means that if these preconditions are not satisfied, the rule is not processed at all, and the actions in the `else` part of the rule are never executed, whether or not the conditions in the condition part of the rule are met.

Adding preconditions in the definition part of a rule rather than in the condition part of a rule can provide a small performance improvement, since it potentially reduces the number of conditions to be checked. However, since the logic of the rule is not the same in each case, the method to use depends on the result you want to obtain.

### Example

Consider a rule that sets the decision to the value of a discount. A customer will get a 10% discount if the customer is in the Gold category. Otherwise, the customer gets a 5% discount.

```
definitions
 set 'valued customer' to a customer
if
 the category of 'valued customer' is Gold
then
 set decision to 10;
else
 set decision to 5;
```

In the following example, a second condition has been added in the condition part of the rule. Now, a customer receives a 10% discount only if the customer is in the Gold category and is aged over 20. All other customers receive a 5% discount.

```
definitions
 set 'valued customer' to a customer
if
 the category of 'valued customer' is Gold and 'valued customer' is older
 than 20
then
 set decision to 10;
else
 set decision to 5;
```

Now consider a third example. In the following rule, a minimum age requirement is added as a precondition in the definition part of the rule. This means that a customer must first be over 20 years old to be considered a valued customer. Otherwise, the `valued customer` variable is not initialized and the rest of the rule is not executed. With this rule, a customer under 20 receives no discount at all. A customer who is over 20 and in the Gold category receives a 10% discount. The `else` part of the rule is executed and a 5% discount applied only if the customer is over 20 but not in the Gold category.

```
definitions
 set 'valued customer' to a customer
 where this customer is older than 20;
```

```
if
 the category of 'valued customer' is Gold
then
 set decision to 10;
else
 set decision to 5;
```

**Parent topic:** [Action part](#)

## for each (object) in (list)

This construct specifies the actions that must be performed to every element of a collection.

### Purpose

You use this construct to apply one or more actions to all objects in a collection.

### Syntax

```
for each <object> [called <variable>,) in <list>:
 - <action>*;
```

### Description

The construct should be immediately followed by a colon (:) and a list of one or more actions, each on a separate line, each preceded by a dash. End each action statement with a semi-colon (;).

The implicit variable *this* <object> can be used in each action statement within the *for each* loop to refer to the current object. If you nest a *for each* statement inside another, you can use the *called* construct to define an explicit name for the object in the 'outer' *for each* loop in order to reference it clearly from within the nested *for each* loop.

### Example

The following rule shows how to apply a discount to all the expensive items in a customer's shopping cart, if the customer is a Gold customer.

```
definitions
 set 'smith' to a customer ;
 set 'expensive items' to all items
 in the items of the shopping cart of smith
 where the price of 'each item' is greater than 1000;
if
 the category of smith is Gold
then
 for each item in 'expensive items':
 - set decision to 5;
```

The following rule shows two nested *for each* statements. The *called* <variable> construct is used so that each customer can be explicitly referred to from within the nested *for each* loop.

```
if
 the category of the customer is Gold
then
 for each customer called c, in 'senior customers':
 - for each account in 'new accounts':
 - set decision to the mailing address of c;
```

**Parent topic:** [Action part](#)

## for each (object) called (variable), in (list)

This construct specifies that the rest of the rule must be applied to each element of a collection.

### Purpose

You use this construct at the very beginning of the rule, and before the definitions part, to apply the rest of the rule to every object in a collection.

This construct is the equivalent of the construct set *<variable>* to *<definition>* in *<list>* that can be used in the definitions part of the rule.

### Syntax

```
for each <object> called <variable>, in <list>
```

### Description

When using a for each statement to perform an action on a set of items, a variable is created automatically to represent each item in the collection. You use the called *<variable>* construct to refer to this variable with a custom name. This custom name can also be used to specifically refer to this variable in a nested for each construct.

### Example

The following rule shows how to add races to a list of matching races by looking at each race, checking whether it meets the conditions, and if so, adding it to the decision.

```
for each race called 'the race', in 'the races'
if
 the area of 'the runner' is the area of 'the race'
 and the category of 'the race' is 'the runner category'
then
 add "Here is a race that we recommend for you: " + the name of 'the race'
to decision;
```

**Parent topic:** [Action part](#)

## **print (string)**

This construct prints a string to the standard output.

### **Purpose**

You use this construct to define an action phrase that by default outputs a string to the standard output.

### **Syntax**

```
print "<string>;
```

### **Example**

The following action prints the string "Hello World!" to the standard output.

```
print "Hello World!" ;
```

**Parent topic:** [Action part](#)



## remove (object) from decision

The remove construct removes an item from a list.

### Purpose

You use this construct in the action part of a rule to remove an object from a list of objects.

### Syntax

```
remove <object> from decision;
```

### Description

The <object> specifies the object that you want to remove. The list from which you want to remove this object corresponds to the type of the decision node output, which must be a list.

### Example

The following action removes a race from the list of races of the decision.

```
remove 'the race' from decision;
```

**Parent topic:** [Action part](#)

## set decision to (value)

This construct specifies a value for the decision to make.

### Purpose

You use this construct in the action part of a rule to make a decision by specifying its value.

### Syntax

```
set decision to <value>;
```

### Description

The `decision` keyword refers to the decision that the node makes. You can also use the variable name of the output of this decision in its place. The value can be a literal value, an expression, or a list.

### Example

Consider a decision logic for determining the current age of a customer. This decision logic might have a rule that sets the decision to the age of the customer. As a result, the current age is equal to the age of the customer.

```
set decision to the age of the customer;
```

Another way to write this action is to use the output name of the decision variable:

```
set 'the current age' to the age of the customer;
```

**Parent topic:** [Action part](#)

## then

This construct introduces the part of the rule that defines the actions that are executed if the condition part of a rule is satisfied.

### Purpose

You use this construct to introduce one or more actions to be executed if the conditions of the rule are met.

### Syntax

```
[then]
 <action>;*
```

### Description

Every rule must have an action part. If a rule has no definition part and no condition part, the then keyword is optional and the actions in this part of the rule are always executed whenever the rule is executed.

The actions in the action part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

### Example

The following example shows how to test the age of a customer in a condition, then execute an action if this test is true.

```
if
 the age of the customer is less than 18
then
 print "apply a 10% discount to the shopping cart of this customer";
```

The following example shows a rule that contains only an action with no definition and no condition.

```
set decision to (the number of monthly payments of 'the loan' + 11) / 12;
```

**Parent topic:** [Action part](#)

# Operators

You use operators in rule statements to perform arithmetic operations, associate or negate conditions, and compare expressions.

## Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

## Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

## Date operators

Use date operators to include time logic in rules.

## Number operators

Number operators define conditions based on numbers.

## Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

## Text operators

Text operators define conditions based on strings.

**Parent topic:** [Decision Center modeling language reference](#)

# Arithmetic operators

Arithmetic operators perform operations on numbers and text. You can use them in variable definitions, conditions, and actions.

Table 1. Arithmetic operators

Operator	Description	Example
- <number>	Makes a number negative.	<pre>if   there is more than one error then   set decision to - the number of errors;</pre>
<number> + <number>	Adds a number to another number.	<pre>if   there is at least one additional driver then   set decision to the base rental cost of 'the car' + 100;</pre>
<number> - <number>	Subtracts a number from another number.	<pre>if   the number of 'items purchased' - the number of 'items returned' is more than 0 then...</pre>
<number> / <number>	Divides a number by another number. If both numbers are integers, the result is also an integer (the result of the division without the remainder).	<pre>if   there is at least one order then   set decision to the number of 'items ordered' / the number of orders;</pre>
<number> * <number>	Multiplies a number by another number.	<pre>if   the price of the item * 'sales tax' is more than 100 then...</pre>
<text> + <text>	Concatenates two strings.	<pre>definitions   set decision to 'first name' + ' ' + 'last name'</pre>
<number> + <text>	Concatenates a number and a string. The result is treated	<pre>if</pre>

	as a string.	<pre>+'     the order total is less than 'discount threshold' then     print "Spend just \$" + ('discount threshold' - order total) + " more to receive a 10% discount!";</pre>
<text> + <number>	Concatenates a string and a number. The result is treated as a string.	<pre>if     the order total is more than 'discount threshold' then     print "You received a 10% discount because your order was over \$" + 'discount threshold' ;</pre>
lg(<number>)	Returns the base 10 logarithm of a number.	<pre>set decision to lg('the initial value')</pre>
ln(<number>)	Returns the natural logarithm (base e) of a number.	<pre>set decision to ln('the initial value')</pre>
max(<number>, <number>)	Returns the greater value.	<pre>set decision to max(the age 'the customer', 20)</pre>
min(<number>, <number>)	Returns the smaller value.	<pre>set decision to min(the age 'the customer', 20)</pre>
pow(<number>, <number>)	Returns the power of a number raised to the second argument.	<pre>set decision to pow('the initial value', 3)</pre>
round(<number>, <number>)	Rounds the first number to the number of decimal places that are given as second argument.	<pre>set decision to round('the intermediate total', 2)</pre>
sqrt(<number>)	Returns the positive square root of a number.	<pre>set decision to</pre>

		set decision to sqrt('the side')
--	--	-------------------------------------

Parent topic: [Operators](#)

# Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Table 1. Logical operators

Operator	Description	Example
<condition> and <condition>	Associates two conditions such that the resulting expression is only true if both conditions are true.	<pre>if   the category of the customer is Gold   and the age of the customer is at least 60 then...</pre>
<condition> or <condition>	Associates two conditions such that the resulting expression is true if either (or both) conditions are true.	<pre>if   the category of the customer is Gold   or the age of the customer is at least 60 then...</pre>

Parent topic: [Operators](#)



# Date operators

Use date operators to include time logic in rules.

## Date & time

Table 1. Date operators available for business rules that use time points in Decision Composer

Operator	Description	Example
<date & time> is at the same time as <date & time>	Checks that the two time points are identical, regardless of their time zone.	if the arrival date of 'the customer' is at the same time as the beginning of 'the promotional period'
<date & time> is at <time>	Checks that the time point is at the specified time.	if the return date of 'rented car' is at 05:00:00 PM
<date & time> is on <date>	Checks that the time point is on a specific day.	if the arrival date of 'the customer' is on 01/01/2018
<date & time> is on <day of week>	Checks that the time point is on a specific day of the week.	if the return date of 'rented car' is on Thursday
<date & time> is on <day of week> at <time>	Checks that the time point is on a specific day of the week, at a specific time.	if the return date of 'rented car' is on Thursday at 05:00:00 PM
<date & time> is on day <a number>	Checks that the time point is on a specific day, specified by a number ranging from 1 to 31.	if the arrival date of 'the customer' is on day 1
<date & time> is on <date & time>	Checks that the two time points have the same date.	if the arrival date of 'the customer' is on 01/01/2018 12:00:00 PM
<date & time> is in hour <a number>	Checks that the time point is on a specific hour.	if 'the flight departure' is in hour 13
<date & time> is in minute <a number>	Checks that the time point is on a specific minute.	if 'the flight departure' is in minute 45
<date & time > is in second <a number>	Checks that the time point is on a specific second.	if 'the flight

		departure' is in second 30
<date & time> is in <month>	Checks that the time point is on a specific month.	if 'the flight departure' is in January
<date & time> is in <month> <year>	Checks that the time point is in the specified month of the specified year.	if 'the flight departure' is in January 2019
<date & time> is in year <year>	Checks that the time point is in a specific year.	if 'the flight departure' is 2019
<date & time> is after <date & time>	Checks that the time point comes after another time point.	if 'the flight departure' is after 01/01/2018 12:00:00 PM
<date & time> is after or the same as <date & time>	Checks that the time point comes after another time point, or is the same as that other time point.	if 'the flight departure' is after or the same as 01/01/2018 12:00:00 PM
<date & time> is before <date & time>	Checks that the time point comes before another time point.	if 'the flight departure' is before 01/01/2018 12:00:00 PM
<date & time> is before or the same as <date & time>	Checks that the time point comes before another time point, or is the same as another time point.	if 'the flight departure' is before or the same as 01/01/2018 12:00:00 PM
<date & time> is after <date & time> and before <date & time>	Checks that the time point is in a range in which both time points are excluded.	if 'the flight departure' is after 01/01/2018 12:00:00 PM and before 08/01/2018 12:00:00 PM
<date & time> is after <date & time> and on or before <date & time>	Checks that the time point is in a range in which the first time point is excluded, and the last time point is included.	if 'the flight departure' is after 01/01/2018 12:00:00 PM and on or before 08/01/2018 12:00:00 PM

<date & time> is on or after <date & time> and before <date & time>	Checks that the time point is on or after a specific time point and before another time point.	if 'the flight departure' is on or after 01/01/2018 12:00:00 PM and before 08/01/2018 12:00:00 PM
<date & time> is on or after <date & time> and on or before <date & time>	Checks that the time point is on or after a specific time point and on or before another time point.	if 'the flight departure' is on or after 01/01/2018 12:00:00 PM and on or before 08/01/2018 12:00:00 PM
the local time of <a date & time> in <a string>	Converts the time point to a specific time zone, in the format defined by IANA	set the local time of 'the flight departure' in "Pacific/Midway"
the local time of <a date & time> in the zone of <a date & time>	Converts the first time point to the time zone of the second time point.	set the local time of 'the flight departure' in the zone of 'the flight arrival'
the number of <time unit> between <date & time> and <date & time>	Returns the number of years, months, days, hours, minutes, or seconds between two time points.	if the number of minutes between the connection time of 'the customer' and the beginning of 'the promotional period' is less than 10
the duration in <time unit> between <date & time> and <date & time>	Returns the number of time units between the two time points, where the <i>time unit</i> is one of <i>years, months,</i> <i>days, hours, minutes, or</i> <i>seconds</i> .	set 'the training duration' to the duration in months between the beginning of 'the training' and the end of 'the training'
the <calendar period> of <date & time>	Returns a calendar period in the Gregorian calendar that contains the specified time point, where the <i>calendar</i> <i>period</i> is one of <i>calendar</i> <i>year, calendar month,</i> <i>calendar week, calendar</i> <i>day, hour, minute, or</i> <i>second</i> .	the calendar day of the end of 'the promotional period'
the <time component> of <date & time>	Returns a time component of the specified time point, where the <i>time component</i>	the day of week of the end of 'the .. ..

	is one of <i>year, month, day of month, day of week, hour of day, minute of hour, second of minute, date, or time of day.</i>	<code>promotional period'</code>
--	-------------------------------------------------------------------------------------------------------------------------------	----------------------------------

Date

Table 2. Date operators available for business rules that use dates in Decision Composer

Operator	Description	Example
the <time component> of <date>	Returns a time component of the specified date where the <i>time component</i> is one of <i>year, month, day of month, or day of week.</i>	<code>the day of month of the beginning of 'the promotional period'</code>
<date> is on <day of week>	Checks that the date is on a specific day of the week.	<code>the beginning of 'the promotional period' is on Sunday</code>
<date> is on day <a number>	Checks that the date is on a specific day, specified by a number ranging from 1 to 31.	<code>if the arrival date of 'the customer' is on day 1</code>
<date> is in <month>	Checks that the date is on a specific month.	<code>the beginning of 'the promotional period' is in January</code>
<date> is in <year>	Checks that the date is on a specific year.	<code>the beginning of 'the promotional period' is in the year 2018</code>
<date> is in <month> the year <year>	Checks that the date is in the specified month of the specified year.	<code>if the return date of 'rented car' is in October the year 2007</code>
<date> is after <date>	Checks that the date comes after another date.	<code>the beginning of 'the promotional period' is after 01/01/2018</code>
<date> is after or the same as <date>	Checks that the date comes after another date, or is the same as that other date.	<code>the beginning of 'the promotional period' is after or the same as 01/01/2018</code>
<date> is before <date>	Checks that the date comes before another date.	<code>the beginning of 'the promotional period' is before 01/01/2018</code>

		'the promotional period' is before 01/01/2018
<date> is before or the same as <date>	Checks that the date comes before another date, or is the same as another date.	the beginning of 'the promotional period' is before or the same as the beginning of 'the winter holiday'
<date> is after <date> and before <date>	Checks that the date is in a range in which both dates are excluded.	the beginning of 'the promotional period' is after the beginning of 'the winter holiday' and before 01/01/2018
<date> is after <date> and on or before <date>	Checks that the date is in a range in which the first date is excluded, and the last date is included.	the beginning of 'the promotional period' is after the beginning of 'the winter holiday' and on or before 01/01/2018
<date> is on or after <date> and before <date>	Checks that the date is on or after a specific date and before another date.	the beginning of 'the promotional period' is on or after the beginning of 'the winter holiday' and before 01/01/2018
<date> is on or after <date> and on or before <date>	Checks that the date is on or after a specific date and on or before another date.	the beginning of 'the promotional period' is on or after the beginning of 'the winter holiday' and on or before 01/01/2018
the number of <time unit> between <date> and <date>	Returns the number of years, months, or days between two dates.	if the number of months between the subscription date of 'the customer' and the beginning of 'the promotional period' is less than 3

**Time**

---

Operator	Description	
<time> is after <time>	Checks that the time point comes after another time point.	if 'the flight departure' is after 12:00:00 PM
<time> is after or the same as <time>	Checks that the time point comes after another time point, or is the same as that other time point.	if 'the flight departure' is after or the same as 12:00:00 PM
<time> is before <time>	Checks that the time point comes before another time point.	if 'the flight departure' is before as 12:00:00 PM
<time> is before or the same as <time>	Checks that the time point comes before another time point, or is the same as another time point.	if 'the flight departure' is before or the same as 12:00:00 PM
<time> is after <time> and before <time>	Checks that the time point is in a range in which both time points are excluded.	if 'the flight departure' is after as 12:00:00 PM and before 05:00:00 PM
<time> is after <time> and on or before <time>	Checks that the time point is in a range in which the first time point is excluded, and the last time point is included.	if 'the flight departure' is after as 12:00:00 PM and on or before 05:00:00 PM
<time> is on or after <time> and before <time>	Checks that the time point is on or after a specific time point and before another time point.	if 'the flight departure' is on or after as 12:00:00 PM and before 05:00:00 PM
<time> is on or after <time> and on or before <time>	Checks that the time point is on or after a specific time point and on or before another time point.	if 'the flight departure' is on or after as 12:00:00 PM and on or before 05:00:00 PM

Day of week

Operator	Description	
<day of week> is after <day of week>	Checks that the day of the week comes after another day of the week. <div><b>Note:</b> Sunday is considered to be the first day of the week.</div>	if the day of the return date of 'rented car' is after Friday
<day of week> is after or the same as <day of week>	Checks that the day comes after another day of the week, or is the same as that other day.	if the day of the return date of 'rented car' is after or the same as Friday
<day of week> is	Checks that the day of the	if the day of the return



before <day of week>	week comes before another day of the week. <div><b>Note:</b> Sunday is considered to be the first day of the week.</div>	if the day of the return date of 'rented car' is before Friday
<day of week> is before or the same as <day of week>	Checks that the day comes before another day of the week, or is the same as another day of the week.	if the day of the return date of 'rented car' is before or the same as Friday
<day of week> is after <day of week> and before <day of week>	Checks that the day is in a range in which both days are excluded.	if the day of the return date of 'rented car' is after Friday and before Sunday
<day of week> is after <day of week> and on or before <day of week>	Checks that the day is in a range in which the first day is excluded, and the last day is included.	if the day of the return date of 'rented car' is after Friday and on or before Sunday
<day of week> is on or after <day of week> and before <day of week>	Checks that the day is on or after a specific day of the week and before another day.	if the day of the return date of 'rented car' is on or after Friday and before Sunday
<day of week> is on or after <day of week> and on or before <day of week>	Checks that the day is on or after a specific day of the week and on or before another day.	if the day of the return date of 'rented car' is on or after Friday and on or before Sunday

Month of year

Operator	Description	
<month of year> is after <month of year>	Checks that the month comes after another month of the year.	if 'the promotional period' is after June
<month of year> is after or the same as <month of year>	Checks that the month comes after another month of the year, or is the same as that other month.	if 'the promotional period' is after or the same as June
<month of year> is before <month of year>	Checks that the month comes before another month of the year.	if 'the promotional period' is before May
<month of year> is before or the same as <month of year>	Checks that the month comes before another month of the year, or is the same as another month.	if 'the promotional period' is before or the same as May
<month of year> is after <month of year> and before <month of year>	Checks that the month is in a range in which both months of the year are excluded.	if 'the promotional

		if 'the promotional period' is after May and before September
<month of year> is after <month of year> and on or before <month of year>	Checks that the month is in a range in which the first month is excluded, and the last month is included.	if 'the promotional period' is after May and on or before September
<month of year> is on or after <month of year> and before <month of year>	Checks that the month is on or after a specific month of the year and before another month.	if 'the promotional period' is on or after May and before September
<month of year> is on or after <month of year> and on or before <month of year>	Checks that the month is on or after a specific month of the year and on or before another month.	if 'the promotional period' is on or after May and on or before September

**Year**

Operator	Description	
<year> is after <year>	Checks that the year comes after another year.	if 'the flight departure' is after 2018
<year> is after or the same as <year>	Checks that the year comes after another year, or is the same as that other year.	if 'the flight departure' is after or the same as 2018
<year> is before <year>	Checks that the year comes before another year.	if 'the flight departure' is before 2018
<year> is before or the same as <year>	Checks that the year comes before another year, or is the same as another year.	if 'the flight departure' is before or the same as 2018
<year> is after <year> and before <year>	Checks that the year is in a range in which both year are excluded.	if 'the flight departure' is after 2018 and before 2020
<year> is after <year> and on or before <year>	Checks that the year is in a range in which the first year is excluded, and the last year is included.	if 'the flight departure' is after 2018 and on or before 2020
<year> is on or after <year> and before <year>	Checks that the year is on or after a specific year and before another year.	if 'the flight departure' is on or



		after 2018 and before 2019
<year> is on or after <year> and on or before <year>	Checks that the year is on or after a specific year and on or before another year.	if 'the flight departure' is on or after 2018 and on or before 2019

### Time period

Table 3. Date operators available for business rules that use time periods in Decision Composer

Operator	Description	Example
<time period> is during the same time as <time period>	Checks that the two time periods are identical and on the same time zone.	the calendar week of the meeting is during the same time as the calendar week of the departure
the start of <time period>	Returns the start of the specified time period.	the start of the calendar week of the meeting is before 2017/12/01 12:00:00 AM
the end of <time period>	Returns the end of the specified time period.	the end of the calendar week of the meeting is before 2017/12/01 12:00:00 AM

### Calendar duration

Table 4. Date operators available for business rules that use calendar durations in Decision Composer

Operator	Description	Example
<number> <time unit>	Returns a calendar duration in years, months, days, hours, minutes, or seconds.	7 months
<date & time> - <calendar duration>	Returns a new time point that precedes the specified time point by the specified duration.	the beginning of 'the training' - 1 month
<date & time> + <calendar duration>	Returns a new time point that follows the specified time point by the specified duration.	the beginning of 'the training' + 1 month
<calendar duration> + <calendar duration>	Returns a calendar duration that corresponds to the sum of the two durations.	the duration of 'the training' + the duration of 'the certification'

--	--	--

Parent topic: [Operators](#)

# Number operators

Number operators define conditions based on numbers.

Table 1. Number operators

Operator	Description	Example
<number> does not equal <number>	Tests that a number is not equal to another number.	<pre>if     the number of rentals does not equal 3 then...</pre>
<number> equals <number>	Tests that a number is equal to another number. This is equivalent to the is <number> operator	<pre>if     the number of rentals equals 3 then...</pre>
<number> is <number>	Tests that a number is equal to another number. This is equivalent to the equals <number> operator	<pre>if     the number of rentals is 3 then...</pre>
<number> is at least <number>	Tests that a number is greater than or equal to another number.	<pre>if     the number of rentals is at least 3 then...</pre>
<number> is at least <number> and less than <number>	Tests that a number is included in a range in which the first value is included and the last value is excluded.	<pre>if     the age of the customer is at least 12 and less than 25 then...</pre>
<number> is at most <number>	Tests that a number is less than or equal to another number.	<pre>if     the number of rentals is at most 3 then...</pre>
<number> is between <number> and <number>	Tests that a number is included in a range in which both values are included.	<pre>if     the age of the customer is between 12 and 25 then...</pre>
<number> is less than <number>	Tests that one number is less than another.	<pre>if     the number of rentals is less than 3  then...</pre>

<number> is more than <number>	Tests that one number is greater than another.	if the number of rentals is more than 3 then...
<number> is more than <number> and at most <number>	Tests that a number is included in a range in which the first value is excluded and the last value is included.	if the age of the customer is more than 12 and at most 25 then...
<number> is strictly between <number> and <number>	Tests that a number is included in a range in which the first value is excluded and the last value is excluded.	if the age of the customer is strictly between 12 and 25 then...

Parent topic: [Operators](#)

# Object operators

Object operators define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Table 1. Object operators

Operator	Description	Example
<object> is <object>	Tests that two objects are equivalent.	<pre>if     the current customer is the customer on the rental agreement then...</pre>
<object> is not <object>	Tests that two objects are not equivalent.	<pre>if     the current customer is not the customer on the rental agreement then...</pre>
<object> is one of <list>	Tests that an object is part of a set.	<pre>if     the item of the order is one of 'discounted items' then...</pre>
<object> is not one of <list>	Tests that an object is not part of a collection.	<pre>if     the customer category is not one of 'all categories' then...</pre>
<list> contain <object>	Tests that a collection contains an object. This operator is functionally equivalent to <object> is one of <list>.	<pre>if     the customer categories contain Platinum then...</pre>
<list> do not contain <object>	Tests that a collection does not contain an object. This operator is functionally equivalent to <object> is not one of <list>.	<pre>if     the customer categories do not contain Normal then...</pre>
the number of <objects >	Returns the total number of objects of the specified type.	<pre>if     the number of vehicles is more than 5 then...</pre>

Parent topic: [Operators](#)

# Text operators

Text operators define conditions based on strings.

Table 1. Text Operators in BAL

Operator	Description	Example
<text> contains <text>	Tests that a string contains another string.	<pre>if     the name of the customer contains "Smith" then...</pre>
<text> does not contain <text>	Tests that a string does not contain another string.	<pre>if     the name of the customer does not contain "Smith" then...</pre>
<text> does not end with <text>	Tests that a string does not end with another string.	<pre>if     the rental agreement code of the customer does not end with "XG5" then...</pre>
<text> does not start with <text>	Tests that a string does not start with another string.	<pre>if     the rental agreement code of the customer does not start with "XG5" then...</pre>
<text> ends with <text>	Tests that a string ends with another string.	<pre>if     the rental agreement code of the customer ends with "XG5" then...</pre>
<text> is empty	Tests that a string is empty. An empty string contains no characters and is equivalent to "". A string is not empty if it contains a space (" ")	<pre>if     the rental agreement code of the customer is empty then...</pre>
<text> is not empty	Tests that a string is not empty. A string is not empty if it contains a space (" ")	<pre>if     the rental agreement code of the customer is not empty then...</pre>
print <text>	Prints a string to standard out.	<pre>if</pre>

		<pre>11.... then     print the message     "Hello World"</pre>
<p>&lt;text&gt; starts with &lt;text&gt;</p>	Tests that a string starts with another string.	<pre>if     the rental     agreement code of the     customer starts with     "XG5" then...</pre>
<p>the length of &lt;text&gt;</p>	Returns the number of characters in a string.	<pre>if     the length of     'customer name' is     more than 3 then...</pre>

Parent topic: [Operators](#)

# Literals

The business rule language supports number, string, date, and time data types.

## Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

## Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

## Dates and times

You can use the Date & Time, Date, and Universal Date & Time value types to express date values. You can use the Time value type to express time values.

**Parent topic:** [Decision Center modeling language reference](#)



# Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent.

- 10 (integer)
- 10.5 (decimal, with decimal point separator)
- 150,500.51 (decimal, with '000 grouping separator and decimal point separator)
- -10 (negative integer)
- 10E-5 (exponent)

The lexical representation of numbers (the decimal point separator and the grouping separator) is locale-specific.

**Note:**

Big number literals are not supported in the business rule language. All literals manipulated in a rule must be within the range of the Java™ Double class, otherwise they are rounded to the closest Double value.

**Parent topic:** [Literals](#)

# Strings

String literals are represented by a set of characters enclosed in double quotation marks ("Text string").

To include certain special characters within a String, you must prefix them with a backslash (\). For example, to include a double quotation mark within a String, you would write \". The backslash is required here to indicate that the String has not yet ended.

The following special character sequences are accepted within text strings:

- \n (new line character)
- \t (tab character)
- \b
- \r (carriage return character)
- \f (line feed character)
- \' (single quotation mark)
- \" (double quotation mark)
- \\ (backslash)

**Parent topic:** [Literals](#)

# Dates and times

You can use the Date & Time, Date, and Universal Date & Time value types to express date values. You can use the Time value type to express time values.

