WebSphere® Process Server for Multiplatforms

Version 6.2.0

Developing and Deploying Modules

WebSphere® Process Server for Multiplatforms

Version 6.2.0

Developing and Deploying Modules

Note

Before using this information, be sure to read the general information in the Notices section at the end of this document.

24 April 2009

This edition applies to version 6, release 2, modification 0 of WebSphere Process Server for Multiplatforms (product number 5724-L01) and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, send an e-mail message to doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2005, 2009.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

PDF books and the information center

PDF books are provided as a convenience for printing and offline reading. For the latest information, see the online information center.

As a set, the PDF books contain the same content as the information center.

The PDF documentation is available within a quarter after a major release of the information center, such as Version 6.0 or Version 6.1.

The PDF documentation is updated less frequently than the information center, but more frequently than the Redbooks®. In general, PDF books are updated when enough changes are accumulated for the book.

Links to topics outside a PDF book go to the information center on the Web. Links to targets outside a PDF book are marked by icons that indicate whether the target is a PDF book or a Web page.

Table 1. Icons that prefix links to topics outside this book

Icon	Description

	A link to a Web page, including a page in the information center.
	
	Links to the information center go through an indirection routing service, so that they continue to work even if target topic is moved to a new location.
	
	If you want to find a linked page in a local information center, you can search for the link title. Alternatively, you can search for the topic id. If the search results in several topics for different product variants, you can use the search result **Group by** controls to identify the topic instance that you want to view. For example:
	1. Copy the link URL; for example, right-click the link then select Copy link location. For example: `http://www14.software.ibm.com/webapp/wsbroker/redirect?version=wbpm620&product=wesb-dist&topic=tins_apply_service`
	2. Copy the topic id after `&topic=`. For example: `tins_apply_service`
	3. In the search field of your local information center, paste the topic id. If you have the documentation feature installed locally, the search result will list the topic. For example:
	`1 result(s) found for`
	`Group by: None │ Platform │ Version │ Product`
	`Show Summary`
	`Installing fix packs and refresh packs with the Update Installer`
	4. Click the link in the search result to display the topic.

	A link to a PDF book.

© Copyright IBM Corp. 2005, 2009

Contents

PDF books and the information center iii

Figures vii

Tables ix

Part 1. Developing applications . . . 1

Chapter 1. Developing business integration solutions 3
Business integration programming model. 5
Business integration architecture and patterns . . . 6
  Business integration scenarios 6
  Roles, products, and technical challenges 7
  The business object framework 8
  Service component architecture 10
  Business processes 15
  Human tasks 15
Building business integration applications 15

Chapter 2. Developing service modules 17
Overview of developing modules 17
Developing service components 19
Invoking components 20
Dynamically invoking a component 22
Overview of isolating modules and targets 24
HTTP bindings 27

Chapter 3. Programming guides and techniques 29
Programming business objects 29
  Arrays in business objects 29
  Creating nested business objects 30
  Business objects: schema enhancement and industry schema support 34
  Overriding a Service Data Object to Java conversion. 52
  Overriding the generated Service Component Architecture implementation. 53
  Runtime rules used for Java to Service Data Objects conversion 54
XML document validation 55
Protocol header propagation from non-SCA export bindings 57
Business rule management 59
  Programming model 60
  Examples 86
  Common operations classes 152

Chapter 4. Developing client applications for business processes and tasks 163
Comparison of the programming interfaces for interacting with business processes and human tasks 163
Queries on business process and task data . . . 165
  Comparison of the programming interfaces for retrieving process and task data 165
  Query tables in Business Process Choreographer 167
  Business Process Choreographer EJB query API 217
Developing EJB client applications for business processes and human tasks. 229
  Accessing the EJB APIs 230
  Querying business-process and task-related objects. 236
  Developing applications for business processes 240
  Developing applications for human tasks . . . 260
  Developing applications for business processes and human tasks 277
  Handling exceptions and faults 283
Developing Web service API client applications . . 286
  Web service components and sequence of control 286
  Overview of the Web services APIs 287
  Requirements for business processes and human tasks 287
  Developing client applications. 288
  Copying artifacts 288
  Developing client applications in the Java Web services environment 295
  Developing client applications in the .NET environment. 304
  Querying business-process and task-related objects. 309
Developing client applications using the Business Process Choreographer JMS API 312
  Requirements for business processes. 312
  Authorization for JMS renderings 313
  Accessing the JMS interface 313
  Copying artifacts for JMS client applications . . 316
  Checking the response message for business exceptions 317
  Example: executing a long running process using the Business Process Choreographer JMS API. 317
Developing Web applications for business processes and human tasks, using JSF components . 318
  Business Process Choreographer Explorer components 321
  Error handling in JSF components 322
  Default converters and labels for client model objects. 323
  Adding the List component to a JSF application 324
  Adding the Details component to a JSF application 330
  Adding the CommandBar component to a JSF application 332

© Copyright IBM Corp. 2005, 2009

Adding the Message component to a JSF application 336
Developing JSP pages for task and process messages 339
User-defined JSP fragments. 340
Creating plug-ins to customize human task functionality. 341
Creating API event handlers 341
Creating notification event handlers 344
Installing API event handler and notification event handler plug-ins 345
Registering API event handler and notification event handler plug-ins with task templates, task models, and tasks 346
Creating, installing, and running plug-ins to post-process people query results. 347

Part 2. Deploying applications. . . . 351

Chapter 5. Overview of preparing and installing modules 353
Libraries and JAR files overview 353
EAR file overview. 355
Preparing to deploy to a server 355
Considerations for installing service applications on clusters 357

Chapter 6. Deploying a module. . . . 359
Installing versioned SCA modules in a production environment. 359
Installing an SCA module with the console . . . 361
Creating an installable EAR file using serviceDeploy 362
Deploying applications using Apache Ant tasks 362

Chapter 7. Installing business process and human task applications 365
How business process and human task applications are installed in a network deployment environment 365
Deployment of business processes and human tasks 366
Installing business process and human task applications interactively 366
Configuring process application data source and set reference settings 367
Uninstalling business process and human task applications, using the administrative console . . 368
Uninstalling business process and human task applications, using an administrative command . . 369

Chapter 8. Adapters and their installation 371

Chapter 9. Troubleshooting a failed deployment 373
Deleting J2C activation specifications 374
Deleting SIBus destinations. 375

Part 3. Appendixes 377

Notices 379

Figures

1. IBM tools span the entire BPM life cycle, enabling you to model, assemble, deploy and manage your processes. 4
2. WebSphere Process Server component-based framework 11
3. SCA in WebSphere Process Server 12
4. Assembly diagram 13
5. Simple invocation model 24
6. Multiple applications invoking a single service 25
7. Isolated invocation model invoking UpdateCalculateFinal 26
8. Isolated invocation model invoking UpdatedCalculateFinal 27
9. Propagating context including protocol header 58
10. Class diagram of BusinessRuleGroup and related classes. 62
11. Class diagram of Property and related classes 63
12. Class diagram for Operation and related classes 66
13. Class diagram of BusinessRule and related classes 67
14. Class diagram of BusinessRule and related classes 69
15. Class diagram of DecisionTable and related classes 70
16. Class diagram for TreeNode and related classes 73
17. Class diagram of TreeAction and related classes 76
18. Class diagram for DecisionTableRule and related classes. 77
19. Class diagram for Template and Parameter and related classes. 79
20. Class diagram for BusinessRuleManager and package 80
21. Class diagram for QueryNodeFactory and related classes. 82
22. Class diagram for BusinessRuleManagementException and related classes. 84
23. Query tables in Business Process Choreographer 167
24. Composite query table content 174
25. Composite query table with selection criteria 176
26. Filters in composite query tables 179
27. Filters and selection criteria in expressions 186
28. Instance-based authorization for query tables 191
29. Relationship amongst module, component and library 354

© Copyright IBM Corp. 2005, 2009

Tables

1. Icons that prefix links to topics outside this book iii
2. Data abstractions and the corresponding implementations 9
3. WSDL type to Java class conversion 55
4. Business Rule Group problems 84
5. Rule set and Decision Table problems 85
6. Properties of predefined query tables 169
7. Predefined query tables containing instance data. 170
8. Predefined query tables containing template data. 170
9. Properties of supplemental query tables 172
10. Valid contents of a composite query table 177
11. Invalid contents of a composite query table 177
12. Properties of composite query tables 177
13. Query table development steps 181
14. Attributes for query table expressions 185
15. Types of authorization for query tables 190
16. Work item types 192
17. Work items and people assignment criteria 193
18. Attribute types 194
19. Database type to attribute type mapping 195
20. Database types to attribute types mapping example 195
21. Attribute type to literal values mapping 196
22. Attribute type to user parameter values mapping 197
23. Attribute type to Java object type mapping 198
24. Attribute type compatibility. 199
25. Methods for queries run on query tables 200
26. Parameters of the query table API 202
27. Query table API parameters: Filter options 203
28. Query table API parameters: Authorization option defaults for instance-based authorization 205
29. Query table API parameters: AdminAuthorizationOptions 206
30. User parameters for the query table API 207
31. Entity result set properties of a query table API entity. 208
32. Entity properties of a query table API entity 208
33. Row result set properties of a query table API row 209
34. Methods for meta data retrieval on query tables 210
35. Meta data related to query table structure 211
36. Meta data related to query table internationalization 211
37. Query performance impact of composite query table options 214
38. Query performance impact of query table API options 215
39. Query table performance: Other considerations 216
40. 217
41. API methods for process templates 258
42. API methods are related to starting process instances 258
43. API methods for controlling the life cycle of process instances 258
44. API methods for controlling the life cycle of activity instances 259
45. API methods for variables and custom properties. 260
46. API methods for task templates 275
47. API methods for task instances 275
48. API methods for working with escalations 276
49. API methods for variables and custom properties. 276
50. Mapping of the reference bindings to JNDI names 320
51. How Business Process Choreographer interfaces are mapped to client model objects . 323
52. bpe:list attributes 329
53. bpe:column attributes 330
54. bpe:details attributes 331
55. bpe:property attributes 332
56. bpe:commandbar attributes 335
57. bpe:command attributes 336
58. bpe:form attributes. 339

© Copyright IBM Corp. 2005, 2009

Part 1. Developing applications

© Copyright IBM Corp. 2005, 2009

Chapter 1. Developing business integration solutions

This section discusses the fundamentals of the business integration programming model. It introduces the Service Component Architecture (SCA) and discusses patterns related to business integration.

Business integration is the discipline that enables companies to identify, consolidate, and optimize business processes. The objective is to improve productivity and maximize organizational effectiveness. Interest in business integration has become more acute as companies merge and consolidate, and as they grow a library of disparate information assets. These assets often lack consistency and coordination, thus giving rise to “islands of information.”

Business integration has strong links to Business Process Management (BPM) and Service-Oriented Architecture (SOA). Depending on the nature of the company and the extent of the integration needs, business integration poses different requirements for IT departments. Some projects may deal with only a few aspects, whereas some larger projects may encompass many of these requirements. Here are some of the most common aspects of business integration projects:

- **Application integration** is a common requirement. The complexity of application integration projects varies from simple cases, in which you need to ensure that a small number of applications can share information, to more complex situations, in which transactions and data exchanges need to be reflected simultaneously on multiple back-end applications. Complex application integration often requires complex unit-of-work management as well as transformation and mapping.
- **Process automation** is another key aspect that ensures that activities performed by an individual or organization systematically trigger consequential activities elsewhere. This ensures the successful completion of the overall business process. For example, when a company hires an employee, payroll information has to be updated, appropriate actions need to be taken by the security department, the necessary tools need to be given to the employee, and so on. Some activities in a process might capture human input and interaction, whereas others might invoke scripts on back-end systems and other services in the environment.
- **Connectivity** is an abstract, yet critical, aspect both in a company and in terms of business partners. By connectivity, we mean both the flow of information between organizations or companies and the ability to access distributed IT services.

Some of the technical challenges of business integration implementations can be summarized as follows:

- Dealing with different data formats and therefore not being able to perform efficient data transformation
- Dealing with different protocols and mechanisms for accessing IT services that may have been developed using very different technologies
- Orchestrating different IT services that may be geographically distributed or offered by different organizations
- Providing rules and mechanisms to classify and manage the services that are available (governance)

© Copyright IBM Corp. 2005, 2009

As such, business integration encompasses many of the themes and elements that are also common to SOA. IBM's vision of business integration builds on many of the same foundational concepts that are found in SOA. One of the immediate consequences of this vision is that business integration solutions may require a variety of products for their realization. IBM® provides a portfolio of tools and runtime platforms to support all the various stages and operational aspects.

To paraphrase IBM's vision of business integration, it should enable companies to define, create, merge, consolidate, and streamline business processes using applications that run on a SOA IT infrastructure. Business integration work is truly role-based. At the macro level, it involves modeling, developing, governance, managing, and monitoring business process applications. With the help of proper tools and procedures, it enables you to automate business processes involving people and heterogeneous systems, both inside and outside the enterprise. One of the key aspects of business integration is the ability to optimize your business operations so that they are efficient, scalable, reliable, and flexible enough to handle change.

Business integration requires development tools, runtime servers, monitoring tools, a service repository, toolkits, and process templates. Because there are so many aspects to business integration, you will find that you have to utilize more than one development tool to develop a solution. These tools enable integration developers to assemble complex business solutions. A server is a high-performance business engine or service container that runs complex applications. Management always wants to know who is doing what in the organization, and that is where monitoring tools come into play. As enterprises create these business processes or services, governance, classification, and storage of these services becomes critical. That function is served by a service repository. Specific toolkits to create specialized parts of the solution, such as connectors or adapters to existing systems, are often required.

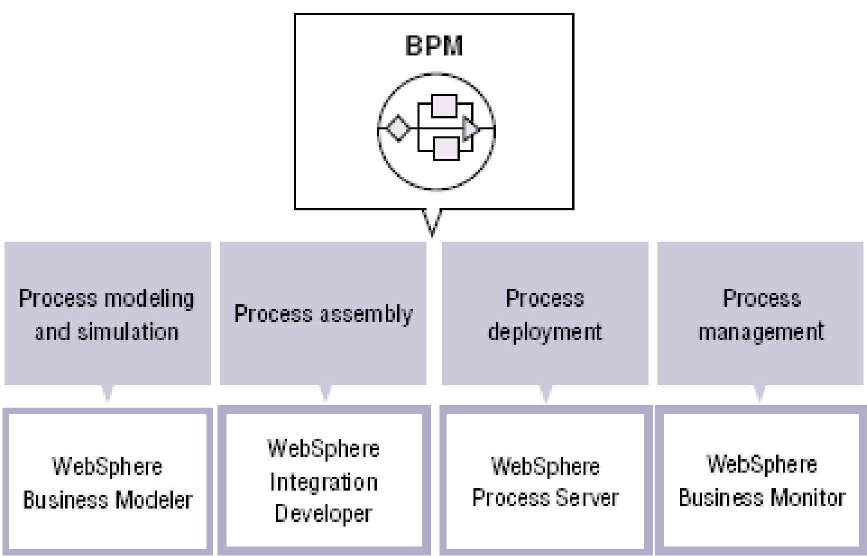


Figure 1. IBM tools span the entire BPM life cycle, enabling you to model, assemble, deploy and manage your processes.

Business integration is not based on a single product. It involves almost everybody and all business aspects within an organization and across organizations. Business integration encompasses many of the services and elements in the SOA reference architecture.

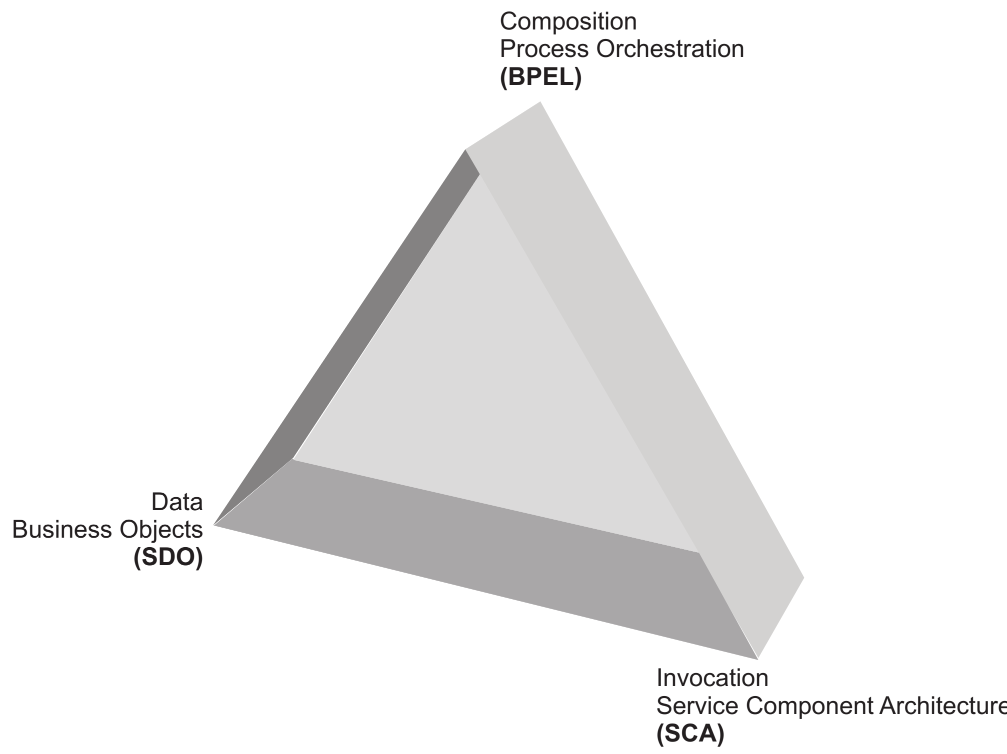
For more details about these concepts, along with programming examples, see:

- *WebSphere® Business Integration Primer: Process Server, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1: Development*, IBM Redbooks, SG24-7608-00, June 2008.

Business integration programming model

Business integration is not an easy task. There are so many technologies and so many ways to represent or interact with data that achieving integration is no easy task. If you take the three aspects of a programming model, which are data, invocation, and composition, and apply some of the new paradigms of a services-based approach, the new programming model for SOA starts to emerge.

First, we see that data is primarily represented by Extensible Markup Language (XML) and is programmed with Service Data Objects (SDOs) or through native XML facilities such as XPath or XSLT (Extensible Stylesheet Language Transformation). Second, service invocation maps to Service Component Architecture (SCA). Finally, composition is embodied in process orchestration using Business Process Execution Language (BPEL). The figure shows the three aspects of this new programming model.



Service Component Architecture

In addition to providing a consistent syntax and mechanism for service invocation, the SCA is the invocation framework that provides a way for developers to encapsulate service implementations in reusable components. It enables developers to define interfaces, implementations, and references in a technology-agnostic way, giving you the opportunity to bind the elements to whichever technology you choose. SCA separates business logic from infrastructure so that application programmers can focus on solving business problems.

Business integration architecture and patterns

A typical business integration project involves coordinating several different IT assets, potentially running on different platforms, and having been developed at different times using different technologies. Being able to easily manipulate and exchange information with a diverse set of components is a major technical challenge. It is best addressed by the programming model used to develop business integration solutions.

This section introduces the Service Component Architecture (SCA) and discusses patterns related to business integration. Patterns seem to permeate our lives. Sewing patterns, think-and-learn patterns for children, home construction patterns, wood-carving patterns, flight patterns, wind patterns, practice patterns in medicine, customer buying patterns, workflow patterns, design patterns in computer science, and many more exist.

Patterns have proven successful in helping solution designers and developers. Therefore, it is not surprising that we now have business integration patterns and enterprise integration patterns. There are a wide array of patterns that are applicable to business integration, including patterns for request and response routing, channel patterns (such as publish/subscribe), and many more. Abstract patterns provide a template for resolving a certain category of problems, whereas concrete patterns provide more specific indications of how to implement a specific solution. This section focuses on patterns that deal with data and service invocation, which are at the foundation of the programming model of the IBM software strategy for WebSphere business integration.

Business integration scenarios

Enterprises have many different software systems that they use to run their business. In addition, they have their own ways of integrating these business components.

The two most prevalent business process integration scenarios are as follows:

- **Integration broker**: In this scenario, the business integration solution acts as an intermediary located among a variety of “back-end” applications. For example, you might need to ensure that when a customer places an order using the online order management application, the transaction updates relevant information in your Customer Relationship Management (CRM) back end. In this scenario, the integration solution needs to be able to capture and possibly transform the necessary information from the order management application and invoke the appropriate services in the CRM application.
- **Process automation**: In this scenario, the integration solution acts as the glue among different IT services that would otherwise be unrelated. For example, when a company hires an employee, the following sequence of actions needs to occur:

– The employee's information is added to the payroll system.
– The employee needs to be granted physical access to the facilities, and a badge needs to be provided.
– The company might need to assign a set of physical assets to the employee (office space, a computer, and so on).
– The IT department needs to create a user profile for the employee and grant access to a series of applications.

  Automating this process is also a common use case in a business integration scenario. In this scenario, the solution implements an automated flow that is triggered by the employee's addition to the payroll system. Subsequently, the flow triggers the other steps by creating work items for the people who are responsible for taking action or by calling the appropriate services.

In both scenarios, the integration solution needs to do the following:

1. Work with disparate sources of information and different data formats, and be able to convert information between different formats
2. Be able to invoke a variety of services, potentially using different invocation mechanisms and protocols.

Roles, products, and technical challenges

Successful business integration projects depend on the blending of specialized development roles, programming techniques, and tool suites.

Business integration projects require a few basic ingredients:

- A clear separation of roles in the development organization to promote specialization, which typically improves the quality of the individual components that are developed
- A common business object (BO) model that enables business information to be represented in a common logical model
- A programming model that strongly separates interfaces from implementations and that supports a generic service invocation mechanism that is totally independent of the implementation and that only involves dealing with interfaces
- An integrated set of tools and products that supports development roles and preserves their separation

The following sections elaborate on each of these ingredients.

Clear Separation of Roles

A business integration project requires people in four collaborative, but distinctly separate, roles:

- **Business analyst**: Business analysts are domain experts responsible for capturing the business aspects of a process and for creating a process model that adequately represents the process itself. Their focus is to optimize the financial performance of a process. Business analysts are not concerned with the technical aspects of implementing processes.
- **Component developer**: Component developers are responsible for implementing individual services and components. Their focus is the specific technology used for the implementation. This role requires a strong programming background.
- **Integration specialist**: This relatively new role describes the person who is responsible for assembling a set of existing components into a larger business

integration solution. Integration developers do not need to know the technical details of each of the components and services they reuse and wire together. Ideally, integration developers are concerned only with understanding the interfaces of the services that they are assembling. Integration developers should rely on integration tools for the assembly process.
- **Solution deployer**: Solution deployers and administrators are concerned with making business integration solutions available to end users. Ideally, a solution deployer is primarily concerned with binding a solution to the physical resources ready for it to function (databases, queue managers, and so on) and not with having a deep understanding of the internals of a solution. The solution deployer's focus is quality of service (QoS).

A Common Business Object Model

As we discussed, the key aspects of a business integration project include the ability to coordinate the invocation of several components and the ability to handle the data exchange among those. In particular, different components can use different techniques to represent business items such as the data in an order, a customer's information, and so on. For example, you might have to integrate a Java™ application that uses entity Enterprise Java Beans (EJBs) to represent business items and an existing application that organizes information in COBOL copybooks. Therefore, a platform that aims to simplify the creation of integration solutions should also provide a generic way to represent business items, irrespective of the techniques used by the back-end systems for data handling. This goal is achieved in WebSphere Process Server and WebSphere Enterprise Service Bus thanks to the *business object framework*.

The business object framework enables developers to use XML Schemas to define the structure of business data and access and manipulate instances of these data structures (business objects) via XPath or Java code. The business object framework is based on the Service Data Object (SDO) standard.

The Service Component Architecture (SCA) Programming Model

The SCA programming model represents the foundation for any solution to be developed on WebSphere Process Server and WebSphere Enterprise Service Bus. SCA provides a way for developers to encapsulate service implementations in reusable components. It enables you to define interfaces, implementations, and references in a technology-agnostic way, giving you the opportunity to bind the elements to whichever technology you choose. There is also an SCA client programming model that enables the invocation of those components. In particular, it enables runtime infrastructures based on Java to interact with non-Java runtimes. SCA uses business objects as the data items for service invocation.