Several types of date and time are available.

In the following table, a is the symbol for AM or PM, and z is the time zone information, for example +0200.

Table 1. Date and time format

Value type	Format
Date & Time	Standard date format that includes time, and is written as: m/d/y h:mm:ss a
Date	Simple date format that does not include time, and is written as: m/d/y
Time	Time format that is written as either: h:mm:ss a h:mm a
Universal Date & Time	Date and time format that includes the time zone, and is written as: m/d/y h:mm:ss a z

**Note:**

The business rule language supports only the Gregorian calendar format.

**Parent topic:** [Literals](#)

## Punctuation in rules

The business rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules, mandatory punctuation is usually predicted in the completion menu.

### Backslash \

The backslash is used to prefix special characters in text strings.

### Comma ,

The comma marks the end of Where statements.

#### Example

This example shows a Where statement that is closed by a comma.

```
if
 there is at least one car
 where
 the color of this car is blue,
then ...
```

Some language constructs may be terminated by an optional comma.

#### Example

This example shows a construct that is terminated by an optional comma. This may prevent the construct becoming ambiguous in a particular context.

```
all following conditions are true:
- the color of the car is blue
- the price of the car is more than 1000,
```

### Double quotation mark "

Double quotation marks are used to enclose string literals; that is, text strings.

#### Example

This example shows an action phrase in which the message to be displayed is a text string enclosed in double quotation marks.

```
print "You received a discount!";
```

### Parentheses ( )

Parentheses can be used to group any expression. They can help clarify the default precedence of logical operators linking conditions to one another. You can also use parentheses to regroup condition statements and hence change the order in which they are interpreted.

#### Example

This example shows nested parentheses in an action phrase.

```
print "the customer: " + (the name of the customer of (the shopping cart));
```

### Semicolon ;

The semicolon marks the end of variable definitions and action phrases.

### Single quotation mark '

Single quotation marks are used to enclose variables. Single quotation marks are optional for single-word variables and mandatory for variables that consist of several words. Automatic variables are not enclosed in single quotation marks.

#### Example

In this example, the variables `customer` and `the cart` are defined. The `customer` variable does not need to be enclosed in single quotation marks because it is a single word, whereas the `the cart` needs to be delimited because it consists of two words.

*definitions*

```
set customer to a customer;
set 'the cart' to the shopping cart of customer;
```

**Parent topic:** [Decision Center modeling language reference](#)

# Decision Center Messages

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRAB</a>	Messages pertaining to BRLDF
<a href="#">GBRAD</a>	Messages pertaining to decision tables
<a href="#">GBRD</a>	Messages pertaining to governance
<a href="#">GBRIE</a>	Messages pertaining to simulation
<a href="#">GBRM</a>	Messages pertaining to the model and the business object model (BOM)
<a href="#">GBRP</a>	Messages pertaining to the samples
<a href="#">GBRT</a>	Messages pertaining to testing and simulation

# Decision Center Messages - Messages pertaining to BRLDF

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRAB0001E</a>	Bad index for token: {0} (line: {2}, column: {3}).
<a href="#">GBRAB0002E</a>	Cannot add a child to token {0} (line: {2}, column: {3}).
<a href="#">GBRAB0003E</a>	Reference to unknown operator "{0}" in grammar (type: {1}) at: {2}.
<a href="#">GBRAB0004E</a>	Reference to unknown terminal "{0}" in grammar (type: {1}) at: {2}.
<a href="#">GBRAB0005E</a>	Invalid numeric value.
<a href="#">GBRAB0006E</a>	Empty numeric value.
<a href="#">GBRAB0007E</a>	Empty value.
<a href="#">GBRAB0008E</a>	The schema has no element.
<a href="#">GBRAB0009E</a>	Element type not found: {0}.
<a href="#">GBRAB0010E</a>	Grammar not set on this syntax tree.
<a href="#">GBRAB0011E</a>	There is no root named "{0}" in grammar.
<a href="#">GBRAB0012E</a>	Current grammar node is not a list: "{0}" (type: {1}) in path: {2}
<a href="#">GBRAB0013E</a>	The grammar node "{0}" has not been found in path: {1}
<a href="#">GBRAB0014E</a>	Bad index for syntax tree node {0} : {1}.
<a href="#">GBRAB0015E</a>	Node not found in the abstract syntax tree: {0}.

# Decision Center Messages - Messages pertaining to decision tables

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRAD0001E</a>	Exception raised while loading "{0}". See log file.
<a href="#">GBRAD0002E</a>	Exception raised while checking "{0}". See log file.



# Decision Center Messages - Messages pertaining to governance

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRDR0001E</a>	
<a href="#">GBRDR0002E</a>	
<a href="#">GBRDR0004E</a>	
<a href="#">GBRDR0005E</a>	
<a href="#">GBRDR0006E</a>	
<a href="#">GBRDR0007E</a>	
<a href="#">GBRDR0008E</a>	
<a href="#">GBRDR0009E</a>	
<a href="#">GBRDR0010E</a>	
<a href="#">GBRDR0011E</a>	
<a href="#">GBRDR0012E</a>	
<a href="#">GBRDR0013E</a>	
<a href="#">GBRDR0014E</a>	
<a href="#">GBRDR0015E</a>	
<a href="#">GBRDR0016E</a>	
<a href="#">GBRDR0017E</a>	
<a href="#">GBRDR0018E</a>	
<a href="#">GBRDR0019E</a>	
<a href="#">GBRDR0020E</a>	
<a href="#">GBRDR0021E</a>	
<a href="#">GBRDR0022E</a>	
<a href="#">GBRDR0023E</a>	
<a href="#">GBRDR0024E</a>	
<a href="#">GBRDR0025E</a>	The element may have been deleted by another user, or you do not have permission to access it.
<a href="#">GBRDR0026E</a>	
<a href="#">GBRDR0027E</a>	
<a href="#">GBRDR0028E</a>	
<a href="#">GBRDR0029E</a>	
<a href="#">GBRDR0030E</a>	
<a href="#">GBRDR0031E</a>	
<a href="#">GBRDR0032E</a>	
<a href="#">GBRDR0033E</a>	
<a href="#">GBRDR0034E</a>	
<a href="#">GBRDR0035E</a>	
<a href="#">GBRDR0036E</a>	

<a href="#">GBRDR0037E</a>	
<a href="#">GBRDR0038E</a>	
<a href="#">GBRDR0039E</a>	
<a href="#">GBRDR0040E</a>	
<a href="#">GBRDR0041E</a>	
<a href="#">GBRDR0042E</a>	
<a href="#">GBRDR0043E</a>	
<a href="#">GBRDR0044E</a>	
<a href="#">GBRDR0045E</a>	
<a href="#">GBRDR0046E</a>	
<a href="#">GBRDR0047E</a>	
<a href="#">GBRDR0048E</a>	
<a href="#">GBRDR0049E</a>	
<a href="#">GBRDR0050E</a>	
<a href="#">GBRDR0051E</a>	
<a href="#">GBRDR0052E</a>	
<a href="#">GBRDR0053E</a>	
<a href="#">GBRDR0054E</a>	
<a href="#">GBRDR0055E</a>	
<a href="#">GBRDR0056E</a>	
<a href="#">GBRDR0057E</a>	
<a href="#">GBRDR0058E</a>	
<a href="#">GBRDR0059E</a>	
<a href="#">GBRDR0060E</a>	
<a href="#">GBRDR0061E</a>	
<a href="#">GBRDR0062E</a>	
<a href="#">GBRDR0063E</a>	
<a href="#">GBRDR0064E</a>	
<a href="#">GBRDR0065E</a>	
<a href="#">GBRDR0066E</a>	
<a href="#">GBRDR0067E</a>	
<a href="#">GBRDR0068E</a>	
<a href="#">GBRDR0069E</a>	
<a href="#">GBRDR0070E</a>	
<a href="#">GBRDR0071E</a>	
<a href="#">GBRDR0072E</a>	
<a href="#">GBRDR0073E</a>	
<a href="#">GBRDR0074E</a>	
<a href="#">GBRDR0075E</a>	
<a href="#">GBRDR0076E</a>	
<a href="#">GBRDR0077E</a>	
<a href="#">GBRDR0078E</a>	
<a href="#">GBRDR0079E</a>	
<a href="#">GBRDR0080E</a>	
<a href="#">GBRDR0081E</a>	
<a href="#">GBRDR0082E</a>	

<a href="#">GBRDR0083E</a>	
<a href="#">GBRDR0084E</a>	
<a href="#">GBRDR0085E</a>	
<a href="#">GBRDR0086E</a>	
<a href="#">GBRDR0087E</a>	
<a href="#">GBRDR0088E</a>	
<a href="#">GBRDR0089E</a>	
<a href="#">GBRDR0090E</a>	
<a href="#">GBRDR0091E</a>	
<a href="#">GBRDR0092E</a>	
<a href="#">GBRDR0093E</a>	
<a href="#">GBRDR0094E</a>	
<a href="#">GBRDR0095E</a>	
<a href="#">GBRDR0096E</a>	
<a href="#">GBRDR0097E</a>	
<a href="#">GBRDR0098E</a>	
<a href="#">GBRDR0099E</a>	
<a href="#">GBRDR0100E</a>	
<a href="#">GBRDR0101E</a>	
<a href="#">GBRDR0102E</a>	
<a href="#">GBRDR0103E</a>	
<a href="#">GBRDR0104E</a>	

# Decision Center Messages - Messages pertaining to simulation

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">1E</a>	Error occurred during the serialization
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">2E</a>	Error occurred during the deserialization
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">3E</a>	Error occurred when creating the JAXBContext instance for the {0} class
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">4E</a>	Error occurred when retrieving the job XML descriptor
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">5E</a>	Error occurred when setting the job XML descriptor
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">6E</a>	Error occurred when parsing the scenario provider xml file
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">7E</a>	Unable to deserialize the input parameter of type {0}
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">8E</a>	Unable to load the class {0}
<a href="#">GBRI</a> <a href="#">E000</a> <a href="#">9E</a>	The input parameter format of the scenario suite is unsupported or not specified
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">0E</a>	Error occurred when parsing the input parameter {0}
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">1E</a>	An unknown item {0} was found when deserializing the array object
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">2E</a>	The mandatory attribute {0} is missing in xml element {1}
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">3E</a>	An unknown input {0} was found in the scenario
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">4E</a>	Error occurred when creating the custom serializer for input parameters
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">5E</a>	Error occurred while starting the job
<a href="#">GBRI</a> <a href="#">E001</a> <a href="#">6E</a>	Error occurred while stopping the job with execution identifier {0}

<a href="#">GBRI E001 7E</a>	Error occurred while deleting data for the job with execution identifier {0}
<a href="#">GBRI E001 8E</a>	Error occurred while getting the status for the job with execution identifier {0}
<a href="#">GBRI E001 9E</a>	Error occurred while getting the top-level errors for the job with execution identifier {0}
<a href="#">GBRI E002 0E</a>	Error occurred while getting the KPI results for the job with execution identifier {0}
<a href="#">GBRI E002 1E</a>	Error occurred while getting the scenario results for the job with execution identifier {0}
<a href="#">GBRI E002 2E</a>	Error occurred while getting the execution details for the job with execution identifier {0}
<a href="#">GBRI E020 1E</a>	Error occurred while generating RuleApp
<a href="#">GBRI E020 2E</a>	Error occurred when starting the job
<a href="#">GBRI E020 3E</a>	Unable to execute ruleset
<a href="#">GBRI E020 4E</a>	Error occurred while trying to clean up the rule session
<a href="#">GBRI E020 5E</a>	Unable to create a stateful rule session for ruleset path: {0}
<a href="#">GBRI E020 6E</a>	Error occurred when stopping the job with execution identifier {0}
<a href="#">GBRI E020 7E</a>	No scenario suite format in the scenario suite descriptor
<a href="#">GBRI E020 8E</a>	Error occurred while saving scenario suite metrics
<a href="#">GBRI E020 9E</a>	JobContext instance is not available, it must not have been injected properly (current class: {0})
<a href="#">GBRI E021 0E</a>	No job property found in JobContext instance
<a href="#">GBRI E021 1E</a>	No scenario file bytes
<a href="#">GBRI E021 2I</a>	Managed XOM class loader does not exist, the ruleset may not be linked to any XOM resource or library
<a href="#">GBRI E021 3E</a>	Unable to retrieve the rule session for the job with execution identifier {0}
<a href="#">GBRI E021 4E</a>	{0} artifact class could not be instantiated
<a href="#">GBRI E021 5E</a>	Error occurred when getting the managed XOM class loader on the rule session for the job with execution identifier {0}

<a href="#">GBRI E021 6E</a>	Item to process does not have the expected type: expecting type {0} but found {1}
<a href="#">GBRI E021 7E</a>	Unexpected null item to process
<a href="#">GBRI E021 8E</a>	Error occurred while executing ruleset
<a href="#">GBRI E021 9E</a>	Invalid scenario suite format: the job XML descriptor is missing
<a href="#">GBRI E022 0E</a>	Error occurred while deploying RuleApp
<a href="#">GBRI E022 1E</a>	Error occurred while undeploying RuleApp
<a href="#">GBRI E022 2E</a>	The object factory cannot be used to create the object instances because of the following issues: {0}
<a href="#">GBRI E022 3W</a>	Rule session could not be cleaned up
<a href="#">GBRI E022 4E</a>	Invalid scenario suite descriptor: either a ruleset archive or a ruleset path should be included
<a href="#">GBRI E022 5E</a>	{0} property not found in the job properties: {1}
<a href="#">GBRI E022 6W</a>	{0} ruleset path does not seem to represent a temporary ruleset but RuleApp will be undeployed anyway
<a href="#">GBRI E022 7E</a>	The ruleset required for the job execution could not be deployed
<a href="#">GBRI E022 8E</a>	The RuleApp with ruleset path {0} could not be undeployed
<a href="#">GBRI E022 9E</a>	A session factory is expected
<a href="#">GBRI E023 0E</a>	Error occurred while adding RuleApp {0} to repository
<a href="#">GBRI E023 1E</a>	Error occurred while removing RuleApp {0} from the repository
<a href="#">GBRI E023 2E</a>	Error occurred while creating repository factory
<a href="#">GBRI E023 3I</a>	{0} RuleApp successfully deployed
<a href="#">GBRI E023 4I</a>	{0} RuleApp successfully undeployed
<a href="#">GBRI E023 5E</a>	Resource {0} does not have the expected type: expecting type {1} but found {2}
<a href="#">GBRI E023 6E</a>	Unable to save error for the job with execution identifier {0}
<a href="#">GBRI E023 7E</a>	Unable to serialize {0} resource

<a href="#">GBRI E023 7E</a>	Unable to serialize {0} resource
<a href="#">GBRI E023 8E</a>	Unable to deserialize {0} resource
<a href="#">GBRI E023 9E</a>	Ruleset path not specified
<a href="#">GBRI E024 0E</a>	Ruleset archive not specified
<a href="#">GBRI E024 1E</a>	The RuleApp already exists and we are not allowed to replace it
<a href="#">GBRI E024 2E</a>	{0} is not a valid value for resource {1}
<a href="#">GBRI E024 3E</a>	Unable to retrieve the object model services for ruleset path {0}
<a href="#">GBRI E024 4E</a>	Unable to read the Excel Scenario File
<a href="#">GBRI E024 5E</a>	Unable to retrieve the number of scenarios in the Excel Scenario File
<a href="#">GBRI E024 6E</a>	Error occurred while getting the output parameters as XML BOM
<a href="#">GBRI E024 7E</a>	Error occurred while invoking the metrics computation function
<a href="#">GBRI E024 8E</a>	Metrics list does not have the expected type: expecting a Map instance but found {0}
<a href="#">GBRI E024 9W</a>	The metrics computation function returned <null>
<a href="#">GBRI E025 0E</a>	Error occurred while computing metrics
<a href="#">GBRI E025 1W</a>	Unable to add job parameter with name {0} since there is already a job parameter with the same name
<a href="#">GBRI E025 2E</a>	Error occurred while disposing KPI calculator
<a href="#">GBRI E025 3E</a>	Unable to retrieve the main job execution identifier for the job with execution identifier {0}
<a href="#">GBRI E025 4E</a>	KPI calculator cannot be disposed, since no KPI calculator was found for {0} job execution identifier
<a href="#">GBRI E025 5E</a>	KPI calculator cannot be accessed for {0} job execution identifier, it may not have been created properly
<a href="#">GBRI E025 6E</a>	Error occurred while parsing KPI definitions
<a href="#">GBRI E025 7E</a>	Unable to retrieve the ruleset path for the job with execution identifier {0}
<a href="#">GBRI</a>	Error occurred while resetting rule session

<a href="#">E0258E</a>	
<a href="#">GBRI E0259E</a>	Error occurred while computing KPIs
<a href="#">GBRI E0260E</a>	Unable to serialize KPI result for {0} KPI to JSON
<a href="#">GBRI E0261E</a>	Error occurred while extracting execution details
<a href="#">GBRI E0262E</a>	Unable to retrieve job parameters for the job with execution identifier {0}
<a href="#">GBRI E0263E</a>	Error occurred while saving KPI results
<a href="#">GBRI E0264E</a>	Error occurred while updating scenario suite metrics
<a href="#">GBRI E0265E</a>	Error occurred while creating rule session request
<a href="#">GBRI E0266E</a>	No scenario suite descriptor
<a href="#">GBRI E0267E</a>	The rule session is null for the job with execution identifier {0}
<a href="#">GBRI E0268E</a>	Error occurred while saving scenario results
<a href="#">GBRI E0269E</a>	Error occurred while saving scenario execution details
<a href="#">GBRI E0270W</a>	Unable to determine whether the RuleApp should be undeployed or not for job with execution identifier {0}
<a href="#">GBRI E0271E</a>	Error occurred while setting the execution trace filter
<a href="#">GBRI E0272E</a>	Job property {0} with string value {1} cannot be parsed as {2}
<a href="#">GBRI E0273E</a>	Error occurred while processing metrics
<a href="#">GBRI E0274W</a>	{0} property not found in the job properties ({1})
<a href="#">GBRI E0275W</a>	Using {0} as default value for job property {1}
<a href="#">GBRI E0276E</a>	{0} is not a valid value for job property {1}
<a href="#">GBRI E0277E</a>	Error occurred while computing partial KPIs
<a href="#">GBRI E0278E</a>	Error occurred while computing final KPIs
<a href="#">GBRI</a>	Error occurred while initializing KPI calculator



<a href="#">GBRI E027 9E</a>	Error occurred while initializing the calculator.
<a href="#">GBRI E028 0E</a>	Unable to retrieve the job execution instance for the job with execution identifier {0}
<a href="#">GBRI E028 1E</a>	Error occurred while updating KPI results
<a href="#">GBRI E028 2E</a>	Error occurred while getting the XOM parameters
<a href="#">GBRI E028 3E</a>	No XOM parameters
<a href="#">GBRI E028 4E</a>	Unable to retrieve the managed XOM class loader for the job with execution identifier {0}
<a href="#">GBRI E028 5E</a>	The session request is null
<a href="#">GBRI E028 6E</a>	The session response is null
<a href="#">GBRI E028 7E</a>	Error occurred while saving error
<a href="#">GBRI E028 8E</a>	{0} batch thread pool service initialization failed
<a href="#">GBRI E028 9E</a>	Managed thread failed to complete
<a href="#">GBRI E029 0E</a>	Parallel managed thread failed to complete
<a href="#">GBRI E029 1E</a>	Error occurred while retrieving {0} batch thread pool service
<a href="#">GBRI E029 2E</a>	Error occurred while retrieving job parameters
<a href="#">GBRI E029 3E</a>	Error occurred while getting the input parameters as XML BOM
<a href="#">GBRI E029 4E</a>	Job data cannot be deleted because the job status is not final - job may be still running
<a href="#">GBRI E029 5E</a>	Unable to read the XML Scenario File
<a href="#">GBRI E029 6E</a>	Error occurred while getting the built-in serializer
<a href="#">GBRI E029 7E</a>	Error occurred while getting the custom serializer
<a href="#">GBRI E029 8E</a>	Error occurred while retrieving the domains cache
<a href="#">GBRI E029 9I</a>	Using a thread pool size value of {0} as found in configuration for {1} batch thread pool service
<a href="#">GBRI</a>	Invalid thread pool size value {0} found in configuration for {1} batch thread pool service - the default value

<a href="#">E0300E</a>	({2}) will be used
<a href="#">GBRIE0301W</a>	No thread pool size value found in configuration for {0} batch thread pool service - the default value ({1}) will be used
<a href="#">GBRIE0302E</a>	Error occurred while shutting down {0} batch thread pool service
<a href="#">GBRIE0303E</a>	Main execution thread failed to complete
<a href="#">GBRIE0304E</a>	Parallel execution thread failed to complete
<a href="#">GBRIE0305E</a>	Domains values list does not have the expected type: expecting a Map instance but found {0}
<a href="#">GBRIE0306E</a>	Error occurred while invoking the domains cache build function
<a href="#">GBRIE0307I</a>	The executor service used for running batch jobs has been shut down
<a href="#">GBRIE0308E</a>	A thread request has been received by the batch thread pool service {0} but the executor service is not initialized or has been shut down
<a href="#">GBRIE0309E</a>	Error occurred while releasing resources
<a href="#">GBRIE0310E</a>	Error occurred while initializing the persistence
<a href="#">GBRIE0311E</a>	Error occurred while invoking the test evaluation function
<a href="#">GBRIE0312E</a>	The test result object does not have the expected type: expecting a Map instance but found {0}
<a href="#">GBRIE0313W</a>	The test evaluation function returned <null>
<a href="#">GBRIE0314E</a>	Error occurred while evaluating tests
<a href="#">GBRIE0315E</a>	Error occurred while retrieving execution details
<a href="#">GBRIE0316E</a>	The execution detail object does not have the expected type: expecting a Set instance but found {0}
<a href="#">GBRIE0317W</a>	The execution details retrieval function returned <null>
<a href="#">GBRIE0318E</a>	Error occurred while serializing the object by using BOM serialization
<a href="#">GBRIE0319E</a>	Invalid decision mode value {0}
<a href="#">GBRIE0320E</a>	Error occurred while serializing the observed value for test "{0}"
<a href="#">GBRIE032</a>	Error occurred while serializing the expected values for test "{0}"

<a href="#">1E</a>	
<a href="#">GBRI E032 2E</a>	Error occurred while setting the decimal scale: {0} does not represent a valid decimal scale
<a href="#">GBRI E032 3E</a>	Invalid scenario suite descriptor: should include the engine type that corresponds to the used ruleset archive or ruleset path
<a href="#">GBRI E032 4E</a>	No engine type
<a href="#">GBRI E032 5E</a>	No test identifier
<a href="#">GBRI E032 6E</a>	Unable to retrieve the ruleset signature for ruleset path {0}
<a href="#">GBRI E032 7E</a>	Unable to retrieve the ruleset signature in the Excel scenario file
<a href="#">GBRI E032 8E</a>	The ruleset signature in the Excel scenario file is not compatible with the ruleset signature of the deployed ruleset. The ruleset signature in the Excel scenario file is {0}, but the ruleset signature of the deployed ruleset is {1}
<a href="#">GBRI E032 9E</a>	Error occurred while validating the object factory for the following ruleset signature: {0}
<a href="#">GBRI E033 0W</a>	Error occurred while trying to dump scenario input data for debug purposes.
<a href="#">GBRI E033 1I</a>	The DEBUG mode is enabled. Please do not use it for production.
<a href="#">GBRI E033 2I</a>	DEBUG mode: initialization properties: {0}
<a href="#">GBRI E033 3E</a>	DEBUG mode: failed to instantiate a valid XOM repository DAO instance by using the application's configuration: {0}
<a href="#">GBRI E033 4E</a>	DEBUG mode: failed to instantiate a valid XOM repository DAO instance by using the default DAO configuration: {0}. Other methods may be used to dump XOM files.
<a href="#">GBRI E033 5I</a>	DEBUG mode: successfully instantiated a valid XOM repository DAO instance.
<a href="#">GBRI E033 6I</a>	The DEBUG mode is disabled.
<a href="#">GBRI E033 7I</a>	DEBUG mode: cannot compress the folder because no file was found.
<a href="#">GBRI E033 8E</a>	DEBUG mode: an error occurred while compressing the folder.
<a href="#">GBRI E033 9E</a>	DEBUG mode: successfully completed DEBUG session {0}.
<a href="#">GBRI E034 0W</a>	BOM serialization post-processing may not complete correctly : unable to find reference node {0}
<a href="#">GBRI E034 1E</a>	Error occurred during BOM serialization post-processing
<a href="#">GBRI E040 1E</a>	Initialization failed

<a href="#">GBRI E040 2I</a>	Properties used for initialization: {0}
<a href="#">GBRI E040 3I</a>	Logging started. Decision Runner Service version: {0}
<a href="#">GBRI E040 4E</a>	The API version "{0}" specified in the request is unknown to the server. The request cannot be processed.
<a href="#">GBRI E040 5W</a>	An unknown http request is routed to the Decision Runner Rest API: {0}
<a href="#">GBRI E040 6E</a>	The content type "{0}" in the request {1} is not supported
<a href="#">GBRI E040 7E</a>	Unable to serialize an internal server error. Refer to the server log for more information.
<a href="#">GBRI E040 8E</a>	An internal error occurred in Decision Batch Runner: {0}
<a href="#">GBRI E040 9E</a>	Unable to serialize object for {0}
<a href="#">GBRI E041 0E</a>	Unable to deserialize the job descriptor.
<a href="#">GBRI E041 1E</a>	An error occurred when constructing a scenario suite descriptor from the multipart request.
<a href="#">GBRI E041 2I</a>	Servlet request parameter {0} set to {1}.
<a href="#">GBRI E060 1E</a>	Persistence check failed. Diagnostic report: {0}
<a href="#">GBRI E060 2I</a>	Diagnostic Report "{0}".
<a href="#">GBRI E060 3E</a>	Cannot retrieve in database the job execution ID: {0}.
<a href="#">GBRI E060 4E</a>	Cannot find the job instance that is associated with the job execution ID: {0}.
<a href="#">GBRI E060 5E</a>	Persistence initialization failed.
<a href="#">GBRI E060 6E</a>	Decision Runner does not support the {0} persistence type.
<a href="#">GBRI E060 7E</a>	Cannot find job parameters that are associated with the job execution ID: {0}.
<a href="#">GBRI E060 8E</a>	An error occurred while creating the schema for the persistence.
<a href="#">GBRI E070 1E</a>	JNDI lookup on the data source "{0}" failed.
<a href="#">GBRI E070 2E</a>	Failed to execute SQL statement "{0}".

<a href="#">GBRI E070 3E</a>	Cannot load elements by the job execution ID: "{0}".
<a href="#">GBRI E070 4E</a>	Cannot create the JDBC driver "{1}" from class loader "{0}" with the URL "{2}" and properties {3}.
<a href="#">GBRI E070 5E</a>	Cannot create the job instance with name "{0}" and apptag "{1}".
<a href="#">GBRI E070 6E</a>	Cannot delete the job data for the job execution ID: {0}.
<a href="#">GBRI E070 7E</a>	Cannot create the job execution.
<a href="#">GBRI E070 8E</a>	Cannot create the step execution instance data.
<a href="#">GBRI E070 9E</a>	Cannot retrieve the most recent step executions for the job instance ID: {0}.
<a href="#">GBRI E071 0E</a>	Cannot retrieve the running executions for the job with name: {0}.
<a href="#">GBRI E071 1E</a>	Cannot retrieve the job status for the job execution with ID: {0}.
<a href="#">GBRI E071 2E</a>	Cannot create the sub job instance with name "{0}" and apptag "{1}".
<a href="#">GBRI E071 3E</a>	Cannot retrieve job execution with the given job execution ID: {0}.
<a href="#">GBRI E071 4E</a>	Cannot retrieve job executions with the given job execution ID: {0}.
<a href="#">GBRI E071 5E</a>	Cannot retrieve scenario suite element with the given element ID: {0}.
<a href="#">GBRI E071 6E</a>	Cannot retrieve step executions with the given job execution ID: {0}.
<a href="#">GBRI E071 7E</a>	Error occurred when decrypting password.
<a href="#">GBRI E071 8E</a>	Cannot create the job status with the given job instance ID: {0}.
<a href="#">GBRI E071 9E</a>	Cannot create the step status with the given step execution ID: {0}.
<a href="#">GBRI E072 0E</a>	Cannot retrieve the IDs of the sub job instances with the given job execution ID: {0}.
<a href="#">GBRI E072 1E</a>	Cannot count a job instance with the given job name: {0}.
<a href="#">GBRI E072 2E</a>	Cannot count a job instance with the given job name: "{0}" and apptag: "{1}".
<a href="#">GBRI E072 3E</a>	Cannot retrieve the job instance IDs with the given job name: "{0}" and apptag: "{1}".

<a href="#">GBRI E072 4E</a>	Cannot update the checkpoint data with the given key: {0}.
<a href="#">GBRI E072 5E</a>	Cannot retrieve the step status with the given job instance ID: "{0}" and step name: "{1}".
<a href="#">GBRI E072 6E</a>	Cannot retrieve the external job instance data.
<a href="#">GBRI E072 7E</a>	Cannot retrieve the job current tag with the given job instance ID: {0}.
<a href="#">GBRI E072 8E</a>	Cannot retrieve the parameters with the given job execution ID: {0}.
<a href="#">GBRI E072 9E</a>	Cannot retrieve the job status with the given job instance ID: {0}.
<a href="#">GBRI E073 0E</a>	Cannot retrieve the checkpoint data with the given key: {0}.
<a href="#">GBRI E073 1E</a>	Cannot create checkpoint data with the given key: {0}.
<a href="#">GBRI E073 2E</a>	Cannot retrieve the most recent execution ID with the given job instance ID: {0}.
<a href="#">GBRI E073 3E</a>	Cannot retrieve the batch status with the given key: {0}.
<a href="#">GBRI E073 4E</a>	Cannot mark the started job with the given key: {0} and timestamp: {1}.
<a href="#">GBRI E073 5E</a>	Cannot load all the elements.
<a href="#">GBRI E073 6E</a>	Cannot count elements.
<a href="#">GBRI E073 7E</a>	Cannot load elements with pagination.
<a href="#">GBRI E073 8E</a>	Cannot insert the scenario suite elements list.
<a href="#">GBRI E073 9E</a>	Cannot update the scenario suite element list.
<a href="#">GBRI E074 0E</a>	Cannot remove the scenario suite element list.
<a href="#">GBRI E074 1E</a>	Cannot purge jBatch data.
<a href="#">GBRI E074 2E</a>	The database "{0}" is not supported.
<a href="#">GBRI E074 3E</a>	The used database is not supported.
<a href="#">GBRI E074 4E</a>	No SQL script file found for the given schema name : {0}.
<a href="#">GBRI E074 5E</a>	Cannot update the job status with the given job instance ID: {0}.

<a href="#">GBRI E074 5E</a>	Cannot update the job status with the given job instance ID: {0}.
<a href="#">GBRI E074 6E</a>	Cannot update the step status with the given step execution ID: {0}.
<a href="#">GBRI E074 7E</a>	Cannot update the step execution with the given step execution ID: {0}.
<a href="#">GBRI E074 8E</a>	Cannot update the batch status with the given key: {0}.
<a href="#">GBRI E074 9E</a>	Cannot update the final execution status with the given key: {0}.
<a href="#">GBRI E075 0E</a>	Cannot retrieve the job instance IDs with the given job name: "{0}".
<a href="#">GBRI E075 1E</a>	Cannot retrieve the exit status with the given key: {0}.
<a href="#">GBRI E075 2E</a>	Cannot retrieve the timestamp with the given key: {0} and type: {1}.
<a href="#">GBRI E075 3E</a>	Cannot retrieve the tag with the given job execution ID: {0}.
<a href="#">GBRI E075 4E</a>	Cannot retrieve the step execution with the given step execution ID: {0}.
<a href="#">GBRI E080 1E</a>	{0} is not a valid Decision Runner URL. CDIHttpClient cannot be created.
<a href="#">GBRI E080 2E</a>	Unable to serialize object for URL {0}.
<a href="#">GBRI E080 3E</a>	Unable to deserialize object for URL {0}.
<a href="#">GBRI E080 4E</a>	An internal error occurred: {0} However an exception prevents the deserializing of this error.
<a href="#">GBRI E080 5E</a>	An error occurred. The server returned HTTP status {0}: {1}.
<a href="#">GBRI E080 6E</a>	An error occurred when connecting to the server.
<a href="#">GBRI E080 7E</a>	An error occurred when getting the response body.



# Decision Center Messages - Messages pertaining to the model and the business object model (BOM)

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRMI0001E</a>	Unknown attribute name
<a href="#">GBRMI0002E</a>	unknown component property name
<a href="#">GBRMI0003E</a>	Unknown type when creating the property {0}
<a href="#">GBRMI0004E</a>	Parameter declaration is too late
<a href="#">GBRMI0005E</a>	unknown method name
<a href="#">GBRMI0006E</a>	Wrong domain definition
<a href="#">GBRMI0007E</a>	Unexpected domain outside domain intersection or collection domain
<a href="#">GBRMI0008E</a>	Unexpected modifier {0}
<a href="#">GBRMI0009E</a>	invalid super class {0}
<a href="#">GBRMI0010E</a>	invalid exception class {0}
<a href="#">GBRMI0011E</a>	Type {0} cannot be changed: the metaclass is not an IlrMutableClass
<a href="#">GBRMI0012E</a>	The type of attribute {0} is unknown when trying to read the default value
<a href="#">GBRMI0013E</a>	Unknown type when creating the attribute {0}
<a href="#">GBRMI0014E</a>	Unknown return type when creating the indexed property {0}
<a href="#">GBRMI0015E</a>	Unknown return type when creating the method {0}
<a href="#">GBRMI0016E</a>	Cannot change the type of the member {0} to {1}
<a href="#">GBRMI0017E</a>	{0} errors found while parsing object model
<a href="#">GBRMI0018E</a>	Method {0} not found for indexed component property {1}
<a href="#">GBRMI0019E</a>	Method {0} not found for component property {1}
<a href="#">GBRMI0020W</a>	Cannot find formatter {0}
<a href="#">GBRMI0021W</a>	Cannot instantiate formatter {0}
<a href="#">GBRMI0022W</a>	Class {0} already exists. The new declaration is ignored.
<a href="#">GBRMI0023W</a>	Modifier {0} specified more than once
<a href="#">GBRMI0024E</a>	Cannot not bind non generic type {0}
<a href="#">GBRMI0025E</a>	Cannot not bind generic type {0} with parameters {1}
<a href="#">GBRMI0026E</a>	Cannot bind {0} which is not a class
<a href="#">GBRMI0027E</a>	Cannot include {0}: file not found
<a href="#">GBRMI0028E</a>	Cannot include {0}: I/O Exception {1}
<a href="#">GBRMI0029E</a>	Mismatch in properties
<a href="#">GBRMI0030E</a>	Unresolved type {0}
<a href="#">GBRMI0031E</a>	identifier expected, found "{0}"
<a href="#">GBRMI0032E</a>	String literal expected, found "{0}"



<a href="#">GBRMI0033E</a>	Word expected, found "{0}"
<a href="#">GBRMI0034E</a>	End of file expected, found "{0}"
<a href="#">GBRMI0035E</a>	integer value expected, found "{0}"
<a href="#">GBRMI0036E</a>	Symbol "{0}" expected, found "{1}"
<a href="#">GBRMI0037E</a>	Keyword "{0}" expected, found "{1}"
<a href="#">GBRMI0038E</a>	Word "{0}" expected, found {1}
<a href="#">GBRMI0039E</a>	Expected "class" or "interface" expected, found "{0}"
<a href="#">GBRMI0040E</a>	Class name expected, found primitive type {0}
<a href="#">GBRMI0041E</a>	Minimal cardinality has to be more than 0. It cannot be {0}
<a href="#">GBRMI0042E</a>	invalid destructor declaration
<a href="#">GBRMI0043E</a>	An operator has to be a method
<a href="#">GBRMM0001E</a>	No BOM found
<a href="#">GBRMM0002E</a>	Cannot migrate type {0}
<a href="#">GBRMM0003E</a>	Cannot migrate attribute {0}
<a href="#">GBRMM0004E</a>	Cannot migrate constructor {0}
<a href="#">GBRMM0005E</a>	Cannot migrate method {0}
<a href="#">GBRMM0006E</a>	Cannot migrate static reference "{0}": cannot find the corresponding attribute
<a href="#">GBRMM0007W</a>	Enum "{0}" must not contain a public constructor
<a href="#">GBRMM0008W</a>	Initial value "{0}" of type "{1}" is ignored as its type is not compatible with attribute "{2}"
<a href="#">GBRMO0001J</a>	Object model
<a href="#">GBRMO0002E</a>	Loop in inheritance tree
<a href="#">GBRMO0003E</a>	LinkageError {1} while trying to load the class {0}
<a href="#">GBRMO0004E</a>	ClassNotFoundException while trying to load the class {0}
<a href="#">GBRMO0005E</a>	Cannot find attribute {0} referenced in domain
<a href="#">GBRMO0006E</a>	Referenced attribute {0} must be static
<a href="#">GBRMO0007E</a>	Referenced attribute is of type {0}, whereas type {1} is expected
<a href="#">GBRMO0008E</a>	Actual value {0} is of type {1}, whereas type {2} is expected
<a href="#">GBRMO0009E</a>	Actual value {0} is not well formatted: reading it back produces {1}
<a href="#">GBRMO0010W</a>	Class {0} is deprecated. {1}
<a href="#">GBRMO0011W</a>	Member {0} is deprecated. {1}
<a href="#">GBRMO0012W</a>	Referenced type {0} is not defined
<a href="#">GBRMO0013E</a>	Type is not known
<a href="#">GBRMO0014E</a>	Error {1} when loading members of class {0}: {2}

<a href="#">GBRMO0015</a> E	Error {1} when loading members of class {0} : {2}
<a href="#">GBRMO0016</a> E	Error {1} when loading members of class {0} : {2}
<a href="#">GBRMO0017</a> E	There are two namespaces (probably a class and a package) with the same name {0}
<a href="#">GBRMO0018</a> E	Cannot merge class "{0}"
<a href="#">GBRMO0019</a> E	Cannot find referenced attribute ""{0}" from restricted attribute "{1}"
<a href="#">GBRMO0020</a> E	Cannot create package "{0}"
<a href="#">GBRMO0021</a> E	Member "{0}" has two parameters with the same name "{1}"
<a href="#">GBRMO0023</a> E	Enum class {0} cannot inherit from class {1} which has an incompatible domain
<a href="#">GBRMO0024</a> E	The domain of enum class {0} cannot contain the attribute {1} which is not in the domain of enum class {2}
<a href="#">GBRMO0025</a> E	The BOM element "{0}" has a legacy translation property "{1}". Translation properties are not supported anymore.

# Decision Center Messages - Messages pertaining to the samples

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

I D	Description
G B R P A C O O O O O O 1 I	Starting {0} on cluster {1} ...
G B R P A C O O O O O O 2 E	No member in cluster!
G B R P A C O O O O O O 3 I	Stopping {0} on cluster {1} ...
G B R P A C O O O O O O 4 E	Application MBean is not running on server!
G B R P A C O O O O O O 5 I	Invoking synchronization for node {0} ...
G B R P A C O O O O O O O O	Checking for existence of node {0}

06	
GBBRRPUCUOIOI7E	Node not found for name {0}
GBBRRPUCUOIOI8	Checking for existence of cluster {0}
GBBRRPUCUOIOI9W	The cluster {0} already exists
GBBRRPUCUOIOIT0	Creating the ServerCluster {0}
GBBRRPUCUOIOIT1	Creating server {0} on node {1}
GBBRRPUCUOIOIT2E	ClusterMgr MBean not found for cell {0}
GBBRRPUCUOIOIT3	Synchronization done.
GBBRRP	Checking to see if server {0} is already running on node {1}

00014	
GBRPCC00015W	Server {0} already running on node {1}
GBRPCC00016I	Checking to see if server {0} is already exists on node {1}
GBRPCC00017I	Checking for the existence of a NodeSync MBean on node {0}
GBRPCC00018E	NodeSync MBean not found for name {0}
GBRPCC00019I	Creating a server {0} .....
GBRPCC00020I	Updating virtual host configuration ...
GBRPCC00021I	Create HostAlias for {0} [Host: * port: {1} ]
GB	Starting server {0}

<div>Progress bar for empty row</div>	
<div>Progress bar for Invoking start for cluster {0}</div>	Invoking start for cluster {0}
<div>Progress bar for Error during start, may be already started!</div>	Error during start, may be already started!
<div>Progress bar for Stopping {0} ...</div>	Stopping {0} ...
<div>Progress bar for Removing {0} ...</div>	Removing {0} ...
<div>Progress bar for {0} not found on node {1}</div>	{0} not found on node {1}
<div>Progress bar for The key {0} is not correctly set in your properties file</div>	The key {0} is not correctly set in your properties file
<div>Progress bar for The cluster is up and running!</div>	The cluster is up and running!



6/1/2025 10:00:00 AM	
6/1/2025 10:00:00 AM	{0} nodeagent is now available
6/1/2025 10:00:00 AM	Unable to locate jrules-mbean-descriptors.jar in ODM home folder
6/1/2025 10:00:00 AM	Checking SDK version for node {0}
6/1/2025 10:00:00 AM	The SDK version {0} for node {1} is not compatible with ODM
6/1/2025 10:00:00 AM	The node {0} is configured with SDK versions {1}
6/1/2025 10:00:00 AM	The environment variable JAVA_HOME is not set. You must set this environment variable before running the script.
6/1/2025 10:00:00 AM	Creating a dedicated profile under {0}.
6/1/2025 10:00:00 AM	The profile under {0} is being deleted.



3	
4	Cell name: {0}
5	Node name: {0}
6	Server name: {0}
7	This is expected to take a while.
8	Restarting the server to finalize the configuration.
9	The server is started.
10	The server has stopped.
11	Application Manager was not found for cell {0} node {1} and server {2}.

<a href="#">B R P S O O I 1 1 E</a>	
<a href="#">G B R P S O O I 2</a>	Activation was successful for {0}.
<a href="#">G B R P S O O I 3</a>	Deploying {0}.
<a href="#">G B R P S O O I 4</a>	Undeploying {0}.
<a href="#">G B R P S O O I 5</a>	Updating {0}.
<a href="#">G B R P S O O I 6 W</a>	Failed to stop {0}.
<a href="#">G B R P S O O I 7</a>	{0} was successfully installed.
<a href="#">G B R P S O O I 8</a>	Removing {0}.

<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">G</a>	Creating {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">W</a>	{0} was not found.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a>	Enabling Java2 security.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">E</a>	The error {1} occurred while activating {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">E</a>	An error occurred while saving the configuration {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">E</a>	{0} is an unknown application.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">H</a> <a href="#">I</a> <a href="#">E</a>	A profile {0} already exists under {1}. A new product offering installation has been detected. You can recreate the sample server by running ODM_HOME/shared/tools/ant/bin/ant recreateserver in the ODM_HOME/shared/bin directory. Recreating the sample server deletes the existing profile and recreates a new profile. If you do not want to delete the existing profile, you can update the "profile.name" property in <InstallDir>/shared/samplesServer/was/build.properties, restarting the server and creating a new profile with the updated name.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">I</a>	The property was.home is not set in the file <Installation_Dir>/shared/samplesServer/was/build.properties.

S O O 2 6 E	
G B R P I S O O 2 7 E	You must have write access on {0} to proceed. Depending on your operating system, you might be able to run the command in elevated mode to access the directory and the directory contents.
G B R P I S O O 2 8 E	Checking {0} has failed with the message {1}.
G B R P I S O O 2 9 E	{0} is completed.
G B R P I S O O 3 0 E	A profile {0} already exists under {1}. Remove the existing profile or use a different profile name.
G B R P I S O O 3 1 E	Decision Server Rules is not installed on {0}. Unable to create a stand-alone profile for Decision Server Rules.
G B R P I S O O 3 2 E	Decision Server is not installed under {0}. Unable to create a standalone profile for Decision Server.
G B R P I S O O 3 3 E	Decision Center is not installed in {0}. Unable to create a stand-alone profile for Decision Center.

3 E	
G B R A S I L I A N E	You are upgrading, you have to destroy your profile and re-launch.
G B R A S I L I A N E	Your profile under {0} still contains logs but has been removed from WAS. Please clean completely the directory an re-launch.
G B R A S I L I A N E	Make sure to use an IBM WebSphere server where the SDK Java Technology Edition version {0} or later is installed.
G B R A S I L I A N E	You are upgrading, you have to destroy your server under {0} and re-launch.
G B R A S I L I A N E	Creating a dedicated profile under {0}.
G B R A S I L I A N E	No WLP ant task can be found in {0}. Check the wlp.home set in {1} is correct.
G B R A S I L I A N E	The port {0} on host {1} seems to be already in use. To change it, you have to update both {2} and {3}
G	Adding users to monitor administrative group

BBBBHHHH OOOO II	
GBBBHHHH OOOO 2W	User already mapped to monitor administrative group!
GBBBHHHH OOOO 3W	{0} authorization group does not exist!
GBBBHHHH OOOO 4W	No authorisation group for {0}
GBBBHHHH OOOO 5I	Remove user {0} from role {1} in authorization group {2}
GBBBHHHH OOOO 6I	Creating {0} on node {1}
GBBBHHHH OOOO 7I	{0} created
GBBBHHHH OOOO	JDBC Provider already exists!

<a href="#">8</a> <a href="#">W</a>	
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">9</a> <a href="#">I</a>	Creating datasource with JDBCProvider {0}
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">O</a> <a href="#">I</a>	Deleting {0} ...
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">1</a> <a href="#">W</a>	{0} does not exist!
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">1</a> <a href="#">2</a> <a href="#">W</a>	{0} already exists!
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">3</a> <a href="#">I</a>	Installing {0} ...
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">4</a> <a href="#">I</a>	Uninstalling {0} ...
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">T</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">O</a> <a href="#">1</a> <a href="#">5</a> <a href="#">W</a>	Unable to start Decison applications, may be already started
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a>	XU already deployed, doing nothing XU related

P T T O O 1 6 W	
G B R P T T O O 1 7 I	Install resource adapter on the node: {0}
G B R P T T O O 1 8 I	Create a J2CConnectionFactory named {0}
G B R P T T O O 1 9 I	Start application {0} on server {1} ...
G B R P T T O O 2 0 I	Stop application {0} on server {1} ...
G B R P T T O O 2 1 W	Unable to stop Decison applications, may be already stopped
G B R P T T O O 2 2 W	The application {0} is not installed
G B R P T T O O 2 3 I	Uninstalling {0} ...



<div>G B R P T I O O 2 4 W</div>	Unable to start {0}, may be already started
<div>G B R P T I O O 2 5 W</div>	Unable to stop {0}, may be already stopped
<div>G B R P T I O O 2 6</div>	Loading properties file name {0}
<div>G B R P T I O O 2 7 E</div>	Unsupported database {0}
<div>G B R P T I O O 2 8</div>	Jython scripts are in {0}
<div>G B R P T I O O 2 9</div>	Installing DecisionCenter enterprise application
<div>G B R P T I O O 3 0 E</div>	An error occur during the DecisionCenter enterprise application installation
<div>G B R P T I O O</div>	This script does not contain shared functions

031E	
GBRPT000321	Starting server JVM configuration script on node: {0} and server: {1}
GBRPT00033E	An error occur during the server JVM configuration
GBRPT00034E	Bad number of argument (must be {0})
GBRPT000351	Installing DecisionCenter enterprise application on cluster
GBRPT000361	Creating default users for DecisionCenter: rtsAdmin, rtsConfig, rtsUser1
GBRPT00037E	An error occur the default DecisionCenter users creation
GBRPT000381	Configuring DecisionCenter JDBC datasource
GB	Standalone server datasource configuration

G B R P T H I O O L U M N	
G B R P T H I O O L U M N	Cluster datasource configuration
G B R P T H I O O L U M N	An error occur during the JDBC datasource creation
G B R P T H I O O L U M N	Processing sanity check before augmentation
G B R P T H I O O L U M N	The specified ODM_HOME looks correct
G B R P T H I O O L U M N	The specified ODM_HOME does not look correct, unable to find {0}
G B R P T H I O O L U M N	An error occur during sanity checks
G B R P T H I O O L U M N	The specified file does not exist!

<a href="#">7</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">4</a> <a href="#">8</a>	Do not check databaseConfigFile, we are in a management profile augmentation
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">4</a> <a href="#">9</a>	Enabling application security
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">0</a> <a href="#">5</a> <a href="#">0</a> <a href="#">F</a>	An error occur during the application security settings
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">1</a>	Deploying the Rule Execution Servcer Mbean descriptor
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">2</a>	An error occur during the Rule Execution Servcer Mbean descriptor deployment
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">3</a>	Installing DecisionServer eXecution Unit JCA adapter
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">4</a> <a href="#">E</a>	An error occur during the DecisionServer eXecution Unit JCA adapter installation
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a>	Installing DecisionServer HTDS enterprise application

00551	
GBRPT00056E	An error occur during the DecisionCenter enterprise application installation
GBRPT00057E	An error occur during the DecisionServer console installation
GBRPT00058I	Uploading {0} on node {1}
GBRPT00059I	Installing eXecution Unit on node {0}

# Decision Center Messages - Messages pertaining to testing and simulation

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	An error has been detected. Contact your Administrator as this error typically occurs when trying to retrieve the signature of the ruleset. The XOM might not be correctly deployed to the SSP or to Rule Execution Server. Check the error trace for more detailed information.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	The simulation part definition contains an error. A scenario suite part is defined with a first index {0} and last index {1}. The first index must be a positive value and the last index must be greater than or equal to the first index. Contact your Administrator to solve the issue with the scenario provider implementation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">03</a> <a href="#">E</a>	The simulation part definition contains an error. A balanced list of scenario part is defined using a total number of scenario of {0} and a number of parts of {1}. The total number of scenarios must be greater than 0 and the number of parts must be less than or equal to the total number of scenarios. Contact your Administrator to solve the issue with the scenario provider implementation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">04</a> <a href="#">E</a>	SSP has stopped the execution of the scenario suite because no resources were available on the server. Contact your Administrator to add more resources to the work managers used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">05</a> <a href="#">E</a>	An error has been detected in one of the work managers used by the SSP. Contact your Administrator to fix the problem.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">06</a> <a href="#">E</a>	The execution of the scenario suite has been stopped by the SSP, because there were no resources available on the server. Contact your Administrator to add more resources to the thread pool used by the SSP or wait for the resources to be available before running your scenario suite again. A resource is made available again when the execution of a scenario suite is finished.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">07</a> <a href="#">I</a>	The resource allocation policy manager has rejected the execution request.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">00</a> <a href="#">08</a> <a href="#">I</a>	The execution request has been accepted by the resource allocation policy manager. The resource allocation policy for this job is: Execution Priority {0}, Maximum number of threads to use for parallel execution {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a>	Decision Engine is not supported for the supplied scenario provider "{0}"

<a href="#">0009E</a>	
<a href="#">G BR TE 02 01 E</a>	Cannot convert string value {0} to an instance of {1}
<a href="#">G BR TE 02 02 E</a>	Cannot parse XML string {0}
<a href="#">G BR TE 02 03 E</a>	Cannot compute equality. There are too many possible matching values because too many expected objects have no value provided.
<a href="#">G BR TE 02 04 E</a>	Cannot cast object {0} to an instance of {1}
<a href="#">G BR TE 02 05 E</a>	Cannot parse the Gregorian calendar {0}. The <time> XML element is missing.
<a href="#">G BR TE 02 06 E</a>	Cannot compare complex object {0} directly. This object should be compared field by field.
<a href="#">G BR TE 02 07 E</a>	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
<a href="#">G BR TE 02 08 W</a>	No converter defined for type: "{0}". The toString() method is used to compare the expected value "{1}" with the observed value "{2}".
<a href="#">G BR TE 02 09 W</a>	The equals method is used to compare object {0}.
<a href="#">G BR TE 02 10 W</a>	Cannot compare object {0} using BOM serialization because fields {1} cannot not be found in this object.
<a href="#">G BR TE 02 11 W</a>	Ensure that the equals method is correctly implemented for this object.
<a href="#">G</a>	Cannot compare object {0} using BOM serialization because all the corresponding expected fields are empty in the

<a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">12</a> <a href="#">E</a>	Excel scenario file.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">13</a> <a href="#">W</a>	Object {0} is not compared because all the corresponding expected fields are empty in the Excel scenario file.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">02</a> <a href="#">14</a> <a href="#">E</a>	Cannot serialize object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">01</a> <a href="#">I</a>	GET request handled...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">02</a> <a href="#">I</a>	Logout requested, invalidating the session
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">03</a> <a href="#">I</a>	PING requested
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">04</a> <a href="#">I</a>	POST request handled, deserializing the request object...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">05</a> <a href="#">I</a>	ServiceCall deserialized from input stream, starting the service...
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">06</a> <a href="#">I</a>	Service method called
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">07</a> <a href="#">I</a>	Writing result {0} to the response stream
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">08</a> <a href="#">I</a>	OK, result sent, response stream flushed
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">09</a>	SSP: No value defined in web.xml for parameter {0}: using default value {1}



<a href="#">W</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">10</a> <a href="#">I</a>	SSP: Using {0} rule sessions
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">11</a> <a href="#">E</a>	SSP: Unable to initialize Trace DAO
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">12</a> <a href="#">I</a>	SSP: Successfully initialized the trace DAO factory: {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">13</a> <a href="#">E</a>	SSP: Unable to instantiate job results store from class: {0}. The in-memory store is used instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">14</a> <a href="#">I</a>	SSP: Using the job results store: {0} instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">15</a> <a href="#">W</a>	Value of parameter {0} in web.xml is not a valid integer. Using default value {1}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">16</a> <a href="#">I</a>	SSP: Using {0} as the custom policy manager for allocation of resources
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">17</a> <a href="#">W</a>	SSP: An exception occurred while creating a new instance of the custom policy manager {0} for allocation of scenario suite resources. The default allocation policy manager is used instead.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">18</a> <a href="#">I</a>	SSP: The default policy manager for resource allocation is used with a setting of {0} for the maximum number of threads per parallel simulation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">19</a> <a href="#">I</a>	SSP pool size: {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a> <a href="#">06</a> <a href="#">20</a> <a href="#">I</a>	SSP: High-priority work manager name : {0}
<a href="#">G</a> <a href="#">BR</a> <a href="#">TE</a>	SSP: Normal-priority work manager name : {0}

<a href="#">06 21 J</a>	
<a href="#">G BR TE 06 22 J</a>	SSP: Low-priority work manager name : {0}
<a href="#">G BR TE 06 23 E</a>	SSP: Cannot retrieve the high-priority work manager {0} in the JNDI directory
<a href="#">G BR TE 06 24 E</a>	SSP: Cannot retrieve the normal-priority work manager {0} in the JNDI directory
<a href="#">G BR TE 06 25 E</a>	SSP: Cannot retrieve the low-priority work manager {0} in the JNDI directory
<a href="#">G BR TG 00 01 E</a>	Cannot find sheet {0} in Excel scenario file.
<a href="#">G BR TG 00 02 E</a>	Index out of bounds: scenario {0} requested, while valid index are from {1} to {2} for the scenario file.
<a href="#">G BR TG 00 03 E</a>	Class {0}, referenced in the headers of sheet {1}, is not defined in the business object model.
<a href="#">G BR TG 00 04 E</a>	Missing property {0} that defines the name of the sheet for input objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G BR TG 00 05 E</a>	Object "{0}" referenced in cell {1} of sheet {2} cannot be found in sheet {3}.
<a href="#">G BR TG 00 06 E</a>	Missing property {0} that defines the name of the sheet for scenarios. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G BR TG 00 07 E</a>	Missing property {0} that defines the name of the sheet for output objects of type {1}. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G BR TG 00 08 E</a>	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results.

<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">08</a> <a href="#">E</a>	Missing property {0} in the Excel sheet {1} to define the name of the sheet for expected results.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">09</a> <a href="#">E</a>	Incorrect direction ({0}) defined for parameter {1} in property {2} of the Excel sheet {3}. The list of correct directions is: {4}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">10</a> <a href="#">E</a>	The Excel scenario provider does not support rulesets that define no input parameters.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">11</a> <a href="#">E</a>	Missing property {0} that defines the locale of the scenario file. Please define the missing property in the Custom tab of the Excel file properties dialog.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">12</a> <a href="#">E</a>	{0} is not a valid name for the scenario description column: there is an input parameter defined with the same name. Please choose a different name for the scenario description column.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">13</a> <a href="#">E</a>	Header rows are missing in the sheet {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">14</a> <a href="#">E</a>	Invalid value {0} for property {1} in the Excel sheet {2} to define the type of parameter {3}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">15</a> <a href="#">E</a>	Invalid value {0} for property {1} in the Custom tab of the Excel file properties dialog to define the ruleset output (direction OUT) parameters. Valid values are comma separated lists of PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION items.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">16</a> <a href="#">E</a>	Invalid value {0} in the headers of column {1} in the Excel sheet {2}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">17</a> <a href="#">E</a>	Mandatory value missing in cell {1} of sheet {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">18</a> <a href="#">E</a>	The BOM of the ruleset to test is not suitable for generating an Excel Scenario File Template. Please check additional messages.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">19</a>	Ruleset input parameter {0}, which is an array, is not supported in flat mode.