Tools and Products

IBM WebSphere Integration Developer is the integrated development environment that has all the necessary tools to create and compose business integration solutions based on the technologies just mentioned. These solutions typically are deployed to the WebSphere Process Server or, in some cases, to the WebSphere Enterprise Service Bus.

The business object framework

The computer software industry has developed several programming models and frameworks that enable developers to encapsulate business object (BO)

information. In general, a BO framework should provide database independence, transparently map custom business objects to database tables or to data structures in enterprise information systems, and bind business objects to user interfaces. Of late, XML schemas are perhaps the most popular and accepted way to represent the structure of a business object.

From a tooling perspective, WebSphere Integration Developer provides developers with a common BO model for representing different kinds of entities from different domains. At development time, WebSphere Integration Developer represents business objects as XML schemas. At runtime, however, those same business objects are represented in memory by a Java instance of an SDO. SDO is a standard specification that IBM and BEA Systems have jointly developed and agreed on. IBM has extended the SDO specification by including some additional services that facilitate the manipulation of data within the business objects.

Before we get into the BO framework, let's look at the basic types of data that get manipulated:

- **Instance data** is the actual data and data structures, from simple, basic objects with scalar properties to large, complex hierarchies of objects. This also includes data definitions such as a description of the basic attribute types, complex type information, cardinality, and default values.
- **Instance metadata** is instance-specific data. Incremental information is added to the base data, such as change tracking (also known as change summary), context information associated with how the object or data was created, and message headers and footers.
- **Type metadata** is usually application-specific information, such as attribute-level mappings to destination enterprise information system (EIS) data columns (for example, mapping a BO field name to a SAP table column name).
- **Services** are basically helper services that get data, set data, change summary, or provide data definition type access.

The table shows how the basic types of data are implemented in the WebSphere platform.

Table 2. Data abstractions and the corresponding implementations

Data Abstraction	Implementation
Instance data	Business object (SDO)
Instance metadata	Business graph
Type metadata	Enterprise metadata, Business object type metadata
Services	Business object services

Working with the IBM business object framework

As we mentioned, the WebSphere Process Server BO framework is an extension of the SDO standard. Therefore, business objects exchanged between WebSphere Process Server components are instances of the `commonj.sdo.DataObject` class. However, the WebSphere Process Server BO framework adds several services and functions that simplify and enrich the basic DataObject functionality.

To facilitate the creation and manipulation of business objects, the WebSphere BO framework extends SDO specifications by providing a set of Java services. These services are part of the package named `com.ibm.websphere.bo`:

- **BOFactory**: The key service that provides various ways to create instances of business objects.
- **BOXMLSerializer**: Provides ways to ″inflate″ a business object from a stream or to write the content of a business object, in XML format, to a stream.
- **BOCopy**: Provides methods that make copies of business objects (″deep″ and ″shallow″ semantics).
- **BODataObject**: Gives you access to the data object aspects of a business object, such as the change summary, the business graph, and the event summary.
- **BOXMLDocument**: The front end to the service that lets you manipulate the business object as an XML document.
- **BOChangeSummary** and **BOEventSummary**: Simplifies access to and manipulation of the change summary and event summary portion of a business object.
- **BOEquality**: A service that enables you to determine whether two business objects contain the same information. It supports both deep and shallow equality.
- **BOType** and **BOTypeMetaData**: These services materialize instances of `commonj.sdo`.Type and let you manipulate the associated metadata. Instances of Type can then be used to create business objects ″by type″.
- **BOInstanceValidator** : Validates the data in a business object to see if it conforms to the XSD.

Service component architecture

SCA is an abstraction you can implement in many different ways. It does not mandate any particular technology, programming language, invocation protocol, or transport mechanism. SCA components are described using Service Component Definition Language (SCDL), which is an XML-based language.

An SCA component has the following characteristics:

- It wraps an implementation artifact, which contains the logic that the component can execute.
- It exposes one or more interfaces.
- It can expose one or more references to other components. The implementation's logic determines whether a component exposes a reference. If the implementation requires invoking other services, the SCA component needs to expose a reference.

This information focuses on the SCA implementation that WebSphere Process Server offers and the WebSphere Integration Developer tool that is available to create and combine SCA components. WebSphere Process Server and WebSphere Integration Developer support the following implementation artifacts:

- Plain Java objects
- Business processes
- Business state machines
- Human tasks
- Business rules
- Mediation flow

SCA separates business logic from infrastructure so that application programmers can focus on solving business problems. IBM's WebSphere Process Server is based on that same premise. Figure 2 shows the architectural model of WebSphere Process Server.

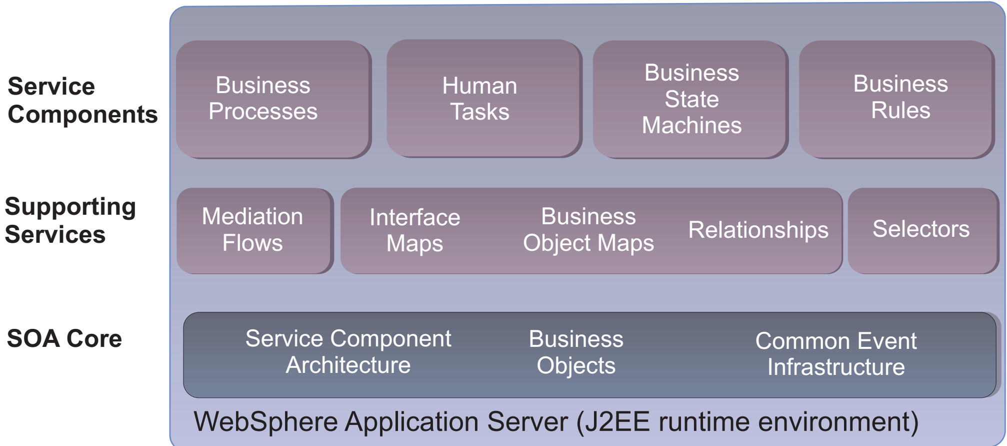


Figure 2. WebSphere Process Server component-based framework

In the WebSphere environment, the SCA framework is based on the Java 2 Platform, Enterprise Edition (J2EE) runtime environment of WebSphere Application Server. The overall WebSphere Process Server framework consists of SOA Core, Supporting Services, and the Service Components. The same framework with a subset of this overall capability, targeted more specifically at the connectivity and application integration needs of business integration, is available in WebSphere Enterprise Service Bus.

The interface of an SCA component, as illustrated in Figure 3 on page 12, can be represented as one of the following:

- A Java interface
- A WSDL port type (in WSDL 2.0, port type is called interface)

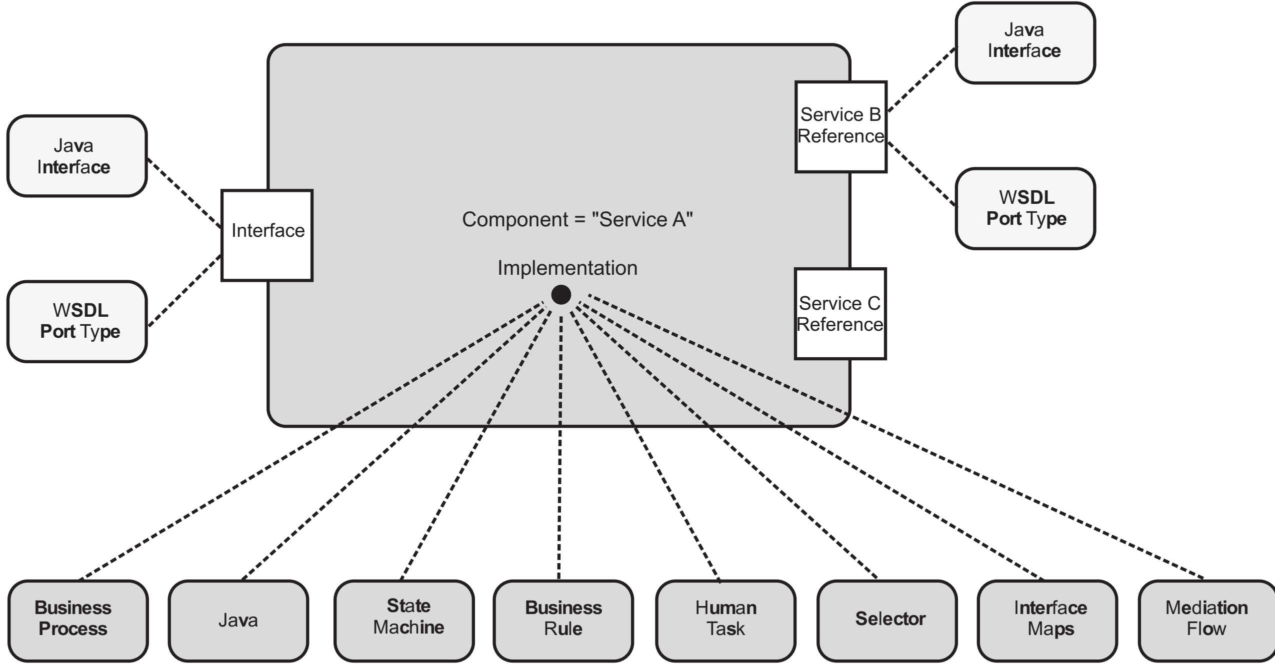


Figure 3. SCA in WebSphere Process Server

An SCA module is a group of components wired together by directly linking references and implementations. In WebSphere Integration Developer, each SCA module has an assembly diagram associated with it, which represents the integrated business application, consisting of SCA components and the wires that connect them. One of the main responsibilities of the integration developer is to create the assembly diagram by connecting the components that form the solution. WebSphere Integration Developer provides a graphical Assembly Editor to assist with this task. When creating the assembly diagram, the integration developer can proceed in one of two ways:

- **Top-down** defines the components, their interfaces, and their interactions before creating the implementation. The integration developer can define the structure of the process, identify the necessary components and their implementation types, and then generate an implementation skeleton.
- **Bottom-up** combines existing components. In this case, the integration developer simply needs to drag and drop existing implementations onto the assembly diagram.

The bottom-up approach is more commonly used when customers have existing services that they want to reuse and combine. When you need to create new business objects from scratch, you are likely to adopt the top-down approach.

The SCA Programming Model: Fundamentals

The concept of a software *component* forms the basis of the SCA programming model. As we mentioned, a component is a unit that implements some logic and makes it available to other components through an interface. A component may also require the services made available by other components. In that case, the component exposes a *reference* to these services.

In SCA, every component must expose at least one interface. The assembly diagram shown in Figure 4 on page 13 has three components, C1, C2, and C3. Each component has an interface that is represented by the letter I in a circle. A

component can also refer to other components. References are represented by the letter R in a square. References and interfaces are then linked in an assembly diagram. Essentially, the integration developer "resolves" the references by connecting them with the interfaces of the components that implement the required logic.

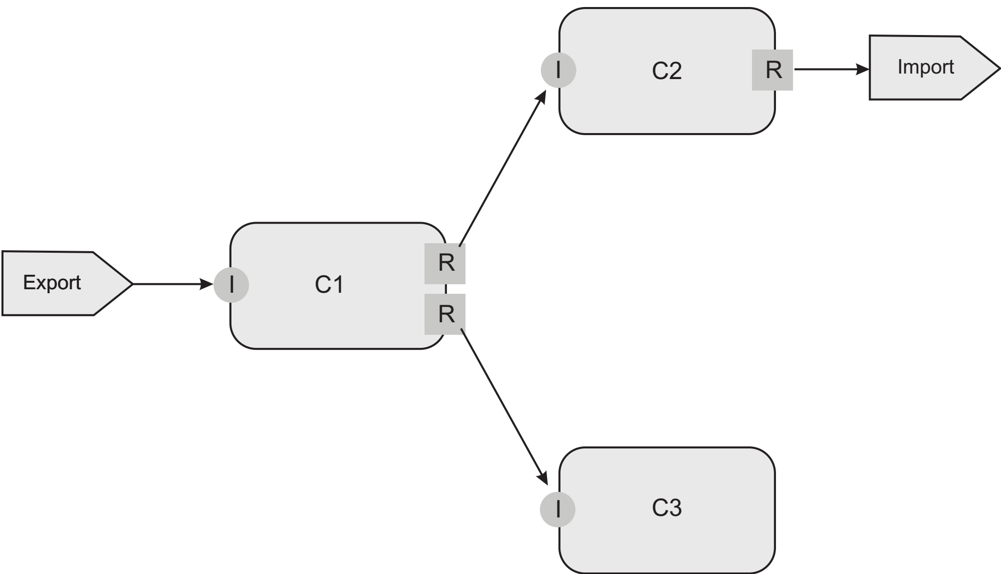


Figure 4. Assembly diagram

Invoking SCA Components

To provide access to the services to be invoked, the SCA programming model includes a *ServiceManager* class, which enables developers to look up available services by name. Here is a typical Java code fragment illustrating service lookup. The ServiceManager is used to obtain a reference to the BOFactory service, which is a system-provided service:

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
        smgr.locateService("com/ibm/websphere/bo/BOFactory");
```

Note: The package for ServiceManager is `com.ibm.websphere.sca`.

Developers can use a similar mechanism to obtain references to their own services by specifying the name of the service referenced in the *locateService* method. After you have obtained a reference to a service using the ServiceManager class, you can invoke any of the available operations on that service in a way that is independent of the invocation protocol and the type of implementation.

SCA components can be called using three different invocation styles:

- **Synchronous invocation**: When using this invocation style, the caller waits synchronously for the response to be returned. This is the classic invocation mechanism.
- **Asynchronous invocation**: This mechanism allows the caller to invoke a service without waiting for the response to be produced right away. Instead of getting the response, the caller gets a "ticket," which can be used later to retrieve the

response. The caller retrieves the response by calling a special operation that must be provided by the callee for this purpose.
- **Asynchronous invocation with callback**: This invocation style is similar to the preceding one, but it delegates the responsibility of returning the response to the callee. The caller needs to expose a special operation (the callback operation) that the callee can invoke when the response is ready.

Imports

Sometimes, business logic is provided by components or functions that are available on external systems, such as legacy applications, or other external implementations. In those cases, the integration developer cannot resolve the reference by connecting a reference to a component containing the implementation he or she needs to connect the reference to a component that "points to" the external implementation. Such a component is called an *import*. When you define an import, you need to specify how the external service can be accessed in terms of location and the invocation protocol.

Exports

Similarly, if your component has to be accessed by external applications, which is quite often the case, you must make it accessible. That is done by using a special component that exposes your logic to the "outside world." Such a component is called an *export*. These can also be invoked synchronously or asynchronously.

Stand-alone references

In WebSphere Process Server, an SCA service module is packaged as a J2EE EAR file that contains several other J2EE submodules. J2EE elements, such as a WAR file, can be packaged along with the SCA module. Non-SCA artifacts such as JSPs can also be packaged together with an SCA service module. This lets them invoke SCA services through the SCA client programming model using a special type of component called a stand-alone reference.

The SCA programming model is strongly declarative. Integration developers can configure aspects such as transactional behavior of invocations, propagation of security credentials, whether an invocation should be synchronous or asynchronous in a declarative way, directly in the assembly diagram. The SCA runtime, not the developers, is responsible for taking care of implementing the behavior specified in these modifiers. The declarative flexibility of SCA is one of the most powerful features of this programming model. Developers can concentrate on implementing business logic, rather than focusing on addressing technical aspects, such as being able to accommodate asynchronous invocation mechanisms. All these aspects are automatically taken care of by the SCA runtime.

Qualifiers

The qualifiers govern the interaction between a service client and a target service. Qualifiers can be specified on service component references, interfaces, and implementations and are usually external to an implementation.

The different categories of qualifiers include the following:
- Transaction, which specifies the way transactional contexts are handled in an SCA invocation
- Activity session, which specifies how Activity Session contexts are propagated.

- Security, which specifies the permissions
- Asynchronous reliability provides rules for asynchronous message delivery

SCA allows these quality of service (QoS) qualifiers to be applied to components declaratively (without requiring programming or a change to the services implementation code). This is done in WebSphere Integration Developer. Usually, you apply QoS qualifiers when you are ready to consider solution deployment. For more information, see Quality of service qualifier reference.

Business processes

Business processes, specifically, BPEL-based business processes, form the cornerstone of service components in the SCA.

Whether it is a simple order approval or a complex manufacturing process, enterprises have always had business processes. A *business process* is a set of activities, related to the business, that are invoked in a specific sequence to achieve a business goal. In the business integration world, a business process is defined using some kind of markup language.

These business processes can invoke other supporting services or other service components such as business state machines, human tasks, business rules, or data maps. And, when deployed, these processes can either get done quickly or run over a long period of time. Sometimes, these processes can run for years.

Like most components in the J2EE world, business processes run in a container. In the IBM WebSphere platform, this specific container is called Business Process Choreographer. In WebSphere Process Server, Business Process Choreographer is responsible for executing business processes and human tasks.

Human tasks

A human task is a component that allows people and services to interact.

Some human tasks represent to dos for people. These tasks can be initiated either by a person or by an automated service. Human tasks can be used to implement activities in business processes that require human interactions, such as manual exception handling and approvals. Other human tasks can be used to invoke a service, or to coordinate the collaboration between people. However, regardless of how a task is initiated, a person from a group of people, to which the task is assigned, performs the work associated with the task.

People are assigned to human tasks either statically, or by specifying criteria, such as a role or a group, that are resolved at runtime using a people directory. Alternatively the input data of a human task, or the data of a business process is used to find the right people to work on a task.

Building business integration applications

Business integration means integrating applications, data, and processes within an enterprise or among a set of enterprises. Integration also means developing processes, because there is some logic to the sequence of the applications that are assembled to integrate them. WebSphere Integration Developer is used to create business integration applications.

This section provides overview information of the development process for a business integration module.

The typical development flow for modules and mediation modules is as follows:

1. Start WebSphere Integration Developer and open a workspace.
2. Switch to the Business Integration perspective for development.
3. Create a library to store artifacts, such as business objects and interfaces that are shared among multiple modules.
4. Create a new module or mediation module.
5. Create the business objects to contain the application data, for example, customer or order data.
6. Create the interface and define the interface operations for each component. The interface determines what data can be passed from one component to another.
7. Create and implement the service components.
8. Build the module assembly by adding the service components, imports, and exports to the assembly diagram. Wire the components together. Bind the imports and exports to a protocol.
9. Test the module in the integrated test environment.
10. Deploy the module to WebSphere Process Server.
11. Share the tested module with others on the team by putting it in a repository.

Chapter 2. Developing service modules

A service component must be contained within a service module. Developing service modules to contain service components is key to providing services to other modules.

Before you begin

This task assumes that an analysis of requirements shows that implementing a service component for use by other modules is beneficial.

About this task

After analyzing your requirements, you might decide that providing and using service components is an efficient way to process information. If you determine that reusable service components would benefit your environment, create a service module to contain the service components.

Procedure

1. Identify service components other modules can use.

   After you have identified the service components, continue with "Developing service components".
2. Identify service components within an application that could use service components in other service modules.

   After you have identified the service components and their target components, continue with " Invoking components" or "Dynamically invoking components,"
3. Connect the client components with the target components through wires.

Overview of developing modules

A module is a basic deployment unit for a WebSphere Process Server application. A module can contain components, libraries, and staging modules used by the application.

Developing modules involves ensuring that the components, staging modules, and libraries (collections of artifacts referenced by the module) required by the application are available on the production server.

WebSphere Integration Developer is the main tool for developing modules for deployment to WebSphere Process Server. Although you can develop modules in other environments, it is best to use WebSphere Integration Developer.

WebSphere Process Server supports modules for business services and mediation modules. Both modules and mediation modules are types of Service Component Architecture (SCA) module. A mediation module allows communication between applications by transforming the service invocation to a format understood by the target, passing the request to the target and returning the result to the originator. A module for a business service implements the logic of a business process. However, a module can also include the same mediation logic that can be packaged in a mediation module.

© Copyright IBM Corp. 2005, 2009

The following sections address how to implement and update modules for WebSphere Process Server.

Components

SCA modules contain components, which are the basic building blocks to encapsulate reusable business logic. Components provide and consume services and are associated with interfaces, references, and implementations. The interface defines a contract between a service component and a calling component.

With WebSphere Process Server, a module can either export a service component for use by other modules or import a service component for use. To invoke a service component, a calling module references the interface to the service component. The references to the interfaces are resolved by configuring the references from the calling module to their respective interfaces.

To develop a module you must do the following activities:

1. Define or identify interfaces for the components in the module.
2. Define or manipulate business objects used by components.
3. Define or modify components through their interfaces.

   Note: A component is defined through its interface.
4. Optional: Export or import service components.
5. Create an enterprise archive (EAR) file to deploy to the run time. You create the file using either the export EAR feature in WebSphere Integration Developer or the serviceDeploy command.

Development types

WebSphere Process Server provides a component programming model to facilitate a service-oriented programming paradigm. To use this model, a provider exports interfaces of a service component so that a consumer can import those interfaces and use the service component as if it were local. A developer uses either strongly-typed interfaces or dynamically-typed interfaces to implement or invoke the service component. The interfaces and their methods are described in the References section within this information center.

After installing service modules to your servers, you can use the administrative console to change the target component for a reference from an application. The new target must accept the same business object type and perform the same operation that the reference from the application is requesting.

Service component development considerations

When developing a service component, ask yourself the following questions:

- Will this service component be exported and used by another module?

  If so, make sure the interface you define for the component can be used by another module.
- Will the service component take a relatively long time to run?

  If so, consider implementing an asynchronous interface to the service component.
- Is it beneficial to decentralize the service component?

If so, consider having a copy of the service component in a service module that is deployed on a cluster of servers to benefit from parallel processing.

- Does your application require a mixture of 1-phase and 2-phase commit resources?

  If so, make sure you enable last participant support for the application.

  Note: If you create your application using WebSphere Integration Developer or create the installable EAR file using the serviceDeploy command, these tools automatically enable the support for the application. See the topic, "Using one-phase and two-phase commit resources in the same transaction" in the WebSphere Application Server Network Deployment information center.

Developing service components

Develop service components to provide reusable logic to multiple applications within your server.

Before you begin

This task assumes that you have already developed and identified processing that is useful for multiple modules.

About this task

Multiple modules can use a service component. Exporting a service component makes it available to other modules that refer to the service component through an interface. This task describes how to build the service component so that other modules can use it.

Note: A single service component can contain multiple interfaces.

Procedure

1. Define the data object to move data between the caller and the service component.

   The data object and its type is part of the interface between the callers and the service component.
2. Define an interface that the callers will use to reference the service component.

   This interface definition names the service component and lists any methods available within the service component.
3. Generate the class that implements calling the service.
4. Develop the implementation of the generated class.
5. Save the component interfaces and implementations in files with a .java extension.
6. Package the service module and necessary resources in a JAR file.

   See "Deploying a module to a production server" in this information center for a description of steps 6 through 8.
7. Run the serviceDeploy command to create an installable EAR file containing the application.
8. Install the application on the server node.
9. Optional: Configure the wires between the callers and the corresponding service component, if calling a service component in another service module.

The “Administering” section of this information center describes configuring the wires.

Examples of developing components

This example shows a synchronous service component that implements a single method, CustomerInfo. The first section defines the interface to the service component that implements a method called getCustomerInfo.

```
public interface CustomerInfo {
 public Customer getCustomerInfo(String customerID);
}
```

The following block of code implements the service component.

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

x

The following section is the implementation of the class associated with StockQuote.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

What to do next

Invoke the service.

Invoking components

Components with modules can use components on any node of a WebSphere Process Server cluster.

Before you begin

Before invoking a component, make sure that the module containing the component is installed on WebSphere Process Server.

About this task

Components can use any service component available within a WebSphere Process Server cluster by using the name of the component and passing the data type the component expects. Invoking a component in this environment involves locating and then creating the reference to the required component.

Note: A component in a module can invoke a component within the same module, known as an intra-module invocation. Implement external calls (inter-module invocations) by exporting the interface in the providing component and importing the interface in the calling component.

Important: When invoking a component that resides on a different server than the one on which the calling module is running, you must perform additional configurations to the servers. The configurations required depend on whether the component is called asynchronously or synchronously. How to configure the application servers in this case is described in related tasks.

Procedure

1. Determine the components required by the calling module.

   Note the name of the interface within a component and the data type that interface requires.
2. Define a data object.

   Although the input or return can be a Java class, a service data object is optimal.
3. Locate the component.
   a. Use the ServiceManager class to obtain the references available to the calling module.
   b. Use the locateService() method to find the component.

      Depending on the component, the interface can either be a Web Service Descriptor Language (WSDL) port type or a Java interface.
4. Invoke the component synchronously.

   You can either invoke the component through a Java interface or use the invoke() method to dynamically invoke the component.
5. Process the return.

   The component might generate an exception, so the client has to be able to process that possibility.

Example of invoking a component

The following example creates a ServiceManager class.

```
ServiceManager serviceManager = new ServiceManager();
```

The following example uses the ServiceManager class to obtain a list of components from a file that contains the component references.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

The following code locates a component that implements the StockQuote Java interface.

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

The following code locates a component that implements either a Java or WSDL port type interface. The calling module uses the Service interface to interact with the component.

Tip: If the component implements a Java interface, the component can be invoked through either the interface or the invoke() method.

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

The following example shows MyValue, code that calls another component.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

     // invoke
         CustomerInfo cInfo =
      (CustomerInfo)serviceManager.locateService("customerInfo");
         customer = cInfo.getCustomerInfo(customerID);

     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuote sQuote =
      (StockQuote)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuote(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // assign
          value = quote * customer.getNumShares();
     } else {

         // throw
         throw new MyValueException(customer.getErrorMsg());
     }
     // reply
         return value;
 }
}
```

What to do next

Configure the wires between the calling module references and the component interfaces.

Dynamically invoking a component

When an module invokes a component that has a Web Service Descriptor Language (WSDL) port type interface, the module must invoke the component dynamically using the invoke() method.

Before you begin

This task assumes that a calling component is invoking a component dynamically.

About this task

With a WSDL port type interface, a calling component must use the invoke() method to invoke the component. A calling module can also invoke a component that has a Java interface this way.

Procedure

1. Determine the module that contains the component required.
2. Determine the array required by the component.

   The input array can be one of three types:
   - Primitive uppercase Java types or arrays of this type
   - Ordinary Java classes or arrays of the classes
   - Service Data Objects (SDOs)
3. Define an array to contain the response from the component.

   The response array can be of the same types as the input array.
4. Use the invoke() method to invoke the required component and pass the array object to the component.
5. Process the result.

Examples of dynamically invoking a component

In the following example, a module uses the invoke() method to call a component that uses primitive uppercase Java data types.

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();

  BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

  DataObject paramObject = boFactory.createbyType(t);

  paramObject.set(0,"input1")
  paramObject.set(1,"input2")
  paramObject.set(2,"input3")

  service.invoke("methodMultiParamater",paramObject);
```

The following example uses the invoke method with a WSDL port type interface as the target.

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

 DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
   dob.setString("attribute1", stringArg);

 DataObject wrapBo = factory.createByElement
  ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
   wrapBo.set("input1", dob); //wrapBo encapsulates all the parameters of methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

 DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

Overview of isolating modules and targets

When developing modules, you will identify services that multiple modules can use. Leveraging services this way minimizes your development cycle and costs. When you have a service used by many modules, you should isolate the invoking modules from the target so that if the target is upgraded, switching to the new service is transparent to the calling module. This topic contrasts the simple invocation model and the isolated invocation model and provides an example of how isolation can be useful. While describing a specific example, this is not the only way to isolate modules from targets.

Simple invocation model

While developing a module, you might use services that are located in other modules. You do this by importing the service into the module and then invoking that service. The imported service is "wired" to the service exported by the other module either in WebSphere Integration Developer or by binding the service in the administrative console. Simple invocation model illustrates this model.

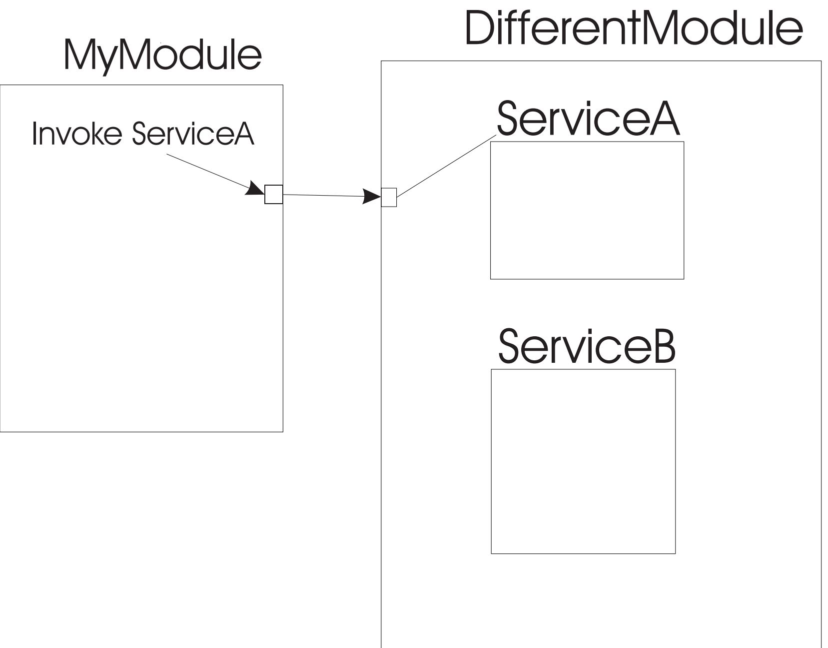


Figure 5. Simple invocation model

Isolated invocation model

To change the target of an invocation without stopping invoking modules, you can isolate the invoking modules from the target of the invocation. This allows the modules to continue processing while you change the target because you are not changing the module itself but the downstream target. Example of isolating applications shows how isolation allows you to change the target without affecting the status of the invoking module.

Example of isolating applications

Using the simple invocation model, multiple modules invoking the same service would look much like Multiple applications invoking a single service. MODA, MODB, and MODC all invoke CalculateFinalCost.

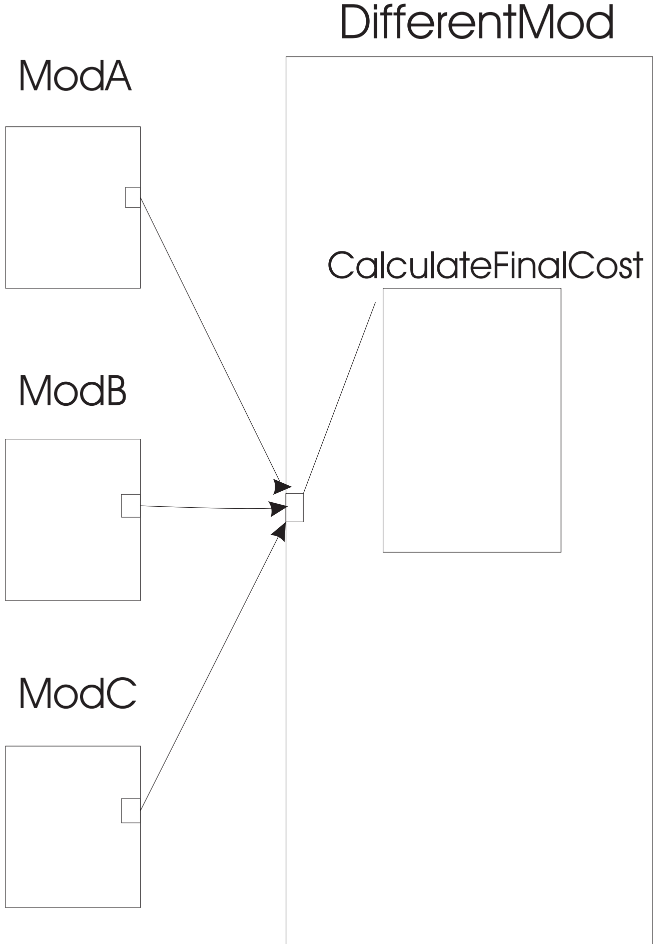


Figure 6. Multiple applications invoking a single service

The service provided by CalculateFinalCost needs updating so that new costs are reflected in all modules that use the service. The development team builds and tests a new service UpdatedCalculateFinal to incorporate the changes. You are ready to bring the new service into production. Without isolation, you would have to update all of the modules invoking CalculateFinalCost to invoke UpdateCalculateFinal. With isolation, you only have to change the binding that connects the buffer module to the target.

Note: Changing the service this way allows you to continue to provide the original service to other modules that may need it.

Using isolation, you create a buffer module between the applications and the target (see Isolated invocation model invoking UpdateCalculateFinal).

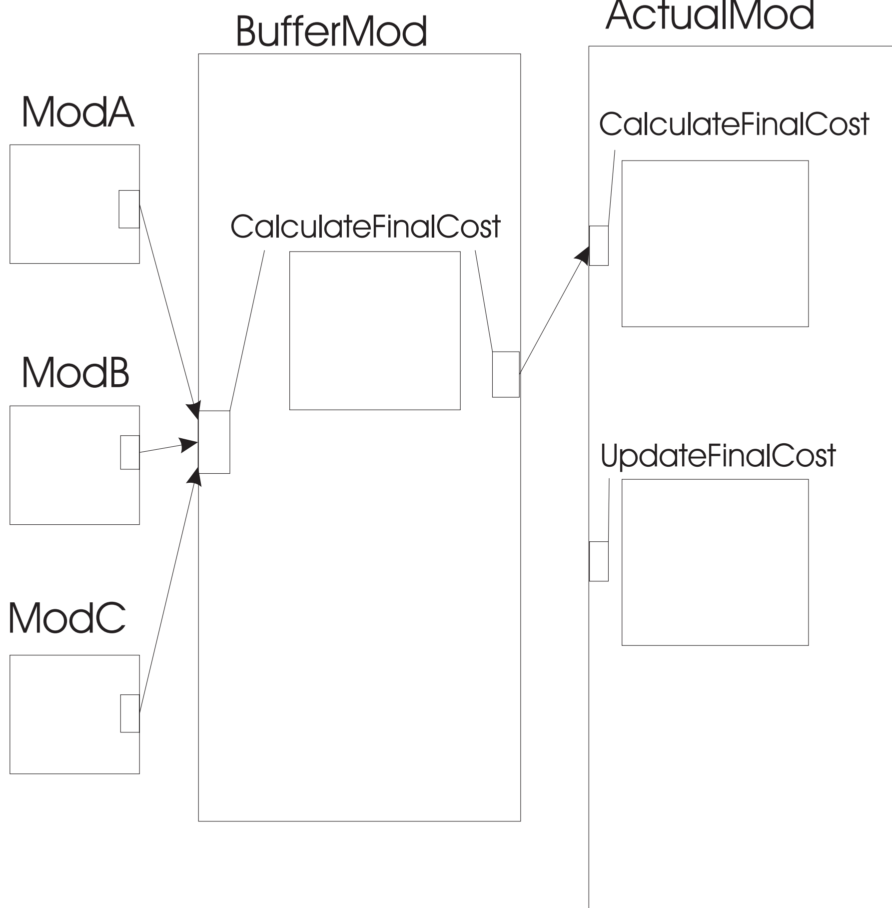


Figure 7. Isolated invocation model invoking UpdateCalculateFinal

With this model, the invoking modules do not change, you just have to change the binding from the buffer module import to the target (see Isolated invocation model invoking UpdatedCalculateFinal).

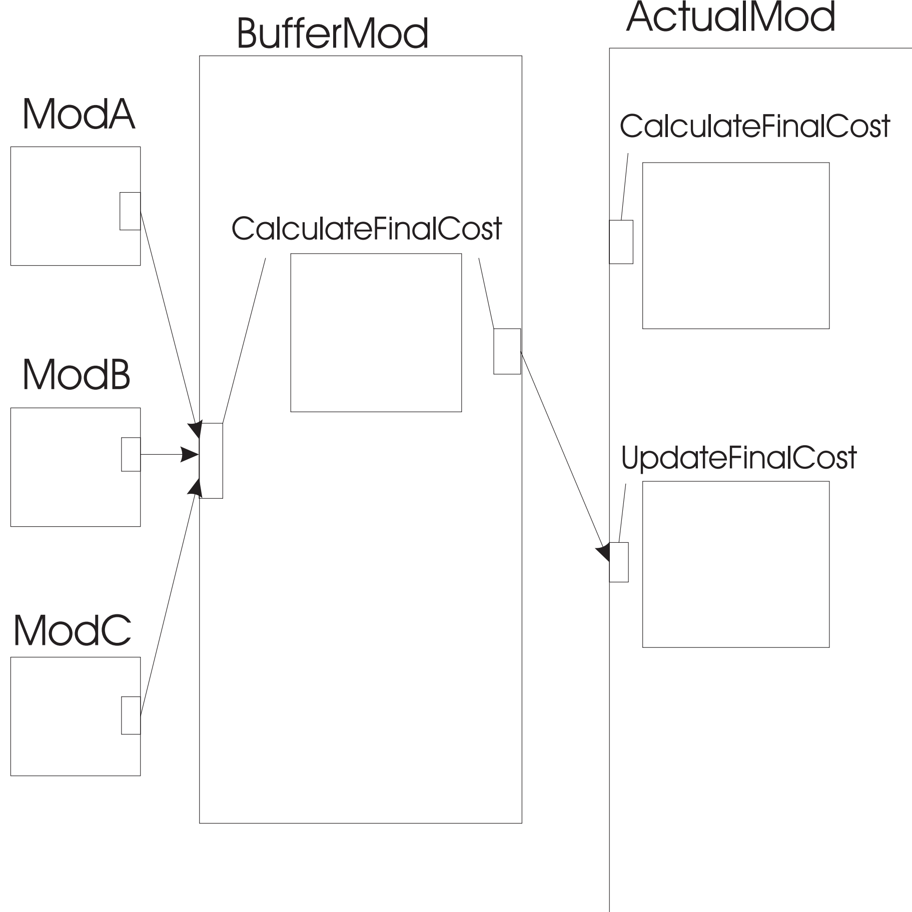


Figure 8. Isolated invocation model invoking UpdatedCalculateFinal

If the buffer module invokes the target synchronously, when you restart the buffer module (whether a mediation module or a service for business module) the results returned to the original application come from the new target. If the buffer module invokes the target asynchronously, the results returned to the original application come from the new target on the next invocation.

HTTP bindings

The HTTP binding is designed to provide Service Component Architecture (SCA) connectivity to HTTP. This allows existing or newly-developed HTTP applications to participate in Service Oriented Architecture (SOA) environments.

In addition, a network of SCA runtime environments can communicate across an existing HTTP infrastructure.

The HTTP binding exposes several HTTP features:

- Messages are presented to mediation components in a manner that preserves HTTP format and message header information. This provides a more familiar view to HTTP application programmers, users and administrators.
- An existing data binding framework is extended for HTTP conventions and provides mapping between SCA messages and HTTP message headers and bodies.

- Imports and exports can be configured to support a range of common HTTP features.
- When you install an SCA module containing HTTP imports or exports, the runtime environment is automatically configured appropriately to allow connectivity to HTTP.

Detailed instructions on creating HTTP imports and exports can be found in the information center at **WebSphere Integration Developer** > **Developing integration applications** > **HTTP data binding**.

Chapter 3. Programming guides and techniques

This section includes programming guides and examples.

The following subtopics provide information for programming various components, applications, and business integration solutions.

Important: See the Reference section of the infocenter for details of the application programming interfaces (APIs) and system programming interfaces (SPIs) that are supported by WebSphere Process Server and WebSphere Enterprise Service Bus.

Programming business objects

The WebSphere Process Server business object framework is an extension of the Service Data Object (SDO) standard. This sections provides information on programming business objects and SDOs.

The following subtopics contain information for programming business objects and SDOs.

Arrays in business objects

You can define arrays for an element in a business object so that the element can contain more than one instance of data.

You can use a List type to create an array for a single named element in a business object. This will allow you to use that element to contain multiple instances of data. For example, you can use an array to store several telephone numbers within an element named `telephone` that is defined as a string in its business object wrapper. You can also define the size of the array by specifying the number of data instances using the `maxOccurs` value. The following example code shows how you would create such an array that will hold three instances of data for that element:

```
<xsd:element name="telephone" type="xsd:string" maxOccurs="3"/>
```

This will create a list index for the element `telephone` that can hold up to three data instances. You may also use the value `minOccurs` if you are only planning to have one item in the array.

The resulting array consists of two items:
- the contents of the array
- the array itself.

In order to create this array, however, you need perform an intermediate step by defining a wrapper. This wrapper, in effect, replaces the property of the element with an array object. In the example above, you can create an `ArrayOfTelephone` object to define the element `telephone` as an array. The following code example shows how you accomplish this task:

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="Customer">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
```

© Copyright IBM Corp. 2005, 2009

```
          <xsd:element name="ArrayOfTelephone" type="ArrayOfTelephone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="ArrayOfTelephone">
      <xsd:sequence maxOccurs="3">
        <xsd:element name="telephone" type="xsd:string" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
 </xsd:schema>
```

The `telephone` element now appears as a child of the `ArrayOfTelephone` wrapper object.

Note that in the example above, the `telephone` element contains a property named `nillable`. You can set this property to `true` if you want to certain items in the array index to contain no data. The following example code shows how the data in an array may be represented:

```
<Customer>
   <name>Bob</name>
   <ArrayOfTelephone>
     <telephone>111-1111</telephone>
     <telephone xsi:nil="true"/>
     <telephone>333-3333</telephone>
   </ArrayOfTelephone>
 </Customer>
```

In this case, the first and third items in the array index for the `telephone` element contain data, while the second item does not contain any data. If you had not used the `nillable` property for the `telephone` element, then you would have had to have the first two elements contain data.

You can use the Service Data Object (SDO) Sequence APIs in WebSphere Process Server as an alternative method to handle sequences in business object arrays. The following example code will create an array for the `telephone` element with data identical to that shown above:

```
DataObject customer = ...
 customer.setString("name", "Bob");

 DataObject tele_array = customer.createDataObject("ArrayOfTelephone");
 Sequence seq = tele_array.getSequence();  // The array is sequenced
 seq.add("telephone", "111-1111");
 seq.add("telephone", null);
 seq.add("telephone", "333-3333");
```

You can return the data for a given element array index by using code similar to the example below:

```
String tele3 = tele_array.get("telephone[3]");  // tele3 = "333-3333"
```

In this example, a string named `tele3` will return the data ″333-3333″.

You can fill the data items for the array in the list index by using fixed width or delimited data placed in a JMS or MQ message queue. You can also accomplish this task by using a flat text file that contains the properly formatted data

Creating nested business objects

You can use the `setWithCreate` function to create nested business objects within a parent business object.

You can create nested business objects from a parent business object without having to write code that details intermediate child objects. For instance, you can set a nested business object two levels below the parent object without having to define a dependent business object one level below the parent object. Use the `setWithCreate` function to accomplish this task for:

- a single instance
- multiple instances
- a wildcard value
- a model group

The following topics describe how you can do each of these.

Single instance of a nested business object

Use the `setWithCreate` function to create a single instance of nested business object.

Before you begin

The example code below shows how you would normally have to create code for an intermediate (child) object from a higher level (parent) object in order to create a third-level (grandchild) object. The XSD file would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

About this task

If you used the traditional ″top-down″ method to set the business object data, you would have to process the following code specifying the child and grandchild objects prior to setting the data in the grandchild object:

```
DataObject parent = ...
DataObject child = parent.createDataObject("child");
DataObject grandchild = child.createDataObject("grandChild");
grandchild.setString("name", "Bob");
```

You can use a more efficient method by using the `setWithCreate` function to simultaneously define the grandchild object and set its data, without having to specify the intermediate child object. The following example code shows how you would accomplish this task:

```
DataObject parent = ...
parent.setString("child/grandchild/name", "Bob");
```

Results

The lower-level business object data is set without having to reference the intermediate-level business object. An exception occurs if the path is not valid.

Creating multiple instances of nested business objects

Use the `setWithCreate` function to create a multiple instances of nested business object.

Before you begin

The example XSD file below contains nested objects one (child) and two (grandchild) levels below the top (parent) business object:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child" maxOccurs="5"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Note that the parent object can have up to five child objects, as specified in the `maxOccurs` value.

About this task

You can create a list with a more stringent policy that will not allow for missing sequences in an array. You can use the `setWithGet` method, and at the same time specify the data that will appear in a particular list index item:

```
DataObject parent = ...
parent.setString("child[3]/grandchild/name", "Bob");
```

In this case, the resulting array would be of size three, but the values for `child[1]` and `child[2]` list index items are undefined. You may want the items to either be a null value or have an associated data value. In the scenario above, an exception will be thrown because the values for the first two array index items are undefined.

You can remedy this situation by defining the values in the index of the list. If the index item refers to an existing element in the array if that element is not null (i.e,

it contains data), it will be used. If it is null, it will be created and used. If the index of the list is one greater than the size of the list, a new value will be created and added. The following example code shows what will happen in a list that of size two, where `child[1]` is designated null and `child[2]` contains data:

```
DataObject parent = ...
// child[1] = null
// child[2] = existing Child
// This code will work because child[1] is null and will be created.
parent.setString("child[1]/grandchild/name", "Bob");

// This code will work because child[2] exists and will be used.
parent.setString("child[2]/grandchild/name", "Dan");

// This code will work because the child list is of size 2, and adding
// one more list item will increase the list size.
parent.setString("child[3]/grandchild/name", "Sam");
```

Results

You have overridden the values for the two existing items and added a third item to the list index. If, however, you then add another item that is not of size four, or which is greater than the size specified in `maxOccurs`, then an exception will be thrown. The more stringent policy of this method is demonstrated in the following example code.

Note: The code below is assumed to be appended to the existing code above:

```
// This code will throw an exception because the list is of size 3
// and you have not created an item to increase the size to 4.
parent.setString("child[5]/grandchild/name", "Billy");
```

Using a nested business object defined by a wildcard

You can specify the type `xsd:any` in a parent object to specify a child object, but only if the child object already exists.

About this task

The `setWithCreate` function used to define nested business objects for single and multiple instances does not work if you are using a wildcard value of `xsd:any` in the Service Data Object. This is illustrated in the following example code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Results

An exception will be thrown if the child data object does not exist.

Using business objects in model groups

You should the model group path patterns when working with nested business objects that are part of a model group.

About this task

Model groups use the tag `xsd:choice` that you can use to create business objects from a parent business object. The Eclipse Modeling Framework (EMF), however, can cause naming conflicts that may generate an exception. The following example code illustrates how this can occur:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Note that there can be multiple instances of the elements named ″child1″ and ″child2″,

You should use the Service Data Object (SDO) path patterns for model groups to resolve these conflicts.

Results

You would get arrays that use the SDO path pattern that is used to handle model groups, as shown in the example code below:

```
set("child1/grandchild/name", "Bob");

set("child11/grandchild/name", "Joe");
```

Business objects: schema enhancement and industry schema support

The Service Data Objects (SDO) framework provides the basis for the Business Object data used by WebSphere Process Server

This guide provides information about problem areas in handling the schema constructs for some features. For information about how a business object is defined, business object development guidelines, and how to use business object programming APIs, refer to the articles in the ″Related information″ section below.

Related information

Web Services Description Language (WSDL) 1.1

Introduction to Service Data Objects

Examining business objects in WebSphere Process Server

Differentiating identically named elements

You must provide unique names for business object elements and attributes.

In the Service Data Object (SDO) framework, elements and attributes are created as properties. In the following code examples, the XSDs create types that have one property named foo:

```
<xsd:complexType name="ElementFoo">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeFoo">
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

In these cases, you can access the property using the XML Path Language (XPath). However, valid schema types can have an attribute and element of the same name, as in the following example:

```
<xsd:complexType name="DuplicateNames">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

In XPath, you must be able to differentiate identically named elements from attributes. This is achieved by beginning one of the names with an at sign (@). The following snippet shows how to access the identically named element and attribute:

```
1 DataObject duplicateNames = ...

2 // Displays "elem_value"
3 System.out.println(duplicateNames.get("foo"));

4 // Displays "attr_value"
5 System.out.println(duplicateNames.get("@foo"));
```

Use this naming scheme for all methods that take a String value that is an SDO XPath.

Model group support (all, choice, sequence, and group references):

The SDO specification requires model groups (all, choice, sequence, and group references) to be expanded in place and not describe types or properties.

Basically, this means that any of those structures that are within the same containing structures are "flattened". This "flattening" puts all the child structures at the same level. This can produce duplicate naming issues in an SDO whose structure is derived from the flattened data. When an XSD does not flatten the groups, there is still a separation of duplicates that are contained by different parents.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
```

```
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Since the multiple occurrences of option1 and option2 are contained in different choice blocks and even have a separating element between them, XSD and XML have no problem distinguishing between them. But when SDO flattens these groups, all the option properties are now under the same container of MultipleGroup.