<a href="#">E</a>	
<a href="#">G BR TG 00 20 E</a>	The data for your scenarios is not suitable for generating a flat Excel Scenario File Template. However, you may use it to generate a Tabbed Excel Scenario File Template. Please talk to your developer for assistance if this format is not currently available.
<a href="#">G BR TG 00 21 E</a>	The business object model of the ruleset to test is not suitable for generating a flat Excel Scenario File Template. However, you can use it to generate a Tabbed Excel Scenario File Template. Please check additional messages.
<a href="#">G BR TG 00 22 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}.
<a href="#">G BR TG 00 23 E</a>	Unable to create XML reader.
<a href="#">G BR TG 00 24 E</a>	The provided data must be a String representation of the XML when using BOM model type.
<a href="#">G BR TG 00 25 E</a>	The provided data must be a String representation of the XML when using XML model type.
<a href="#">G BR TG 00 26 E</a>	Unable to translate XML into BOM XML.
<a href="#">G BR TG 00 27 E</a>	Unable to find parameter in the signature.
<a href="#">G BR TG 00 28 E</a>	Ruleset Path not found.
<a href="#">G BR TG 00 29 E</a>	Unexpected exception during the BOM serialization.
<a href="#">G BR TG 00 30 E</a>	Unable to read XML.
<a href="#">G BR TG</a>	BOM type not found. Create a special converter for this type: "{0}".

<a href="#">0031E</a>	
<a href="#">G BR TG 0032E</a>	Unknown property: "{0}".
<a href="#">G BR TG 0033E</a>	Unable to find conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0034E</a>	SecurityException on the constructor for conversion class for type "{0}": "{1}"
<a href="#">G BR TG 0035E</a>	No default public constructor for conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0036E</a>	Exception during the construction of conversion class for type "{0}": "{1}".
<a href="#">G BR TG 0037E</a>	The conversion class for type "{0}" does not implement IlrBOM2DVSCConverter interface: "{1}"
<a href="#">G BR TG 0038I</a>	Additional converter loaded for type: "{0}".
<a href="#">G BR TG 0039E</a>	Unable to read inline BOM type "{0}".
<a href="#">G BR TG 0040E</a>	Missing default constructor for type "{0}".
<a href="#">G BR TG 0041E</a>	Unsupported BOM type "{0}".
<a href="#">G BR TG 0042E</a>	Field "{0}" in class "{1}" not found.

<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">43</a> <a href="#">E</a>	Ruleset input parameter {0}, which is a collection, is not supported in flat mode.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">44</a> <a href="#">E</a>	It is impossible to create scenario data because of cross references between the following objects defined in the scenario: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">45</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a byte (integer between -128 and 127) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">46</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a short (integer between -32,768 and 32,767) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">47</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: an integer (between -2,147,483,648 and 2,147,483,647) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">48</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a long (integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">49</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a single character was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">50</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a boolean value (TRUE or FALSE) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">51</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a double (decimal) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">52</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a float (decimal) was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a> <a href="#">53</a> <a href="#">E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a date was expected.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TG</a> <a href="#">00</a>	Too many domains are defined for the input data, they cannot all be stored in the scenario suite.

<a href="#">54 E</a>	
<a href="#">G BR TG 00 55 E</a>	Only domains with less than {0} entries can be stored in the Excel file.
<a href="#">G BR TG 00 56 W</a>	BOM type not found. Create a special converter for this type: "{0}".
<a href="#">G BR TG 00 57 W</a>	Unsupported business object model type "{0}".
<a href="#">G BR TG 00 58 E</a>	Cannot read the Excel file: Try to open the file in Microsoft Excel. If you can open the file in Microsoft Excel but you still receive this error, please contact your Administrator.
<a href="#">G BR TG 00 59 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigDecimal (decimal) was expected.
<a href="#">G BR TG 00 60 E</a>	Invalid value {0} in cell {1} of the Excel sheet {2}: a BigInteger (integer) was expected.
<a href="#">G BR TG 00 61 E</a>	The default Excel format is not compatible with the ruleset parameter "{1}" of type {2}: it cannot be instantiated because of a cross-reference issue that involves an attribute of type {0}.
<a href="#">G BR TG 00 62 E</a>	An unexpected error occurred while checking whether the Excel Flat format is compatible with the ruleset parameter "{0}" of type {1}.
<a href="#">G BR TG 00 63 E</a>	An unexpected error occurred while invoking method "{0}" on instance "{1}".
<a href="#">G BR TG 02 01 E</a>	Exception {0} raised during the generation of a test rule: {1}.
<a href="#">G BR TG 02 02 W</a>	It was not possible to find a navigation phrase in the vocabulary for member {0} in BOM class {1}: a default verbalization is used for the Excel Scenario File.
<a href="#">G</a>	You have reached the maximum number of columns allowed in an Excel sheet.

<a href="#">BR</a> <a href="#">TR</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TR</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	Exception thrown during the business object model serialization process of the execution trace: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">01</a> <a href="#">E</a>	{0}: the construction of {1} instances is not supported by the Excel Scenario Provider for testing and simulation.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">02</a> <a href="#">E</a>	{0}: cannot find the type {1} in the business object model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">03</a> <a href="#">E</a>	{0}: cannot find a testing and simulation constructor for the business object model class. If there are no constructors, create one. If there are several constructors in the business object model class, you must specify a constructor by selecting the "Testing and simulation constructor" option in the business object model editor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">04</a> <a href="#">E</a>	{0}: It is not possible to create instances of the business object model class. There is a cross-reference problem with the testing and simulation constructor, because at least one argument references the business object model class directly or indirectly. Check the arguments of the testing and simulation constructor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">05</a> <a href="#">E</a>	{0}: there are no constructor arguments or attributes that enable the creation of instances of the business object model class. Select a testing and simulation constructor with arguments, or create attributes that are nonstatic and writable.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">06</a> <a href="#">W</a>	{0}: the arguments of the constructor have generic names (arg1, arg2, etc.). Rename the arguments by reusing the names of the corresponding attributes or with your own meaningful names.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">07</a> <a href="#">E</a>	{0}: the Excel Format does not support multidimensional arrays.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">08</a> <a href="#">E</a>	{0}: cannot define the contents of the collection. Add a domain to the collection.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">09</a> <a href="#">W</a>	{0}: cannot use the default Excel format because of a cross-reference problem with an attribute. You can use the Excel Tabbed Format.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TV</a> <a href="#">00</a> <a href="#">10</a>	{0}: input parameters that are collections are not supported because the domain of the collection cannot be defined. Replace the input parameter with JavaBeans that hold the collection in a single attribute, and define the collection domain of this attribute.



<a href="#">E</a>	
<a href="#">G BR TV 00 11 E</a>	There are no input parameters. Add at least one ruleset parameter of direction IN or IN_OUT.
<a href="#">G BR TV 00 12 W</a>	Generic collections as output ruleset parameters are not supported.
<a href="#">G BR TV 00 13 W</a>	Generic collections as attributes of a BOM class are not supported.
<a href="#">G BR TV 00 14 W</a>	No supported child attribute found for this ruleset parameter.
<a href="#">G BR TV 00 15 W</a>	No supported child attribute found for this attribute.
<a href="#">G BR TV 00 16 W</a>	Multidimensional arrays as ruleset parameters are not supported.
<a href="#">G BR TV 00 17 W</a>	Multidimensionals arrays ruleset parameter attributes are not supported.
<a href="#">G BR TV 00 18 W</a>	Ruleset parameters of type "{0}" are not supported.
<a href="#">G BR TV 00 19 W</a>	Ruleset parameter attributes of type "{0}" are not supported.
<a href="#">G BR TV 00 21 E</a>	{0}: argument "{1}" of the constructor does not correspond to any attribute in the {2} class or its superclasses. Rename the arguments by reusing the names of the corresponding attributes or create virtual attributes that correspond to the arguments in the business object model.
<a href="#">G BR TV 00 22 W</a>	{0}: is not ignored because it is not optional.
<a href="#">G BR TV</a>	Engine type "{0}" is not supported.

<a href="#">0023E</a>	
<a href="#">G BR TV 00 24 E</a>	{0}: cannot create an instance for BOM class {1} because it is an abstract class. Use a concrete class instead of an abstract class.
<a href="#">G BR TX 00 01 E</a>	Column index is out of bounds. The allowable column range is [0..{0}].
<a href="#">G BR TX 00 02 E</a>	Row index is out of bounds. The allowable row range is [0..{0}].
<a href="#">G BR TX 00 03 E</a>	Could not create a Sheet in a readable Excel file.
<a href="#">G BR TX 00 04 E</a>	Could not retrieve the drawing patriarch for the Excel Sheet.
<a href="#">G BR TX 00 05 E</a>	Unable to deserialize this BOM type descriptor: {0}.
<a href="#">G BR TX 00 05 W</a>	Cell at [{0},{1}] is unexpectedly out of range.
<a href="#">G BR TX 00 06 E</a>	Unable to create a new XOM array instance for this BOM type: {0}.
<a href="#">G BR TX 00 06 W</a>	Cell at [{0},{1}] is unexpectedly within range.
<a href="#">G BR TX 00 07 E</a>	Unable to retrieve the XOM class mapping for this BOM type: {0}.
<a href="#">G BR TX 00 08 E</a>	Unable to create the object instance for this BOM type: {0}.
<a href="#">G</a>	Unable to cast the value at index {0} in the Input Record into {1}.

<a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">09</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">10</a> <a href="#">E</a>	The Input Record does not contain a value at index: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">11</a> <a href="#">E</a>	BOM support has not been enabled or the BOM is missing for the ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">12</a> <a href="#">E</a>	Unable to retrieve the type of the array elements for this BOM type: {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">13</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is an array.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">14</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is a collection.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">15</a> <a href="#">E</a>	Unable to determine whether the BOM type {0} is a collection with a single factory parameter.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">16</a> <a href="#">E</a>	Unable to retrieve the BOM class that declares the field {0} of the BOM type {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">17</a> <a href="#">E</a>	Unable to retrieve the ruleset archive from Rule Execution Server.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">18</a> <a href="#">E</a>	Unable to parse the XML descriptor of the ruleset signature.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">19</a> <a href="#">E</a>	Unable to extract the XML descriptor of the ruleset signature from the ruleset archive.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">20</a>	Unable to serialize the ruleset parameter {0}.

<a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">21</a> <a href="#">E</a>	Unable to extract the BOM from the ruleset archive.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">22</a> <a href="#">E</a>	The BOM type for the ruleset parameter {0} is missing.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">23</a> <a href="#">E</a>	Unable to retrieve the BusinessDataIEService instance for ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">24</a> <a href="#">E</a>	Unable to convert BOM instance into XOM instance.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">25</a> <a href="#">E</a>	The BOM is missing for ruleset {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">00</a> <a href="#">26</a> <a href="#">E</a>	Unable to transform the input map into a compliant map.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">01</a> <a href="#">E</a>	Cannot create a ruleset parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">02</a> <a href="#">E</a>	Cannot extract the ruleset parameter name from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION with a string that is not empty after it has been trimmed.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">03</a> <a href="#">E</a>	Cannot extract the ruleset parameter type from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">04</a> <a href="#">E</a>	Cannot extract the ruleset parameter direction from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_DIRECTION.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">05</a> <a href="#">E</a>	Cannot create an object factory parameter from description {0}. Use a correct description: PARAMETER_NAME: [PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a>	Cannot extract the name of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL with a string that is not empty after it has been

<a href="#">TX</a> <a href="#">02</a> <a href="#">06</a> <a href="#">E</a>	trimmed.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">07</a> <a href="#">E</a>	Cannot extract the type of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">08</a> <a href="#">E</a>	Cannot extract the direction of the object factory parameter from description {0}. Use a correct description: PARAMETER_NAME:[PARAMETER_TYPE]:PARAMETER_IS_OPTIONAL.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">09</a> <a href="#">E</a>	Cannot find BOM file <{0}> in the class path.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">10</a> <a href="#">E</a>	The created object instance for class {0} must not be null.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">11</a> <a href="#">E</a>	Value "{0}" is not valid for class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">12</a> <a href="#">E</a>	Mandatory parameter "{0}" is missing for building an instance of {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">13</a> <a href="#">E</a>	There is no attribute "{0}" in the BOM class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">14</a> <a href="#">E</a>	Maps are not supported.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">15</a> <a href="#">E</a>	Value of mandatory parameter {0} for class {1} must be a string.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">16</a> <a href="#">E</a>	The object factory signature for class {0} already contains a parameter that is named {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">17</a> <a href="#">E</a>	Class {0} is not considered a simple type.

<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">18</a> <a href="#">E</a>	The node kind for class {0} must not be null.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">19</a> <a href="#">E</a>	Cannot convert date {0} to a string format for class {1}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">20</a> <a href="#">E</a>	The object factory signature for class {0} must contain only one parameter.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">21</a> <a href="#">E</a>	Cannot convert an instance of BOM class {0} into a XOM instance.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">22</a> <a href="#">E</a>	Cannot create a node instance for the XMLGregorianCalendar class.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">23</a> <a href="#">E</a>	The argument "{0}" of the constructor does not correspond to any attribute in the {1} class or its super classes. Rename the arguments by reusing the names of the corresponding attributes, or create virtual attributes corresponding to the arguments in the Business Object Model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">24</a> <a href="#">E</a>	Cannot find class {0} in the business object model.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">25</a> <a href="#">E</a>	Cannot find the default constructor for class {0} in the business object model. If there are no constructors, create one. If there are several constructors in the class, you must specify a constructor by selecting the "DVS constructor" option in the business object model editor.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">26</a> <a href="#">E</a>	Cannot get metadata for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">27</a> <a href="#">E</a>	Cannot get array type for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">28</a> <a href="#">E</a>	Cannot get object value for object {0}.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a>	Cannot create a node instance for the Calendar class using value {0}.

<a href="#">02</a> <a href="#">29</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">30</a> <a href="#">E</a>	{0} does not represent a valid Calendar value.
<a href="#">G</a> <a href="#">BR</a> <a href="#">TX</a> <a href="#">02</a> <a href="#">31</a> <a href="#">E</a>	Cannot create an instance for BOM class {0} because it is an abstract class. Use a concrete class instead of an abstract class.

## Rule Execution Server reference

Refer to these topics for predefined ruleset properties and RuleApp properties, for execution unit (XU) persistence properties and configuration properties, RuleApp management using the MBeans Accessor API, Ant tasks, public API, and guidance on the graphical user interface in Eclipse.

### [Hosted transparent decision services](#)

A hosted transparent decision service (HTDS) is an execution component that is linked to a ruleset path through a JMX management bean (MBean). Using such a transparent decision service, you can make rulesets available as web services without any code deployment.

### [Messages](#)

Each listed message provides a detailed description about the error message and some steps to try and resolve the error.



## Hosted transparent decision services

A hosted transparent decision service (HTDS) is an execution component that is linked to a ruleset path through a JMX management bean (MBean). Using such a transparent decision service, you can make rulesets available as web services without any code deployment.

Hosted transparent decision services serialize Java™ primitive and simple types to XML and return status codes to the calling application.

### XML serialization of Java types

Simple and primitive Java types are serialized to XML to generate Web Service Description Language (WSDL) or Web Application Description Language (WADL) format.

### HTTP status codes

Status codes return the results of your WSDL, WADL, or OpenAPI requests.

**Parent topic:** [Rule Execution Server reference](#)

# XML serialization of Java types

Simple and primitive Java™ types are serialized to XML to generate Web Service Description Language (WSDL) or Web Application Description Language (WADL) format.

Simple Java types are the finest-grain types from which complex Java types can be expressed. The following table shows how the first-level elements of the primitive Java types are mapped to XML types for serialization into WSDL or WADL format. You retrieve rulesets in WSDL format to present them as hosted transparent decision services. You retrieve rulesets in WADL format to execute them by using Representational State Transfer (REST) APIs.

**Restriction:** If you use Java primitive types as input values in your execution requests, make sure that each value falls within its corresponding data type range. Otherwise, you might get unexpected output.

Table 1. XML serialization of Java simple types and primitive types

Java types	XML serialization	JAXB types for deserialization
boolean java.lang.Boolean	xsd:boolean	java.lang.Boolean
byte	xsd:int	java.lang.Integer
char java.lang.Character	xsd:int	java.lang.Character
java.util.Date	xsd:dateTime	java.util.Date
java.math.BigDecimal	xsd:decimal	java.math.BigDecimal
double java.lang.Double	xsd:double	java.lang.Double
float java.lang.Float	xsd:float	java.lang.Float
int java.lang.Integer	xsd:int	java.lang.Integer
java.math.BigInteger	xsd:integer	java.math.BigInteger
long java.lang.Long	xsd:long	java.lang.Long
short java.lang.Short	xsd:short	java.lang.Short
java.lang.String	xsd:string	java.lang.String

Parent topic: [Hosted transparent decision services](#)

# HTTP status codes

Status codes return the results of your WSDL, WADL, or OpenAPI requests.

Hosted transparent decision services and REST methods return HTTP status codes to notify the calling client of the failure or success of your requests for WSDL, WADL, or OpenAPI format.

Table 1. HTTP status codes

Status code	Description
200 OK	The request completed successfully.
201 Created	The request completed successfully. A new resource was created.
204 No Content	The request completed successfully, but content is not available.
400 Bad Request	The REST request contains parameters that are not valid or the request parameters are missing.
401 Unauthorized	The caller is not authorized to do the request.
403 Forbidden	The caller is not authorized to complete the request.
404 Not Found	The requested resource does not exist.
406 Not Acceptable	The request contains an unsupported content type or content encoding.
412 Invalid syntax	The request uses an invalid regular expression in the <i>fieldname</i> parameter for collection types.
500 Internal Server Error	A problem has occurred. Additional information is provided in the stack trace.

Parent topic: [Hosted transparent decision services](#)

# Rule Execution Server Messages

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRBA</a>	Messages pertaining to BAI
<a href="#">GBRP</a>	Messages pertaining to the samples
<a href="#">GBRSE</a>	Messages pertaining to rule execution
<a href="#">GBRXC</a>	Messages pertaining to Rule Execution Server console
<a href="#">GBRXH</a>	Messages pertaining to HTDS
<a href="#">GBRXR</a>	Messages pertaining to rule session
<a href="#">GBRXU</a>	Messages pertaining to the XU
<a href="#">GBRXX</a>	Messages pertaining to the common API

# Rule Execution Server Messages - Messages pertaining to BAI

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRBA2001E</a>	An unexpected error occured when retrieving the log level from the plugin configuration: {0}.
<a href="#">GBRBA2002I</a>	The plugin property configuration file {0} was not found.
<a href="#">GBRBA2003E</a>	An unexpected error occurred while reading the plugin configuration file {0}: {1}
<a href="#">GBRBA2004E</a>	An unexpected exception occurred when logging a message: {0}.
<a href="#">GBRBA2005E</a>	An unexpected error occurred when starting the ODM emitter plugin for Business Automation Insights: {0}
<a href="#">GBRBA2006E</a>	The value {0} is not a valid value for the log level. Allowed values can be found in the documentation of the Java class {1}.
<a href="#">GBRBA2007E</a>	An unexpected error occurred when setting the log level of the ODM emitter plugin to {0}: {1}.
<a href="#">GBRBA2008I</a>	Setting the log level of the plugin to {0}.
<a href="#">GBRBA2009E</a>	An unexpected error occurred while releasing the resources used by the plugin: {0}
<a href="#">GBRBA2010E</a>	Unexpected exception when building event for ruleset {0}: {1}
<a href="#">GBRBA2011W</a>	Event emission skipped for unsupported ruleset {0}
<a href="#">GBRBA2012E</a>	Unexpected exception during the JSON serialization of the ruleset {0} input parameters: {1}
<a href="#">GBRBA2013E</a>	Unexpected exception during the JSON serialization of the ruleset {0} output parameters: {1}
<a href="#">GBRBA2014E</a>	Unexpected exception when getting execution trace for ruleset {0}: {1}
<a href="#">GBRBA2015E</a>	Unexpected exception when serializing the execution trace for ruleset {0}: {1}
<a href="#">GBRBA2016E</a>	Unexpected exception when emitting event for ruleset {0}: {1}
<a href="#">GBRBA2017E</a>	Unexpected exception during processing before execution of ruleset {0}: {1}
<a href="#">GBRBA2018E</a>	Unexpected exception during processing after execution of ruleset {0}: {1}
<a href="#">GBRBA2019E</a>	Unexpected exception during processing of failed execution of ruleset {0}: {1}
<a href="#">GBRBA2020E</a>	After execution of ruleset {0}, cannot find the data stored before the execution.
<a href="#">GBRBA2021E</a>	An unexpected error occured when reading the enabled property: {0}

# Rule Execution Server Messages - Messages pertaining to the samples

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

I D	Description
G B B R R L C U 0 0 0 0 1	Starting {0} on cluster {1} ...
G B B R R L C U 0 0 0 0 2 E	No member in cluster!
G B B R R L C U 0 0 0 0 3 I	Stopping {0} on cluster {1} ...
G B B R R L C U 0 0 0 0 4 E	Application MBean is not running on server!
G B B R R L C U 0 0 0 0 5 I	Invoking synchronization for node {0} ...
G B B R R L C U	Checking for existence of node {0}

00006	
GBRPUUOONEN	Node not found for name {0}
GBRPUUOONEN	Checking for existence of cluster {0}
GBRPUUOONEN	The cluster {0} already exists
GBRPUUOONEN	Creating the ServerCluster {0}
GBRPUUOONEN	Creating server {0} on node {1}
GBRPUUOONEN	ClusterMgr MBean not found for cell {0}
GBRPUUOONEN	Synchronization done.
GB	Checking to see if server {0} is already running on node {1}

<div>RR CCCC 114</div>	
<div>GB RR PP CCCC 000115W</div>	Server {0} already running on node {1}
<div>GB RR PP CCCC 000116</div>	Checking to see if server {0} is already exists on node {1}
<div>GB RR PP CCCC 000117</div>	Checking for the existence of a NodeSync MBean on node {0}
<div>GB RR PP CCCC 000118E</div>	NodeSync MBean not found for name {0}
<div>GB RR PP CCCC 000119</div>	Creating a server {0} .....
<div>GB RR PP CCCC 000120</div>	Updating virtual host configuration ...
<div>GB RR PP CCCC 000121</div>	Create HostAlias for {0} [Host: * port: {1} ]







<div>GRPC 6371</div>	
<div>GRPC 371</div>	{0} nodeagent is now available
<div>GRPC 881</div>	Unable to locate jrules-mbean-descriptors.jar in ODM home folder
<div>GRPC 401</div>	Checking SDK version for node {0}
<div>GRPC 411</div>	The SDK version {0} for node {1} is not compatible with ODM
<div>GRPC 421</div>	The node {0} is configured with SDK versions {1}
<div>GRPC 111</div>	The environment variable JAVA_HOME is not set. You must set this environment variable before running the script.
<div>GRPC 121</div>	Creating a dedicated profile under {0}.

<div>3</div>	The profile under {0} is being deleted.
<div>4</div>	Cell name: {0}
<div>5</div>	Node name: {0}
<div>6</div>	Server name: {0}
<div>7</div>	This is expected to take a while.
<div>8</div>	Restarting the server to finalize the configuration.
<div>9</div>	The server is started.
<div>10</div>	The server has stopped.

<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">I</a> <a href="#">E</a>	Application Manager was not found for cell {0} node {1} and server {2}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">2</a>	Activation was successful for {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">3</a>	Deploying {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">4</a>	Undeploying {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">5</a>	Updating {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">6</a> <a href="#">W</a>	Failed to stop {0}.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a> <a href="#">7</a>	{0} was successfully installed.
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">S</a> <a href="#">O</a> <a href="#">O</a> <a href="#">I</a> <a href="#">I</a>	Removing {0}.

8	
GBRPSIOI19	Creating {0}.
GBRPSIOI20W	{0} was not found.
GBRPSIOI21I	Enabling Java2 security.
GBRPSIOI22E	The error {1} occurred while activating {0}.
GBRPSIOI23E	An error occurred while saving the configuration {0}.
GBRPSIOI24E	{0} is an unknown application.
GBRPSIOI25E	A profile {0} already exists under {1}. A new product offering installation has been detected. You can recreate the sample server by running ODM_HOME/shared/tools/ant/bin/ant recreateserver in the ODM_HOME/shared/bin directory. Recreating the sample server deletes the existing profile and recreates a new profile. If you do not want to delete the existing profile, you can update the "profile.name" property in <InstallDir>/shared/samplesServer/was/build.properties, restarting the server and creating a new profile with the updated name.
GBR	The property was.home is not set in the file <Installation_Dir>/shared/samplesServer/was/build.properties.







G B B R P H I O O O I I	Adding users to monitor administrative group
G B B R P H I O O O O O N W	User already mapped to monitor administrative group!
G B B R P H I O O O O O O W	{0} authorization group does not exist!
G B B R P H I O O O O O O O W	No authorisation group for {0}
G B B R P H I O O O O O O O I	Remove user {0} from role {1} in authorization group {2}
G B B R P H I O O O O O O O I	Creating {0} on node {1}
G B B R P H I O O O O O O O I	{0} created
G B B R P H I O	JDBC Provider already exists!

<a href="#">0008W</a>	
<a href="#">GBRPTT00009I</a>	Creating datasource with JDBCProvider {0}
<a href="#">GBRPTT00010I</a>	Deleting {0} ...
<a href="#">GBRPTT00011W</a>	{0} does not exist!
<a href="#">GBRPTT00012W</a>	{0} already exists!
<a href="#">GBRPTT00013I</a>	Installing {0} ...
<a href="#">GBRPTT00014I</a>	Uninstalling {0} ...
<a href="#">GBRPTT00015W</a>	Unable to start Decison applications, may be already started
<a href="#">GB</a>	XU already deployed, doing nothing XU related

<a href="#">RRPTT00016W</a>	
<a href="#">GBRPTT0017I</a>	Install resource adapter on the node: {0}
<a href="#">GBRPTT0018I</a>	Create a J2CConnectionFactory named {0}
<a href="#">GBRPTT0019I</a>	Start application {0} on server {1} ...
<a href="#">GBRPTT0020I</a>	Stop application {0} on server {1} ...
<a href="#">GBRPTT0021W</a>	Unable to stop Decison applications, may be already stopped
<a href="#">GBRPTT0022W</a>	The application {0} is not installed
<a href="#">GBRPTT0023I</a>	Uninstalling {0} ...

<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">4</a> <a href="#">W</a>	Unable to start {0}, may be already started
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">5</a> <a href="#">W</a>	Unable to stop {0}, may be already stopped
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">6</a>	Loading properties file name {0}
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">7</a> <a href="#">E</a>	Unsupported database {0}
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">8</a>	Jython scripts are in {0}
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">2</a> <a href="#">9</a>	Installing DecisionCenter enterprise application
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">3</a> <a href="#">0</a> <a href="#">0</a> <a href="#">E</a>	An error occur during the DecisionCenter enterprise application installation
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a>	This script does not contain shared functions

<a href="#">0031E</a>	
<a href="#">GBRPT0032I</a>	Starting server JVM configuration script on node: {0} and server: {1}
<a href="#">GBRPT0033E</a>	An error occur during the server JVM configuration
<a href="#">GBRPT0034E</a>	Bad number of argument (must be {0})
<a href="#">GBRPT0035I</a>	Installing DecisionCenter enterprise application on cluster
<a href="#">GBRPT0036I</a>	Creating default users for DecisionCenter: rtsAdmin, rtsConfig, rtsUser1
<a href="#">GBRPT0037E</a>	An error occur the default DecisionCenter users creation
<a href="#">GBRPT0038I</a>	Configuring DecisionCenter JDBC datasource
<a href="#">G</a>	Standalone server datasource configuration

<a href="#">B R P T I O O 3 9</a>	
<a href="#">G B R P T I O O 4 0</a>	Cluster datasource configuration
<a href="#">G B R P T I O O 4 2 E</a>	An error occur during the JDBC datasource creation
<a href="#">G B R P T I O O 4 3</a>	Processing sanity check before augmentation
<a href="#">G B R P T I O O 4 4</a>	The specified ODM_HOME looks correct
<a href="#">G B R P T I O O 4 5 E</a>	The specified ODM_HOME does not look correct, unable to find {0}
<a href="#">G B R P T I O O 4 6 E</a>	An error occur during sanity checks
<a href="#">G B R P T I O O 4</a>	The specified file does not exist!

<a href="#">7</a> <a href="#">E</a>	
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">4</a> <a href="#">8</a>	Do not check databaseConfigFile, we are in a management profile augmentation
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">4</a> <a href="#">9</a>	Enabling application security
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">0</a> <a href="#">5</a> <a href="#">0</a> <a href="#">E</a>	An error occur during the application security settings
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">1</a>	Deploying the Rule Execution Servcer Mbean descriptor
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">2</a>	An error occur during the Rule Execution Servcer Mbean descriptor deployment
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">3</a>	Installing DecisionServer eXecution Unit JCA adapter
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a> <a href="#">O</a> <a href="#">O</a> <a href="#">5</a> <a href="#">4</a> <a href="#">E</a>	An error occur during the DecisionServer eXecution Unit JCA adapter installation
<a href="#">G</a> <a href="#">B</a> <a href="#">B</a> <a href="#">R</a> <a href="#">P</a> <a href="#">T</a> <a href="#">I</a>	Installing DecisionServer HTDS enterprise application

00551	
GBRPT000556E	An error occur during the DecisionCenter enterprise application installation
GBRPT000557E	An error occur during the DecisionServer console installation
GBRPT000558I	Uploading {0} on node {1}
GBRPT000559I	Installing eXecution Unit on node {0}



# Rule Execution Server Messages - Messages pertaining to rule execution

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRSE0001E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0002E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0003E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0004E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0005E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0006E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0007E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0008E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0009E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0010E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0011E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0012E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0013E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0014E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0015E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0016E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0017E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0018E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0019E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0020E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0021E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0022E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0023E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0024E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0025E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0026E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0027E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0028E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0029E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0030E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0031E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0032E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0033E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0034E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0035E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0036E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0037E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0038E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0039E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0040E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0041E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0042E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0043E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0044E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0045E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0046E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0047E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0048E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0049E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0050E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0051E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0052E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0053E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0054E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0055E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0056E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0057E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0058E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0061E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0064E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0065E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0066E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0067E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0068E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0069E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0070E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0071E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0072E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0073E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0074E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0075E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0076E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0077E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0078E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0079E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0080E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0081E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0082E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0083E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE0084E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0085E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0086E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0087E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0088E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0089E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0090E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0091E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0092E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0093E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0094E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0095E</a>	An unexpected error occurred while loading the RuleApp XML descriptor "{0}".
<a href="#">GBRSE0102E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0103E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0104E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0105E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0106E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0107E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0108E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0109E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0110E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0111E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0112E</a>	Unexpected exception while retrieving the engine type for ruleset archive "{0}".
<a href="#">GBRSE0117E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0118E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0119E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0120E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0121E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE0122E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE 0123E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0124E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0125E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0126E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0127E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0128E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0129E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0130E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0131E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0132E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0134E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0135E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0136E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0137E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0138E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0139E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0140E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0145E</a>	Failed to instantiate a password encryption manager. Will fall back to a non-encryption policy.
<a href="#">GBRSE 0146E</a>	Failed to encrypt password. Returning the password not encrypted.
<a href="#">GBRSE 0147E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0148E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0149E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0150E</a>	Failed to decrypt password. Returning the empty password.
<a href="#">GBRSE 0153E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0154E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0155E</a>	The file which stored the login and password information could not be found. Please re-enter the login and password values for your Rule Execution Server Configurations.
<a href="#">GBRSE 0156E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0157E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0158E</a>	Unexpected exception in Rule Execution Server plug-ins

<a href="#">GBRSE 0159E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0160E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0161E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0162E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0163E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0164E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0165E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0166E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0167E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0168E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0169E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0170E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0171E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0172E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0173E</a>	Unexpected exception in Rule Execution Server plug-ins
<a href="#">GBRSE 0174E</a>	"{0}" : "{1}"
<a href="#">GBRSE 0175E</a>	The ruleset archive cannot be found.
<a href="#">GBRSE 0176E</a>	Not a valid ruleset archive.
<a href="#">GBRSE 0177E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0178E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0179E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0180E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0181E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0182E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0183E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0184E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0185E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0186E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0187E</a>	Unexpected exception in Rule Execution Server UI plug-ins.



<a href="#">GBRSE0188E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0189E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0190E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0191E</a>	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
<a href="#">GBRSE0192E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0193E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0194E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0195E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0196E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0197E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0198E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0199E</a>	An unexpected error occurred while adding the rule project "{0}" to the ruleset archives of the RuleApp project "{1}".
<a href="#">GBRSE0200E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0201E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0202E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0203E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0204E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0205E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0206E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0207E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0208E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0209E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0210E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0211E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0212E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0213E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0214E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE0215E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE0216E</a>	Unexpected exception in Rule Execution Server UI plug-ins.

<a href="#">GBRSE 0217E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0218E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0219E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0220E</a>	Unexpected exception in Rule Execution Server UI plug-ins.
<a href="#">GBRSE 0221E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0222E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0223E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0224E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0225E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0226E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0227E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0228E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0229E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0230E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0231E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0232E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0233E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0234E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0235E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0236E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0237E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0240E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0241E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0242E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0243E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0244E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0245E</a>	Unexpected exception in RES UI client plugins
<a href="#">GBRSE 0262E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE 0263E</a>	Unexpected exception in DVS plugins



<a href="#">GBRSE0264E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0265E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0266E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0267E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0268E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0269E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0270E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0271E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0272E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0273E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0274E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0275E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0276E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0277E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0278E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0279E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0280E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0281E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0282E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0283E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0284E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0285E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0286E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0287E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0288E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0289E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0290E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0291E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0292E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0293E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0294E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0295E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0296E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0297E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0298E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0299E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0300E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0301E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0302E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0303E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0304E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0305E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0306E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0307E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0308E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0309E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0310E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0311E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0312E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0313E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0314E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0315E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0316E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0317E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0318E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0319E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0320E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0321E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0322E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0323E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0324E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0325E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0326E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0327E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0328E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0329E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0330E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0331E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0332E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0333E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0334E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0335E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0336E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0337E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0338E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0339E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0340E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0341E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0342E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0343E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0344E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0345E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0346E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0347E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0348E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0349E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0350E</a>	Unexpected exception in DVS plugins

<a href="#">GBRSE0351E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0352E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0353E</a>	Ruleset cannot be generated
<a href="#">GBRSE0354E</a>	Directory cannot be created: "{0}"
<a href="#">GBRSE0355E</a>	The ruleset archive cannot be parsed, see following errors.
<a href="#">GBRSE0358E</a>	The ruleset archive cannot be parsed, verify the XOM path
<a href="#">GBRSE0359E</a>	The referenced rule project seems to be closed.
<a href="#">GBRSE0360E</a>	The file cannot be written: it is read-only.
<a href="#">GBRSE0365E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0366E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0367E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0368E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0369E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0370E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0371E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0372E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0373E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0374E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0375E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0376E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0377E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0378E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0379E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0380E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0381E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0382E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0383E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0384E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0385E</a>	Unexpected exception in DVS UI plug-ins

<a href="#">GBRSE0386E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0387E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0388E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0389E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0390E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0391E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0392E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0393E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0394E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0395E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0396E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0397E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0398E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0399E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0400E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0401E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0402E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0403E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0404E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0405E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0406E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0407E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0408E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0409E</a>	An unexpected exception occurred while accessing the project {0}.
<a href="#">GBRSE0410E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0411E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0412E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0413E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE0414E</a>	Unexpected exception in DVS UI plug-ins

<a href="#">GBRSE 0415E</a>	An unexpected error occurred while accessing the DVS project {0}.
<a href="#">GBRSE 0416E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0417E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0418E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0419E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0420E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0422E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0423E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0424E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0425E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0426E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0427E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0428E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0429E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0430E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0431E</a>	Unexpected exception in DVS UI plug-ins
<a href="#">GBRSE 0432E</a>	An exception occurred while checking decision operation "{0}" for compatibility with testing and simulation.
<a href="#">GBRSE 0434E</a>	Failed to run command
<a href="#">GBRSE 0435E</a>	Failed to run command
<a href="#">GBRSE 0436E</a>	An error occurs during the repackaging. See the log file for more information.
<a href="#">GBRSE 0437E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0438E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0439E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0440E</a>	An error occurred removing the project {0}. See log file for details.
<a href="#">GBRSE 0441E</a>	Exception while generating deployment.xml file.
<a href="#">GBRSE 0442E</a>	Unexpected exception while reading launch configuration
<a href="#">GBRSE 0443E</a>	There is no entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0444E</a>	An unexpected error occurred while accessing flags for the IType element {0} in DVS project {1}.
<a href="#">GBRSE 0445E</a>	An unexpected error occurred while retrieving the hierarchy of IType element {0} in DVS project {1}.



<a href="#">GBRSE 0446E</a>	An unexpected error occurred while retrieving the IType element {0} in DVS project {1}.
<a href="#">GBRSE 0447E</a>	An unexpected error occurred while retrieving the KPI Result classes compatible with the Decision Center renderer {0} in DVS project {1}.
<a href="#">GBRSE 0448E</a>	Exception while making .zip file for XOM deployment.
<a href="#">GBRSE 0449E</a>	The generation of the DVS KPI implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0450E</a>	An unexpected error occurred during the generation of the DVS KPI implementation classes.
<a href="#">GBRSE 0451E</a>	The generation of the DVS KPI Result Aggregator implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0452E</a>	An unexpected error occurred during the generation of the DVS KPI Result Aggregator implementation classes.
<a href="#">GBRSE 0453E</a>	The generation of the DVS Scenario Provider implementation classes was unexpectedly interrupted.
<a href="#">GBRSE 0454E</a>	An unexpected error occurred during the generation of the Scenario Provider implementation classes.
<a href="#">GBRSE 0455E</a>	An unexpected error occurred while applying the changes brought about by renaming the DVS Format file "{0}".
<a href="#">GBRSE 0457E</a>	An unexpected error occurred while synchronizing the editor of the DVS Customization "{0}".
<a href="#">GBRSE 0458E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Format file "{0}".
<a href="#">GBRSE 0459E</a>	An unexpected error occurred while applying the changes brought about by the DVS Configuration "{0}".
<a href="#">GBRSE 0460E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the DVS Configuration "{0}".
<a href="#">GBRSE 0461E</a>	An unexpected error occurred while applying the changes brought about by the deletion of the Rule project "{0}" to "{1}".
<a href="#">GBRSE 0462E</a>	An unexpected error occurred while checking the nature of the project "{0}".
<a href="#">GBRSE 0467E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE 0471E</a>	An unexpected exception occurred while checking rule project "{0}" for compatibility with testing and simulation.
<a href="#">GBRSE 0477E</a>	Errors when trying to detect the engine type of rulesets declared in RuleApp {0}.
<a href="#">GBRSE 0479E</a>	There is no deployment.xml file for rule project "{0}". To generate this file, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0480E</a>	There is no deployed XOM entry for Rule Execution Server instance "{1}" in the existing deployment.xml file of rule project "{0}". To create the missing entry, deploy the XOM from "{0}" to "{1}".
<a href="#">GBRSE 0481E</a>	The rule project that is passed to the validator for XOM deployment descriptor files is undefined (null value).
<a href="#">GBRSE 0482E</a>	An unexpected error occurred while setting up the test report data for DVS launch configuration "{0}".
<a href="#">GBRSE 0483E</a>	An unexpected error occurred upon a workspace refresh during the execution of DVS launch configuration "{0}".
<a href="#">GBRSE 0484E</a>	An unexpected error occurred while opening the DVS execution report "{0}".
<a href="#">GBRSE 0485E</a>	An unexpected error occurred while refreshing the workspace resource "{0}".
<a href="#">GBRSE 0486E</a>	An unexpected exception occurred while retrieving the signature of the ruleset archive that is located in "{0}". Please check the Error View for more information about the cause of this issue.
<a href="#">GBRSE 0487E</a>	An unexpected exception occurred while looking for decision operations in rule project "{0}".
<a href="#">GBRSE 0488E</a>	An exception occurred while checking rule project "{0}" for compatibility with testing and simulation.

<a href="#">GBRSE0489E</a>	The check for compatibility with testing and simulation was interrupted for rule project "{0}".
<a href="#">GBRSE0490E</a>	The variable "{0}" was not found in rule package "{1}".
<a href="#">GBRSE0491E</a>	The variable "{0}" that is referenced by decision operation "{1}" was not found in the default package.
<a href="#">GBRSE0492E</a>	The variable set "{0}" that is referenced by variable "{1}" in decision operation "{2}" was not found.
<a href="#">GBRSE0493E</a>	An unexpected error occurred while retrieving the JRE upon generation of Java project "{0}".
<a href="#">GBRSE0494E</a>	An unexpected exception occurred while retrieving the Decision Service property for project "{0}".
<a href="#">GBRSE0495E</a>	An unexpected exception occurred while retrieving attribute "{0}" of type "java.lang.String" for launch configuration "{0}".
<a href="#">GBRSE0496E</a>	An unexpected exception occurred while retrieving attribute "{0}" of type "boolean" for launch configuration "{0}".
<a href="#">GBRSE0497E</a>	An unexpected error occurred while checking file location "{0}", where the Excel file with DVS output values is stored, for write access.
<a href="#">GBRSE0498E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0499E</a>	Unexpected exception in DVS plugins
<a href="#">GBRSE0500E</a>	Unexpected exception in SDsExcelRunner.
<a href="#">GBRSE0501E</a>	An unexpected exception occurred when retrieving DVS input and output parameters values.
<a href="#">GBRSE0502E</a>	An unexpected exception occurred when generating the DVS output values Excel file.
<a href="#">GBRSE0503E</a>	Unexpected exception in SDsExcelRunner.
<a href="#">GBRSE0504E</a>	Unexpected exception in IlrExcelRunner.
<a href="#">GBRSE0505E</a>	Unexpected exception in IlrExcelRunner.
<a href="#">GBRSE0506E</a>	Unexpected exception in IlrDVSArchiveRunner.
<a href="#">GBRSE0507E</a>	An unexpected error occurred while exporting the RuleApp "{0}".
<a href="#">GBRSE0508E</a>	Failed to connect to Rule Execution Server at "{0}".
<a href="#">GBRSE0509E</a>	An unexpected error occurred during XOM deployment.



# Rule Execution Server Messages - Messages pertaining to Rule Execution Server console

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRX C0100 E</a>	Unsupported persistence type "{0}". Only FILE and DB are supported.
<a href="#">GBRX C0101 E</a>	Initialization failed
<a href="#">GBRX C0102 E</a>	The automatic creation of the database failed: {0}.
<a href="#">GBRX C0103 E</a>	RuleApp "{0}" was not found.
<a href="#">GBRX C0104 E</a>	The initialization of the persistence layer failed.
<a href="#">GBRX C0105 E</a>	The initialization of the management layer failed.
<a href="#">GBRX C0106 E</a>	The initialization of MBeans failed.
<a href="#">GBRX C0107 E</a>	The initialization of the model failed.
<a href="#">GBRX C0108 E</a>	Error on execution unit (XU): "{0} - {1}".
<a href="#">GBRX C0110 E</a>	The Res console logger configuration file was not found in WEB-INF/classes: "{0}".
<a href="#">GBRX C0111 E</a>	Batch service error.
<a href="#">GBRX C0113 E</a>	An error occurred when the decision service "{0}" was removed. See the log file for more information.
<a href="#">GBRX C0114 E</a>	The persistence check failed: {0}.
<a href="#">GBRX C0115 E</a>	The persistence check failed. Diagnostic report: {0}.
<a href="#">GBRX C0116 E</a>	Access denied. The user "{0}" is not in any of the following roles: "{1}".
<a href="#">GBRX</a>	Cannot find the ruleset that corresponds to the query "{0}" containing "{1}".

<a href="#">C0117 E</a>	
<a href="#">GBRX C0118 E</a>	Initialized default trace factory: {0}
<a href="#">GBRX C0119 E</a>	The initialization of the trace DAO factory failed.
<a href="#">GBRX C0120 E</a>	It was not possible to reinitialize the trace DAO configuration with this init parameter: {0}
<a href="#">GBRX C0121 E</a>	It was not possible to initialize the trace DAO configuration: {0}
<a href="#">GBRX C0123 E</a>	Could not get information from the XOM DAO.
<a href="#">GBRX C0124 E</a>	An invalid port is specified for the broker server. The port must be a number.
<a href="#">GBRX C0125 E</a>	Resource "{0}" not found.
<a href="#">GBRX C0126 E</a>	A resource with the same name and content (identical checksum) already exists in the repository.
<a href="#">GBRX C0127 E</a>	A library with the same name and version already exists in the repository.
<a href="#">GBRX C0128 E</a>	The selected file type is not allowed. Select a JAR or compressed file.
<a href="#">GBRX C0129 E</a>	This URL is already referenced by this library.
<a href="#">GBRX C0130 E</a>	This item is already referenced.
<a href="#">GBRX C0131 E</a>	Timeout expired while the execution unit {0} is contacted to run a test.
<a href="#">GBRX C0132 W</a>	This XU uses a different persistence type ({0}) than the Rule Execution Server console ({1}).
<a href="#">GBRX C0133 W</a>	This XU uses a different XOM persistence type ({0}) than the Rule Execution Server console ({1}).
<a href="#">GBRX C0134 W</a>	Invalid value {0} for property {1} in the deployment descriptor. This value should be an integer. The default value {2} is used instead.
<a href="#">GBRX C0135 I</a>	Properties used for initialization: {0}
<a href="#">GBRX C0136 I</a>	The initialization was successful. {0} RuleApps are available in the RES repository.
<a href="#">GBRX C0137 I</a>	Logging started. Rule Execution Server console version: {0}
<a href="#">GBRX C0138 I</a>	Unregister MBeans before stopping the application.
<a href="#">GBRX C0139</a>	Connection to the Rule Execution Server Console from "{0}".

<a href="#"> </a>	
<a href="#">GBRX C0140</a> <a href="#"> </a>	Rule Execution Server console log file: {0}
<a href="#">GBRX C0143</a> <a href="#"> </a>	Using the following configuration parameters: {0}
<a href="#">GBRX C0144</a> <a href="#"> </a>	The trace DAO factory was initialized: {0}
<a href="#">GBRX C0146</a> <a href="#"> </a>	XOM resource added: "{0}"
<a href="#">GBRX C0147</a> <a href="#"> </a>	XOM library added: "{0}"
<a href="#">GBRX C0148</a> <a href="#"> </a>	XOM resource removed: "{0}"
<a href="#">GBRX C0149</a> <a href="#"> </a>	XOM library removed: "{0}"
<a href="#">GBRX C0150</a> <a href="#"> </a>	... and {0} more log messages
<a href="#">GBRX C0151</a> <a href="#"> </a>	Persistence type: {0}
<a href="#">GBRX C0152</a> <a href="#"> </a>	Persistence properties: {0}
<a href="#">GBRX C0153</a> <a href="#"> </a>	Execution unit location: {0}
<a href="#">GBRX C0154</a> <a href="#"> </a>	Reading Decision Warehouse configurations from web.xml
<a href="#">GBRX C0155</a> <a href="#"> </a>	The Decision Warehouse configuration {0} was loaded.
<a href="#">GBRX C0156</a> <a href="#">E</a>	An error occurred. See the exception message for details. {0}
<a href="#">GBRX C0157</a> <a href="#">E</a>	The logger configuration file cannot be read.
<a href="#">GBRX C0158</a> <a href="#">E</a>	The property "{0}" cannot be found in the web.xml file.
<a href="#">GBRX C0159</a> <a href="#">E</a>	The security manager exists and the logging permission is not correct.
<a href="#">GBRX C0160</a> <a href="#">E</a>	An error occurred while the schema was uploaded.
<a href="#">GBRX C0161</a> <a href="#">E</a>	An error occurred while the trace schema was uploaded.
<a href="#">GBRX C0162</a> <a href="#">E</a>	An error occurred while the ruleset archive was parsed.
<a href="#">GBRX C0163</a> <a href="#">E</a>	An error occurred during upload.

<a href="#">GBRX C0164 E</a>	An error occurred while property "{0}" was updated. Error message {1}.
<a href="#">GBRX C0165 E</a>	The change to the trace DAO configuration failed: {0}.
<a href="#">GBRX C0166 E</a>	An error occurred while the tree data was built: {0}.
<a href="#">GBRX C0167 E</a>	An error occurred when the server status was set.
<a href="#">GBRX C0168 E</a>	Date "{0}" could not be parsed against pattern {1}.
<a href="#">GBRX C0169 E</a>	Unable to create RTS link: {0}.
<a href="#">GBRX C0170 E</a>	The WADL file was not found.
<a href="#">GBRX C0171 E</a>	WADL file.
<a href="#">GBRX C0172 E</a>	Time string < {0} > is not a correct time, formatted as HH:mm:ss. Ignore it.
<a href="#">GBRX C0173 E</a>	Time string < {0} > does not conform to format HH:mm:ss. Ignore it.
<a href="#">GBRX C0174 E</a>	The file handler for the Rule Execution Server console was not found.
<a href="#">GBRX C0175 I</a>	The operating system is {0} {1} {2}.
<a href="#">GBRX C0176 I</a>	The JVM is {1} {2} {0}.
<a href="#">GBRX C0177 I</a>	The class path is {0}.
<a href="#">GBRX C0178 E</a>	The logging configuration property "{0}" for the Rule Execution Server console was not found. Verify the WEB-INF/web.xml file for the console.
<a href="#">GBRX C0179 E</a>	The value of property "{0}", defined in the "{1}" file, could not be resolved.
<a href="#">GBRX C0180 W</a>	The logging file path "{0}" was not found.
<a href="#">GBRX C0181 W</a>	The property "{0}" was not found in the "{1}" file.
<a href="#">GBRX C0182 E</a>	The security token is missing or incorrect. You are potentially under a Cross-site request forgery attack. If you are trying to use the REST API, please use /apiauth root context.
<a href="#">GBRX C0183 W</a>	An unauthorized access has been detected from {0} because the security token is incorrect or the request contains an invalid referer header. The console is potentially under a Cross-site request forgery attack.
<a href="#">GBRX C0184 E</a>	An error occurred when validating the basic authentication header of a request.

<a href="#">GBRX C0185 W</a>	Decision Runner persistence initialization failed: persistence type is not supported
<a href="#">GBRX C0186 I</a>	RESMGMT {0}: Adding RuleApp.
<a href="#">GBRX C0187 I</a>	RESMGMT {1}: Changing RuleApp "{0}".
<a href="#">GBRX C0188 I</a>	RESMGMT {1}: Removing RuleApp "{0}".
<a href="#">GBRX C0189 I</a>	RESMGMT {2}: Adding RuleApp property "{1}" to "{0}".
<a href="#">GBRX C0190 I</a>	RESMGMT {2}: Changing RuleApp property "{1}" in "{0}".
<a href="#">GBRX C0191 I</a>	RESMGMT {2}: Removing RuleApp property "{1}" from "{0}".
<a href="#">GBRX C0192 I</a>	RESMGMT {1}: RuleApp "{0}" is added.
<a href="#">GBRX C0193 I</a>	RESMGMT {1}: RuleApp "{0}" is changed.
<a href="#">GBRX C0194 I</a>	RESMGMT {1}: RuleApp "{0}" is removed.
<a href="#">GBRX C0195 I</a>	RESMGMT {2}: RuleApp property "{1}" is added to "{0}".
<a href="#">GBRX C0196 I</a>	RESMGMT {2}: RuleApp property "{1}" is changed in "{0}".
<a href="#">GBRX C0197 I</a>	RESMGMT {2}: RuleApp property "{1}" is removed from "{0}".
<a href="#">GBRX C0198 I</a>	RESMGMT {1}: Adding ruleset "{0}".
<a href="#">GBRX C0199 I</a>	RESMGMT {1}: Changing ruleset "{0}".
<a href="#">GBRX C0200 I</a>	RESMGMT {1}: Removing ruleset "{0}".
<a href="#">GBRX C0201 I</a>	RESMGMT {2}: Adding ruleset property "{1}" to "{0}".
<a href="#">GBRX C0202 I</a>	RESMGMT {2}: Changing ruleset property "{1}" in "{0}".
<a href="#">GBRX C0203 I</a>	RESMGMT {2}: Removing ruleset property "{1}" from "{0}".
<a href="#">GBRX C0204 I</a>	RESMGMT {1}: Ruleset "{0}" is added.
<a href="#">GBRX C0205 I</a>	RESMGMT {1}: Ruleset "{0}" is changed.
<a href="#">GBRX</a>	RESMGMT {1}: Ruleset "{0}" is removed.

<a href="#">C0206</a> I	
<a href="#">GBRX</a> <a href="#">C0207</a> I	RESMGMT {2}: Ruleset property "{1}" is added to "{0}".
<a href="#">GBRX</a> <a href="#">C0208</a> I	RESMGMT {2}: Ruleset property "{1}" is changed in "{0}".
<a href="#">GBRX</a> <a href="#">C0209</a> I	RESMGMT {2}: Ruleset property "{1}" is removed from "{0}".
<a href="#">GBRX</a> <a href="#">C0210</a> I	RESMGMT {2}: Sending a notification "ruleset {0} is changed" from the TCPIP notification server to all execution units: {1}.
<a href="#">GBRX</a> <a href="#">C0211</a> I	RESMGMT {2}: Notification "ruleset {0} is changed" is sent from the TCPIP notification server to all execution units: {1}.
<a href="#">GBRX</a> <a href="#">C0212</a> I	AUTO CREATE: The repository table is created successfully.
<a href="#">GBRX</a> <a href="#">C0213</a> I	AUTO CREATE: The XOM repository table is created successfully.
<a href="#">GBRX</a> <a href="#">C0214</a> I	AUTO CREATE: The CDI repository table is created successfully.
<a href="#">GBRX</a> <a href="#">C0215</a> I	AUTO CREATE: The TRACE repository table is created successfully.
<a href="#">GBRX</a> <a href="#">C0216</a> E	[Unexpected server error: {0}]
<a href="#">GBRX</a> <a href="#">C0217</a> W	Role cannot be checked for the user "{0}" for HTTP request "{1}".
<a href="#">GBRX</a> <a href="#">C0218</a> W	Remote user is not set after logging in for HTTP request "{0}".
<a href="#">GBRX</a> <a href="#">C0219</a> E	No selected ruleset when downloading the ruleapp archive {0}.
<a href="#">GBRX</a> <a href="#">C0219</a> I	GBRXC0219I: User ({0}) has signed on.
<a href="#">GBRX</a> <a href="#">C0220</a> E	The initialization of the XOM persistence failed.
<a href="#">GBRX</a> <a href="#">C0221</a> E	The XOM persistence was not initialized. See the server log for more information.
<a href="#">GBRX</a> <a href="#">C0222</a> E	The file size exceeded the limit while uploading the RuleApp archive. For more information, see the server log and the product documentation.
<a href="#">GBRX</a> <a href="#">C0301</a> E	No Decision Warehouse configurations were found. Define the ilog.rules.res.trace.DATAWAREHOUSE_CONFIGURATIONS context parameter in the web.xml file.
<a href="#">GBRX</a> <a href="#">C0302</a> E	Decision Warehouse configuration {0} is missing from the web.xml file. This configuration will be ignored.
<a href="#">GBRX</a> <a href="#">C0303</a> E	Cannot extract property from token {0} in configuration {1}. Properties must be defined in the form key {2} value. Configuration {1} will be ignored.
<a href="#">GBRX</a>	No properties are defined for configuration {0} in the web.xml file. This configuration will be ignored.

<a href="#">C0304 E</a>	
<a href="#">GBRX C0306 E</a>	The default Decision Warehouse configuration {0} is not available. The first configuration defined in web.xml, {1}, is used instead.
<a href="#">GBRX C0307 E</a>	Cannot find the index file for the online help at {0}. Online help will not be available.
<a href="#">GBRX C0308 E</a>	Cannot start notification server on port {0}.
<a href="#">GBRX C0501 E</a>	No panel is currently selected.
<a href="#">GBRX C0502 E</a>	The tabbed panel name cannot be null.
<a href="#">GBRX C0503 E</a>	No parent UIForm found for the tabbed panel.
<a href="#">GBRX C0504 E</a>	The row index is invalid.
<a href="#">GBRX C0505 E</a>	The tabbed panel must be configured.
<a href="#">GBRX C0506 E</a>	The Rule Execution Server console application is not granted the required Java permission to access environment variables for configuration purpose.
<a href="#">GBRX C0507 E</a>	The Rule Execution Server console application is not granted the required Java permission to access Java system properties for configuration purpose.
<a href="#">GBRX C0508 I</a>	Retrieving messages from the execution units.
<a href="#">GBRX C0509 I</a>	Retrieving messages from the Rule Execution Server management console.
<a href="#">GBRX C0510 W</a>	The following message was logged without specifying a required level: "{0}". The INFO level will be used as default.
<a href="#">GBRX C0511 E</a>	Access denied.
<a href="#">GBRX C0512 W</a>	Logging out the user by using the web container mechanism did not succeed.
<a href="#">GBRX C0513 W</a>	Session invalidation did not succeed.
<a href="#">GBRX C0514 W</a>	Logout was requested by using a third-party provider but no redirect URL was specified. Users might not be logged out correctly.
<a href="#">GBRX C0515 E</a>	Logout by using a third-party provider did not succeed.
<a href="#">GBRX C0516 W</a>	Host name could not be identified from HTTP referer: {0}



# Rule Execution Server Messages - Messages pertaining to HTDS

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">001</a> <a href="#">E</a>	Unable to generate WSDL from ruleset {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">002</a> <a href="#">E</a>	The URL to access the web service cannot be empty.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">003</a> <a href="#">E</a>	Error when generating the ruleset parameters schema.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">004</a> <a href="#">E</a>	{0} has no XML type.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">005</a> <a href="#">E</a>	The kind of the parameter {0} is not supported. Must be XML and Simple Java types (int, short, long, float, double, Boolean, java.lang.Integer, java.lang.Short, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean, java.lang.String, java.math.BigInteger, java.math.BigDecimal, java.util.Date).
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">006</a> <a href="#">E</a>	Error when parsing XML file {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">007</a> <a href="#">E</a>	Unable to create the SAX parser.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">009</a> <a href="#">E</a>	Arrays of XML type is not supported but found parameter {0}: {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">011</a> <a href="#">E</a>	An error occurred when the WSDL file was generated, due to a JAX-B problem: {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">012</a> <a href="#">E</a>	Rule Virtual Engine archives are not supported.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">013</a> <a href="#">E</a>	Multidimensional arrays are not supported.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">014</a> <a href="#">E</a>	Unable to generate WADL from ruleset {0}.
<a href="#">GBR</a> <a href="#">XH0</a>	An error occurred when the WADL file was generated, due to a JAX-B problem: {0}.