Even without duplicate names, there is also the semantic issue that the flattening of these groups cause. Take the following XSD for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://SimpleChoice">
  <xsd:complexType name="SimpleChoice">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Asking the user to rename duplicate names or add special annotations to XSDs is impractical because in many cases, like standards and industry schemas, the user does not control the XSDs they are working with.

To create consistency for all properties, business objects include a method to access each individual occurrence of the duplicate named properties through XPath. Following the EMF naming convention, any duplicate property names encountered will have the next unused digit appended to their name So for example, the following XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://TieredGroup">
  <xsd:complexType name="TieredGroup">
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:sequence>
          <xsd:element name="low" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:choice minOccurs="0">
            <xsd:element name="width" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
        <xsd:element name="high" minOccurs="1"
    maxOccurs="1" type="xsd:string"/>
        <xsd:sequence>
          <xsd:element name="width" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="high" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="center" minOccurs="1"
```

```
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="width" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

The preceding XSD produces the following DataObject model:

```
DataObject - TieredGroup
 Property[0] - low - string
 Property[1] - width - string
 Property[2] - high - string
 Property[3] - high1 - string
 Property[4] - width1 - string
 Property[5] - high2 - string
 Property[6] - center - string
 Property[7] - width2 - string
```

Where **width**, **width1**, and **width2** are the names of the of properties named width starting from the first one in the XSD going down, likewise with **high**, **high1**, **high2**.

These new property names are just the names used for reference and XPath and do not affect serialized content. The ″true″ names of each of these properties that appear in the serialized XML are the values given in the XSD. So for the XML instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:TieredGroup xsi:type="p:TieredGroup"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://TieredGroup">
  <width>foo</width>
  <high>bar</high>
</p:TieredGroup>
```

In order to access those properties you would use the following code:

```
 DataObject tieredGroup = ...

 // Displays "foo"
 System.out.println(tieredGroup.get("width1"));

 // Displays "bar"
 System.out.println(tieredGroup.get("high2"));
```

Differentiating identically named properties

When multiple XSDs with the same namespace define the same named types, an incorrect type can be accidentally referenced.

Address1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="city" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

Address2.xsd:
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="state" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Business objects do not support duplicate names for any global XSD structures (such as complexType, simpleType, element, attribute, and so on) through the BOFactory.create() APIs. These duplicate global structures can still be created as the child to other structures if the proper APIs are used, as shown in the following examples

Customer1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer1"
 targetNamespace="http://Customer1">
  <xsd:import schemaLocation="./Address1.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Customer2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer2"
 targetNamespace="http://Customer2">
  <xsd:import schemaLocation="./Address2.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

When populating both of the Customer address fields and then calling BOFactory.create() to make the Address, the resulting child business object types can be incorrectly set. You can avoid this by calling the createDataObject("address") API on the Customer DataObject. This will be guaranteed to produce a child of the correct type because business objects will follow the import's schemaLocation.

```
DataObject customer1 = ...

// Incorrect way to create Address child
// This may create a type of Address1.xsd Address or maybe Address2.xsd Address
DataObject incorrect = boFactory.create("", "Address");
customer1.set("address", incorrect);

// Correct way to create Address child
// This is guaranteed to create a type of Address1.xsd Address
customer1.createDataObject("address");
```

Resolving property names that contain periods

Property names in an XSD may contain a period (".") as one of many valid characters, while in a SDO they are also used to show indexing in a property of multiple cardinality. This may cause resolution problems in certain situations.

Property names in Service Data Objects (SDOs) are based on the names of the elements and attribute that are generated from in the XSD. Business objects will handle the ″.″ character properly, with one exception: if an XSD has a single cardinality property named ″<name>.<#>″ and a multiple cardinality property named ″<name>″.

An XPath such as ″foo.0″ would not resolve properly if there is a single cardinality property named ″foo.0″ and multiple cardinality property named ″foo″. In this case, the single cardinality Property named ″foo.0″ would be the one resolved. Although this should be a rare occurrence, you can avoid it entirely if you use the ″foo[1]″ syntax to access their multiple cardinality property. SDOs will not support the ″.″ syntax for indexing, so you should use the ″[]″ for indexing.

Serializing and deserializing unions with xsi:type:

In XSD, a union is a way to merge the lexical spaces of several simple datatypes known as members.

The following example XSD shows a union that has the members of an integer and a date.

```
<xsd:simpleType name="integerOrDate">
 <xsd:union memberTypes="xsd:integer xsd:date"/>
</xsd:simpleType>
```

This multiple typing can cause confusion during deserialization and when manipulating the data.

Business objects support SDO's using xsi:type for serialization and will follow the same algorithm for determining the type on a deserialization if the xsi:type is not present in the XML data.

So to guarantee that the data (the number ″42″ in this example) would be deserialized as an integer, you can use the xsi:type specified in the input XML. You can also order the member list of the union in the XSD so that the integer comes before the string. The following example shows how both methods are implemented:

```
<integerOrString xsi:type="xsd:integer">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:integer xsd:string"/>
</xsd:simpleType>
```

Likewise, if the user wanted the data to be deserialized as a string, then either of the following changes would cause that behavior:

```
<integerOrString xsi:type="xsd:string">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:string xsd:integer"/>
</xsd:simpleType>
```

Note that if a string type is the first member of the union, it never has any information loss. It can also hold any data that will always be chosen by the no xsi:type algorithm. If you want to use a type other than string, you must either use xsi:type in the XML or reorder the member types in the XSD to give the other members a chance to accept the data.

Using the Sequence object to set data order

Some XSDs are defined in a way that makes the order that the data occurs in the XML have special significance.

One example of order significance in XSDs is mixed content. If the text data appears before or after an element, it may have different meaning than if it occurs in a different location. For these situations, SDO generates an object known as a Sequence, which is used to set the data in an ordered fashion.

SDO Sequences should not be confused with XSD sequences. XSD sequences are just model groups that are flattened out before SDO model generation. The presence of an XSD sequence does not relate to the presence of an SDO Sequence.

The following conditions in an XSD cause an SDO Sequence to be generated:

A complexType with mixed content:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://MixedContent"
 targetNamespace="http://MixedContent">
  <xsd:complexType name="MixedContent" mixed="true">
    <xsd:sequence>
      <xsd:element name="element1" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element2" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element3" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MixedContent" type="tns:MixedContent"/>
</xsd:schema>
```

A schema that has 1 or more <any/> tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <xsd:any/>
      <xsd:element name="marker1" type="xsd:string"/>
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

A model group array (an all, choice, sequence, or group reference with maxOccurs > 1):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence maxOccurs="3">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

An <all/> tag of maxOccurs <= 1 that contains more than one element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://All">
```

```
  <xsd:complexType name="All">
    <xsd:all>
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>
```

Specific information about using <any/> and sequence together will be discussed in the topic listed at the bottom of this page.. The general information that follows in the remainder of this section will describe how to work with the other Sequence conditions, but will still apply to <any/> as well.

How do I know if my DataObject has a sequence?:

There are two simple APIs to choose from that can determine if a DataObject is sequenced: DataObject noSequence and DataObject withSequence.

You would use DataObject noSequence and DataObject withSequence as shown in the following example:

```
DataObject noSequence = ...
DataObject withSequence = ...

// Displays false
System.out.println(noSequence.getType().isSequenced());

// Displays true
System.out.println(withSequence.getType().isSequenced());

// Displays true
System.out.println(noSequence.getSequence() == null);

// Displays false
System.out.println(withSequence.getSequence() == null);
```

Why do I need to know a DataObject has a Sequence?:

If you are working on a DataObject that has a Sequence, it is important to know the order in which the data is set. Because of this, care needs to be taken in the order in which the values are set.

A DataObject that is not sequenced allows random order set access. This functions like a Map where all the keys are set to the same values. It does not matter in what order the keys were set, the data in the map is the same and would be serialized to XML identically.

When a DataObject is sequenced, the order in which the data was set is recorded in the Sequence, much like adding data to a List. This provides two ways to access the data, by name/value pairs (the DataObject APIs) and by order in which it was set (the Sequence APIs). You can use the DataObject set(...) or Sequence add(...) APIs to preserve the structure. This ordering affects the way that the XML is serialized.

Take for example, the <all/> tag XSD below. When the set methods are called in the following order it produces the following XML when serialized:

```
DataObject all = ...
all.set("element1", "foo");
all.set("element2", "bar");
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element1>foo</element1>
  <element2>bar</element2>
</p:All>
```

If instead, the set methods are called in the opposite order, then the following XML will be produced when the business object is serialized:

```
 DataObject all = ...
 all.set("element2", "bar");
 all.set("element1", "foo");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element2>bar</element2>
  <element1>foo</element1>
</p:All>
```

If the order of the Sequence ever needs to be changed, then the Sequence class has basic add, remove, and move methods to allow the user to alter the order of Sequence.

How do I work with mixed content?:

For mixed content, Sequence has a specific API for adding text: addText(...).

All other APIs work equally with text as they do with Properties. The getProperty(int) API will return null for mixed content text data. The following example of mixed content code can be used to print all the mixed content text from a DataObject:

```
 DataObject mixedContent = ...
 Sequence seq = mixedContent.getSequence();

 for (int i=0; i < seq.size(); i++)
 {
     Property prop = seq.getProperty(i);
     Object value = seq.getValue(i);

     if (prop == null)
     {
         System.out.println("Found mixed content text: "+value);
     }
     else
     {
         System.out.println("Found Property "+prop.getName()+": "+value);
     }
 }
```

How do I work with a model group array?:

A model group array is created when a model group has a value for maxOccurs > 1.

Since model groups are flattened and not expressed in a DataObject, the properties inside of the model group become multiple cardinality properties so that their isMany() methods return true if they are not already true. Their minOccurs and maxOccurs facets become multiplied by that of the containing model group.

Choice will multiply the maxOccurs facet in the same way as the other model groups, but will always use 0 as the multiplication value for minOccurs because any data in a choice may not be selected.

For example, the following XSD has a model group array:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

As stated, **element1** and **element2** will now be multiple cardinality so that a get(...) accessor would return a List. **Element1** has a default minOccurs of 1 and a default maxOccurs of 1. **Element2** has a minOccurs of 0 and a maxOccurs of 3. In the following example, their new minOccurs and maxOccurs will be as follows:

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(2*1)=2 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(2*0)=0 - maxOccurs=(5*3)=15
```

If the type were Choice:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:choice minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Would generate the following minOccurs due to the exclusion of choice that only **element1** can be picked each time or only **element2** could be picked each time, so to pass validation both need to be able to have 0 occurrences:

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(0*1)=0 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(0*0)=0 - maxOccurs=(5*3)=15
```

Using AnySimpleType for simple types

AnySimpleType is handled no differently from any other simple type (string, int, boolean, and so on) by the SDO APIs.

The only differences between anySimpleType and the other simple types are in its instance data and serialization/deserialization. These should be internal concepts to business object only, and used to determine if data being mapped to or from the field is valid. If a string type were to have a set(...) method called on it, the data would first be converted to a string and the original data type would be lost:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://StringType">
  <xsd:complexType name="StringType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:string"/>
```

```
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject stringType = ...

 // Set the data to a String
 stringType.set("foo", "bar");

 // The instance data will always be type String, regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Set the data to an Integer
 stringType.set("foo", new Integer(42));

 // The instance data will always be type String, regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());
```

An anySimpleType instead does not loose the original data type of what is being set:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnySimpleType">
  <xsd:complexType name="AnySimpleType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:anySimpleType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anySimpleType = ...

 // Set the data to a String
 stringType.set("foo", "bar");

 // The instance data will always be of the type of date used in the set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Set the data to an Integer
 stringType.set("foo", new Integer(42));

 // The instance data will always be of the type of date used in the set
 // Displays "java.lang.Integer"
 System.out.println(stringType.get("foo").getClass().getName());
```

This data type is also preserved across serialization and deserialization by xsi:type. Consequently, any time you serialize an anySimpleType element, it will have an xsi:type that matches that defined in the SDO specification based on its Java type:

In the following example, you serialize the business object above so that the data would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:StringType xsi:type="p:StringType"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:p="http://StringType">
  <foo xsi:type="xsd:int">42</foo>
<p:StringType></p:StringType>
```

The xsi:type will be used during deserialization to load the data as the appropriate Java instance class. If no xsi:type is specified, then the default deserialization type will be string.

For the other simple types, determining mappability is a constant. For instance, A boolean can always map to a string. AnySimpleType can contain any of the simple types, however, so a mapping may or may not be possible based on the instance data in the field.

Use the property Type's URI and Name to determine if a property is of type anySimpleType. They will be ″commonj.sdo″ and ″Object″. To determine if data is valid to be inserted into anySimpleType, check to see if it is not an instance of a DataObject. All data that can be represented as a String and is not a DataObject is allowed to be set into an anySimpleType field.

This leads to the following mapping rules:

- anySimpleType can always map to anySimpleType.
- any other simple type can always map to anySimpleType.
- anySimpleType can always map to string because all simple types must be able to be converted into a string.
- anySimpleType may or may not be able to map to any of the other simple types depending on its value in the business object. This means that this mapping cannot be determined at design time, only at runtime.

Related information

Assigning from and to xs:any

Using AnyType for complex types

The anyType tag is not handled differently from any other complex type by the SDO APIs.

The only differences between anyType and any other complex types are in its instance data and serialization/deserialization, which should be internal concepts to business object only, and determining if data being mapped to or from the field is valid.Complex types are limited to a single type: Customer, Address, and so on. The anyType, however, allows any DataObject regardless of type. If maxOccurs > 1, then each DataObject in the list can of a different type.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnyType">
  <xsd:complexType name="AnyType">
    <xsd:sequence>
      <xsd:element name="person" type="xsd:anyType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://Customer">
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Employee" targetNamespace="http://Employee">
  <xsd:complexType name="Employee">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anyType = ...
 DataObject customer = ...
 DataObject employee = ...

 // Set the person to a Customer
 anyType.set("person", customer);

 // The instance data will be a Customer
 // Displays "Customer"
 System.out.println(anyType.getDataObject("person").getName());

 // Set the person to an Employee
 anyType.set("person", employee);

 // The instance data will be an Employee
 // Displays "Employee"
 System.out.println(anyType.getDataObject("person").getName());
```

Just like anySimpleType, anyType uses xsi:type during serialization to assure that the intended type of DataObject is maintained when deserialized. So when you set to ″Customer,″ the XML would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:customer="http://Customer"
    xmlns:p="http://AnyType">
  <person xsi:type="customer:Customer">
    <name>foo</name>
  </person>
</p:AnyType>
```

And when set to ″Employee″:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:employee="http://Employee"
    xmlns:p="http://AnyType">
  <person xsi:type="employee:Employee">
    <id>foo</id>
  </person>
</p:AnyType>
```

AnyType also allows for the setting of simple type values through wrapper DataObjects. These wrapper DataObjects have a single property named ″value″ (element) that holds the simple type value. The SDO APIs have been overridden to automatically wrap and unwrap these simple types and wrapper DataObjects when using the get<Type>/set<Type> APIs. The non-type casting get/set APIs will not perform this wrapping.

```
 DataObject anyType = ...

 // Calling a set<Type> API on an anyType Property causes automatic
 // creation of a wrapper DataObject
 anyType.setString("person", "foo");
```

```
// The regular get/set APIs are not overridden, so they will return
// the wrapper DataObject
DataObject wrapped = anyType.get("person");

// The wrapped DataObject will have the "value" Property
// Displays "foo"
System.out.println(wrapped.getString("value"));

// The get<Type> API will automatically unwrap the DataObject
// Displays "foo"
System.out.println(anyType.getString("person"));
```

When the wrapper DataObject is serialized, it will be serialized just like anySimpleType mapping of Java instance classes to XSD types in the xsi:type field. So this setting would serialize as:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd=" http://www.w3.org/2001/XMLSchema"
 xmlns:p="http://AnyType">
  <person xsi:type="xsd:string">foo</person>
</p:AnyType>
```

If no xsi:type is given or if an incorrect xsi:type is given, then an exception will be thrown. In addition to automatic wrapping, the wrapper can be manually created for use with the set() API through BOFactory createDataTypeWrapper(Type, Object) where Type is the SDO simple type of the data to be wrapped and Object is the data to be wrapped.

```
Type stringType = boType.getType("http://www.w3.org/2001/XMLSchema", "string");
DataObject stringType = boFactory.createByMessage(stringType, "foo");
```

To determine if a DataObject is a wrapper type, the BOType isDataTypeWrapper(Type) can be called.

```
DataObject stringType = ...
boolean isWrapper = boType.isDataTypeWrapper(stringType.getType());
```

For the other complex types, in order to move data from one field to another, the data must be of the same type. AnyType can contain any complex types, however, so a direct move with no mapping may or may not be possible based on the instance data in the field.

You can use the property Type’s URI and Name to determine if a property is of type anyType. They will be ″commonj.sdo″ and ″DataObject″. All data is valid to be inserted into an anyType. This leads to the following mapping rules:

- anyType can always map to anyType.
- any complex type can always map to anyType.
- any simple type can always map to anyType.
- anyType may or may not be able to map to any of the other simple or complex types depending on its value in the BO instance. This means that this mapping cannot be determined at design time, only at runtime.

Using Any to set global elements for complex types

You can use the <any/> tag to set global elements to a complex type.

An occurrence of the any tag makes the DataObject Type isOpen() method and the isSequenced() method return true. If the value for maxOccurs is > 1 on an any tag, it has no effect on the structure of the DataObject; it is only used as information

during validation. Similarly, the occurrence of multiple any tags in a type does not change the structure of the DataObject; they are used only for validating the location of open data that was set.

How do I know if my DataObject has an any tag?:

You can easily determine if instances of a DataObject have any values set within them by checking the instance properties to see if any of the open properties are attributes.

DataObject does not provide a mechanism for determining if a DataObject Type has an any tag. DataObjects only have the concept of "open" that applies to both any and anyAttribute and allows the free additional of any properties. While the presence of an any tag causes a DataObject to have isOpen() = true and isSequenced() = true, it might just have an anyAttribute tag and one of the reasons for being sequenced discussed in the Sequences topics. The following example demonstrates these concepts:

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// any values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have any values set

// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Elements
for (int i=definedPropertyCount; i < instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isElement(prop))
    {
        return true;  // Found an any value
    }
}

return false;  // Does not have any values set
```

How do I get/set any values?:

Performing a get on data that was set in an any field can be done in the same manner as any other element value if the name is known.

You can perform a get with the XPath "<name>" and it will be resolved. If the name is unknown, then the value can be found by checking the instance properties as in above. If there are multiple any tags, or an any tag with maxOccurs > 1, then the DataObject sequence will have to be used instead if it is important to determine which any tag the data originated from.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
```

```
    <xsd:sequence>
      <!-- Handle all these any one way -->
      <xsd:any maxOccurs="3"/>
      <xsd:element name="marker1" type="xsd:string"/>
      <!-- Handle this any in another -->
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Because the <any/> tag causes the DataObject to be sequenced determining which any value was set can be done by checking the Sequence for the position of the any properties.

You can determine which any tag the instance data for the following XSD belongs to by using the following code:

```
DataObject anyElemAny = ...
Seqeuence seq = anyElemAny.getSequence();

// Until we encounter the marker1 element, all the open data
// found belongs to the first any tag
boolean foundMarker1 = false;

for (int i=0; i<seq.size(); i++)
{
    Property prop = seq.getProperty(i);

    // Check to see if the property is an open property
    if (prop.isOpenContent())
    {
        if (!foundMarker1)
        {
            // Must be the first any because it occurs
            // before the marker1 element
            System.out.println("Found first any data: "+seq.getValue(i));
        }
        else
        {
            // Must be the second any because it occurs
            // after the marker1 element
            System.out.println("Found second any data: "+seq.getValue(i));
         }
    }
    else
    {
        // Must be the marker1 element
        System.out.println("Found marker1 data: "+seq.getValue(i));
        foundMarker1 = true;
    }
}
```

Setting an <any/> value is done by creating a global element property and adding that value to the sequence.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://GlobalElems"
 targetNamespace="http://GlobalElems">
  <xsd:element name="globalElement1" type="xsd:string"/>
  <xsd:element name="globalElement2" type="xsd:string"/>
</xsd:schema>

 DataObject anyElemAny = ...
 Seqeuence seq = anyElemAny.getSequence();
```

```
// Get the global element Property for globalElement1
Property globalProp1 = boXsdHelper.getGlobalProperty(http://GlobalElems,
"globalElement1", true);

// Get the global element Property for globalElement2
Property globalProp2 = boXsdHelper.getGlobalProperty(http://GlobalElems,
"globalElement2", true);

// Add the data to the sequence for the first any
seq.add(globalProp1, "foo");
seq.add(globalProp1, "bar");

// Add the data for the marker1
seq.add("marker1", "separator");  // or anyElemAny.set("marker1", "separator")

// Add the data to the sequence for the second any
seq.add(globalProp2, "baz");

// The data can now be accessed by a get call
System.out.println(dobj.get("globalElement1"); // Displays "[foo, bar]"
System.out.println(dobj.get("marker1");        // Displays "separator"
System.out.println(dobj.get("globalElement2"); // Displays "baz"
```

What are valid mappings for data in an any?:

An <any/> tag is a set of name/value pairs. The only valid mapping that can be determined at design time for <any/> is another <any/> or anyType that has the same maxOccurs value.

Individually, the values contained in an instance of a DataObject for any are basic complex types that follow all the rules of complex type mapping. Some of these complex types may be wrapped simple types, so these will follow the rules of simple type mapping.

Using AnyAttribute to set global attributes for complex types

The <anyAttribute/> tag allows a complex type to have any number of global attributes set to it.

Similar to the <any/> tag, the occurrence of the <anyAttribute/> tag makes the DataObject Type isOpen() method return true. Unlike the <any/> tag, however, the <anyAttribute/> tag does not cause a DataObject to be sequenced because attributes in XSD are not ordered constructs.

How do I tell if my DataObject has an anyAttribute tag?:

You can easily determine if instances of a DataObject have anyAttribute values set within them by checking the instance properties to see if any of the open properties are attributes.

DataObject does not provide a mechanism for determining if a DataObject Type has an anyAttribute tag. DataObjects only have the concept of ″open″ that applies to both any and <anyAttribute/> and allows the free additional of any Properties. While it is true that if a DataObject has isOpen() = true and isSequenced() = false, then it must have an anyAttribute tag, if isOpen() = true and isSequenced() = true, the DataObject Type might or might not have an anyAttribute tag.

DataObject provides metadata query methods to programmatically answer this and other questions about the XSD structure that was used to generate the DataObject. The InfoSet model can be queried if it is necessary to know if the anyAttribute tag is present. Because anyAttribute is singular and either is or is not true, business

objects will also provide a BOXSDHelper hasAnyAttribute(Type) method to allow determination as to if setting an open attribute on this DataObject would produce a valid result. The following example code demonstrates these concepts:

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// anyAttribute values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have anyAttribute values set

// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Attributes
for (int i=definedPropertyCount; i<instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isAttribute(prop))
    {
        return true;  // Found an anyAttribute value
    }
}

return false;  // Does not have anyAttribute values set
```

How do I get/set anyAttribute values?:

Setting an <anyAttribute/> value is done in the same way as setting an <any/>, but instead of a global element a global attribute is used.

Performing a get on data that was set in an anyAttribute field can be done in the same manner as any other attribute value if the name is known. You can perform a get with the XPath "@<name>" and it will be resolved. If the name is unknown, using the above code the values can be iterated and accessed one by one. The example code below shows this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyAttrOnlyMixed"
 targetNamespace="http://AnyAttrOnly">
  <xsd:complexType name="AnyAttrOnly">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://GlobalAttrs">
  <xsd:attribute name="globalAttribute" type="xsd:string"/>
</xsd:schema>

 DataObject dobj = ...

 // Get the global attribute Property that is going to be set
 Property globalProp = boXsdHelper.getGlobalProperty(http://GlobalAttrs,
```

```
"globalAttribute", false);

// Set the value on the dobj, just like any other data
dobj.set(globalProp, "foo");

// The data can now be accessed by a get call
System.out.println(dobj.get("@globalAttribute")); // Displays "foo"
```

What are valid mappings for data in an anyAttribute?:

The AnyAttribute tag is similar to the any tag, and comprises a set of name/value pairs. Consequently, the only valid mapping for anyAttribute is another anyAttribute.