<a href="#">015E</a>	
<a href="#">GBRXH0016E</a>	Unable to invoke interaction function name. Caused by {0}.
<a href="#">GBRXH0101E</a>	Initialization failed.
<a href="#">GBRXH0106E</a>	The requested url is not found.
<a href="#">GBRXH0107E</a>	Error when executing the ruleset {0}.
<a href="#">GBRXH0108E</a>	Error when extracting the ruleset parameter value from the request.
<a href="#">GBRXH0109E</a>	Error when writing the ruleset parameter value to the response.
<a href="#">GBRXH0110E</a>	The web service URL {0} is not correct. Verify your web service client URL.
<a href="#">GBRXH0112E</a>	The SOAP error message cannot be created.
<a href="#">GBRXH0114E</a>	A servlet parameter is not valid. The parameter {0} is not supported.
<a href="#">GBRXH0116E</a>	The MBean failed to unregister for the objectname "{0}".
<a href="#">GBRXH0117E</a>	The ruleset parameter {0} cannot be found in the ruleset signature.
<a href="#">GBRXH0118E</a>	Cannot get the information about the ruleset {0}. Make sure that you have deployed the ruleset and that the enable property is set correctly.
<a href="#">GBRXH0119E</a>	The TraceDAO factory is not created due to this exception {0}.
<a href="#">GBRXH0120E</a>	Execution using JSON format is not supported for a ruleset with XML parameters.
<a href="#">GBRXH0200E</a>	The XOM class {0} cannot be deserialized. Verify that the XOM library is set correctly.
<a href="#">GBRXH0300I</a>	Properties used for initialization: {0}.
<a href="#">GBRXH0301I</a>	Logging started. Decision service version: {0} {1} {2}.

<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">302I</a>	Logging started. Decision Service version: {0} {1} {2}
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">303I</a>	Unregistering the MBeans before stopping the application.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">304I</a>	XMLEventReader.{0}
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">310E</a>	The root element name is not valid: {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">311E</a>	The root element namespace is not valid: {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">312E</a>	Unable to generate XSDs files. Verify that WADL file is generated correctly.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">313E</a>	Exception Error. Please see the exception message for details : {0}
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">316I</a>	The operating system is {0} {1} {2}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">317I</a>	The JVM is {1} {2} {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">318I</a>	The class path is {0}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">319E</a>	The GET /rest/formats method expects the parameter 'ruleset' to be set to an actual ruleset path.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">320E</a>	Failed to register a specific date/time module onto JSON serializer.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">321E</a>	Failed to generate a sample request due to the use of type "{0}": make sure to annotate your model and reference the proper XML adapters (see product documentation for more details).
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">400E</a>	Generation of OpenAPI description files is not supported for rulesets with XML parameters.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">401E</a>	The endpoint {0} passed as a query parameter is not a valid URL.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">402E</a>	Could not resolve class for type "{0}".
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">500E</a>	The targeted ruleset could not be found, make sure to use a valid URL.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">501E</a>	An error occurred during the initialization of the execution, make sure that the ruleset matches the execution requirements.

<a href="#">GBR XH0 502 E</a>	An error occurred while interpreting the request: either it contains invalid data, or it does not correctly include the mandatory parameters, or there is a problem with your execution object model.
<a href="#">GBR XH0 503 E</a>	An unexpected error occurred while checking the request syntax.
<a href="#">GBR XH0 504 E</a>	An error occurred during the execution of the ruleset. Make sure that the request is valid, and review the ruleset and execution object model if applicable.
<a href="#">GBR XH0 505 E</a>	An error occurred when generating the WADL.
<a href="#">GBR XH0 506 E</a>	An error occurred while interpreting the request due to the use of type "{0}": make sure to annotate your model and reference the proper XML adapters (see product documentation for more details).
<a href="#">GBR XH0 507 E</a>	An error occurred during the execution of the ruleset due to the use of type "{0}": make sure to annotate your model and reference the proper XML adapters (see product documentation for more details).
<a href="#">GBR XH0 508 E</a>	{0} is not a valid value for the number of calls to reset hosted transparent decision service application's JSON mapper. The configured threshold should be a positive long value.
<a href="#">GBR XH0 509 E</a>	{0} is not a valid value for the elapsed hours to reset hosted transparent decision service application's JSON mapper. The configured threshold should be a positive double value.
<a href="#">GBR XH0 510I</a>	{0}, the current number of calls, is greater than the defined threshold ({1}): the hosted transparent decision service application will reset its JSON mapper.
<a href="#">GBR XH0 511I</a>	The hosted transparent decision service application is not configured to reset its JSON mapper.
<a href="#">GBR XH0 512I</a>	The hosted transparent decision service application is configured to reset its JSON mapper when there are more than {0} XOM Classloader instances.
<a href="#">GBR XH0 513I</a>	The hosted transparent decision service application is configured to reset its JSON mapper every {0,number,#.#####} hours.
<a href="#">GBR XH0 514I</a>	The hosted transparent decision service application is configured to reset its JSON mapper every {0} calls.
<a href="#">GBR XH0 515 E</a>	"{1}" is not a valid value for the hosted transparent decision service configuration property "{0}". A positive long value is expected.
<a href="#">GBR XH0 516I</a>	More than {0,number,#.#####} hours have passed: the hosted transparent decision service application will reset its JSON mapper.
<a href="#">GBR XH0 517 E</a>	"{1}" is not a valid value for the hosted transparent decision service configuration property "{0}". A positive double value is expected.
<a href="#">GBR XH0 518I</a>	{0}, the current number of XOM Classloader instances, is greater than the defined threshold ({1}): the hosted transparent decision service application will reset its JSON mapper.
<a href="#">GBR XH0 519 E</a>	{0} is not a valid value for the number of XOM Classloader instances to reset hosted transparent decision service application's JSON mapper. The configured threshold should be a positive long value.
<a href="#">GBR</a>	The hosted transparent decision service application is not granted the required java permission to access

<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">520</a> <a href="#">E</a>	The hosted transparent decision service application is not granted the required java permission to access environment variables for configuration purpose.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">521</a> <a href="#">E</a>	The hosted transparent decision service application is not granted the required Java permission to access Java system properties for configuration purpose.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">522</a> <a href="#">E</a>	An unexpected error occurred while retrieving the HTDS description property for the language "{0}": {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">523</a> <a href="#">E</a>	An unexpected error occurred while retrieving the HTDS title property for the language "{0}": {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">524</a> <a href="#">E</a>	An unexpected error occurred while retrieving the HTDS page title property for the language "{0}": {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">525</a> <a href="#">E</a>	An unexpected error occurred while retrieving the HTDS REST page title property for the language "{0}": {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">526</a> <a href="#">E</a>	An unexpected error occurred while retrieving the HTDS Error page title property for the language "{0}": {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">527</a> <a href="#">E</a>	Sending a REST error response to the client with HTTP code {0} and the following body: {1}.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">528</a> <a href="#">E</a>	Sending SOAP error messages to client due to the following error: {0}
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">529</a> <a href="#">E</a>	An error occurred during the execution of the ruleset. Check the ruleset, the request, and the execution object model to see if they are valid for the ruleset execution. Check if there is any mismatch between them.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">530</a> <a href="#">E</a>	The REST request sent to the The hosted transparent decision service application contains an empty JSON payload.
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">531</a> <a href="#">W</a>	Multiple environment variables can be used to initialize the hosted transparent decision service application parameter "{0}". Possible matches are: {1}. The environment variable "{2}" will be used to initialize the configuration property "{0}" with the value "{3}".
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">532</a> <a href="#">W</a>	Multiple System properties can be used to initialize the hosted transparent decision service application parameter "{0}". Possible matches are: {1}. The System property "{2}" will be used to initialize configuration property "{0}" with the value "{3}".
<a href="#">GBR</a> <a href="#">XH0</a> <a href="#">533</a> <a href="#">W</a>	Multiple configuration parameters can be used to initialize the hosted transparent decision service application parameter "{0}". Possible matches are: {1}. The configuration parameter "{2}" will be used to initialize configuration property "{0}" with the value "{3}".

# Rule Execution Server Messages - Messages pertaining to rule session

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRXR0001E</a>	Missing "{0}". You might be executing Java SE rule sessions by using a deployment artifact that is intended for Java EE rule sessions.
<a href="#">GBRXR0002E</a>	An error occurred while the rule session was called
<a href="#">GBRXR0003E</a>	An error occurred while the rule session was created.
<a href="#">GBRXR0004E</a>	An error occurred while the rule session interceptor was created or called
<a href="#">GBRXR0005E</a>	Unable to find RuleApp {0} for use by the interceptor.
<a href="#">GBRXR0006E</a>	Unable to find interceptor class {0}.
<a href="#">GBRXR0007E</a>	Unable to parse monitoring filters: {0}.
<a href="#">GBRXR0008W</a>	Session factory returned a null trace DAO.
<a href="#">GBRXR0009W</a>	Exception while saving with trace DAO: {0}. Trace might not have been saved.
<a href="#">GBRXR0010W</a>	Unable to create trace DAO: {0}.
<a href="#">GBRXR0011W</a>	BOM support is not enabled for this ruleset. In/out parameters will not be stored as BOM XML. The toString() method will be used.
<a href="#">GBRXR0012W</a>	Exception caught while trying to convert to BOM XML: {0}. The method toString() will be used to serialize in/out parameters.
<a href="#">GBRXR0013E</a>	XU JNDI name {0} does not reference an XU connection factory.
<a href="#">GBRXR0014W</a>	An exception occurred while trying to close the session: {0}.
<a href="#">GBRXR0015E</a>	The persistence type is not valid : {0}.
<a href="#">GBRXR0016W</a>	IlrJ2SESessionFactory.setOutput(PrintWriter) is deprecated, use IlrJ2SESessionFactory(PrintWriter) instead.
<a href="#">GBRXR0017E</a>	An error occurred when the trace was serialized to XML.
<a href="#">GBRXR0018I</a>	Decision Warehouse trace instrumentation: {0}.
<a href="#">GBRXR0019W</a>	Warning on session response {0}: {1}.
<a href="#">GBRXR0020I</a>	Decision Warehouse instrumentation warning: {0}.
<a href="#">GBRXR0021E</a>	An error occurred when the execution trace was saved for execution ID: {0}.

<a href="#">GBR XR002 2E</a>	An error occurred during access to the trace Data Access Object (DAO): {0}
<a href="#">GBR XR002 3E</a>	An error occurred when the process tried to save the trace.
<a href="#">GBR XR002 4E</a>	The value for the TCP/IP port is not valid : {0}.
<a href="#">GBR XR002 5E</a>	The value for the TCP/IP retry interval is not valid : {0}.
<a href="#">GBR XR002 6E</a>	"{0}": The ruleset archive cannot be null.
<a href="#">GBR XR002 7I</a>	Loading execution unit (XU) settings from the file descriptor.
<a href="#">GBR XR002 8I</a>	Loading default settings from the file : {0}.
<a href="#">GBR XR002 9I</a>	Copying configuration settings from the rule session factory to the execution unit (XU).
<a href="#">GBR XR003 0I</a>	Found user settings in file : {0}.
<a href="#">GBR XR003 1I</a>	Found default settings in file : {0}.
<a href="#">GBR XR003 2I</a>	The execution of the rule session for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 3I</a>	Setting the parameters as XOM BOM for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 4I</a>	Getting the parameters as XOM BOM for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 5I</a>	Storing the Decision Warehouse trace for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 6I</a>	Invoking the pre-interceptor for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 7I</a>	Invoking the post-interceptor for the ruleset {1} is : {0} ms.
<a href="#">GBR XR003 8I</a>	The ruleset property "{0}" is set to "true" for ruleset "{1}": execution traces will be stored in Decision Warehouse.
<a href="#">GBR XR003 9E</a>	The root cause of the previous error was: {0}

# Rule Execution Server Messages - Messages pertaining to the XU

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GBRXU0001E</a>	The interaction {0} has failed.
<a href="#">GBRXU0002E</a>	The XU resource adapter does not support: {0}.
<a href="#">GBRXU0004E</a>	Transaction error.
<a href="#">GBRXU0005E</a>	A concurrent transaction has been detected.
<a href="#">GBRXU0006E</a>	The transaction has not started.
<a href="#">GBRXU0008E</a>	It was not possible to retrieve the information about the ruleset {0}.
<a href="#">GBRXU0009E</a>	It was not possible to create the ruleset: {0}.
<a href="#">GBRXU0010E</a>	Invalid ConnectionSpec: {0}.
<a href="#">GBRXU0011E</a>	It was not possible to create the connection: {0}.
<a href="#">GBRXU0012E</a>	Invalid InteractionSpec: {0}.
<a href="#">GBRXU0013E</a>	The input or output record is invalid.
<a href="#">GBRXU0014E</a>	The parsing of the ruleset {0} failed {1}.
<a href="#">GBRXU0016E</a>	The rule engine failed.
<a href="#">GBRXU0017E</a>	It was not possible to create a resource information provider.
<a href="#">GBRXU0018E</a>	A parsing error occurred on the plug-ins property: {0}.
<a href="#">GBRXU0019E</a>	The property "pluginClass" is required for a plug-in. That property was not found in {0}.



<a href="#">GBRX U002 0E</a>	It was not possible to start plug-in {0}.
<a href="#">GBRX U002 1E</a>	It was not possible to destroy plug-in {0}.
<a href="#">GBRX U002 2E</a>	It was not possible to create the plug-in with properties {0}.
<a href="#">GBRX U002 3E</a>	An error occurred during the execution test, due to: {0}.
<a href="#">GBRX U002 6E</a>	It was not possible to start the MBean plug-in.
<a href="#">GBRX U002 7E</a>	It was not possible to destroy the MBean plug-in.
<a href="#">GBRX U002 8E</a>	It was not possible to unregister the MBean by using the MBean manager {0} with the objectname {1}.
<a href="#">GBRX U002 9E</a>	It was not possible to register the MBean by using the MBean manager {0} with the name {1}.
<a href="#">GBRX U003 3E</a>	Notification of a ruleset archive update failed for the canonical ruleset path: {0}, due to: {1}.
<a href="#">GBRX U003 6E</a>	It was not possible to close the interaction.
<a href="#">GBRX U003 7E</a>	It was not possible to close the connection.
<a href="#">GBRX U004 0E</a>	It was not possible to reconnect {0}.
<a href="#">GBRX U004 1E</a>	It was not possible to enable the rule engine trace.
<a href="#">GBRX U004 3E</a>	The connection type is invalid.
<a href="#">GBRX U004 4E</a>	The connection is closed.
<a href="#">GBRX U004 5E</a>	A problem occurred when the plug-in manager started.
<a href="#">GBRX U004 6E</a>	A problem occurred when the plug-in manager was updated.
<a href="#">GBRX U004 8E</a>	A syntax error occurred in the ruleset path {0}.
<a href="#">GBRX U004 9E</a>	It was not possible to resolve the ruleset path {0}.
<a href="#">GBRX U005 0E</a>	It was not possible to create a ruleset information provider for the persistence type {0}.
<a href="#">GBRX U005 4E</a>	A parsing error occurred on the plug-ins: a comma must separate the blocks of properties.



<a href="#">GBRX U005 5E</a>	A parsing error occurred on the plug-ins: unexpected comma at the end.
<a href="#">GBRX U005 6E</a>	A parsing error occurred on the plug-ins: the block of properties must not be empty.
<a href="#">GBRX U005 7E</a>	A parsing error occurred on the plug-ins: the block of properties must end with a closing bracket.
<a href="#">GBRX U005 8E</a>	A parsing error occurred on the plug-ins: the block of properties must start with an opening bracket.
<a href="#">GBRX U005 9E</a>	It was not possible to retrieve ruleset information: {0}.
<a href="#">GBRX U006 0E</a>	Privileged action error.
<a href="#">GBRX U007 0E</a>	The connection association failed.
<a href="#">GBRX U007 1E</a>	The rollback failed.
<a href="#">GBRX U010 0E</a>	Invalid configuration property: {0} = {1}.
<a href="#">GBRX U010 1E</a>	Invalid event mask: {0}.
<a href="#">GBRX U020 0E</a>	The default connection manager pool is full.
<a href="#">GBRX U020 1E</a>	The default connection manager failed to clean up the connection.
<a href="#">GBRX U020 2E</a>	The default connection manager failed to destroy the connection.
<a href="#">GBRX U020 3E</a>	The default connection manager failed to create the connection pool.
<a href="#">GBRX U030 0E</a>	The root element must be <connector>.
<a href="#">GBRX U030 1E</a>	A <resourceadapter> element cannot be found.
<a href="#">GBRX U030 2E</a>	A <managedconnectionfactory-class> element cannot be found or is invalid.
<a href="#">GBRX U030 3E</a>	The document is not XML valid.
<a href="#">GBRX U030 4E</a>	The configuration property name is empty or missing.
<a href="#">GBRX U030 5E</a>	The configuration property {0} is empty or has no value.
<a href="#">GBRX U030 6E</a>	The configuration property type {0} is invalid.

<a href="#">GBRX U030 7E</a>	This is not a JCA1.5 XU configuration file.
<a href="#">GBRX U030 8E</a>	It was not possible to parse the resource adapter configuration.
<a href="#">GBRX U030 9E</a>	It was not possible to find the XML element: {0}.
<a href="#">GBRX U040 0E</a>	Ruleset parameter error: {0}.
<a href="#">GBRX U040 1E</a>	The transformation of {2} (id={0}, kind={1}) into an IlrXmlObject instance failed.
<a href="#">GBRX U040 2E</a>	The creation of an IlrXmlDocumentDriver instance failed.
<a href="#">GBRX U040 3E</a>	An error occurred in the pool of XML document drivers.
<a href="#">GBRX U040 4E</a>	The IlrXmlDocumentDriver reservation has timed out.
<a href="#">GBRX U040 5E</a>	The evaluation of {0} failed: {1}.
<a href="#">GBRX U040 6E</a>	The ruleset is already being parsed.
<a href="#">GBRX U040 7E</a>	WSDL binding is not supported.
<a href="#">GBRX U040 8E</a>	The asynchronous execution of the ruleset failed.
<a href="#">GBRX U040 9E</a>	BOM support is not enabled for this ruleset.
<a href="#">GBRX U041 0E</a>	The ruleset execution trace is not activated.
<a href="#">GBRX U041 1E</a>	An error occurred during the ruleset execution.
<a href="#">GBRX U041 2E</a>	The activation of the ruleset execution trace failed.
<a href="#">GBRX U041 3E</a>	The interaction is already closed.
<a href="#">GBRX U041 4E</a>	The interaction is closed.
<a href="#">GBRX U041 5E</a>	The interaction failed to close.
<a href="#">GBRX U041 6E</a>	The XU dump failed.
<a href="#">GBRX U041 7E</a>	Failed to dump system information.
<a href="#">GBRX</a>	The license check failed.

<a href="#">U0418E</a>	
<a href="#">GBRX U0419E</a>	The creation of the ConnectionManager factory failed.
<a href="#">GBRX U0420E</a>	This interaction function name {1} already exists for the plug-in {0}.
<a href="#">GBRX U0421E</a>	An error occurred during creation of the ruleset cache.
<a href="#">GBRX U0422E</a>	An error occurred in the ruleset cache.
<a href="#">GBRX U0424E</a>	The ruleset path cannot be null.
<a href="#">GBRX U0425E</a>	The RulesetUsageMonitor property is not enabled.
<a href="#">GBRX U0426E</a>	Unknown ruleset engine {1} for {0}.
<a href="#">GBRX U0427E</a>	It was not possible to set the configuration parameters: {0}.
<a href="#">GBRX U0428E</a>	An error occurred in the pool.
<a href="#">GBRX U0429E</a>	The pool is full.
<a href="#">GBRX U0430E</a>	The pool is full and timeout ({0}) has been reached.
<a href="#">GBRX U0431I</a>	The wait timeout of the pool is set to {0}.
<a href="#">GBRX U0432W</a>	The pool property {0} is deprecated.
<a href="#">GBRX U0434E</a>	XU client error.
<a href="#">GBRX U0435E</a>	The creation of the engine manager failed.
<a href="#">GBRX U0436E</a>	The ruleset parsing process failed.
<a href="#">GBRX U0437E</a>	The operation {0} is not supported.
<a href="#">GBRX U0438E</a>	The plug-in operation {0} has returned false.
<a href="#">GBRX U0439E</a>	The XU configuration property {0} could not be accessed.
<a href="#">GBRX U0440E</a>	The XU configuration property {0} could not be set.
<a href="#">GBRX U044</a>	Invalid XU connection factory.

<a href="#">1E</a>	
<a href="#">GBRX U044 2E</a>	Transactions are not supported.
<a href="#">GBRX U044 3E</a>	It is not possible to monitor ruleset usage.
<a href="#">GBRX U044 4E</a>	Failed to solve the DNS name {0}.
<a href="#">GBRX U044 5E</a>	An error occurred in the cache for compiled archives.
<a href="#">GBRX U044 6E</a>	XU Info scheduler error.
<a href="#">GBRX U044 7E</a>	Ruleset execution not allowed.
<a href="#">GBRX U044 8E</a>	Plug-in error.
<a href="#">GBRX U044 9E</a>	Failed to create a timer.
<a href="#">GBRX U045 0E</a>	Failed to create a BOM converter.
<a href="#">GBRX U045 1E</a>	The XML document must not be empty.
<a href="#">GBRX U045 2E</a>	Ruleset parameter {0} does not exist in the signature of ruleset {1}.
<a href="#">GBRX U045 3E</a>	Error while invoking method {0} on the Engine class.
<a href="#">GBRX U045 4E</a>	Cannot invoke method on the Engine class - no method {0} with compatible signature found.
<a href="#">GBRX U045 5E</a>	The rule engine must be decision engine.
<a href="#">GBRX U045 6E</a>	Missing BusinessDataIEService.
<a href="#">GBRX U045 7E</a>	Input parameter objects must be of Node type.
<a href="#">GBRX U045 8E</a>	Cannot create BOM reflect.
<a href="#">GBRX U045 9E</a>	There is no BOM reflect in the ruleset.
<a href="#">GBRX U046 0E</a>	Removal of the execution unit (XU): You can no longer use the XU.
<a href="#">GBRX U046 1E</a>	The execution unit (XU) was already removed.
<a href="#">GBRX U046</a>	Cannot find version {0} of the decision engine runtime library.

<a href="#">2E</a>	
<a href="#">GBRX U046 3E</a>	Cannot retrieve version {0} of the decision engine runtime library.
<a href="#">GBRX U046 4E</a>	Cannot find file {0}.
<a href="#">GBRX U046 5E</a>	Invalid version format for calling the decision engine runtime library: {0}.
<a href="#">GBRX U046 6E</a>	Cannot find JAR {0}.
<a href="#">GBRX U046 7E</a>	The creation of EngineDefinition failed.
<a href="#">GBRX U046 8E</a>	The start of engine manager failed.
<a href="#">GBRX U046 9E</a>	Cannot get the version of the decision engine runtime library.
<a href="#">GBRX U047 0E</a>	The list of JARs that is required for version {0} of the decision engine runtime library is missing.
<a href="#">GBRX U047 1E</a>	Cannot retrieve the XML type of the parameter {0}.
<a href="#">GBRX U047 2E</a>	Cannot create XML document driver factory.
<a href="#">GBRX U047 3E</a>	Cannot retrieve the list of rules.
<a href="#">GBRX U047 4E</a>	Cannot create the XU EngineDefinition factory.
<a href="#">GBRX U047 5E</a>	Cannot retrieve the required decision engine runtime version for the archive.
<a href="#">GBRX U047 6E</a>	Cannot convert object to its BOM representation in XML format.
<a href="#">GBRX U047 7E</a>	Cannot convert BOM representation in XML format to XOM.
<a href="#">GBRX U047 8E</a>	Cannot find file {0} at location: {1}.
<a href="#">GBRX U047 9E</a>	Cannot convert an Object to its Node representation.
<a href="#">GBRX U048 0E</a>	Cannot convert a Node to an Object.
<a href="#">GBRX U048 1E</a>	Cannot find a resource at location {0}.
<a href="#">GBRX U048 2E</a>	The directory property of the cache of compiled ruleset archives is not defined.
<a href="#">GBRX U048 3E</a>	Failed to copy the resource {0} to a temporary location.

<a href="#">GBRX U048 4E</a>	Failed to determine the build version of the decision engine run time.
<a href="#">GBRX U300 1W</a>	The loading of ruleset {0} raised a warning: {1}.
<a href="#">GBRX U300 2W</a>	An MBean of a previous XU is already registered. It will be replaced by a new MBean.
<a href="#">GBRX U300 3W</a>	A nonserializable object {0} is added to the working memory. Transaction support might fail.
<a href="#">GBRX U300 4W</a>	The XU client does not have the same version as the XU: {0} / {1}.
<a href="#">GBRX U300 5W</a>	Use a file repository with ruleset archives expanded, only for test purposes.
<a href="#">GBRX U300 6W</a>	Warning during the deserialization of the XML string : {0}.
<a href="#">GBRX U300 7W</a>	The plug-in of class {0} does not define the getSupportedFunctionNames method.
<a href="#">GBRX U300 8W</a>	XU transaction support is deprecated and will be removed in a future release.
<a href="#">GBRX U300 9W</a>	The engine version number {0} is lower than the number of the version {1} that produced the ruleset archive.
<a href="#">GBRX U301 0W</a>	A warning was raised during BOM conversion: {0}.
<a href="#">GBRX U301 1W</a>	Directory {0} for the decision engine runtime library does not exist.
<a href="#">GBRX U301 2W</a>	The property ruleset.engine.version of the decision engine archive is not set for the ruleset : {0}.
<a href="#">GBRX U301 3W</a>	The index file of the decision engine runtime library is empty.
<a href="#">GBRX U301 4W</a>	The Classic Rule Engine is deprecated.
<a href="#">GBRX U400 1I</a>	Sets the XU configuration property {0} to {1}.
<a href="#">GBRX U400 2I</a>	Starts the MBean plug-in.
<a href="#">GBRX U400 3I</a>	Destroys the MBean plug-in.
<a href="#">GBRX U400 4I</a>	Creates the XU plug-in with properties {0}.
<a href="#">GBRX U400 5I</a>	Starts the XU plug-in {0}.
<a href="#">GBRX U400 6I</a>	Destroys the XU plug-in {0}.
<a href="#">GBRX U400 7I</a>	A destroyed SPI connection was found in the pool.

<a href="#">GBRX U400 7I</a>	A destroyed SPI connection was found in the pool.
<a href="#">GBRX U400 8I</a>	Logging started {0} XU - {1} - {2}.
<a href="#">GBRX U401 1I</a>	The rule engine resource adapter handles the low-level details of ruleset execution and provides management access to its resources. You can access configuration and runtime data either through a JMX MBean or by using TCP/IP management.
<a href="#">GBRX U401 8I</a>	The operating system is {0} {1} {2}.
<a href="#">GBRX U401 9I</a>	The class path is {0}.
<a href="#">GBRX U402 0I</a>	The JVM is {1} {2} {0}.
<a href="#">GBRX U501 8I</a>	Disconnect.
<a href="#">GBRX U501 9I</a>	Adds rule {3}.
<a href="#">GBRX U502 0I</a>	Removes rule {3}.
<a href="#">GBRX U502 1I</a>	Defines function {3}.
<a href="#">GBRX U502 2I</a>	Adds object {3}.
<a href="#">GBRX U502 3I</a>	Adds logical object {3}.
<a href="#">GBRX U502 4I</a>	Removed object {3}.
<a href="#">GBRX U502 5I</a>	Updated object {3}.
<a href="#">GBRX U502 6I</a>	Removes all.
<a href="#">GBRX U502 7I</a>	Resets.
<a href="#">GBRX U502 8I</a>	Adds rule instance {3}. The previous rule is {4}.
<a href="#">GBRX U502 9I</a>	Rule instance {3} is removed.
<a href="#">GBRX U503 0I</a>	All rule instances are removed.
<a href="#">GBRX U503 1I</a>	The execution of the rule instance {3} starts.
<a href="#">GBRX U503 2I</a>	The execution of the rule instance {3} ends.
<a href="#">GBRX</a>	The execution of the IRL task {3} ends.