Individually, the values contained in the anyAttribute data are basic simple types that follow all the rules of simple type mapping

Overriding a Service Data Object to Java conversion

Sometimes, the conversion the system creates between a Service Data Object (SDO) and a Java type object may not meet your needs. Use this procedure to replace the default implementation with your own.

Before you begin

Make sure that you have generated the WSDL to Java type conversion using either WebSphere Integration Developer or the genMapper command.

About this task

You override a generated component that maps a WSDL type to a Java type by replacing the generated code with code that meets your needs. Consider using your own map if you have defined your own Java classes. Use this procedure to make the changes.

Procedure

1. Locate the generated component. The component is named *java_class*Mapper.component.
2. Edit the component using a text editor.
3. Comment out the generated code and provide your own method.

   Do not change the file name that contains the component implementation.

Example

This is an example of a generated component to replace:

```
private Object datatojava_get_customerAcct(DataObject myCustomerID,
   String integer)
{

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

What to do next

Copy the component and other files to the directory in which the containing module resides, and either wire the component in WebSphere Integration Developer or generate an enterprise archive (EAR) file using the serviceDeploy command.

Overriding the generated Service Component Architecture implementation

Sometimes, the conversion the system creates between a Java code and a Service Data Object (SDO) may not meet your needs. Use this procedure to replace the default Service Component Architecture (SCA) implementation with your own.

Before you begin

Make sure that you have generated the Java to Web Services Definition Language (WSDL) type conversion using either WebSphere Integration Developer or the genMapper command.

About this task

You override a generated component that maps a Java type to a WSDL type by replacing the generated code with code that meets your needs. Consider using your own map if you have defined your own Java classes. Use this procedure to make the changes.

Procedure

1. Locate the generated component. The component is named *java_class*Mapper.component.
2. Edit the component using a text editor.
3. Comment out the generated code and provide your own method.

   Do not change the file name that contains the component implementation.

Example

This is an example of a generated component to replace:

```
private DataObject javatodata_setAccount_output(Object myAccount) {

 // You can override this code for custom mapping.
 // Comment out this code and write custom code.

 // You can also change the Java type that is passed to the
 // converter, which the converter tries to create.


return SDOJavaObjectMediator.java2Data(myAccount);


}
```

What to do next

Copy the component and other files to the directory in which the containing module resides, and either wire the component in WebSphere Integration Developer or generate an enterprise archive (EAR) file using the serviceDeploy command.

Runtime rules used for Java to Service Data Objects conversion

To correctly override generated code, or to determine possible runtime exceptions related to Java to Service Data Object (SDO) conversions, an understanding of the rules involved is important. The majority of the conversions are straightforward, but there are some complex cases where the runtime provides the best possibility when it converts the generated code.

Basic types and classes

The runtime performs a straightforward conversion between Service Data Objects and basic Java types and classes. Basic types and classes include:
- Char or java.lang.Character
- Boolean
- Java.lang.Boolean
- Byte or java.lang.Byte
- Short or java.lang.Short
- Int or java.lang.Integer
- Long or java.lang.Long
- Float or java.lang.Float
- Double or java.lang.Double
- Java.lang.String
- Java.math.BigInteger
- Java.math.BigDecimal
- Java.util.Calendar
- Java.util.Date
- Java.xml.namespace.QName
- Java.net.URI
- Byte[]

User-defined Java classes and arrays

When converting from a Java class or array to an SDO, the runtime creates a data object that has a URI that is generated by inverting the package name of the Java type and has a type equal to the name of the Java class. For example, the Java class com.ibm.xsd.Customer is converted to an SDO and URI http://xsd.ibm.com with type Customer. The runtime then inspects the contents of the Java class members and assigns the values to properties in the SDO.

When converting from an SDO to a Java type, the runtime generates the package name by inverting the URI and the name of the type equals the type of the SDO. For example, the data object with type Customer and URI http://xsd.ibm.com generates an instance of the Java package com.ibm.xsd.Customer. The runtime then extracts values from the properties of the SDO and assign those properties to fields in the instance of the Java class.

When the Java class is a user-defined interface, you must override the generated code and provide a concrete class that the runtime can instantiate. If the runtime cannot create the concrete class, an exception occurs.

Java.lang.Object

When a Java type is java.lang.Object the generated type is xsd:anyType. A module can invoke this interface with any SDO. The runtime attempts to instantiate a

concrete class the same way it does for user-defined Java classes and arrays, if the runtime can find that class. Otherwise, the runtime passes the SDO to the Java interface.

Even if the method returns a java.lang.Object type, the runtime converts to an SDO only if the method returns a concrete type. The runtime uses a similar conversion to that for converting user-defined Java classes and arrays to SDOs, as described by the next paragraph.

When converting from a Java class or array to an SDO, the runtime creates a data object that has a URI that is generated by inverting the package name of the Java type and has a type equal to the name of the Java class. For example, the Java class com.ibm.xsd.Customer is converted to an SDO and URI http://xsd.ibm.com with type Customer. The runtime then inspects the contents of the Java class members and assigns the values to properties in the SDO.

In either case, if the runtime is unable to complete the conversion an exception occurs.

Java.util container classes

When converting to a concrete Java container class such as Vector, HashMap, HashSet and the like, the runtime instantiates the appropriate container class. The runtime uses a method similar to that used for user-defined Java classes and arrays to populate the container class. If the runtime cannot locate a concrete Java class, the runtime populates the container class with the SDO.

When converting concrete Java container classes to SDOs, the runtime uses the generated schemas shown in "Java to XML conversion."

Java.util interfaces

For certain container interfaces in the java.util package, the runtime instantiates the following concrete classes:

Table 3. WSDL type to Java class conversion

Interface	Default concrete classes
Collection	HashSet
Map	HashMap
List	ArrayList
Set	HashSet

XML document validation

XML documents and business objects can be validated using the validation service.

In addition, other services require certain minimum standards or they throw a runtime exception. One of these is `BOXMLSerializer`.

You can use the `BOXMLSerializer` to validate XML documents before they are processed by a service request. The `BOXMLSerializer` validates the structure of XML documents to determine if any of the following types of errors are present:

- Invalid XML documents, such as those that are missing certain element tags.

- Not well-formed XML documents, such as those that contain missing closing tags.
- Documents containing parsing errors, such as errors in entity declaration.

When an error is discovered by the `BOXMLSerializer`, an exception will be thrown with problem details.

The validation can be performed for import and/or export of XML documents for the following services:

- HTTP
- JAXRPC web services
- JAX-WS web services
- JMS services
- MQ services

For the HTTP, JAXRPC, and JAX-WS services, the `BOXMLSerializer` will generate exceptions in the following manner:

- Imports –
  1. The SCA component invokes the service.
  2. The service invokes a destination URL.
  3. The destination URL responds with an invalid XML exception.
  4. The service fails with a runtime exception and message.
- Exports –
  1. The service client invokes the service export.
  2. The service client sends an invalid XML
  3. The export fails for the service and generates an exception and message.

For the JMS and MQ messaging services, the exceptions are generated in the following manner:

- Imports –
  1. The import invokes the JMS or MQ service.
  2. The service returns a response.
  3. The service returns an invalid XML exception.
  4. The import fails and generates a message.
- Exports –
  1. The MQ or JMS client invokes an export.
  2. The client sends invalid XML.
  3. The export fails and generates an exception and message.

You can view the logs for any messages generated by an XML validation exception. The examples below are messages generated by improper XML coding that was validated by the `BOXMLSerializer`

- JAXWS import

```
javax.xml.ws.WebServiceException: org.apache.axiom.om.OMException:
 javax.xml.stream.XMLStreamException: Element type "TestResponse" must be
 followed by either attribute specifications, ">" or "/>".

javax.xml.ws.WebServiceException: org.apache.axiom.soap.SOAPProcessingException:
 First Element must contain the local name, Envelope
```

- JAXRPC import

```
[9/11/08 15:16:27:417 CDT] 0000003e ExceptionUtil E
 CNTR0020E: EJB threw an unexpected (non-declared)
  exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
  Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXParseException: Element type "TestResponse"
   must be followed by either
  attribute specifications, ">" or "/>". Message being parsed:
    <?xml version="1.0"?><TestResponse
  xmlns="http://WSXMLValidation"<firstName>Bob</firstName>
    <lastName>Smith</lastName></TestResponse>
 faultActor: null
 faultDetail:
[9/11/08 15:16:35:135 CDT] 0000003f ExceptionUtil E   CNTR0020E: EJB threw an
  unexpected (non-declared) exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
 Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXException: WSWS3066E: Error: Expected 'envelope'
   but found  TestResponse
  Message being parsed: <?xml version="1.0"?><TestResponse
   xmlns="http://WSXMLValidation">
  <firstName>Bob</firstName><middleName>John</middleName>
   <lastName>Smith</lastName>
   </TestResponse>
 faultActor: null
 faultDetail:
```

- JAXRPC/JAXWS export

```
[9/11/08 15:35:13:401 CDT] 00000064 WebServicesSe E
  com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
  getSoapAction WSWS3112E:
   Error: Generating WebServicesFault due to missing SOAPAction.
                  WebServicesFault
 faultCode: Client.NoSOAPAction
 faultString: WSWS3147E: Error: no SOAPAction header!
 faultActor: null
 faultDetail:
```

For more information about validation services, see the BOInstanceValidator interface in the Generated API and SPI documentation in the Reference section.

Protocol header propagation from non-SCA export bindings

The context service is responsible for propagating the context (including the protocol headers, such as the JMS header, and the user context, such as account ID) along a Service Component Architecture (SCA) invocation path. The context service offers a set of APIs and configurable settings.

When the context service propagation is bi-directional, the response context will always overwrite the current context. When you are running an invocation from one SCA component to another, a response will contain a different context. A service component will have an incoming context, but when you invoke another service, the other service will overwrite the original outgoing context. The response context becomes the new context.

When the context service propagation is one-way, the original context remains the same.

The lifecycle of the context service is associated with an invocation. A request has associated context, and the lifecycle of that context is bound to the processing of that particular request. When that request is finished processing, then the lifecycle of that context ends.

For a short-running Business Process Execution Language (BPEL) process, the response context overwrites the request context. It takes the response context back from the first request and pushes it to the next request. For a long-running BPEL process, the response context is discarded by the BPEL framework. It stores the original context and uses that context when making other outgoing calls.

Example

For example, the context including a protocol header is propagated throughout the invocation path starting with a request coming in to BPEL from a SOAP web service. BPEL processes it, and calls out of BPEL are made sequentially to a Web service binding outbound, and then to another Web service binding outbound. A request from the SOAP Web service uses the context service to pass on the protocol header. It takes the context service from the inbound request and pushes the protocol header outbound.

You can see similar behavior with another SCA component in place of the BPEL in this example.

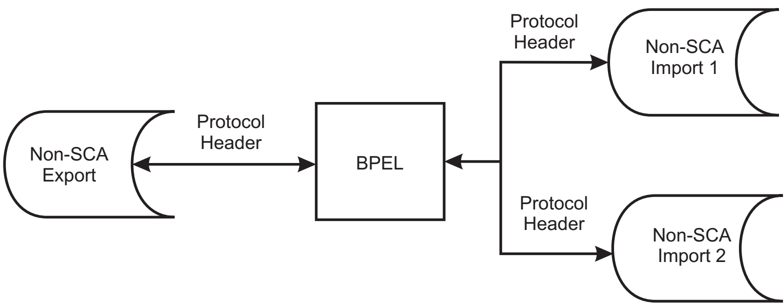


Figure 9. Propagating context including protocol header

Here is a code example.

```
//Import the necessary classes;
import com.ibm.bpm.context.ContextService;
import com.ibm.websphere.sca.ServiceManager;
import com.ibm.bpm.context.cobo.ContextObject;
import com.ibm.bpm.context.cobo.ContextObjectFactory;
import com.ibm.bpm.context.cobo.HeaderInfoType;
import com.ibm.bpm.context.cobo.UserDefinedContextType;

//Locate ContextService;
ContextService contextService = (ContextService)ServiceManager.INSTANCE.locateService
  ("com/ibm/bpm/context/ContextService");

// Get header info
HeaderInfo headerInfo = contextService.getHeaderInfo();
// Get user defined context in current execution context
UserDefinedContextType userDefinedContext = contextService.getUserDefinedContext();
if(userDefinedContext == null){ // create a new context if context is null
userDefinedContext = ContextObjectFactory.eINSTANCE.createUserDefinedContextType()
}

// Do some modification to header info and userDefinedContext
```

```
// Set user defined context back to the current execution context.
contextService.setUserDefinedContext(userDefinedContext);