<a href="#">U503 3I</a>	
<a href="#">GBRX U503 4I</a>	The execution of the ruleflow {3} starts.
<a href="#">GBRX U503 5I</a>	The execution of the ruleflow {3} ends.
<a href="#">GBRX U503 6I</a>	The execution of the sequential instance, rule= {3}, tuple= {4}, priority= {5} begins.
<a href="#">GBRX U503 7I</a>	The execution of the sequential instance, rule= {3}, tuple= {4}, priority= {5} ends.
<a href="#">GBRX U503 8I</a>	Connection established.
<a href="#">GBRX U503 9I</a>	Parameters {3} are set.
<a href="#">GBRX U504 0I</a>	The engine reset starts.
<a href="#">GBRX U504 1I</a>	The engine reset is finished.
<a href="#">GBRX U504 2I</a>	Session {0} starts.
<a href="#">GBRX U504 3I</a>	Session {0} stops.
<a href="#">GBRX U504 4I</a>	The CCI connection is closed. Measures: {0}.
<a href="#">GBRX U504 5I</a>	Rule engine, new ruleset ({0}). Measures: {1}.
<a href="#">GBRX U504 6I</a>	Rule engine, new instance ({0}). Measures: {1}.
<a href="#">GBRX U504 7I</a>	Rule engine, executes ({0}). Measures: {1}.
<a href="#">GBRX U504 8I</a>	Rule engine, executes initial actions ({0}). Measures: {1}.
<a href="#">GBRX U504 9I</a>	Rule engine, executes main ({0}). Measures: {1}.
<a href="#">GBRX U505 0I</a>	Rule engine, executes task ({0}). Measures: {1}.
<a href="#">GBRX U505 1I</a>	Rule engine, adds ({0}). Measures: {1}.
<a href="#">GBRX U505 2I</a>	Rule engine, removes ({0}). Measures: {1}.
<a href="#">GBRX U505 3I</a>	Rule engine, updates ({0}). Measures: {1}.
<a href="#">GBRX U505</a>	Rule engine, sets parameters ({0}). Measures: {1}.



<a href="#">4I</a>	
<a href="#">GBRX U505 5I</a>	{0}, retrieves IRL ({1}). Measures: {2}.
<a href="#">GBRX U505 6I</a>	{0}, retrieves XOM ({1}). Measures: {2}.
<a href="#">GBRX U505 7I</a>	{0}, retrieves information about ruleset properties ({1}). Measures: {2}.
<a href="#">GBRX U505 8I</a>	TimeStamp
<a href="#">GBRX U505 9I</a>	Duration
<a href="#">GBRX U506 0I</a>	Number of executed rules
<a href="#">GBRX U506 1I</a>	Number of rules that were executed before this execution
<a href="#">GBRX U506 2I</a>	Number of rules that were executed per millisecond
<a href="#">GBRX U506 3I</a>	Number of backed-up context managers
<a href="#">GBRX U506 4I</a>	Number of context managers
<a href="#">GBRX U506 5I</a>	The state of the XU connection ({0}) was backed up. Measures: {1}.
<a href="#">GBRX U506 6I</a>	The state of the XU connection ({0}) was rolled back. Measures: {1}.
<a href="#">GBRX U506 7I</a>	The state of the XU connection ({0}) was committed. Measures: {1}.
<a href="#">GBRX U506 8I</a>	The ruleset archive has changed: {0}.
<a href="#">GBRX U506 9I</a>	Execution result: {0}.
<a href="#">GBRX U507 0I</a>	Execution result: {0}.
<a href="#">GBRX U507 1I</a>	Execution result: {0}.
<a href="#">GBRX U507 2I</a>	{0}.
<a href="#">GBRX U507 3I</a>	Parsing ruleset {0} with the following properties: {1}.
<a href="#">GBRX U507 4I</a>	Ruleset {0} was parsed successfully.
<a href="#">GBRX U507</a>	The parsing of ruleset {0} failed.

<a href="#">5I</a>	
<a href="#">GBRX U600 1I</a>	Local transaction start: {1} {0}.
<a href="#">GBRX U600 2I</a>	Local transaction start: {0}.
<a href="#">GBRX U600 3I</a>	Local transaction commit: {1} {0}.
<a href="#">GBRX U600 4I</a>	Local transaction committed: {0}.
<a href="#">GBRX U600 5I</a>	Local transaction rollback: {1} {0}.
<a href="#">GBRX U600 6I</a>	Local transaction rolled back: {0}.
<a href="#">GBRX U600 7I</a>	Closing CCI connection {0}.
<a href="#">GBRX U600 8I</a>	Connection closed: id={0}.
<a href="#">GBRX U600 9I</a>	SPI connection {0} clean-up.
<a href="#">GBRX U601 0I</a>	SPI connection {0} cleaned up.
<a href="#">GBRX U601 1I</a>	SPI connection {0} creates CCI connection {1}.
<a href="#">GBRX U601 2I</a>	Initializes the pool {0}.
<a href="#">GBRX U601 3I</a>	Pool {0}, properties: {1}.
<a href="#">GBRX U601 4I</a>	waitTimeout: {0}.
<a href="#">GBRX U601 5I</a>	The XU logger uses JDK logging: {0}.
<a href="#">GBRX U601 6I</a>	The JDK logging level is set to: {0}.
<a href="#">GBRX U601 7I</a>	Loading ruleset archive {0} with {1}.
<a href="#">GBRX U601 8I</a>	The ruleset archive {0} has been loaded with {1}.
<a href="#">GBRX U601 9I</a>	The version of the application that generated the ruleset archive {0} is {1}.{2}.
<a href="#">GBRX U602 0I</a>	It is not possible to retrieve the number of the version that produced the ruleset archive for {0}.
<a href="#">GBRX</a>	Open connection: id={0}, type={1}, ruleset path={2}, client class loader={3}.

<a href="#">U602 1 </a>	
<a href="#">GBRX U602 2 </a>	Ruleset {0} was parsed in {1} s.
<a href="#">GBRX U602 3 </a>	Set property {0} to {1}.
<a href="#">GBRX U602 4 </a>	The MBean was registered by the MBean manager {0} with the name {1}.
<a href="#">GBRX U602 5 </a>	The MBean was unregistered by the MBean manager {0} with the objectname {1}.
<a href="#">GBRX U602 6 </a>	The execution of the IRL task {3} begins.
<a href="#">GBRX U602 7 </a>	Version {0} of the decision engine runtime library cannot be found at path {1}.
<a href="#">GBRX U602 8 </a>	Loading ruleset archive {0} with version {1} of the decision engine runtime library.
<a href="#">GBRX U602 9 </a>	The ruleset {0} is no longer in the ruleset cache. Instead, it is now kept as a weak reference in the deprecated ruleset cache.
<a href="#">GBRX U603 0 </a>	The ruleset {0} is now kept as a strong reference because it is in use or the ruleset.maxIdleTime is not yet reached.
<a href="#">GBRX U603 1 </a>	The ruleset {0} was released.
<a href="#">GBRX U603 2 </a>	The ruleset {0} is idle.
<a href="#">GBRX U603 3 </a>	The ruleset {0} is now kept as a weak reference because the ruleset.maxIdleTime was exceeded.
<a href="#">GBRX U603 4 </a>	The XU setting rulesetUsageMonitorEnabled is set to "false", so the ruleset archive property ruleset.maxIdleTime does not influence the removal of the ruleset from the cache.
<a href="#">GBRX U603 5 </a>	Start the maintenance task of the ruleset cache on {0}.
<a href="#">GBRX U603 6 </a>	RESMGMT {0}: Processing the notification "ruleset {1} is changed" is completed by the execution unit.
<a href="#">GBRX U603 7 </a>	The ruleset {0} is missing in the cache.
<a href="#">GBRX U603 8 </a>	Cache before addition of ruleset {0}:
<a href="#">GBRX U603 9 </a>	Cache after addition of ruleset {0}:
<a href="#">GBRX U604 0 </a>	Ruleset cache size: {0}.
<a href="#">GBRX U604 1 </a>	Deprecated ruleset cache size: {0}.
<a href="#">GBRX U604</a>	Entry number {0}: ruleset {1} {2} {3} {4}.

<a href="#">2I</a>	
<a href="#">GBRX</a> <a href="#">U604</a> <a href="#">3I</a>	RESMGMT {0}: The notification "ruleset {1} is changed" is processed by the execution unit.
<a href="#">GBRX</a> <a href="#">U604</a> <a href="#">4I</a>	The duration of the operation {0} is {1} ms.
<a href="#">GBRX</a> <a href="#">U604</a> <a href="#">5I</a>	The duration for opening a connection to the XU is {0} ms.
<a href="#">GBRX</a> <a href="#">U604</a> <a href="#">6I</a>	RESMGMT {0}: The notification "ruleset {1} is changed" is processed by the ruleset cache.
<a href="#">GBRX</a> <a href="#">U604</a> <a href="#">7I</a>	RESMGMT {0}: Processing the notification "ruleset {1} is changed" is completed by the ruleset cache.

# Rule Execution Server Messages - Messages pertaining to the common API

A message ID consists of 10 alphanumeric characters that uniquely identify the message. The message ID is composed of:

- 3-character product identifier
- 2-character component or subsystem identifier
- 4-digit serial number
- 1-character type code indicating the severity of the message

The severity of the message is indicated by the last character in the message ID.

- I: Informational -- Provides information or feedback about normal events that occur.
- W: Warning -- Indicates that potentially undesirable conditions have occurred, but processing can continue.
- E: Error -- Indicates that a problem has occurred that requires intervention or correction before processing can continue.

ID	Description
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">0E</a>	An error occurred when the execution unit was notified for the ruleset path: {0} caused by: {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">1W</a>	An error occurred when properties were retrieved: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">2I</a>	XU management plugin started successfully.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">2W</a>	The XU management plugin cannot access the XU configuration property {0}. This property is ignored.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">3I</a>	Notification received for the ruleset path {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">3W</a>	Property {0} will not be displayed in the Server Info page of the Rule Execution Server console.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">4E</a>	Unable to parse the {0} value. Use the default value: {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">5E</a>	Unable to start the XU management plugin because of the following exception: {0} {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">6I</a>	XU management plugin stopped successfully.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">7I</a>	The XU management plugin is creating a notification client with properties: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">8W</a>	The locale is not specified in the incoming message. The default locale is used.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">001</a> <a href="#">9E</a>	The ruleset path is not specified in the incoming message.
<a href="#">GB</a>	Invalid data in the incoming message.

<a href="#">RX</a> <a href="#">002</a> <a href="#">0E</a>	
<a href="#">GB</a> <a href="#">RX</a> <a href="#">002</a> <a href="#">1E</a>	Internal error in the XU client.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">005</a> <a href="#">0I</a>	Not available.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">005</a> <a href="#">1I</a>	RESMGMT {2}: The execution unit {1} received a notification "ruleset {0} is changed".
<a href="#">GB</a> <a href="#">RX</a> <a href="#">005</a> <a href="#">2I</a>	RESMGMT {2}: The notification "ruleset {0} is changed" is processed by the execution unit {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">0I</a>	Client session {0} created for {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">1I</a>	Client session {0} opened for {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">2W</a>	Client {0} was unable to perform handshake with the notification server. It will disconnect and then try to reconnect. For details, see the exception trace.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">3I</a>	Client {0} performed handshake with the notification server.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">4I</a>	Client session {0} closed for {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">5W</a>	Exception in client session {0} of {1}. The session is closing. For details, see the exception trace.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">6I</a>	Client session {0} of {1}: message received: {2}
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">7W</a>	Client session {0} of {1}: the message received has an unsupported type and will be ignored.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">010</a> <a href="#">8W</a>	Client session {0} of {1}: The message received has an unsupported type and will be ignored.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">011</a> <a href="#">1E</a>	An exception occurred in client session {0} of {1} while an incoming message was processed. For details, see the exception trace.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">011</a> <a href="#">2I</a>	Response sent from client session {0} of {1} with correlation ID {2}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">011</a> <a href="#">3E</a>	Error in client session {0} of {1} when a response is sent.

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">011</a> <a href="#">5I</a>	Client {0} failed to connect to the notification server. Trying to reconnect in {1} milliseconds. For details, see the exception trace: {2}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">011</a> <a href="#">6I</a>	Client {0} is already connected to the notification server. The connection request is ignored.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">011</a> <a href="#">7W</a>	Client session {0} of {1} has been disconnected, and is trying to reconnect.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">011</a> <a href="#">8W</a>	Client {0}: Error creating a client socket. For details, see the exception trace.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">011</a> <a href="#">9I</a>	Client {0} is connected to server {1}:{2}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">012</a> <a href="#">0I</a>	Client {0} is disconnected.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">0E</a>	{0} is not a valid value for the notification server port. Specify a value greater than 0.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">1E</a>	The server is already running, therefore it cannot be started.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">2I</a>	Notification server started on port {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">3I</a>	The notification server is stopping.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">4I</a>	Notification server is stopped.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">5E</a>	Client {0} is not connected to the notification server.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">6E</a>	Error in session {0} of server when a request with header {1} is sent to client {2}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">7I</a>	Server session {0} is created.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">8I</a>	Server session {0} opened.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">015</a> <a href="#">9I</a>	The server session {0} is closed.
<a href="#">GB</a> <a href="#">RXX</a>	Exception in server session {0}. The session is closing. For details, see the exception trace.

<a href="#">0160W</a>	
<a href="#">GB RXX 0161I</a>	Server session {0}: message received: {1}
<a href="#">GB RXX 0162W</a>	In the server session {0}, the message that was received has an unsupported type. The type will be ignored.
<a href="#">GB RXX 0164W</a>	Server session {0}: The message received has an unsupported type and it will be ignored.
<a href="#">GB RXX 0165I</a>	Notification server is broadcasting a notification with header {0}.
<a href="#">GB RXX 0166I</a>	Notification server is sending a request with header {0} to {1}.
<a href="#">GB RXX 0167I</a>	Client {0} is not compatible with the notification server, all messages will be ignored.
<a href="#">GB RXX 0168I</a>	Closing the session because a keep-alive response message was not received within {0} second(s).
<a href="#">GB RXX 0201W</a>	Wrong library name "{0}" used during the load of library "{1}".
<a href="#">GB RXX 0202E</a>	The path specified for the XOM DAO must be a directory: "{0}".
<a href="#">GB RXX 0203E</a>	Cannot initialize XOM DAO with the path "{0}".
<a href="#">GB RXX 0204E</a>	Unknown error.
<a href="#">GB RXX 0205E</a>	Cannot add XOM resource "{0}".
<a href="#">GB RXX 0206E</a>	Cannot remove XOM resource "{0}".
<a href="#">GB RXX 0207E</a>	Cannot find the XOM resource "{0}" in the file repository "{1}".
<a href="#">GB RXX 0208E</a>	Cannot retrieve the content of the XOM resource "{0}".
<a href="#">GB RXX 0209E</a>	Unable to integrate file "{0}" in the file repository "{1}".



<a href="#">GB RXX 021 0E</a>	Cannot add library "{0}".
<a href="#">GB RXX 021 1E</a>	Unable to read library "{0}".
<a href="#">GB RXX 021 2E</a>	Cannot remove library "{0}".
<a href="#">GB RXX 021 3E</a>	Error on the XOM resource "{0}". The content cannot be empty.
<a href="#">GB RXX 021 4E</a>	Cannot retrieve the XOM resource identifier from the database.
<a href="#">GB RXX 021 5E</a>	Unsupported merging policy "{0}".
<a href="#">GB RXX 021 6E</a>	Error during the library creation.
<a href="#">GB RXX 021 7E</a>	Could not read internal URL "{0}" because the XOM repository is not set.
<a href="#">GB RXX 021 8I</a>	Requested for {0} ClassLoader (Client ClassLoader First): {1}.
<a href="#">GB RXX 021 9I</a>	Requested for {0} ClassLoader (Client ClassLoader Last): {1}.
<a href="#">GB RXX 022 0E</a>	JNDI lookup on the data source "{0}" failed.
<a href="#">GB RXX 022 1E</a>	Cannot connect to the database.
<a href="#">GB RXX 022 2E</a>	Cannot create the DAO "{0}".
<a href="#">GB RXX 022 3E</a>	Cannot create the JDBC driver "{1}" from class loader "{0}" with the URL "{2}" and properties {3}.
<a href="#">GB RXX 022 4E</a>	Failed to parse the XML DAO descriptor "{0}".
<a href="#">GB RXX 022 5E</a>	Cannot retrieve the database metadata.
<a href="#">GB RXX 022</a>	Cannot save the display name "{1}" for the element "{0}".

<a href="#">6E</a>	
<a href="#">GB RXX 022 7E</a>	Failed to set autoCommit to false on the database connection.
<a href="#">GB RXX 022 8E</a>	Cannot commit the transaction.
<a href="#">GB RXX 022 9E</a>	Cannot remove ruleset "{0}".
<a href="#">GB RXX 023 0E</a>	Cannot remove RuleApp "{0}".
<a href="#">GB RXX 023 1E</a>	Cannot add ruleset "{0}".
<a href="#">GB RXX 023 2E</a>	Cannot update RuleApp "{0}".
<a href="#">GB RXX 023 3E</a>	Cannot add RuleApp "{0}".
<a href="#">GB RXX 023 4E</a>	Cannot save the description "{1}" for the element "{0}".
<a href="#">GB RXX 023 5E</a>	Failed to load the repository.
<a href="#">GB RXX 023 6E</a>	Wrong path used: "{0}".
<a href="#">GB RXX 023 7E</a>	Cannot retrieve the ruleset archive for the ruleset "{0}".
<a href="#">GB RXX 023 8E</a>	A ruleset archive must be associated with the ruleset {0}.
<a href="#">GB RXX 023 9E</a>	Cannot save the ruleset archive for the ruleset "{0}".
<a href="#">GB RXX 024 0E</a>	Cannot find the ruleset "{0}".
<a href="#">GB RXX 024 1E</a>	Error on the ruleset "{0}". The ruleset archive cannot be empty.
<a href="#">GB RXX 024 2E</a>	Cannot retrieve properties for the element "{0}".
<a href="#">GB</a>	Cannot set property "{1}" to "{2}" for the element "{0}".

<a href="#">RXX0243E</a>	
<a href="#">GBRXX0244E</a>	Cannot find the RuleApp "{0}".
<a href="#">GBRXX0245E</a>	Failed to retrieve the canonical ruleset path using "{0}".
<a href="#">GBRXX0246E</a>	Failed to execute SQL statement "{0}".
<a href="#">GBRXX0247E</a>	Database metadata is not available.
<a href="#">GBRXX0248E</a>	Cannot initialize DAO with the path "{0}".
<a href="#">GBRXX0249E</a>	Error on the creation date file at "{0}".
<a href="#">GBRXX0250E</a>	The path specified for the DAO must be a directory: "{0}".
<a href="#">GBRXX0251E</a>	Cannot find the RuleApp "{0}" in the file repository "{1}".
<a href="#">GBRXX0252E</a>	Cannot find the ruleset "{0}" in the file repository "{1}".
<a href="#">GBRXX0253E</a>	The ruleset "{0}" has an unexpected engine type "{1}". Allowed values are: "null" (i.e. property is not defined), "CRE", "DE", "rce", "rve", "rvend", and "rvejb".
<a href="#">GBRXX0254E</a>	Unable to find persistence type "{0}" in property list: {1}
<a href="#">GBRXX0255E</a>	Cannot find class "{0}" to load DAO "{1}".
<a href="#">GBRXX0256E</a>	Unexpected exception during the build of DAO "{0}" using class name "{1}".
<a href="#">GBRXX0257E</a>	Unexpected exception during the build of DAO "{0}".
<a href="#">GBRXX0258E</a>	Unable to find default constructor during DAO creation for class "{0}".
<a href="#">GBRXX025</a>	Could not create a managed XOM class loader for the following path: "{0}".

<a href="#">9E</a>	
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">0W</a>	XOM persistence not initialized for the following reason: {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">1E</a>	Decryption of the JDBC connection password has failed. The encrypted password value was: {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">2I</a>	XOM repository set in file persistence mode: {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">3I</a>	XOM repository set in database persistence mode: {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">4I</a>	XOM repository requested the resource "{0}" in XOM URI "{1}"
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">5I</a>	XOM URI is already loaded in memory "{0}"
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">6I</a>	XOM URI cache was garbage collected. Load the XOM URI again.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">7I</a>	Load resource "{0}" from XOM URI "{1}" in memory
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">8I</a>	Could not find class "{0}" in XOM URI "{1}"
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">026</a> <a href="#">9E</a>	XOM repository is not set
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">0W</a>	Default XOM persistence initialized with type "{0}": {1}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">1E</a>	Unresolved resources or libraries: {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">2E</a>	Cannot delete directory {0}. Check the file system content and permissions.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">3E</a>	An input/output error occurred while creating the list of RuleApp directories from the file based repository. The root directory of the repository is {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">4E</a>	An input/output error occurred while creating the list of RuleApp version directories from the file based repository. The home directory of the RuleApp is {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">027</a> <a href="#">5E</a>	No matching schema found
<a href="#">GB</a>	No schema found

<a href="#">RXX0276E</a>	
<a href="#">GBRXX0277E</a>	Driver name cannot be null.
<a href="#">GBRXX0278E</a>	URL cannot be null.
<a href="#">GBRXX0279E</a>	Executing SQL query: {0}
<a href="#">GBRXX0280E</a>	SQL execution ended after: {0} ms.
<a href="#">GBRXX0281E</a>	SQL execution failed after: {0} ms.
<a href="#">GBRXX0282E</a>	An error occurred when executing the SQL query.
<a href="#">GBRXX0283E</a>	An error occurred when closing the database connection.
<a href="#">GBRXX0284E</a>	An error occurred when closing the SQL statement.
<a href="#">GBRXX0285E</a>	An error occurred when releasing the database from which the ResultSet object was retrieved.
<a href="#">GBRXX0286E</a>	An error occurred when decrypting the password.
<a href="#">GBRXX0287E</a>	The {0} persistence type is not supported.
<a href="#">GBRXX0288E</a>	The database connection is unavailable.
<a href="#">GBRXX0289I</a>	RESMGMT {1}: Adding RuleApp "{0}".
<a href="#">GBRXX0290I</a>	RESMGMT {1}: Removing RuleApp "{0}".
<a href="#">GBRXX0291I</a>	RESMGMT {2}: Adding RuleApp property "{1}" to "{0}".
<a href="#">GBRXX0292I</a>	RESMGMT {2}: Changing RuleApp property "{1}" in "{0}".

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">3I</a>	RESMGMT {2}: Removing RuleApp property "{1}" from "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">4I</a>	RESMGMT {1}: RuleApp "{0}" is added.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">5I</a>	RESMGMT {1}: RuleApp "{0}" is removed.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">6I</a>	RESMGMT {2}: RuleApp property "{1}" is added to "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">7I</a>	RESMGMT {2}: RuleApp property "{1}" is changed in "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">8I</a>	RESMGMT {2}: RuleApp property "{1}" is removed from "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">029</a> <a href="#">9I</a>	RESMGMT {1}: Adding ruleset "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">0I</a>	RESMGMT {1}: Changing ruleset "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">1I</a>	RESMGMT {1}: Removing ruleset "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">2I</a>	RESMGMT {2}: Adding ruleset property "{1}" to "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">3I</a>	RESMGMT {2}: Changing ruleset property "{1}" in "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">4I</a>	RESMGMT {2}: Removing ruleset property "{1}" from "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">5I</a>	RESMGMT {1}: Ruleset "{0}" is added.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">6I</a>	RESMGMT {1}: Ruleset "{0}" is changed.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">7I</a>	RESMGMT {1}: Ruleset "{0}" is removed.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">030</a> <a href="#">8I</a>	RESMGMT {2}: Ruleset property "{1}" is added to "{0}".
<a href="#">GB</a> <a href="#">RXX</a>	RESMGMT {2}: Ruleset property "{1}" is changed in "{0}".

<a href="#">0309I</a>	
<a href="#">GB RXX 031 0I</a>	RESMGMT {2}: Ruleset property "{1}" is removed from "{0}".
<a href="#">GB RXX 031 1I</a>	RESMGMT {1}: Changing RuleApp "{0}".
<a href="#">GB RXX 031 2I</a>	RESMGMT {1}: RuleApp "{0}" is changed.
<a href="#">GB RXX 031 3I</a>	SQL connection closed in {0} ms.
<a href="#">GB RXX 031 4I</a>	SQL statement closed in {0} ms.
<a href="#">GB RXX 031 5I</a>	SQL resultset closed in {0} ms.
<a href="#">GB RXX 031 6E</a>	Failed attempt to close the SQL connection in {0} ms.
<a href="#">GB RXX 031 7E</a>	Failed attempt to close the SQL statement in {0} ms.
<a href="#">GB RXX 031 8E</a>	Failed attempt to close the SQL resultset in {0} ms.
<a href="#">GB RXX 031 9I</a>	DAO rollbacked a JDBC transaction in {0} ms.
<a href="#">GB RXX 032 0I</a>	DAO committed a JDBC transaction in {0} ms.
<a href="#">GB RXX 032 1I</a>	SQL driver connected to {0} in {1} ms.
<a href="#">GB RXX 032 2I</a>	Connection to SQL datasource retrieved in {0} ms.
<a href="#">GB RXX 032 3I</a>	XOM DAO committed a JDBC transaction in {0} ms.
<a href="#">GB RXX 032 4I</a>	DAO executed the SQL statement "{0}" in {1} ms.
<a href="#">GB RXX 032 5I</a>	XOM DAO rollbacked a JDBC transaction in {0} ms.
<a href="#">GB</a>	XOM DAO executed the SOL statement "{0}" in {1} ms.

<a href="#">RX</a> <a href="#">032</a> <a href="#">6I</a>	
<a href="#">GB</a> <a href="#">RX</a> <a href="#">032</a> <a href="#">7E</a>	XOM DAO failed to rollback a JDBC transaction in {0} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">032</a> <a href="#">8E</a>	XOM DAO failed to commit a JDBC transaction in {0} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">032</a> <a href="#">9E</a>	SQL driver failed to connect to {0} in {1} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">0E</a>	XOM DAO failed to commit a JDBC transaction in {0} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">1E</a>	Failed to retrieve connection to SQL datasource in {0} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">2E</a>	DAO failed to execute the SQL statement "{0}" in {1} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">3E</a>	XOM DAO failed to rollback a JDBC transaction in {0} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">4E</a>	XOM DAO failed to execute the SQL statement "{0}" in {1} ms.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">5E</a>	Cannot find the ruleset "{0}" in the classloader "{1}" of the memory repository.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">6E</a>	Cannot find the RuleApp "{0}" in the classloader "{1}" of the memory repository.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">7E</a>	An unexpected exception occurred when creating the XOM classLoader for ruleset "{0}": {1}
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">8I</a>	There is no ruleset archive resource for the path "{0}" in the classLoader "{1}".
<a href="#">GB</a> <a href="#">RX</a> <a href="#">033</a> <a href="#">9E</a>	An unexpected exception occurred when the RuleApp loading strategy "{0}" tried to load "{1}": {2}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">034</a> <a href="#">0I</a>	The RuleApp loading strategy "{0}" did not find any RuleApp archive descriptor "{1}" in the resources of the classloader "{2}".
<a href="#">GB</a> <a href="#">RX</a> <a href="#">034</a> <a href="#">1I</a>	The strategy "{0}" found the following entries in the classLoader "{1}": {2}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">034</a> <a href="#">2E</a>	An unexpected exception occurred when the RuleApp loading strategy "{0}" tried to load the resource "{1}" of the classloader "{2}" as a ruleset archive: {3}.



<a href="#">GB RXX 034 3E</a>	An unexpected exception occurred when loading the content of the resource "{0}" as a RuleApp archive JAR file from the classLoader "{1}" in memory: {2}
<a href="#">GB RXX 034 4E</a>	An unexpected exception occurred when loading the content of the URL "{0}" as a RuleApp archive in memory: {1}.
<a href="#">GB RXX 034 5I</a>	XOM repository set to memory persistence mode
<a href="#">GB RXX 034 6I</a>	XOM repository persistence mode is not set because of an unexpected value for XOM repository DAO: {0}.
<a href="#">GB RXX 034 7E</a>	An unexpected exception occurred when adding the RuleApp "{0}" to the memory repository: {1}
<a href="#">GB RXX 034 8I</a>	The following entries were found in the classLoader "{0}": {1}
<a href="#">GB RXX 034 9I</a>	RuleApp loading strategy "{0}": looking up the RuleApp archive "{1}" as a resource in the classloader "{2}".
<a href="#">GB RXX 035 0I</a>	RuleApp loading strategy "{0}": loading the classloader "{1}" entry "{2}" as a RuleApp archive.
<a href="#">GB RXX 035 1I</a>	RuleApp loading strategy "{0}": looking up for the "META-INF/archive.xml" RuleApp archive descriptor in the classloader "{1}" resources.
<a href="#">GB RXX 035 2I</a>	The RuleApp loading strategy "{0}" did not find any ruleset.
<a href="#">GB RXX 035 3I</a>	The RuleApp loading strategy "{0}" found the ruleset "{1}".
<a href="#">GB RXX 035 4I</a>	The RuleApp loading strategy "{0}" found the following rulesets: {1}.
<a href="#">GB RXX 035 5E</a>	An unexpected exception occurred when the RuleApp loading strategy "{0}" tried to load the resource "{1}" of the classloader "{2}" as a RuleApp archive descriptor: {3}.
<a href="#">GB RXX 035 6E</a>	The RuleApp loading strategy "{0}" failed to load "{1}" as a RuleApp archive: {2}
<a href="#">GB RXX 035 7I</a>	The RuleApp loading strategy "{0}" did not find any content for the resource "{1}" in the classloader "{2}".
<a href="#">GB RXX 035 8I</a>	The RuleApp loading strategy "{0}" did not find any content for the classloader "{1}" entry "{2}".
<a href="#">GB RXX</a>	Trying the RuleApp loading strategy "{0}": looking up for the "{1}META-INF/archive.xml" RuleApp archive descriptor in the resources of the classloader "{2}".

<a href="#">0359I</a>	
<a href="#">GB RXX 036 0I</a>	No ruleset was added to the repository by the default RuleApp loading strategies.
<a href="#">GB RXX 036 1I</a>	The default RuleApp loading strategies added the ruleset "{0}" to the repository.
<a href="#">GB RXX 036 2I</a>	The default RuleApp loading strategies added the following rulesets to the repository: {0}.
<a href="#">GB RXX 036 3E</a>	An unexpected error occurred while cleaning up XOMClassLoaderImpl instances: {0}.
<a href="#">GB RXX 040 1E</a>	Cannot commit the transaction.
<a href="#">GB RXX 040 2E</a>	Cannot create an instance of the trace DAO using the following factory class: "{0}".
<a href="#">GB RXX 040 3E</a>	Error while deleting traces.
<a href="#">GB RXX 040 4E</a>	Failed to execute SQL statement "{0}".
<a href="#">GB RXX 040 5E</a>	Cannot retrieve all traces.
<a href="#">GB RXX 040 6E</a>	Cannot find traces.
<a href="#">GB RXX 040 7E</a>	Invalid query type "{0}" for DAO "{1}".
<a href="#">GB RXX 040 8E</a>	Cannot save the execution trace.
<a href="#">GB RXX 040 9E</a>	Unable to create transaction.
<a href="#">GB RXX 041 0E</a>	No trace DAO factory class is specified in the settings.
<a href="#">GB RXX 041 1E</a>	Could not instantiate the following driver class: {0}.
<a href="#">GB RXX 041 2E</a>	Invalid state: no enclosing transaction.
<a href="#">GB</a>	Trace database connection error.

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">041</a> <a href="#">3E</a>	Invalid database connection error.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">041</a> <a href="#">4W</a>	Cannot serialize system properties. No system properties will be stored in the decision trace.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">041</a> <a href="#">5E</a>	At least one of the two values isn't a number: {0}, {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">041</a> <a href="#">6E</a>	Property "{0}" not found on {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">1E</a>	Invalid path "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">2E</a>	Invalid version "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">3E</a>	The path cannot be null.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">4E</a>	Invalid version "{0}" (max int overflow on major number).
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">5E</a>	Invalid version "{0}" (max int overflow on minor number).
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">6E</a>	Unsupported merging policy "{0}". Supported policies are: "{1}", "{2}", "{3}" and "{4}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">7E</a>	Unsupported versioning policy "{0}". Supported policies are: "{1}" and "{2}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">060</a> <a href="#">8E</a>	Unsupported XOM merging policy "{0}". Supported policies are: "{1}" and "{2}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">080</a> <a href="#">1E</a>	RuleApp "{0}" already exists.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">080</a> <a href="#">2E</a>	The ruleset "{0}" already exists.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">080</a> <a href="#">3E</a>	A RuleApp name cannot be empty.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">080</a> <a href="#">4E</a>	An error occurred when the RuleApp name "{0}" was checked. The character "{1}" is not valid.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">080</a> <a href="#">5E</a>	A ruleset name cannot be empty.

<a href="#">GB RXX 080 6E</a>	An error occurred when the ruleset name "{0}" was checked. The character "{1}" is not valid.
<a href="#">GB RXX 080 7E</a>	A resource provider cannot be null.
<a href="#">GB RXX 080 8E</a>	"{0}": A resource provider error occurred on the ruleset archive.
<a href="#">GB RXX 080 9E</a>	"{0}": A resource provider error occurred on properties.
<a href="#">GB RXX 081 0E</a>	A resource provider error occurred when the ruleset "{0}" was added.
<a href="#">GB RXX 081 1E</a>	A resource provider error occurred when the ruleset "{0}" was removed.
<a href="#">GB RXX 081 2E</a>	A resource provider error occurred when RuleApp "{0}" was added.
<a href="#">GB RXX 081 3E</a>	A resource provider error occurred when RuleApp "{0}" was removed.
<a href="#">GB RXX 081 4E</a>	An error occurred when RuleApps were imported.
<a href="#">GB RXX 081 5E</a>	The version number cannot be null.
<a href="#">GB RXX 081 6E</a>	Unsupported merging policy "{0}".
<a href="#">GB RXX 081 7E</a>	Unsupported versioning policy "{0}".
<a href="#">GB RXX 081 8E</a>	"{0}": A resource provider error occurred on the display name.
<a href="#">GB RXX 081 9E</a>	"{0}": A resource provider error occurred on the description.
<a href="#">GB RXX 082 0E</a>	A resource provider error occurred during the loading.
<a href="#">GB RXX 082 1E</a>	The creation date cannot be null.
<a href="#">GB RXX</a>	The RuleApp cannot be null.

<a href="#">0822E</a>	
<a href="#">GB RXX 082 3E</a>	The ruleset cannot be null.
<a href="#">GB RXX 082 4E</a>	A resource provider error occurred during the update.
<a href="#">GB RXX 082 5E</a>	The engine type cannot be null.
<a href="#">GB RXX 082 6E</a>	Invalid XOM library name "{0}".
<a href="#">GB RXX 082 7E</a>	Invalid XOM resource name "{0}".
<a href="#">GB RXX 082 8E</a>	The URL value cannot be empty.
<a href="#">GB RXX 082 9E</a>	Wrong format for a XOM resource: "{0}".
<a href="#">GB RXX 083 0E</a>	Wrong format for a XOM library: "{0}".
<a href="#">GB RXX 083 1E</a>	Wrong URL format: "{0}".
<a href="#">GB RXX 083 2E</a>	A library name cannot be empty.
<a href="#">GB RXX 083 3E</a>	An error occurred when the library name "{0}" was checked. The character "{1}" is not valid.
<a href="#">GB RXX 083 4E</a>	A resource name cannot be empty.
<a href="#">GB RXX 083 5E</a>	An error occurred when the resource name "{0}" was checked. The character "{1}" is not valid.
<a href="#">GB RXX 083 6E</a>	The RuleApp cannot be null.
<a href="#">GB RXX 083 7E</a>	The library "{0}" already exists.
<a href="#">GB RXX 083 8E</a>	A resource provider error occurred when the XOM library "{0}" was added.

<a href="#">GB RXX 083 9E</a>	A resource provider error occurred when the XOM library "{0}" was removed.
<a href="#">GB RXX 084 0E</a>	A resource provider error occurred when the XOM resource "{0}" was added.
<a href="#">GB RXX 084 1E</a>	A resource provider error occurred when the XOM resource "{0}" was removed.
<a href="#">GB RXX 084 2E</a>	The creation of the IlrRESRulesetArchive instance has failed.
<a href="#">GB RXX 084 3E</a>	A ruleset name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 4E</a>	A RuleApp name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 5E</a>	A library name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 6E</a>	A resource name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 7E</a>	A ruleset display name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 8E</a>	A RuleApp display name cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 084 9E</a>	A ruleset description cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 085 0E</a>	A RuleApp description cannot exceed {0} characters: {1} were found.
<a href="#">GB RXX 085 1E</a>	The name attribute of an instance of {0} cannot be null.
<a href="#">GB RXX 085 2E</a>	The version attribute of an instance of {0} cannot be null.
<a href="#">GB RXX 085 3E</a>	Invalid ruleset operation {0}({1}) performed on repository. Possible ones are: REPLACE({2}), CHANGE_VERSION_AND_ADD({3}) and ADD({4}).
<a href="#">GB RXX 085 4E</a>	Invalid library operation {0}({1}) performed on repository. Possible ones are: ALREADY_PRESENT({2}), REPLACE({3}), CHANGE_VERSION_AND_ADD({4}), ADD_AND_UPDATED({5}), ADD({6}) and CHANGE_VERSION_AND_UPDATED({7}).
<a href="#">GB RXX</a>	Invalid RuleApp operation {0}({1}) performed on repository. Possible ones are: REPLACE({2}), CHANGE_VERSION_AND_ADD({3}), ADD({4}) and UPDATE({5}).

<a href="#">0855E</a>	
<a href="#">GB RXX 100 1E</a>	Error when processing the RuleApp archive: {0}.
<a href="#">GB RXX 100 2E</a>	The RuleApp archive is not valid.
<a href="#">GB RXX 100 3E</a>	Invalid ruleset archive type: "{0}".
<a href="#">GB RXX 100 4E</a>	"{0}": The ruleset archive cannot be null.
<a href="#">GB RXX 100 5E</a>	The OutputStream cannot be null.
<a href="#">GB RXX 100 6E</a>	No RuleApp to write.
<a href="#">GB RXX 100 7E</a>	The factory cannot be null.
<a href="#">GB RXX 100 8E</a>	The InputStream cannot be null.
<a href="#">GB RXX 100 9E</a>	System entity not supported.
<a href="#">GB RXX 101 0E</a>	RuleApp "{0}" was not found.
<a href="#">GB RXX 120 1E</a>	The REST request payload must contain the RuleApp representation.
<a href="#">GB RXX 120 2E</a>	The REST request payload must contain the ruleset representation.
<a href="#">GB RXX 120 3E</a>	The REST request payload must contain the property representation.
<a href="#">GB RXX 120 4E</a>	Unknown RuleApp: {0}.
<a href="#">GB RXX 120 5E</a>	Unknown ruleset: {0}.
<a href="#">GB RXX 120 6E</a>	Could not update the following RuleApp because it does not exist: {0}.

<a href="#">GB RXX 120 7E</a>	Could not update the following ruleset because it does not exist: {0}.
<a href="#">GB RXX 120 8E</a>	Could not update the following property because it does not exist: {0}.
<a href="#">GB RXX 120 9E</a>	Could not add the following RuleApp because it already exists: {0}.
<a href="#">GB RXX 121 0E</a>	Could not add the following ruleset because it already exists: {0}.
<a href="#">GB RXX 121 1E</a>	Could not add the following property because it already exists: {0}.
<a href="#">GB RXX 121 2E</a>	Could not delete the following RuleApp because it does not exist: {0}.
<a href="#">GB RXX 121 3E</a>	Could not delete the following ruleset because it does not exist: {0}.
<a href="#">GB RXX 121 4E</a>	Could not delete the following property because it does not exist: {0}.
<a href="#">GB RXX 121 5E</a>	Could not deploy the ruleset archive because it already exists a ruleset with the following path: {0}.
<a href="#">GB RXX 121 6E</a>	The ruleset representation does not specify the ruleset name.
<a href="#">GB RXX 121 7E</a>	Unknown resource: {0}.
<a href="#">GB RXX 121 8E</a>	Unknown library: {0}.
<a href="#">GB RXX 121 9E</a>	Could not add the following managed XOM library because it already exists: {0}.
<a href="#">GB RXX 122 0E</a>	Could not add the following managed XOM because it already exists: {0}.
<a href="#">GB RXX 122 1E</a>	Could not add the managed XOM {0} because the request body is empty.
<a href="#">GB RXX 122 2E</a>	Incorrect version format: {0}.
<a href="#">GB RXX</a>	The RuleApp version number cannot be null. Location: {0}.



<a href="#">122</a> <a href="#">3E</a>	
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">4E</a>	The RuleApp name cannot be null. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">5E</a>	The ruleset version number cannot be null. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">6E</a>	The ruleset name cannot not be null. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">7E</a>	The ruleset archive cannot be null. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">8E</a>	The property identifier cannot be null. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">122</a> <a href="#">9E</a>	Exception while reading the ruleset archive. Location: {0}. Exception: {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">0E</a>	Invalid ruleset archive. Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">1E</a>	The library name cannot be null.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">2E</a>	The library version number cannot be null.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">3E</a>	The referenced URIs are invalid. Example: "resuri://myFirstXOM.jar/1.0, resuri://mySecondXOM.jar, reslib://MyLib/1.0". Location: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">4E</a>	Invalid merging policy: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">5E</a>	Invalid version policy: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">6E</a>	The referenced URIs are invalid. Example: "resuri://myFirstXOM.jar/1.0, resuri://mySecondXOM.jar, reslib://MyLib/1.0".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">7E</a>	The library content cannot be null. Example: "resuri://myFirstXOM.jar/1.0, resuri://mySecondXOM.jar, reslib://MyLib/1.0".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">8E</a>	The RuleApp representation does not specify the version number.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">123</a> <a href="#">9E</a>	The RuleApp must be not null and well formed.
<a href="#">GB</a>	The RuleApp name or version cannot be changed. You have to create a new one

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">0E</a>	The RuleApp name or version cannot be changed. You have to create a new one.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">1E</a>	Invalid library name: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">2E</a>	Invalid resource name: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">3E</a>	Invalid RuleApp version: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">4E</a>	Invalid RuleApp name: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">5E</a>	Could not put the read only property: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">6E</a>	Could not delete the trace with the following execution identifier because it does not exist: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">7E</a>	{0} is not a valid date.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">8E</a>	{0} is not a valid positive number.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">124</a> <a href="#">9E</a>	The query "q" is empty.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">0E</a>	The format of the query "{0}" is incorrect; {1}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">1E</a>	the query containing "{0}"
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">2E</a>	the keyword "{0}" is used multiple times.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">3E</a>	the keywords "{0}" are used multiple times.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">4E</a>	the value of the keyword: "{0}" is empty.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">5E</a>	the values of the keywords: "{0}" are empty.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">6E</a>	the incorrect element is: "{0}".

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">7E</a>	the incorrect elements are: "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">8E</a>	"{0}" is not a valid diagnostic ID.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">125</a> <a href="#">9E</a>	Cannot include the resources {0} in the library. Resources with the same file name cannot be included in the library. The deployed XOMs are removed.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">130</a> <a href="#">1E</a>	Incorrect {0} syntax or Content-Type not set to the correct format: {1}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">1I</a>	SSP server update task
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">2I</a>	Updating the WAR file...
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">3I</a>	{0} filesets to add.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">4I</a>	->Add in {0} XOMFileSet: {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">5I</a>	->Update Log4jFile: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">6I</a>	->Update WebXMLFile: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">7I</a>	->Update RAFile: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">8I</a>	Updating the EAR file...
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">140</a> <a href="#">9I</a>	Using a temporary file: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">0I</a>	->Updated WAR file: jrules-ssp-server.war
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">1I</a>	Using a temporary directory: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">2I</a>	Updating the web.xml file...
<a href="#">GB</a>	When the web.xml file is specified, other updates might not occur.

<a href="#">RXX</a> <a href="#">141</a> <a href="#">3I</a>	
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">4I</a>	->Update ilog.rules.rsm.extensionsFilename: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">5I</a>	Generating the ra.xml file...
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">6I</a>	Using the specified ra.xml file {0} and copying it to {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">7E</a>	Could not find {0} in file {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">8E</a>	Too many entries during search of {0} in file {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">141</a> <a href="#">9E</a>	Unknown artifact type: {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">142</a> <a href="#">0E</a>	The raFile property must be set.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">142</a> <a href="#">1E</a>	{0}: File {1} does not exist.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">142</a> <a href="#">2E</a>	{0}: {1} is not a file.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">142</a> <a href="#">3E</a>	Bad value for parameter {0}. The value should be true or false instead of {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">142</a> <a href="#">4E</a>	Bad value for parameter {0}. The value should be {1} or {2} instead of {3}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">145</a> <a href="#">1I</a>	Rule Execution Server setup - {0}
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">145</a> <a href="#">2I</a>	No Rule Execution Server console EAR file is specified.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">145</a> <a href="#">3I</a>	Transforming the XU descriptor file {0} into {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">145</a> <a href="#">4I</a>	No execution unit (XU) RAR file is specified.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">145</a> <a href="#">5I</a>	Change persistence type {0}.

<a href="#">GB RXX 145 6I</a>	Creating the XU instance ...
<a href="#">GB RXX 145 7I</a>	No Rule Execution Server console WAR file is specified.
<a href="#">GB RXX 145 8I</a>	Creating the Rule Execution Server console WAR file: {0} => {1} ...
<a href="#">GB RXX 145 9W</a>	The persistenceMode attribute is deprecated and replaced by persistenceType.
<a href="#">GB RXX 146 0W</a>	The Ant task attribute {0} is deprecated and replaced by {1}.
<a href="#">GB RXX 146 1E</a>	Unknown persistence type {0}.
<a href="#">GB RXX 146 2E</a>	Cannot get: {0}.
<a href="#">GB RXX 146 3E</a>	The {0} property is not supported.
<a href="#">GB RXX 146 4E</a>	Not a JCA 1.5 resource-adapter configuration file.
<a href="#">GB RXX 146 5E</a>	Unknown XOM persistence type {0}.
<a href="#">GB RXX 146 6E</a>	The metering plug-in property {0} should be set to value true or false.
<a href="#">GB RXX 146 7E</a>	The metering plug-in property {0} should be set to a valid URL.
<a href="#">GB RXX 146 8E</a>	The required metering plug-in property {0} is missing in the configuration.
<a href="#">GB RXX 146 9I</a>	No WAR file is specified to configure Rule Execution Server for Cloud Product Insights.
<a href="#">GB RXX 147 0I</a>	Configuring the Rule Execution Server WAR file for Cloud Product Insights: {0} => {1} ...
<a href="#">GB RXX 147 1E</a>	The metering plug-in property {0} should be a valid hostname format. Value is {1}.
<a href="#">GB RXX</a>	The parameter {0} is not supported for the ressetup.

<a href="#">147</a> <a href="#">2E</a>	
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">3E</a>	Use of the parameter {0} is required to set the parameter {1}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">4E</a>	The properties "{0}" and "{1}" have the same value "{2}". Set different values for them.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">5I</a>	Configuring the Rule Execution Server WAR file for emitting events to IBM Business Automation Insights (BAI): {0} => {1} ...
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">6I</a>	No WAR file is specified to configure Rule Execution Server for emitting events to IBM Business Automation Insights (BAI).
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">7E</a>	Missing parameter bai.properties.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">147</a> <a href="#">8E</a>	Configuration file {0} for BAI plugin cannot be read.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">160</a> <a href="#">1E</a>	Cannot find key {1} for resource bundle {0}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">160</a> <a href="#">2E</a>	"{0}" could not be translated into "{1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">160</a> <a href="#">3E</a>	Not implemented for SAXResult.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">160</a> <a href="#">4E</a>	Cannot create a temporary directory.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">160</a> <a href="#">5E</a>	No trust manager found.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">170</a> <a href="#">1E</a>	Register MBean failed with type "{0}", properties "{1}", and for the object {2}.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">170</a> <a href="#">2E</a>	Unregister MBean failed for the objectname "{0}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">170</a> <a href="#">3E</a>	Unregister all MBeans failed.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">170</a> <a href="#">4E</a>	The MBean factory cannot be null.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">1E</a>	Error retrieving the warning count for execution unit: "{0} - {1}".

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">2E</a>	Error retrieving the error count for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">3E</a>	Error resetting the warning count for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">4E</a>	Error resetting the error count for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">5E</a>	Error notifying a ruleset archive change for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">6E</a>	Error resetting statistics for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">7E</a>	Error retrieving the statistics for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">8E</a>	Error retrieving the adapter name for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">171</a> <a href="#">9E</a>	Error retrieving the adapter version for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">0E</a>	Error retrieving the adapter short version for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">1E</a>	Error retrieving the adapter vendor name for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">2E</a>	Error retrieving the global logs for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">3E</a>	Error retrieving the logs for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">4E</a>	Error performing the execute test for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">5E</a>	Error resetting the logs for execution unit: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">6E</a>	Error retrieving the ruleset path for decision service: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">172</a> <a href="#">7E</a>	Error retrieving the state for decision service: "{0} - {1}".
<a href="#">GB</a> <a href="#">RXX</a>	Error setting activated attribute for decision service: "{0} - {1}".

<a href="#">1728E</a>	
<a href="#">GB RXX 1729E</a>	Error setting a property for decision service: "{0} - {1}".
<a href="#">GB RXX 1730E</a>	Error retrieving the debug info for execution unit: "{0} - {1}".
<a href="#">GB RXX 1731E</a>	Error removing decision service: "{0} - {1}".
<a href="#">GB RXX 1732E</a>	XU Manager construction error.
<a href="#">GB RXX 1733W</a>	Cannot set the read only property "{0}" to the decision engine ruleset.
<a href="#">GB RXX 1734I</a>	RESMGMT {1}: Adding ruleset "{0}".
<a href="#">GB RXX 1735I</a>	RESMGMT {1}: Ruleset "{0}" is added.
<a href="#">GB RXX 1736I</a>	RESMGMT {1}: Changing ruleset "{0}".
<a href="#">GB RXX 1737I</a>	RESMGMT {1}: Ruleset "{0}" is changed.
<a href="#">GB RXX 1738I</a>	RESMGMT {1}: Removing ruleset "{0}".
<a href="#">GB RXX 1739I</a>	RESMGMT {1}: Ruleset "{0}" is removed.
<a href="#">GB RXX 1740I</a>	RESMGMT {2}: Changing ruleset property "{1}" in "{0}".
<a href="#">GB RXX 1741I</a>	RESMGMT {2}: Ruleset property "{1}" is changed in "{0}".
<a href="#">GB RXX 1742I</a>	RESMGMT {3}: Sending the notification {4} "ruleset {0} is changed" from "{1}" to all execution units: {2}.
<a href="#">GB RXX 1743I</a>	RESMGMT {3}: Notification {4} "ruleset {0} is changed" is sent from "{1}" to all execution units: {2}.
<a href="#">GB RXX 1744I</a>	RESMGMT {2}: Sending the notification "ruleset {0} is changed" to execution unit: {1}.
<a href="#">GB</a>	RESMGMT {2}: Notification "ruleset {0} is changed" is sent to execution unit: {1}.



<a href="#">RX</a> <a href="#">174</a> <a href="#">5I</a>	
<a href="#">GB</a> <a href="#">RX</a> <a href="#">174</a> <a href="#">6I</a>	RESMGMT {2}: Sending the notification {3} "ruleset {0} is changed" to execution unit: {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">174</a> <a href="#">7I</a>	RESMGMT {2}: Notification {3} "ruleset {0} is changed" is sent to execution unit: {1}.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">174</a> <a href="#">8W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRepository.addRuleApp(String ruleAppName, String ruleAppVersion)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">174</a> <a href="#">9W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRepositoryMBean.importRuleApp((byte[]) ruleAppArchive, MergePolicy mergingPolicy, VersionPolicy versioningPolicy)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">0W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRuleAppMBean.addRuleset(String rulesetName, String rulesetVersion, byte[] rulesetArchive)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">1W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRuleAppMBean.addRuleset(String rulesetName, String rulesetVersion, String engineType, byte[] rulesetArchive)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">2W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRuleAppMBean.export()</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">3W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRuleAppMBean.removeRuleset(String rulesetName, String rulesetVersion)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">4W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRuleAppMBean.setProperty(String key, String value)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">5W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXEntityMBean.setDescription(String description)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">6W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXEntityMBean.setDisplayName(String displayName)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">7W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRulesetMBean.setRulesetArchive(byte[] rulesetArchive)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">8W</a>	RESMGMT {0}: As of ODM 8.10.2, the method <code>IlrJMXRulesetMBean.setRESRulesetArchive(String engineType, byte[] content)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">175</a> <a href="#">9W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRulesetMBean.setProperty(String key, String value)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">176</a> <a href="#">0W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRulesetMBean.setStatus(String status)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RX</a> <a href="#">176</a> <a href="#">1W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRulesetMBean.getRulesetArchive()</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.

<a href="#">GB</a> <a href="#">RXX</a> <a href="#">176</a> <a href="#">2W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRulesetMBean.getRESRulesetArchive()</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">176</a> <a href="#">3W</a>	RESMGMT {0}: As of ODM V8.10.2, the method <code>IlrJMXRepository.removeRuleApp(String ruleAppName, String ruleAppVersion)</code> from the Rule Execution Server console management JMX API is deprecated. Use the Rule Execution Server console management REST API instead.
<a href="#">GB</a> <a href="#">RXX</a> <a href="#">180</a> <a href="#">1E</a>	An instance of class "{0}" was created when parsing the REST request. This class is not compatible with the expected type "{1}".

## Credits

The following people have contributed to this product:

Adam Davis

Adrian Vasiliu

Alain Robert

Alexander Kofman

Alexandre Bronetsky

Andy Hebb

Anthony Acremann

Anthony Damiano

Antoine Melki

Antony Viaud

Arun Iyengar

Asma Alibert

Barbara Ressel

Benjamin Herta

Benjamin Ratiarisolo

Benoit Poupon

Catherine Tchong

Cédric Doens

Chabane Hama

Changhai Ke

Chris Backhouse

Christian Leclerc

Christiane Mosbach

Christophe Facquez

Christopher Dolloff

David Audel

David Grove

Didier Vidal

Dominique Breheret

Emi Nakamura

Emmanuel Tissandier

Eric Aubineau

Eric Durocher

Eric Mazeran

Fanny Bischoff

Franck Troalen

François Gadrat

François Tribble

Françoise Tith

Frank Wagner

Fred Desruelle

Frédéric Fusier

Gabriel Rousseau

Gael Crova

Gisèle Eisenmann

Gregory Cuny

Guilhem Molines

Harold Ship

Hugues Citeau

Hui Sajus

Isabelle Murru

James Lentz

James Taylor

Jean-Louis Ardoint

Jean-Louis Sivade

Jean-Michel Bernelas

Jean-Philippe Orsini

Jean-Philippe Vandara

Jeffrey Mariasine

Jérôme Delattre

Johanne Sebaux

Julie Garrone

Julie Tapp

Justine Mills

Laureen Ginier

Laurent Cohen

Laurent Extier

Laurent Grateau

Laurent Villar

Lionel Péron

Lucinda Croft

Margaux Chatelain

Marie Girard

Marie-Françoise Lim

Mark Hiscock

Mark Wilkinson

Mathias Mouly

Melanie Shilpa Rao

Meryanti Tjahyadi

Michael Obadia

Michel Leconte

Mike Johnson

Moussa Mahamadou Yattara

Myriam Batelli

Neil Simpson

Nicolas Carré

Nicolas Duboc

Nicolas Mériaux

Nicolas Peulvast

Nicolas Sarroche

Nicolas Sauterey

Nidia Pinzon Cardenas

Olivier Artigue

Olivier Tardieu

Pascal Ruault

Pascale Dardailler

Patrick Merlin

Peter Gilliver

Philippe Bonnard

Philippe Debras

Philippe Guérin

Pierre Feillet

Pierre-André Paumelle

Pierre Pisot

Pierre-Yves Lochou

Rachel Orti

Rémi Lejeune

Rémi Vankeisbelck

Roselyne Broudeur

Ruth Church

Sabrina Lautier

Said Moutaa

Sean Gleason

Sébastien Grégoire

Sébastien Péron

Sébastien Värnild

Serigne Mbaye

SiaSin Tāy

Sophie Herbin

Stéphane Hillion

Stéphane Mery

Stéphane Rouyer

Sylvain Dehors

Thierry Kormann

Toibibou Houmadi

Uzma Siddiqui

Victoria Genin

Xavier Fabre

Xiaoming Zhang

Yveline Gosselin