// Set header info back to the current execution context.
contextService.setHeaderInfo(headerInfo);
```

Note: In the mediation flow component, `ContextService` APIs must not be used. Use the SMO programming model to access the context.

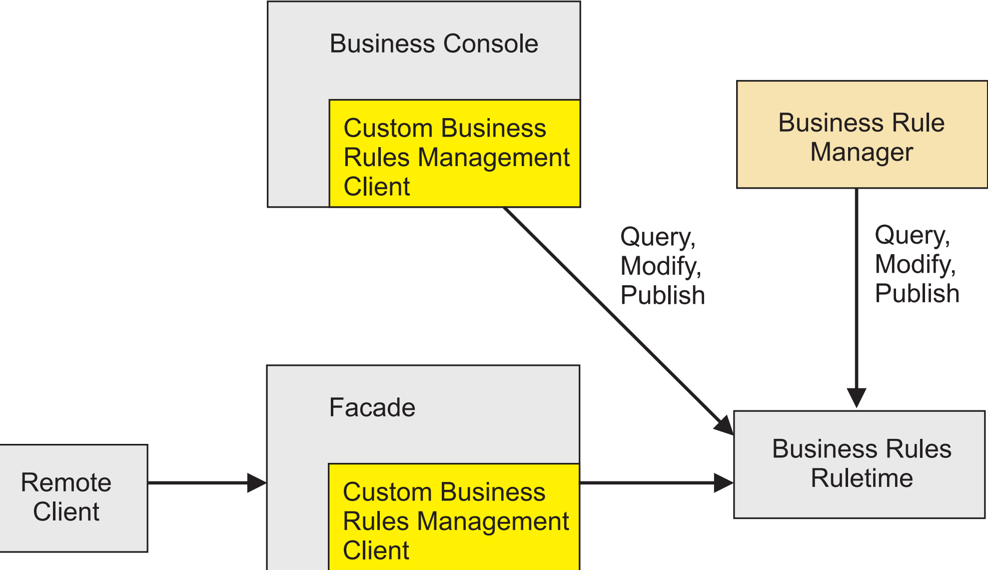
Context services have configurable rules and tables that dictate the binding behavior. For more information, see the Generated API and SPI documentation that is available in the Reference section. During development in WebSphere® Integration Developer, you can set the context service on import-export properties. For more details, see the import and export bindings information in the WebSphere Integration Developer information center.

Business rule management

Public business rule management classes are provided for the building of custom management clients or to automate changes to business rules.

Business rules management classes could be used in a Web application where they are combined with other management capabilities for things such as business process or human tasks in order to manage all components from a single client. Any custom management clients can be used along side the Business Rule Manager Web application included with WebSphere Process Server. The classes could also be used to automate changes to business rules within an application. For example business rules could be changed as the results of a business process that is using the business rules exceeds some threshold or limit.

The business rule management classes must be used in an application installed on WebSphere Process Server. The classes do not provide a remote interface, however they can be wrapped in a facade which is then exposed over a specific protocol for remote execution.



This programming guide is composed of two main sections and an appendix. The first section explains the programming model and how to use the different classes. Class diagrams are provided to show the relationship between classes. The second

section provides examples on using the classes to perform actions such as searching for business rule groups, scheduling a new rule destination, and modifying a rule set or decision table. The appendix contains additional classes that were used in the examples to simplify common operations and additional examples of creating complex queries for searching for business rule groups using wildcards.

Besides this programming guide information on the classes is also available in Javadoc HTML format included with both the WebSphere Process Server v6.1 and test environment included with WebSphere Integration Developer v6.1. This Javadoc documentation is available at ${WebSphere Process Server Install Directory}\web\apidocs or ${WebSphere Integration Developer Install Directory}\runtimes\bi_v61\web\apidocs. The packages com.ibm.wbiserver.brules.mgmt.* contain all of the information.

Programming model

WebSphere Business Integration Business Rules are authored with two different authoring tools and issued by the rule runtime. All three share the same model for the business rule artifacts.

Sharing of the model was deemed critical for not only ease of future maintenance, but for a consistent programming model for the end user. Sharing this model required compromises between the needs of desktop tooling and runtime execution and authoring -- all have clear sets of requirements to meet for their respective environments and these requirements at times conflicted with each other. The artifacts described below as part of the overall programming model represent a balance in meeting the requirements of these different environments.

Modification of business rules is limited to only those items that are defined with templates in the rule sets and decision tables as well as the operation selection table (effective dates and targets). Creation of new rule sets and decision tables is only supported through the copy of an existing rule set or decision table. The business rule group component itself is not eligible for dynamic authoring in the runtime with the exception of the user defined properties and description values. Changes that need to be made to the component (for example, adding a new operation) must be done using WebSphere Integration Developer and then redeployed or reinstalled in the server.

Business Rule Group

The BusinessRuleGroup class represents the business rule group component. The BusinessRuleGroup class can be considered the root object which contains rule sets and decision tables.

Rule sets and decision tables can only be reached through the business rule group that they are associated with. Methods are provided on the class to retrieve information about the business rule group and to reach the rule sets and decision tables. Through the methods the following information can be retrieved:

- Target name space
- Name of business rule group
- Display name
- Name/Display name synchronization
- Description
- Presentation time zone which indicates whether dates should be displayed in UTC format or local to the system

- Operations defined in the interface associated with the business rule group
- Custom properties defined on the business rule group

The different rule sets and decision tables associated with the business rule group can be reached through the business rule group's operation.

There are also methods that allow for information to be updated on the business rule group. Through the methods, the following information can be updated:
- Description
- Display name
- Name/Display name synchronization
- Custom properties defined on the business rule group

The Display name for the business rule group can be set explicitly or it can be set to the value of the Name using the setDisplayNameIsSynchronizedToName method.

Other values cannot be modified as these are part of the business rule group component definition and changes to these values would require a redeploy as well as reinstallation.

The business rule group class also provides a refresh method. This method will make a call to the persistent storage or repository where the business rules are stored and return the business rule group and all of the associated rule sets and decision tables with the persisted information. The returned business rule group is the latest copy and the previous object is obsolete.

The isShell method can be used to tell if a business rule group instance is of a version that is not supported by the current runtime. For example, if a web client was created with the current business rule management classes, and in the future new capabilities are added to the business rule group that are not supported by the classes, a shell business rule group will be created when the business rule group is retrieved. This allows the web client to continue to work with business rules that are supported and still retrieve business rule groups with limited attributes and capabilities. When isShell is true, only the methods getName, getTargetNameSpace, getProperties, getPropertyValue, and getProperty will return values. All other methods will result in an UnsupportedOperationException. Besides using the isShell method, the type of the BusinessRuleGroup can also be checked if it is an instance of BusinessRuleGroupShell in order to determine if it is of a supported version.

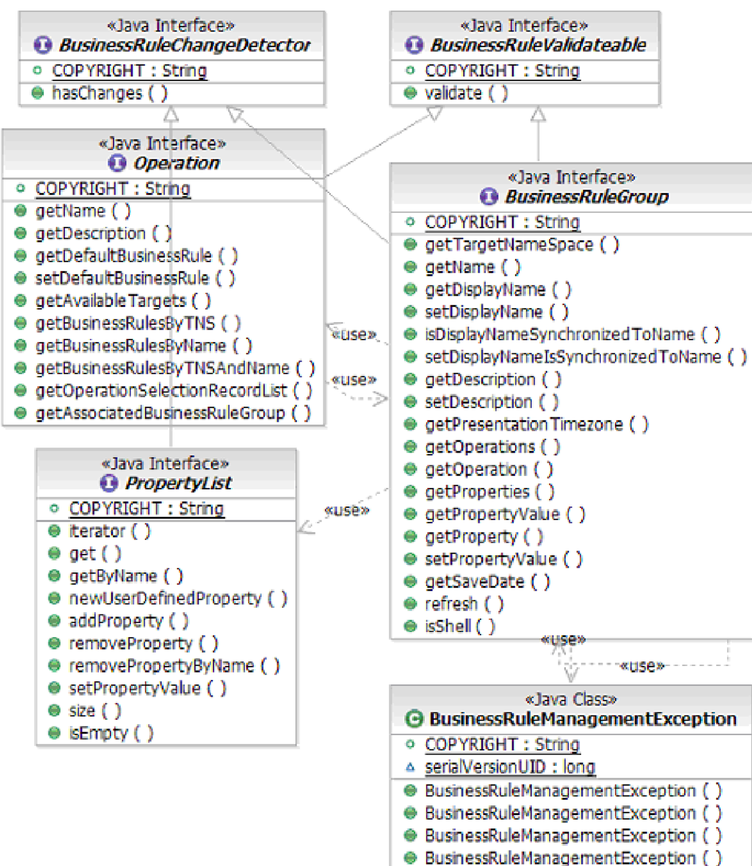


Figure 10. Class diagram of BusinessRuleGroup and related classes

Business Rule Group Properties

The properties on business rule groups are intended to be used for management of business rule groups. Properties set on business rule groups can be used in queries to return only a subset of business rule groups which are to be displayed and then modified.

All properties are of type string and defined as name-value pairs. Each property can only be defined once in a business rule group. For each property defined, it must have a value also defined. The property value can be an empty string or zero in length, but not null. Setting a property to null is the same as deleting the property.

The properties on a business rule group can also be accessed in a rule set or decision table at runtime. This allows a single value to be set at the business rule group to be used within multiple rule sets or decision tables in the business rule group. Only those properties defined on the business rule group are available to enclosed rule sets and decision tables.

There are two types of properties, system and user-defined. The number of system or user-defined properties is not limited on a business rule group. The system properties are used to hold specific component information such as the version of the rule model used in defining the rule logic. This system information is exposed in properties to allow for query across these fields. The system properties begin

with a prefix IBMSystem and are read-only through the business rule group and property classes. System properties can not be added, changed or deleted. An example of a system property is:

Property Name	Property Value
IBMSystemVersion	6.2.0

Note: The values of name, namespace and display name for a business rule group are treated as system properties for query purposes, and will be part of the list of properties that can be retrieved for a business rule group with the getProperties method. These properties are not however, defined as actual property elements in the business rule group artifact and are not seen as properties in WebSphere Integration Developer as they are defined with separate and unique elements on the business rule group. They are solely provided to offer more query options.

User-defined properties are available to be used for holding any customer specific information and can also be used in queries for business rule groups. User-defined properties are available for read-write.

The properties for a business rule group can be retrieved either individually or as a list (PropertyList object). With the PropertyList, methods are provided for retrieving individual properties and adding and removing user-defined properties.

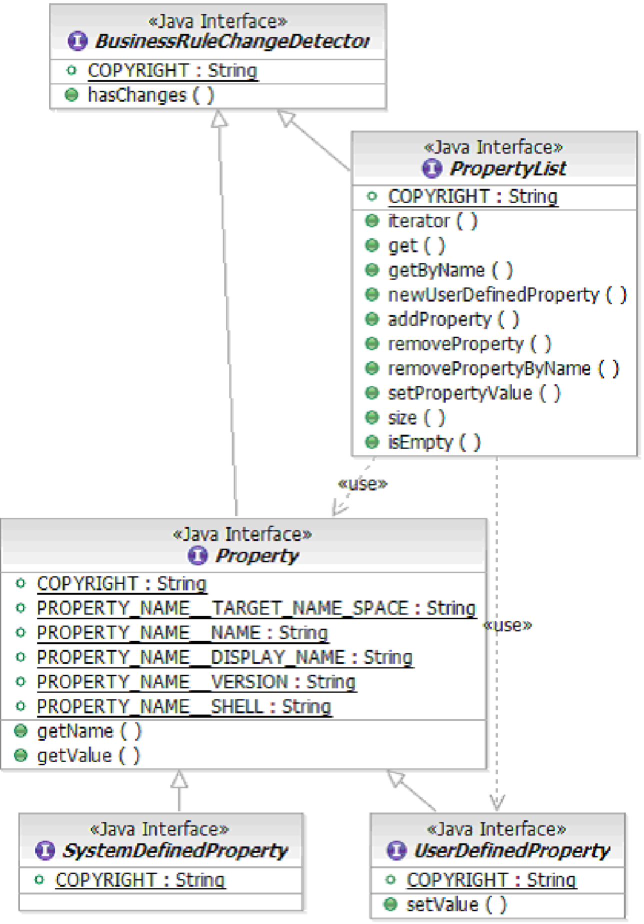


Figure 11. Class diagram of Property and related classes

Operation

Operations are starting points for reaching individual rule sets and decision tables to modify. The operations of a business rule group match the operations listed in the WSDL which is associated with the business rule group component.

For each operation, there are different targets, each of which is a business rule (rule set or decision table):

- Default target (optional)
- List of targets scheduled by date/time ranges (OperationSelectionRecord)
- List of all available targets that can be used for that operation

Each operation must have at least one business rule target specified. This target can be an OperationSelectionRecord with a specific start date and end date when the target should be scheduled to be active. The operation can also have a single default target set which is used during execution when no matching scheduled business rule target is found. The Operation class provides methods for retrieving and setting the default business rule target as well as retrieving the list (OperationSelectionRecordList) of scheduled business rule targets. Besides the default business rule target and the scheduled business rule targets, there is a list of all available business rule targets for the operation. This list will include those business rules targets which are scheduled and the default business rule target as well as any other rule sets or decision tables which are not scheduled for this operation. An unscheduled rule set or decision table is associated with the operation through the available target list by the fact that it implicitly shares the operation information. All business rule targets must support the input and output messages for their operation. With each operation unique on an interface, the rule sets and decision tables for an operation are unique from those rule sets and decision tables of another operation.

Any of the different rule sets and decision tables in the available targets list can be scheduled to be active through the creation of an OperationSelectionRecord. Along with the particular rule set or decision table from the available targets list, a start date and end date must be specified. The start date must be before the end date. The dates can be for a time which covers the current date as well as the past and the future. The time span of the dates cannot overlap with any other OperationSelectionRecords once it is added to the OperationSelectionRecordList and published. The start date and end date values are of type java.util.Date. Any values which are specified will be treated as UTC values according to the java.util.Date class. With the OperationSelectionRecord complete, it can be added to the OperationSelectionRecordList to be scheduled along with other business rule targets. Gaps may exist between the time spans of different OperationSelectionRecords. When a gap is encountered during execution, the default target is used. If no default target has been specified, an exception will be thrown. It is recommended to always specify a default business rule target.

A scheduled business rule target can be removed from the list of scheduled targets by removing the OperationSelectionRecord from the OperationSelectionRecordList. Removing an OperationSelectionRecord will not remove the business rule target from the list of available business rule targets and it will not remove any other OperationSelectionRecords which have the same business rule target scheduled.

Besides retrieving a rule set or decision table through the OperationSelectionRecordList or list of available targets, the Operation class also

allows for business rule targets to be retrieved by name and target namespace property values. Through the methods on the Operation class, those rule sets and decision tables which are listed in the available targets for that operation can be queried. Rule sets and decision tables which might have matching name and target namespace values, but are part of the available target lists of other operations, will not be included in the result set. As a convenience, the getBusinessRulesByName, getBusinessRulesByTNS, and getBusinessRulesByTNSAndName methods are provided to simplify retrieving specific rule sets and decision tables.

The Operation class provides methods that support the following:
- Retrieve the operation name
- Retrieve the operation description
- Retrieve and set the default business rule target
- Retrieve the scheduled business rule targets (OperationSelectionRecordList)
- Retrieve the list of all available business rule targets
- Retrieve a rule set or decision table from the list of all available targets by name or target namespace
- Retrieve the business rule group with which the operation is associated

The OperationSelectionRecordList class provides methods that support the following:
- Retrieve a specific OperationSelectionRecord by index value
- Remove a specific OperationSelectionRecord by index value
- Add a new OperationSelectionRecord to the list

The OperationSelectionRecord class provides methods that support the following:
- Retrieve and set the start date
- Retrieve and set the end date
- Retrieve and set the business rule target
- Retrieve the operation with which the OperationSelectionRecord is associated

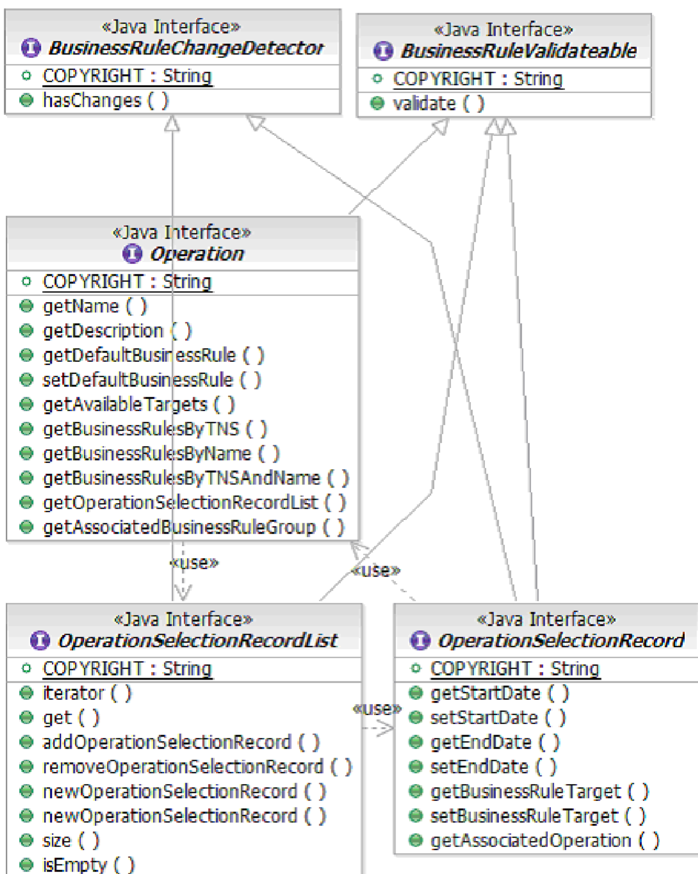


Figure 12. Class diagram for Operation and related classes

Business Rule

The RuleSet and DecisionTable classes are based off a generic BusinessRule class with methods that provide information that is available on both rule sets and decision tables.

Similar to business rule group artifacts, rule sets and decision tables have a name and a target namespace. The combination of these values must be unique when compared to other rule sets and decision tables. For example, two rule sets can share the same target namespace value, but must have different names or a rule set and decision table could have the same name, but have different target namespace values.

A copy of a business rule can be made from an existing business rule for situations where a similar rule is required to be scheduled at a specific time with different parameter values for rules constructed from templates. New rules cannot be fully created from scratch as there must be a backing Java class to provide the implementation for the business rule. The backing Java class is only created at deploy time. When a new rule is created, it is added to the list of available targets for the operation which is associated with the original rule. The additional rule is not persisted however, until the business rule group with which the operation is associated is published.

The new business rule must have either a different target namespace or name from the original rule. The display name for the new business rule can remain the same as the original rule as the combination of the name and namespace provide a key value to identify the business rule. Within the business rule, the different parameter values which have been defined with a template can be modified. Scheduling the business rule at a certain time can be done with the OperationSelectionRecordList or as a default destination with the Operation associated with the business rule.

The BusinessRule class provides methods that support the following:

- Retrieve the target namespace
- Retrieve the name of the rule set or decision table
- Retrieve and set the display name of the rule set or decision table
- Retrieve the type of the business rule, either rule set or decision table
- Retrieve and set the description for the business rule
- Retrieve the operation that the business rule is associated with.
- Create a copy of the business rule with a different name and/or target namespace

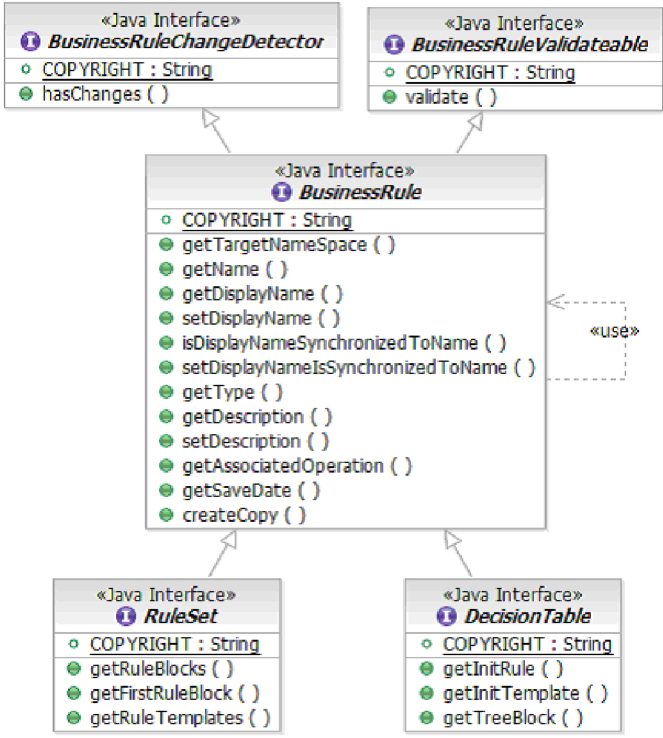


Figure 13. Class diagram of BusinessRule and related classes

Rule set

A rule set is one type of business rule. Rule sets are typically used when multiple rules may need to be executed based on different conditional values. Rule sets are composed of a rule block and rule templates. The rule block (`RuleBlock`) contains the different if-then and action rules which make up the logic of the rule set.

The `RuleSet` class provides methods that support the following:

- Retrieve a list of rule blocks for the rule set

- Retrieve a list of rule templates defined in the rules set

Currently each rule set can only have one rule block, while there can be multiple rule templates defined in the rule set. The rule block contains the set of rules that will be executed when the rule set is invoked. The rule block allows for the order of the rules to be modified. A rule block must have at least one rule defined. The rules (`Rule`) can be defined as template instance rules (TemplateInstanceRule) or hard-coded. If an if-then or action rule has been defined with a template, it can be removed from the rule block. If a new instance of rule was created with a template, it can be added to the rule block.

If a rule is hard-coded and was not defined with a template, it cannot be modified or removed from the rule block. The expectation with these rules is that they have been designed to always be part of the rule set logic and are not to be changed or repeated within the logic.

When a new rule is created with a template, it must have a unique name value. The list of existing rules can be retrieved and checked first before creating the rule.

For hard-coded if-then and action rules, only the name and presentation can be retrieved. The presentation is a string which can be used to display information about the rule in client applications. For if-then or action rules that are defined with a template, the name and presentation can be retrieved as well as additional information. Specific parameter values can be retrieved and changed. With a template (`RuleSetRuleTemplate`) defined in the rule set, another instance of the rule can be created within the rule set and parameter values can be set. For example, if you have a rule saying that a customer of a particular status level receives a discount of a specific amount. This logic could be defined with a single rule template and then repeated with parameter values changed for the status level (gold, silver, bronze, and so on) and the discount amount (15%, 10%, 5%, and so on).

The parameters for a rule defined with a template are specific to the instance of the rule. The template only defines a standard presentation and the number of parameters for the rule. Each rule defined with a template can have different values as explained in the example on discounts for different customer status.

The RuleBlock class provides methods that support the following:
- Retrieve a rule by index
- Add a rule that was defined with a template
- Remove a rule defined with a template
- Modify the order of a rule by one place or to a specific index location

The `RuleSetRule` class provides methods that support the following:
- Retrieve the name of the rule
- Retrieve the display name of the rule
- Retrieve the user presentation
- Retrieve the rule block

The `RuleSetRuleTemplate` class provides methods that support the following:
- Create a rule template instance from this template definition
- Retrieve the parent rule set

The `TemplateInstanceRule` class provides methods that support the following:
- Retrieve the parameters for the rule
- Retrieve the template definition which defined the rule

The `Template` class provides methods that support the following:
- Retrieve the template ID
- Retrieve the name
- Retrieve and set the display name
- Retrieve and set the description
- Retrieve the parameters for this template
- Retrieve the user presentation

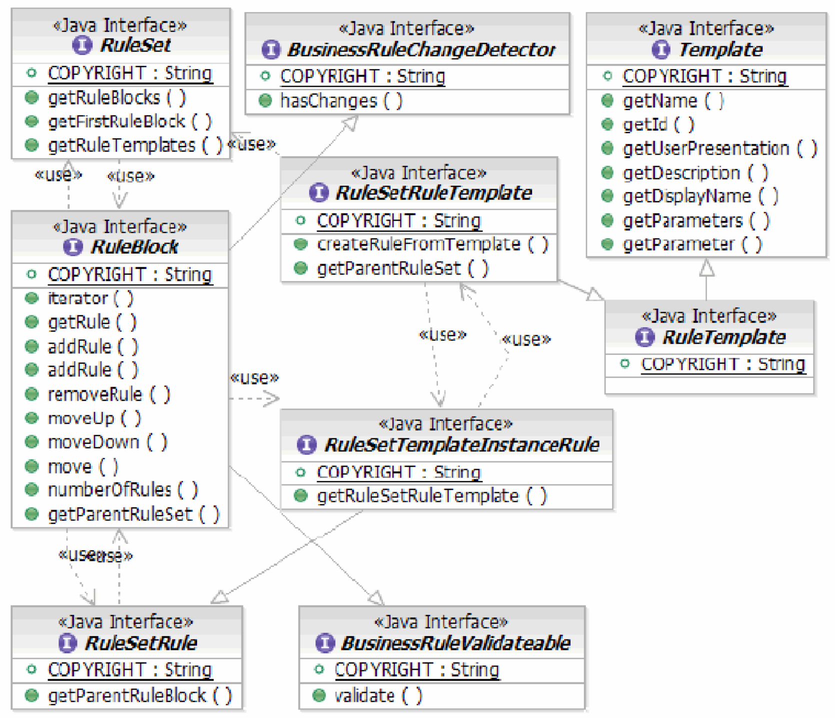


Figure 14. Class diagram of BusinessRule and related classes

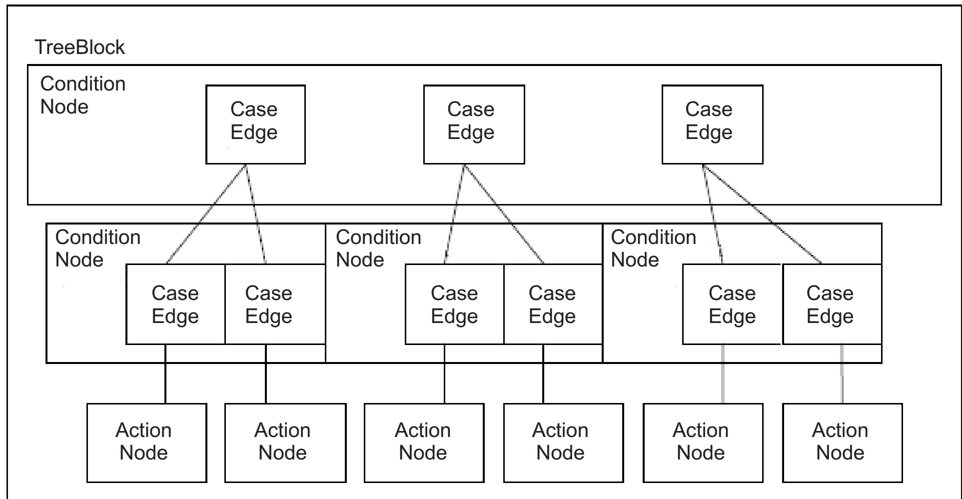
Decision table

Decision tables are another type of business rule which can be managed and modified. Decision tables are typically used when there are a consistent number of conditions which must be evaluated and a specific set of actions to be issued when the conditions are met.

Decision tables are similar to decision trees, however they are balanced. Decision tables always have the same number of conditions to be evaluated and actions to be performed no matter what set of branches are resolved to true. A decision tree may have one branch with more conditions to evaluate than another branch.

Decision tables are structured as a tree of nodes and defined by a `TreeBlock`. There are different `TreeNodes` which make up the `TreeBlock`. `TreeNodes` can be condition nodes or action nodes. Condition nodes are the evaluation branches. At the end of branches, there are action nodes that have the appropriate tree actions to issue

should all of the conditions evaluate to true. There can be any number of levels of condition nodes, but only one level of action nodes.



Decision tables might also have an initialization rule (init rule) which can be issued before the conditions in the table are checked.

The `DecisionTable` class provides methods that support the following:

- Retrieve the tree block of tree nodes (condition and action nodes)
- Retrieve the init rule instance
- Retrieve the init rule template if defined

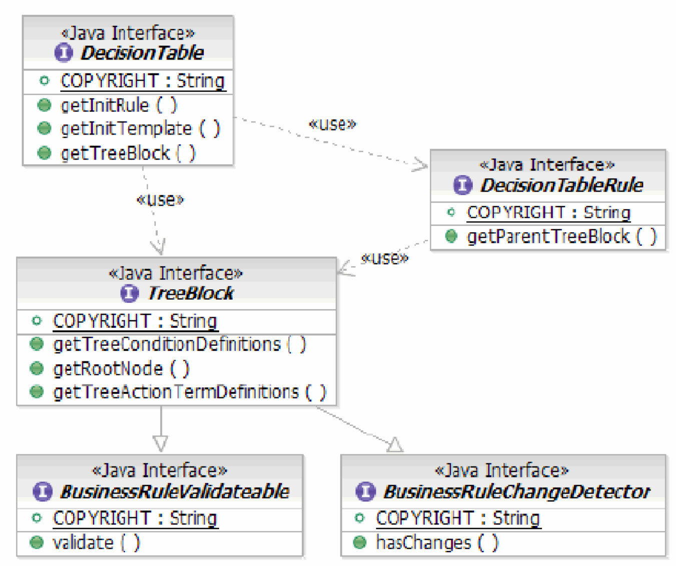
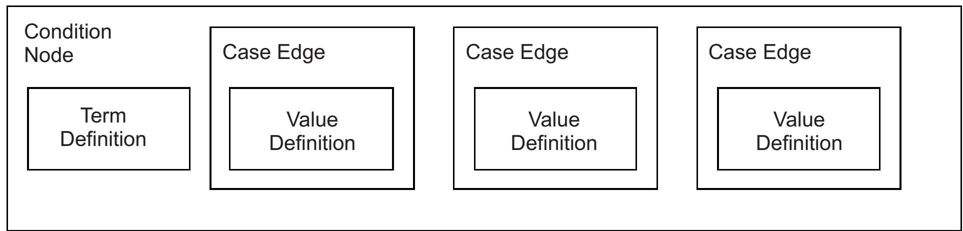


Figure 15. Class diagram of DecisionTable and related classes

The `TreeBlock` of a decision table contains the different condition and action nodes. Each condition node (`ConditionNode`) has a term definition (`TreeConditionTermDefinition`) and one to n case edges (`CaseEdge`). The term

definition contains the left-hand operand for the condition expression. The case edges contain the value definitions which are the different right-hand operands to be used in the condition expression. For example, in the expression (status == "gold") the term definition would be "status" and "gold" would be the value definition in the case edge. For all of the case edges in a condition node, they share the term definition and are only different by the value (`TreeConditionValueDefinition`). Continuing with the example, another case edge in the condition node could have a value "silver". This would be used in an expression too (status == "silver"). The only exception to this behavior is if an otherwise has been defined for the condition node. With an otherwise, there is no value definition as it is used if all other case edges within the condition node evaluate to false. While an otherwise is not a case edge, it does have a `TreeNode` that can be retrieved.



For the term definition, the user presentation can be retrieved and used in client applications. The presentation for the term definition is typically only a representation of the left-hand operand (status in our example) and does not contain any placeholders. For the case edges, a template can be used to define the value definition (`TreeConditionValueTemplate`). A template value definition instance (`TemplateInstanceExpression`) holds the parameter values which are used for execution and can be modified. If an attempt is made to retrieve the value template definition for a `TreeConditionValueDefinition` that was not defined with a template, a null value will be returned. If a template has not been used to define the value condition, a user presentation can still be retrieved and used in client applications if it was specified at authoring time.

The `TreeBlock` class provides methods that support the following:
- Retrieve the root node of the tree
- Retrieve the condition term definitions for the tree block
- Retrieve the action term definitions for the tree block

The root node of the tree is of type `TreeNode` and from here, navigation of the decision table can occur. The `TreeNode` class provides methods that support the following:
- Determine if a node is an otherwise clause
- Retrieve the parent node for the current tree node (condition or action node)
- Retrieve the root node of the tree containing the current tree node

The `ConditionNode` class provides methods that support the following:
- Retrieve the case edges
- Retrieve the term definition
- Retrieve the otherwise case

- Retrieve the templates for the value conditions of the case edges for the condition node
- Add a condition value based on a template to the node
- Remove a condition value based on a template

The `CaseEdge` class provides methods that support the following:

- Retrieve the list of value templates which are available for the value definition
- Retrieve the child node (condition or action node)
- Retrieve the instance of the template definition associated with the value definition
- Retrieve the value definition directly without retrieving the template
- Set the value for the definition to use a specific template instance definition

The `TreeConditionTermDefinition` class provides methods that support the following:

- Retrieve the value definition templates defined for the condition node
- Retrieve the user presentation of the condition term

The `TreeConditionDefinition` class provides methods that support the following:

- Retrieve the term definition for the condition node
- Retrieve the condition value definitions for the condition node from all of the case edges
- Retrieve the orientation (row or column)

The `TreeConditionValueDefinition` class provides methods that support the following:

- Retrieve the specific template instance expression defined for the value
- Retrieve the user

The `Template` class provides methods that support the following:

- Retrieve the system ID for the template
- Retrieve the name of the template
- Retrieve the parameters defined for the template
- Retrieve the presentation for the template

The `TreeConditionValueTemplate` class provides a method that supports the following:

- Create a new template condition value instance

The `TemplateInstanceExpression` class provides methods that support the following:

- Retrieve the parameters for the template instance
- Retrieve the template (`TreeConditionValueTemplate` in the case of a case edge in a decision table) that was used to define the instance

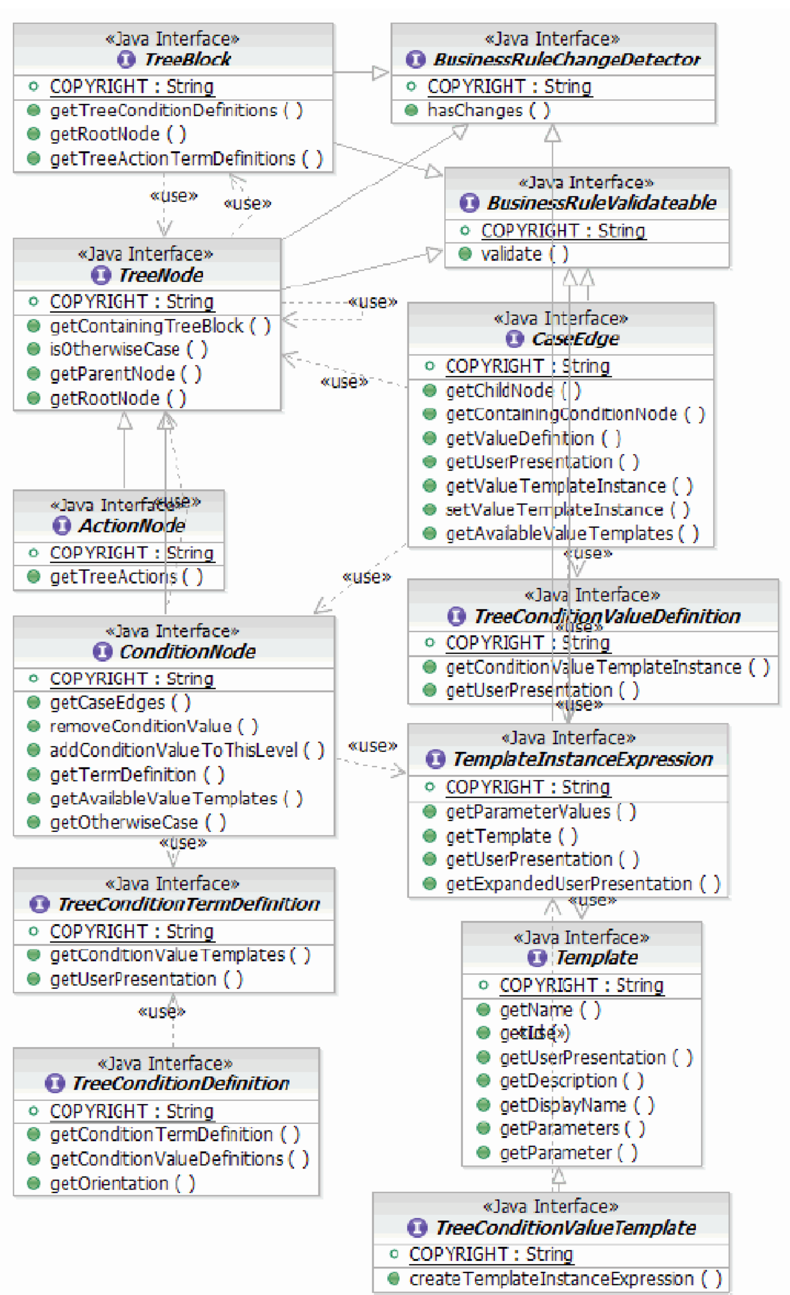


Figure 16. Class diagram for TreeNode and related classes

When a new case edge is added to a condition node, the new case edge must use a template to define the value. For example if a new case edge of "bronze" was to be added for checking 'status', the appropriate template (`TreeConditionValueTemplate`) would need to be used to create a new TemplateInstanceExpression, setting the parameter value to "bronze".

When a new case edge is added, it will also have a child condition node added to it automatically. This child condition node will contain case edges which are based on the case edge definitions that have been defined for condition nodes at that same level. If templates or hard-coded values are used in case edges, they will then be used in the child condition node's case edges as well. The child condition node that is added automatically will also have its own child condition nodes created

automatically. These child condition nodes will also have child condition nodes and so on until all levels of condition nodes have been recreated.

Besides the condition nodes, a decision table and more specifically tree block, also contains a level of action nodes (`ActionNode`). The action nodes are leaf nodes and reside at the end of the branch of condition nodes and the case edges. Should all of the condition values in a line of case edges resolve to true, an action node is reached. The action node will have at least one action (`TreeAction`) defined. For the action, there will be a term definition and value definition. Just as with the condition nodes, the term definition (`TreeActionTermDefinition`) is the left-hand side of the expression and the value definition (`TemplateInstanceExpression`) is the right-hand side of the expression. For example, for the different condition nodes which were checking on the status, there might be actions to define the discount. If the condition was (status == "gold"), the action can be (discountValue = 0.90). For the action the "discountValue" would be the term definition and the "= 0.90" would be the value definition.

The term definition of a tree action is shared with other tree actions in other action nodes. Since every branch of case edges reaches an action, the same term definitions are used. The value definitions however, can be different per tree action and action node. For example the discountValue for a status of "gold" can be "0.90"; however the "discountValue" for a status of "silver" can be "0.95".

Action nodes can have multiple tree actions which have a separate term definition and separate value definition. For example, if the discount was being determined for a rental car, besides setting the discountValue, you can also want to assign a specific level of car. Another tree action could be created to set the "carSize" term to "full size" if the status was "gold" as well as set the "discountValue" to "0.90".

The value definition in a tree action can be created from a template (`TreeActionValueTemplate`). The template definition contains an expression (`TemplateInstanceExpression`) which has the parameters for the expression.

Besides changing the parameters, the entire value definition can be modified with a new value definition instance which is created with another template which was defined for the tree action.

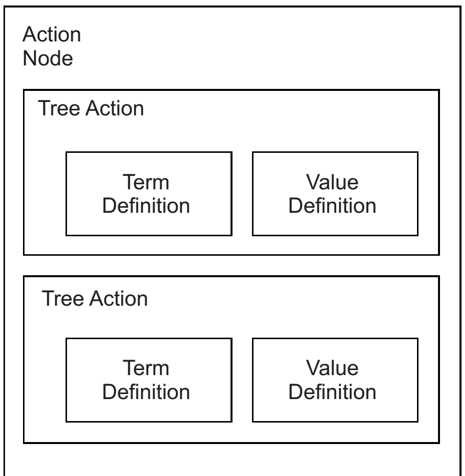
If a value definition is not created with a template, it cannot be changed. For client applications, the user presentation can be used in display if it was specified at author time.

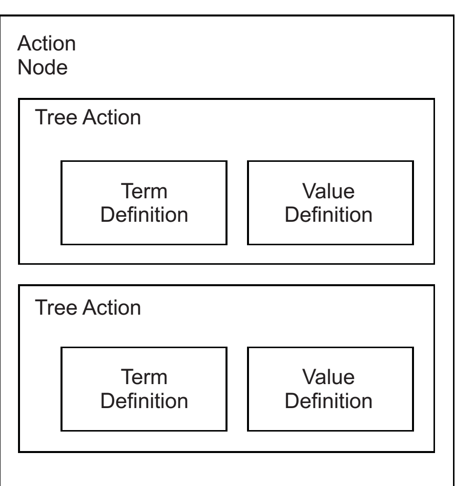
For term definitions in tree actions, if a user presentation has been specified, it can also be used by client applications.

When a new case edge is added to a condition node and the different child condition nodes are created, action nodes will also be created. Unlike the child condition nodes and case edges which are created based on the definition of the case edges already defined for that level, action nodes do not automatically inherit an existing design. Only empty placeholder `TreeActions` are created in the action node. A template (`TreeActionValueTemplate`) must be used to complete the action definition by creating a `TemplateInstanceExpression` for at least one term definition for the action node. Until the tree action is set with a `TemplateInstanceExpression`, the tree action will have null values specified for the user presentation value and template instance value.

When creating a new condition that results in new ActionNodes, the action nodes will be added to the right of existing actions for the immediate parent condition node. For example if a status of “ruby” is added to the decision table and should have a specific discount, the condition to check the status is added at the right of “gold”, “silver”, and “bronze”. The action node for the discount for “ruby” will be added to the right of the action nodes that correspond to the “gold”, “silver” and “bronze” case edges.

When setting new tree actions for action nodes, an algorithm that looks to the rightmost action node at the lowest case edge will return the action node with an empty tree action. The tree action can also be checked that it has null values for the user presentation value and template instance value. Once the tree action is obtained, it can be set with the correct instance of a `TreeActionValueTemplate`.





The `ActionNode` class provides a method that supports the following:
- Retrieve a list of the defined tree actions

The `TreeAction` class provides methods that support the following:
- Retrieve a list of the available value templates defined for the tree action
- Retrieve the term definition
- Retrieve the value template instance defined for the tree action
- Retrieve the user presentation for the value if a value template was not used
- Check if the action is a SCA service invocation (`isValueNotApplicable` method)
- Replace the value template instance with a new instance

The `TreeActionTermDefinition` class provides methods that support the following:
- Retrieve the user presentation for the term value definition
- Retrieve a list of the value templates available for the tree action
- Check if the action is a SCA service invocation (`isTermNotApplicable` method)

The `Template` class provides methods that support the following:
- Retrieve the system ID for the template
- Retrieve the name of the template

- Retrieve the parameters defined for the template
- Retrieve the presentation for the template

The `TreeActionValueTemplate` class provides a method that supports the following:
- Create a new value template instance from the template definition

The `TemplateInstanceExpression` class provides methods that support the following:
- Retrieve the parameters for the template instance
- Retrieve the template (`TreeActionValueTemplate` in the case of a tree action in a decision table) which was used to define the instance

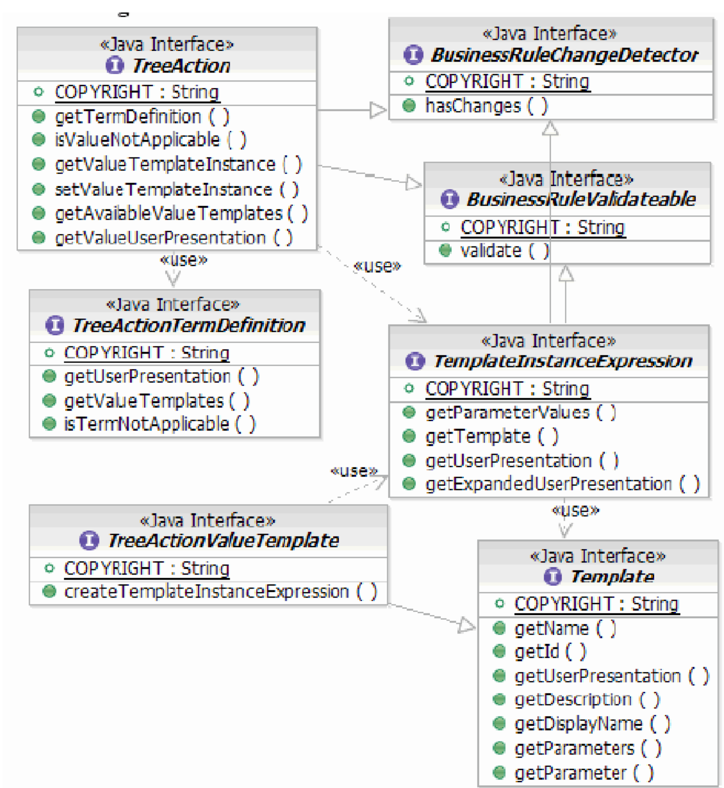


Figure 17. Class diagram of TreeAction and related classes

The definition of an init rule for a decision table follows the same structure as a rule in a rule set. The init rule can be defined with a template (`DecisionTableRuleTemplate`).

If an init rule was not created at authoring time, it can not be added once the rule is deployed.

The `Rule` class provides methods that support the following:
- Retrieve the name of the rule
- Retrieve the user presentation for the rule
- Retrieve the user presentation for the rule with the different parameters for the rule filled in

The DecisionTableRule class provides a method that supports the following:

- Retrieve the tree block containing the init rule

The DecisionTableRuleTemplate class provides a method that supports the following:

- Retrieve the decision table containing the template

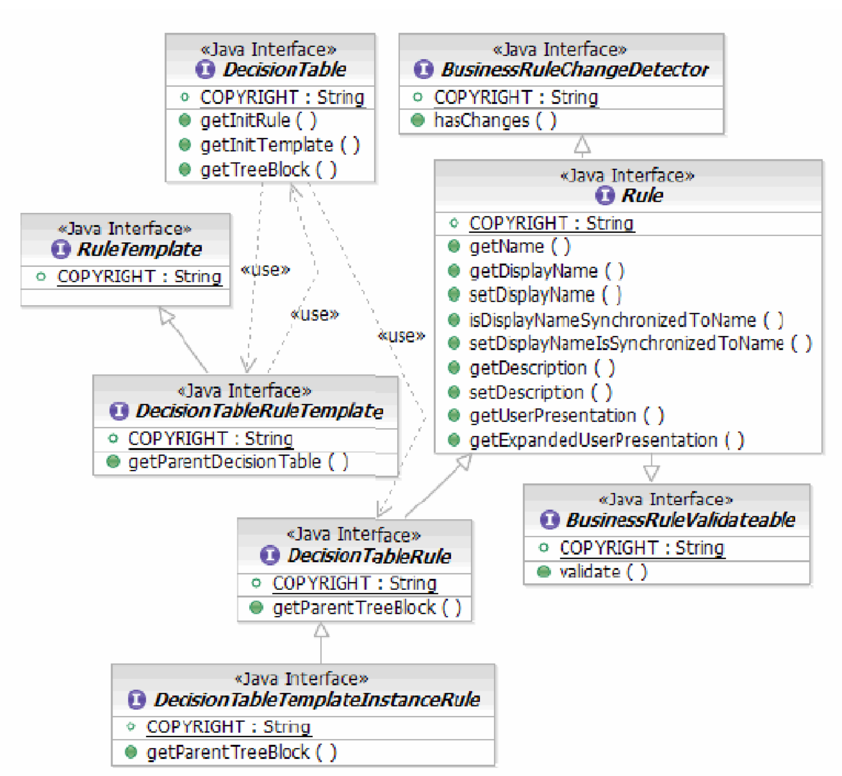


Figure 18. Class diagram for DecisionTableRule and related classes

Templates and Parameters

Templates in rule sets and decision tables are based off of a common definition. Templates have parameters and a user presentation. The template parameter values are defined to allow for changes to be made to the rule once it has been deployed.

The user presentation value provides a string value which can be used for displaying the rule and different parameters in a user-friendly manner. The user presentation, which is a string, has placeholders to allow for the different parameter values to be replaced and displayed correctly. The placeholders are in the format {<parameter index>}. For example, if the presentation string for the init rule is "Base discount is {0} %", the placeholder {0} could be substituted with the parameter value. The presentation string cannot be changed for the rule or the template definition. The placeholder values however, can be modified with the parameter values in a client application per the definition of the template. The different templates include a convenience method (getExpandedUserPresentation) that returns a string which has all of the parameter values correctly placed in the string.

All parameter values have a specific data type, however when retrieving and setting a parameter value, a string object is used. The parameter value can be

treated as string when substituting the value into the user presentation and also when setting the parameter with a new value. The parameter is converted to the correct data type at runtime in order to correctly issue the rule at execution time. During validation, the parameter value will be compared to the data type to insure it is correct. For example, if a parameter is of type `boolean` and is set to “T”, validation will not recognize this value and will return a problem.

In the template definition, the parameter values can be restricted by constraints. The constraints can be defined as a range or an enumeration. The constraints for the parameter will be enforced when the rule is validated. If a template was not used to define the value definition, only a user presentation will be available. A value definition can not have both a template and a user presentation. Should a template be used, the presentation from the template definition is the only presentation which is available.

The `Template` class provides methods that support the following:
- Retrieve the template ID
- Retrieve the name
- Retrieve the parameters
- Retrieve the user presentation

The `Parameter` class provides methods that support the following:
- Retrieve the parameter name
- Retrieve the parameter data type
- Retrieve the constraint for the parameter
- Retrieve the template defining the parameter
- Create a parameter value

The `ParameterValue` class provides methods that support the following:
- Retrieve the parameter name
- Retrieve the parameter value
- Set the parameter value

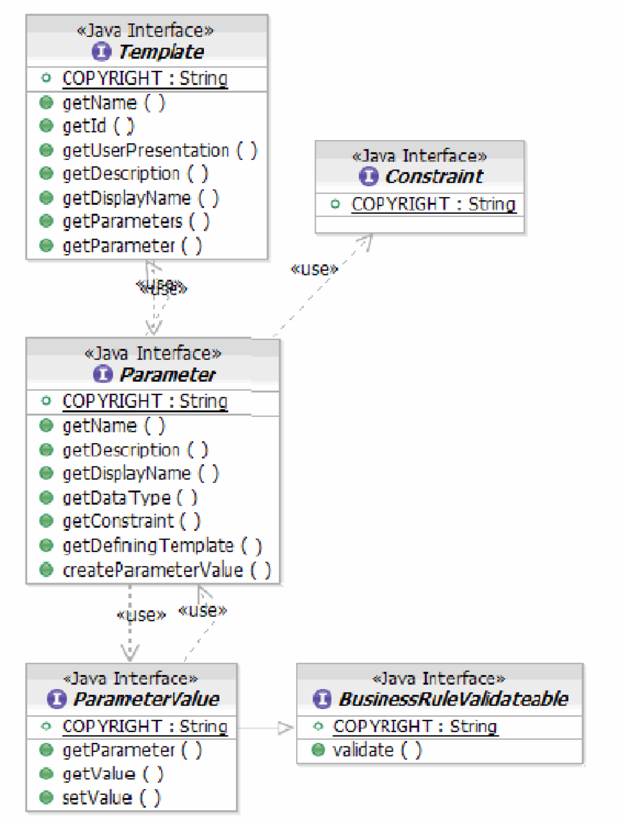


Figure 19. Class diagram for Template and Parameter and related classes

Validation

Many of the main objects have a validate method which allows for the artifacts to be checked for correctness and completeness prior to publishing the artifacts.

The validation that occurs when making changes through the API classes is only a proper subset of the overall validation that occurs during serviceDeploy or when editing the artifacts in WebSphere Integration Developer. This is due to the constraints that are already placed on the business rule group in limiting which aspects are editable at runtime. The user of the classes can validate the business rule group selection table, rule set or decision table whenever it is needed (the rule group component itself is not editable at runtime). When a business rule group is published the rule group selection table, rule sets and decision tables will be validated before being published to the repository.

If the artifacts are invalid, a ValidationException will be thrown with a list of the validation problems. The different validation problems are documented in the Exception Handling section.

Tracking Changes

For all objects, a `hasChanges` method is available to check if there have been any modifications which have occurred to the object and any containing objects.

This method can be used to check for changes and only publish a business rule group if it has items which changed.

BusinessRuleManager

The `BusinessRuleManager` class is the main class for working with the business rule groups, rule sets and decision tables.

The `BusinessRuleManager` has methods which allow for retrieving business rule groups by name, target namespace, or custom properties. It also has a method for publishing changes which have been made to business rule groups, rule sets, or decision tables.

The `BusinessRuleManager` class provides methods that support the following:

- Retrieve all of the business rule groups
- Retrieve business rule groups of a specific target namespace
- Retrieve business rule groups of a specific name
- Retrieve business rule groups of a specific name and target namespace
- Retrieve business rule groups which contain a specific property
- Retrieve business rule groups which contain specific properties
- Publish business rule groups

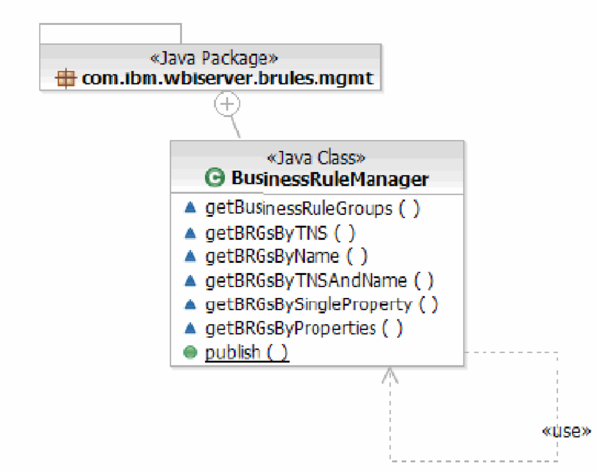


Figure 20. Class diagram for BusinessRuleManager and package

Rule Group Component Query

The rule group component can have user defined properties (name/value pairs) that can be used to narrow the list of business rule groups returned from the class. The fields that can be used in the query and in any combination are as follows:

- Business rule group component target name space
- Business rule group component name
- Property name
- Property value

Each property name can only be defined once per business rule group component.

The query function supported by this class is a small subset of the full SQL language. The user does not provide the SQL statement, but rather provides the values as parameters for a single property or a tree structure containing the information for a multi-property query in the form of nodes. There are logical

operator nodes and property query nodes which all implement the `QueryNode` interface. The logical operator nodes specify the boolean operators (AND, OR, NOT). These are created through the `QueryNodeFactory`. As part of the creation of these logical operator nodes, the left and right hand sides of the operator must be specified with additional `QueryNode` classes. These nodes can either be a property query node or another logical operator node. If a property query node is passed, it will contain the property name, value and operator (EQUAL (==), NOT_EQUAL (!=), LIKE, or NOTLIKE). The overall `QueryNode` is parsed by the class and a query is performed on the underlying data in persistent storage.

Wildcard searches are supported when the LIKE and NOTLIKE operators are used. Both the '%' and '_' characters are supported in wildcard searches. The '%' character is used when there are an infinite number of characters which are not known or should not be considered when searching. For example if a search was to be performed for all business rule groups that have a property with a name of Department and value that begins with "North", the value would be specified as "North%". Another example, suppose that all Departments with a value ending in "Region" was desired. The value would be "%Region". The '%' character can also be used in the middle of a string. For example, if there were business rule groups with properties that had values of "NorthCentralRegion", "NorthEastRegion", and "NorthWestRegion", a value of "North%Region" could be specified.

The '_' character is used when there is a single character which is unknown or should not be considered when searching. For example, if a search for all business rule groups with Department properties with values of "Dept1North", "Dept2North", "Dept3North", and "Dept4North" was desired, a value of "Dept_North" could be specified and all 4 of the business rule groups with these properties will be returned. The '_' character can be used multiple times in a search value with each instance indicating a single character to ignore. The '_' character can be used at the beginning or end of a value. For example if two characters were to be ignored in a value, two '_' could be used such as "Dept__outh".

In order to treat '%' and '_' as literal characters and not wildcards a '\' escape character must be specified preceding the '%' or '_'. For example if the property name was "%Discount", in order to use this in a query, "\%Discount" would need to be specified. If the '\' character is to be used as a literal character, another '\' escape character must be used such as "Orders\\Customer". If a single '\' character is found without a following '%', '_', or '\', an IllegalArgumentException will be thrown.

Wildcard characters can only be used on the left-hand side (property value). Wildcard characters can not be used in property name.

During searches on the value of a specific property or a search for values which do not match a property, the absence of a property causes the artifact to be ignored from consideration in the search. For example, if there are 3 business rule groups (A, B, and C) and only two (A and B) have a property named "Department" with different values ("Accounting" and "Shipping" respectively) a search for all business rule groups which do not have a "Department" property of "Accounting" will only return the business rule group which has the "Department" property defined but does not equal "Accounting" (business rule group B). The business rule group (C) which does not have the "Department" property, will not be returned as it does not have the property defined in any way.

When using properties for searching, there are two special properties named *IBMSystemName* and *IBMSystemTargetNameSpace* which can be used for searching based on the name and namespace of an artifact. These values can also be retrieved with the `getName` and `getTargetNameSpace` methods.

The class supports the following methods for query:

```
List getBRGsByTNS (string tNSName, Operator op, int skip, int threshold)
List getBRGByName(string Name, Operator op, int skip, int threshold)
List getBRGsByTNSAndName (string tNSName, Operator, tNSOp, string
 name, Operator nameOp, int skip, int threshold)
List getBRGsBySingleProperty (string propertyName, string propertyValue,
 Operator op, int skip, int threshold)
List getBRGsByProperties (QueryNode queryTree, int skip, int threshold)
```

The 'skip' and 'threshold' parameters provide the user the capability of fetching a partial result list up to the specified threshold. A value of zero for both of these parameters will return the full result list. The cursor is not maintained in the result set from a query call. If a skip value is used, it is possible that additions or deletions could have been made to the result set such that a subsequent request will return business rule groups which were in an earlier result set.

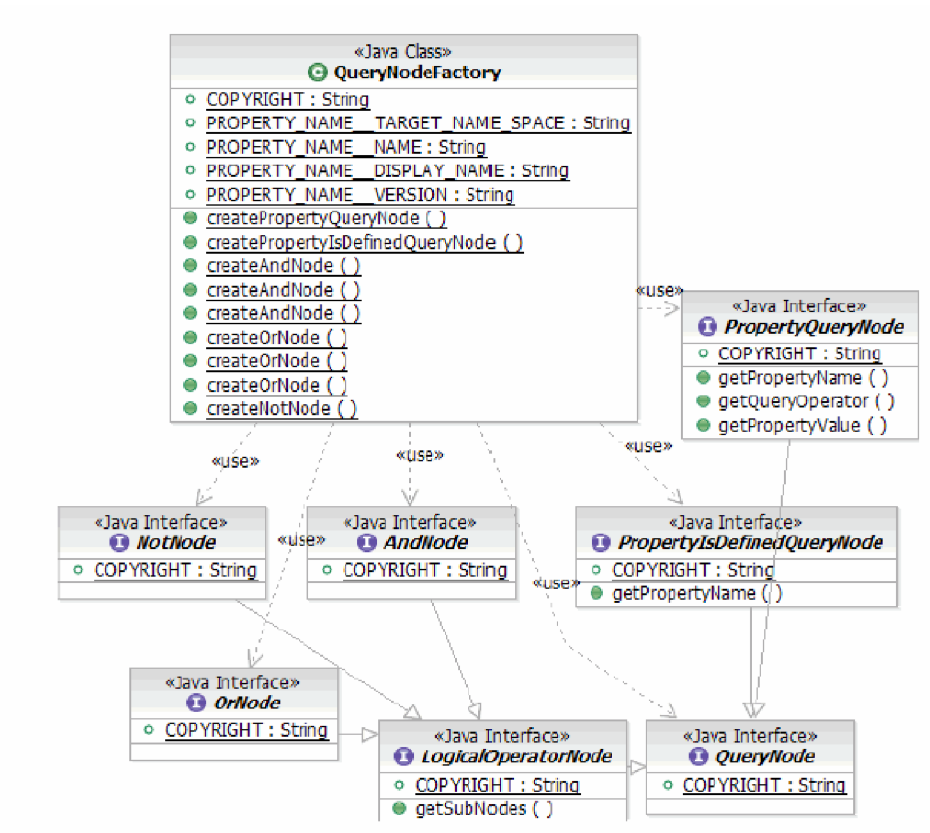


Figure 21. Class diagram for QueryNodeFactory and related classes

The nodes in the tree allow the user to specify a search expression using the boolean operators, wild cards (% and escape) and the property/value pair. The operator is only valid for the values, the operator for the property is always equals (==).

Publishing

Publishing of business rule changes is done at the business rule group component level. The user can publish 1...n business rule group components. Before a publish operation is performed, a validate action is performed on the business rule group and the different objects contained in the business rule group (operation selection table, rule sets, decision table, and so on). Each publish request will occur within a single transaction and if any exceptions are encountered during validation or database publishing the transaction is rolled back and no changes for any business rule group are published to the repository. This allows changes that are dependent on each other within a single component (for example, operation selection table and a rule set) or dependencies between components to occur within one atomic operation.

At publishing time, a check will be made to insure that the items which are to be published have not been changed by another transaction. To reduce the possibilities of a conflict, the publish method will give the user the ability to choose to publish all artifacts whether they are changed or not or only those artifacts that were changed within the business rule group. The default behavior will be to publish all artifacts. If the option is set to publish all artifacts and another transaction had changed the artifacts in the meantime, a `ChangeConflictException` will be thrown. Specifying to only publish those artifacts which have changed will reduce the chance of conflict. Publishing only those artifacts that were changed could result in two users pushing changes to the repository for two different artifacts within a business rule group (for example, two rule sets) which could introduce incompatible changes within the business rule group. Because this potential situation, this option should be used with caution.

Exception Handling

Exceptions can occur when validation is called on an artifact or when an artifact is published. When a validation error occurs, the `ValidationException` is thrown with a list of problems. If there is a problem during publishing due to another transaction publishing the same artifacts, a `ChangeConflictException` is thrown. Anytime another transaction is detected as changing an artifact, the `ChangeConflictException` exception is thrown.

There is a `SystemPropertyNotChangeableException` which is thrown if a property which duplicates a system property name is attempted to be changed. System properties cannot be changed.

There is a `ChangesNotAllowedException` which is thrown if a set operation is attempted on an artifact as it is being published.

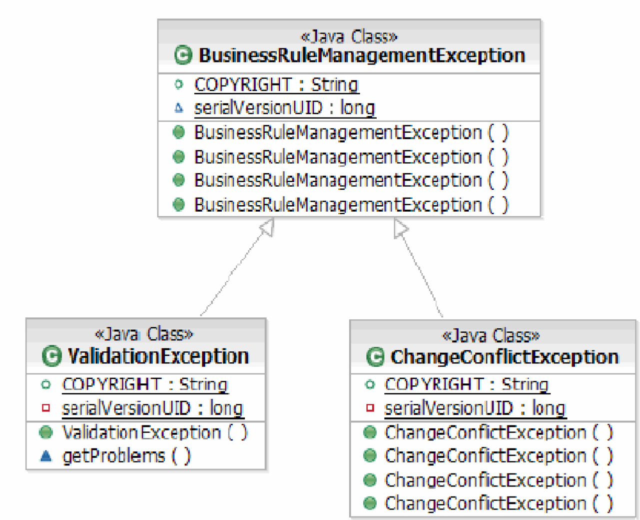


Figure 22. Class diagram for BusinessRuleManagementException and related classes

Business Rule Group problems

Problems which can occur when a business rule group is validated or an attempt to publish the business rule group is made and a portion of the business rule group is not valid.

Table 4. Business Rule Group problems

Exception	Description
ProblemBusRuleNotInAvailTargetList	Problem that occurs when a rule is specified as the default business rule for an operation selection table but the rule artifact is not in the list of available targets for that operation. A valid business rule from the list of available targets on the operation should be specified to avoid this problem.
ProblemDuplicatePropertyName	This problem occurs when a property is attempted to be created which is a duplicate of a system or user-defined property on a business rule group. A unique property name should be used to avoid this problem.
ProblemOperationContainsNoTargets	Problem that occurs when an operation does not have a default rule destination or any scheduled rule destinations set. The operation should be set with at least one rule destination as either the default or at a scheduled time to avoid this problem.
ProblemOverlappingRanges	Problem that occurs when an operation selection record has a start date or end date which overlaps with the range of another operation selection record start date and end date. This overlap in date ranges prevents the business rule runtime from finding the correct rule destination to invoke. The start date or end date of the other operation selection records for an operation should be checked to insure there is not an overlap to avoid this problem.
ProblemStartDateAfterEndDate	This problem occurs when the start date for an operation selection record is after the end date for that selection record. This problem can occur for any operation selection record except the default record which does not have a start date or end date. Specify a start date after the end date on an operation selection record to avoid this problem.

Table 4. Business Rule Group problems (continued)

Exception	Description
ProblemTargetBusRuleNotSet	Problem that occurs when an operation selection record has a rule specified which is not in the list of available target rules. A rule from the available target list should be specified to avoid this problem.
ProblemTNSAndNameAlreadyInUse	Problem that occurs when a new business rule is created with a target name space and name which is already in use by a rule set or decision table. A check is done on all rule sets and decision tables associated with the current business rule group as well as any rule artifact stored in the repository. A different target name space or name should be used to avoid this problem.
ProblemWrongOperationForOpSelectionRecord	Problem that occurs when a new operation selection record is added to an operation selection record list and the operation of the new record does not match the operation of the records in the list. A new operation should be created using the `newOperationSelectionRecord` method on the correct operation selection record list object to avoid this problem.

Rule set and Decision Table problems

Table 5. Rule set and Decision Table problems

Exception	Description
ProblemInvalidBooleanValue	Problem that occurs when a parameter for a rule template in a rule set or an action value or condition value in decision table receives a different value than “true” or “false” for a parameter of type Boolean. Examples of incorrect parameter values would be “T” or “F”. Use values of “true” or “false” when working with a parameter of type Boolean to avoid this problem.
ProblemParmNotDefinedInTemplate	Problem that occurs when a value is specified for a template parameter and the parameter is not defined in the list of valid parameters for the template. The parameters should be checked prior setting in the template. It can occur for RuleTemplate, TreeActionValueTemplate, or TreeConditionValueTemplate templates.
ProblemParmValueListContainsUnexpectedValue	Problem that occurs when valid parameters are passed with a template, however, there are too many parameters for the parameter. The number of parameters should be reduced. It can occur for RuleTemplate, TreeActionValueTemplate, or TreeConditionValueTemplate templates.
ProblemRuleBlockContainsNoRules	This problem occurs when all rules in a rule block in a rule set are removed and the rule set is attempted to be validated or published. The rule block in a rule set must have at least one rule.
ProblemTemplateNotAssociatedWithRuleSet	Problem that occurs when a rule is attempted to be added to a rule set and the rule was created with a template which is not defined with that rule set. When creating a new rule, a template which has been defined in the rule set should be used to avoid this problem.
ProblemRuleNameAlreadyInUse	Problem that occurs when a rule is attempted to be added to a rule block in a rule set and it has the same name as an existing rule in the rule block. The names of the rules should be checked prior to adding a new rule to avoid this problem.

Table 5. Rule set and Decision Table problems (continued)

Exception	Description
ProblemTemplateParameterNotSpecified	This problem occurs when a parameter is not included when a template is updated for a rule in a rule set or action or condition value in a decision table. All parameters for a template should be specified to avoid this problem.
ProblemTypeConversionError	This problem occurs when a parameter for a template cannot be converted to the appropriate type All parameters are treated as string objects and then converted to the parameter type (boolean, byte, short, int, long, float, and double). If the parameter value string cannot be converted to the specified type for this parameter, then this error occurs. To avoid this problem, a string that can be converted to the parameter's type (boolean, byte, short, int, long, float, and double) should be specified.
ProblemValueViolatesParmConstraints	This problem occurs when a parameter is not within the enumeration or range of values which have been defined within the template for that parameter. This problem can occur for parameters restricted with enumerations or ranges in rule templates in a rule set or action value or condition value templates in a decision table. A value which is within the enumeration should be used to avoid this problem.
ProblemInvalidActionValueTemplate	Problem that occurs when a template instance is attempted to be set on the value definition in a tree action but the corresponding template is not available to that tree action. Use the correct template to create a value definition in a tree action in order to avoid this problem.
ProblemInvalidConditionValueTemplate	Problem that occurs when a template instance is attempted to be set on the condition definition in a case edge but the corresponding template is not available to that case edge. Use the correct template to create a condition definition in a case edge in order to avoid this problem.
ProblemTreeActionIsNull	This problem occurs when a new condition value is created and an action is not set with a template instance. Using a template from the ActionNode, create a new template instance and set it in the list of `TreeActions`.

Authorization

The classes do not support any level of authorization. It is up to the client application using the classes to add its own form of authorization.

Examples

A number of examples are provided that show how the different classes can be used to retrieve business rule groups and to make modifications to rule sets and decision tables. The examples are provided in a project interchange file (ZIP) that can be imported into WebSphere Integration Developer where they can be browsed and reused.

The project interchange contains a number of projects.

- **BRMgmtExamples –** Module project with business rules artifacts that are used in the various examples.
- **BRMgmt –** Java project with the examples located in the com.ibm.websphere.sample.brules.mgmt package.
- **BRMgmtDriverWeb –** Web project with interface for executing the samples.

The examples are also provided as an EAR file (BRMgmtExamples.ear) that can be issued once after installed into WebSphere Process Server. A Web interface is provided with the examples. The Web interface is purposely simple as the examples focus on using the classes to retrieve artifacts, make modifications, and publish changes. It is not meant to be a high-functioning Web interface. The classes can however, be easily used to build robust Web interfaces or used in other Java applications focused on modifying the business rules.

Note: You can download the example project interchange and EAR file from Business Rule Management Programming Guide for WebSphere Process Server V6.1.

The example application can be installed on WebSphere Process Server v6.1 and the index page can be accessed at:

http://<hostname>:<port>/BRMgmtDriverWeb/

For example, http://localhost:9080/BRMgmtDriverWeb/

As the examples are issued, changes will be made to the rule artifacts. If all examples are issued, the application will need to be reinstalled to see the same results for all examples again.

The examples are explained in detail with complete code samples as well as the result as displayed in a Web browser.

A number of additional classes were created in order to perform common operations and assist with displaying the information within the example Web application. See the appendix for more information on the `Formatter` and `RuleArtifactUtility` classes.

To fully understand these examples, a study of the different artifacts within WebSphere Integration Developer will greatly help.

Example 1: Retrieve and print all business rule groups

This example will retrieve all business rule groups and print out the attributes, the properties, and the operations for each business rule group.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;
```

For the business rules management classes, be sure to use those classes in the com.ibm.wbiserver.brules.mgmt package and not the com.ibm.wbiserver.brules package or other package. These other packages are for IBM internal classes.

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import
com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;

public class Example1 {
static Formatter out = new Formatter();
static public String executeExample1()
```

```
{
try
{
out.clear();
```

The BusinessRuleManager class is the main class to retrieve business rule groups and to publish changes to business rule groups. This includes working with and changing any rule artifacts such as rule sets and decision tables. There are a number of methods on the BusinessRuleManager class that simplify the retrieval of specific business rule groups by name and namespace and properties.

```
// Retrieve all business rule groups
List<BusinessRuleGroup> brgList = BusinessRuleManager
  .getBusinessRuleGroups(0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Iterate through the list of business rule groups
while (iterator.hasNext())
{
  brg = iterator.next();
  // Output attributes for each business rule group
  out.printlnBold("Business Rule Group");
```

The basic attributes of the business rule group can be retrieved and displayed.

```
out.println("Name: " + brg.getName());
out.println("Namespace: " +
brg.getTargetNameSpace());
out.println("Display Name: " +
brg.getDisplayName());
out.println("Description: " + brg.getDescription());
out.println("Presentation Time zone: "
     + brg.getPresentationTimezone());
out.println("Save Date: " + brg.getSaveDate());
```

The properties for the business rule group can also be retrieved and modified.

```
PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Output property names and values
while (propIterator.hasNext())
{
prop = propIterator.next();
out.println("Property Name: " +
prop.getName());
out.println("Property Value: " +
prop.getValue());
}
```

The operations for the business rule group are also available and are the way to retrieve the business rule artifacts such as rule sets and decision tables.

```
List<Operation> opList = brg.getOperations();

Iteration<Operation> opIterator = opList.iterator();
Operation op = null;
// Output operations for the business rule group
while (opIterator.hasNext())
{
op = opIterator.next();
out.println("Operation: " + op.getName());
}
```

```
out.println("");}
} catch (BusinessRuleManagementException e)
{
e.printStackTrace();
out.println(e.getMessage());
}
return out.toString();
}
}
```

Web browser output for example 1.

```
Executing example1

Business Rule Group
Name: ApprovalValues
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: ApprovalValues
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: Department
Property Value: Accounting
Property Name: RuleType
Property Value: regulatory
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: ApprovalValues
Property Name: IBMSystemDisplayName
Property Value: ApprovalValues
Operation: getApprover

Business Rule Group
Name: ConfigurationValues
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: ConfigurationValues
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: Department
Property Value: General
Property Name: RuleType
Property Value: messages
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: ConfigurationValues
Property Name: IBMSystemDisplayName
Property Value: ConfigurationValues
Operation: getMessages

Business Rule Group
Name: DiscountRules
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: DiscountRules
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: Department
Property Value: Accounting
Property Name: IBMSystemVersion
Property Value: 6.2.0
```

```
Property Name: RuleType
Property Value: monetary
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: DiscountRules
Property Name: IBMSystemDisplayName
Property Value: DiscountRules
Operation: calculateOrderDiscount
Operation: calculateShippingDiscount
```

Example 2: Retrieve and print business rule groups, rule sets and decision tables

Besides the function in example 1, this example will print out the selection table for each operation and then the default business rule destination (either rule set or decision table) and the other business rules scheduled for the operation. It prints out both rule sets and decision tables.

The majority of the example is the same, but provided for completeness.

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
public class Example2
{
 status Formatter out = new Formatter();
 static public String executeExample2()
 {
  try
  {
    out.clear();
```

A specific business rule group is retrieved by name for this example.

```
  // Retrieve all business rule groups
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByName("DiscountRules",
    QueryOperator.EQUAL, 0, 0);

  Iterator<BusinessRuleGroup> iterator = brgList.iterator();

  BusinessRuleGroup brg = null;
  // Iterate through the list of business rule groups
  while (iterator.hasNext())
   {
   brg = iterator.next();
   // Output attributes for each business rule group
   out.printlnBold("Business Rule Group");
   out.println("Name: " + brg.getName());
   out.println("Namespace: " +
    brg.getTargetNameSpace());
   out.println("Display Name: " +
    brg.getDisplayName());
   out.println("Description: " + brg.getDescription());
   out.println("Presentation Time zone: "
```

```
 + brg.getPresentationTimezone());
out.println("Save Date: " + brg.getSaveDate());

PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Output property names and values
while (propIterator.hasNext())
{
  prop = propIterator.next();
  out.println("Property Name: " +
  prop.getName());
  out.println("Property Value: " +
  prop.getValue());
}
```

For each operation, a selection table has a list of the different rule artifacts and the schedule on when they are active. A default business rule can be specified for each operation. There is no requirement that a default business rule be specified or that there is a scheduled business rule, however there must be at least a default business rule or one scheduled business rule. Because of this support, it is best to check for `null` before using the default business rule or check the size of the `OperationSelectionRecordList`.

```
List<Operation> opList = brg.getOperations();

Iterator<Operation> opIterator = opList.iterator();
Operation op = null;
out.println("");
out.printlnBold("Operations");
// Output operations for the business rule group
while (opIterator.hasNext())
 {
   op = opIterator.next();
   out.printBold("Operation: ");
   out.println(op.getName());

   // Retrieve the default business rule for the operation
   BusinessRule defaultRule =
   op.getDefaultBusinessRule();
   // If the default rule is found, print out the business rule
   // using the appropriate method for rule type
   if (defaultRule != null)
   {
     out.printlnBold("Default Destination:");
```

The default business rule is of type `RuleSet` or `DecisionTable` and can be cast to the correct type in order to process the rule artifact.

```
     if (defaultRule instanceof RuleSet)
        out.println(RuleArtifactUtility.
        intRuleSet(defaultRule));
     else
        out.print(RuleArtifactUtility.
        tDecisionTable(defaultRule));
   }
   OperationSelectionRecordList
   opSelectionRecordList = op
     .getOperationSelectionRecordList()
     ;

   Iterator<OperationSelectionRecord>
```

```
      opSelRecordIterator = opSelectionRecordList
        .iterator();
      OperationSelectionRecord record = null;
```

The OperationSelectionRecord is composed of the rule artifact and the schedule on when the rule artifact is active.

```
      while (opSelRecordIterator.hasNext())
      {
        out.printlnBold("Scheduled
        Destination:");
        record = opSelRecordIterator.next();

        out.println("Start Date: " +
        record.getStartDate()
          + " - End Date: " +
          record.getEndDate());
        BusinessRule ruleArtifact = record
          .getBusinessRuleTarget();

        if (ruleArtifact instanceof RuleSet)
          out.println(RuleArtifactUtility.pr
          intRuleSet(ruleArtifact));
        else
          out.print(RuleArtifactUtility.prin
          tDecisionTable(ruleArtifact));
       }
    }
  }
  out.println("");
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 return out.toString();
 }
}
```

Example

Web browser output for example 2.

```
Business Rule Group
Name: DiscountRules
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: DiscountRules
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: Department
Property Value: Accounting
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: RuleType
Property Value: monetary
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: DiscountRules
Property Name: IBMSystemDisplayName
Property Value: DiscountRules

Operations
Operation: calculateOrderDiscount
Default Destination:
Rule Set
Name: calculateOrderDiscount
```

```
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Rule: CopyOrder
Display Name: CopyOrder
Description: null
Expanded User Presentation: null
User Presentation: null
Rule: FreeGiftInitialization
Display Name: FreeGiftInitialization
Description: null
Expanded User Presentation: Product ID for Free Gift = 5001AE80 Quantity = 1 Cost =
0.0 Description = Free gift for discounted order
User Presentation: Product ID for Free Gift = {0} Quantity = {1} Cost = {2}
Description = {3}Parameter Name: param0
Parameter Value: 5001AE80
Parameter Name: param1
Parameter Value: 1
Parameter Name: param2
Parameter Value: 0.0
Parameter Name: param3
Parameter Value: Free gift for discounted order
Rule: Rule1
Display Name: Rule1
Description: null
Expanded User Presentation: If customer is gold status, then apply a discount of 20.0
and include a free gift
User Presentation: If customer is {0} status, then apply a discount of {1} and include a
free gift
Parameter Name: param0
Parameter Value: gold
Parameter Name: param1
Parameter Value: 20.0
Rule: Rule2
Display Name: Rule2
Description: null
Expanded User Presentation: If customer.status == silver, then provide a discount of
15.0
User Presentation: If customer.status == {0}, then provide a discount of {1}
Parameter Name: param0
Parameter Value: silver
Parameter Name: param1
Parameter Value: 15.0
Rule: Rule3
Display Name: Rule3
Description: Template for non-gold customers
Expanded User Presentation: If customer.status == bronze, then provide a discount of
10.0
User Presentation: If customer.status == {0}, then provide a discount of {1}
Parameter Name: param0
Parameter Value: bronze
Parameter Name: param1
Parameter Value: 10.0

Operation: calculateShippingDiscount
Default Destination:
Decision Table
Name: calculateShippingDiscount
Namespace: http://BRSamples/com/ibm/websphere/sample/brules

Init Rule: Rule1
Display Name: Rule1
Description: null
Extended User Presentation: null
User Presentation: null
```

Example 3: Retrieve business rule groups by multiple properties with AND

This example is also similar to example 1, but will only retrieve those business rule groups which have a property named Department and a value of "accounting" and a property named RuleType and a value of "regulatory".

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example3
{
static Formatter out = new Formatter();
static public String executeExample3()
{
  try
  {
    out.clear();
```

Queries for business rule groups are composed of query nodes that follow a tree structure. Each query node has a left-hand term and a right-hand term and condition. Each lefthand term and right-hand term can be another query node. For this example the business rule group is retrieved by the combination of two property values.

```
   // Retrieve business rule groups based on two conditions
   // Create PropertyQueryNodes for each one condition
   PropertyQueryNode propertyNode1 = QueryNodeFactory
     .createPropertyQueryNode("Department",
     QueryOperator.EQUAL,"Accounting");
   PropertyQueryNode propertyNode2 = QueryNodeFactory
     .createPropertyQueryNode("RuleType", QueryOperator.EQUAL,
     "regulatory");
   // Combine the two PropertyQueryNodes with an AND node
   AndNode andNode =
   QueryNodeFactory.createAndNode(propertyNode1, propertyNode2);

   // Use andNode in search for business rule groups
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByProperties(andNode, 0, 0);

   Iterator<BusinessRuleGroup> iterator = brgList.iterator();

   BusinessRuleGroup brg = null;
   // Iterate through the list of business rule groups
   while (iterator.hasNext())
   {
     brg = iterator.next();
     // Output attributes for each business rule group
     out.printlnBold("Business Rule Group");
     out.println("Name: " + brg.getName());
     out.println("Namespace: " +
     brg.getTargetNameSpace());
     out.println("Display Name: " + brg.getDisplayName());
     out.println("Description: " + brg.getDescription());
```

```
    out.println("Presentation Time zone: "
    + brg.getPresentationTimezone());
    out.println("Save Date: " + brg.getSaveDate());

    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
    propList.iterator();
    Property prop = null;
    // Output property names and values
    while (propIterator.hasNext())
    {
      prop = propIterator.next();
      out.println("\t Property Name: " +
      prop.getName());
      out.println("\t Property Value: " +
      prop.getValue());
    }
  }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Example

Web browser output for example 3.

```
Executing example3

Business Rule Group
Name: ApprovalValues
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: ApprovalValues
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: Department
Property Value: Accounting
Property Name: RuleType
Property Value: regulatory
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: ApprovalValues
Property Name: IBMSystemDisplayName
Property Value: ApprovalValues
```

Example 4: Retrieve business rule groups by multiple properties with OR

This example is similar to example 3; however it will only retrieve those business rule groups which have a property named Department and a value of “accounting” or a property named RuleType and a value of “monetary”.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
```

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example4
{
static Formatter out = new Formatter();
static public String executeExample4()
{
  try
  {
    out.clear();
```

Different properties make up the query and return different business rule groups.

```
    // Retrieve business rule groups based on two conditions
    // Create PropertyQueryNodes for each one condition
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"Accounting");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
      QueryOperator.EQUAL,"monetary");
    // Combine the two PropertyQueryNodes with an OR node
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1,
      propertyNode2);
    // Use orNode in search for business rule groups
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(orNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Iterate through the list of business rule groups
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Output attributes for each business rule group
      out.printlnBold("Business Rule Group");
      out.println("Name: " + brg.getName());
      out.println("Namespace: " +
      brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());

      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Output property names and values
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
        out.println("\t Property Name: " +
        prop.getName());
        out.println("\t Property Value: " +
        prop.getValue());
```

```
      }
      out.println("");
    }
  } catch (BusinessRuleManagementException e)
 {
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Example

Web browser output for example 4.

```
Executing example4

Business Rule Group
Name: ApprovalValues
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: ApprovalValues
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: Department
Property Value: Accounting
Property Name: RuleType
Property Value: regulatory
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: ApprovalValues
Property Name: IBMSystemDisplayName
Property Value: ApprovalValues

Business Rule Group
Name: DiscountRules
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: DiscountRules
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: Department
Property Value: Accounting
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: RuleType
Property Value: monetary
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: DiscountRules
Property Name: IBMSystemDisplayName
Property Value: DiscountRules
```

Example 5: Retrieve business rule groups with a complex query

This example is a combination of examples 3 and 4 and it is meant to show how more complex queries can be created. In this example a search is performed with a query that combines 2 query conditions. The first query condition is to retrieve those business rule groups which have a property named Department and a value of “General” or a property named MissingProperty and a value of “somevalue”.

This query condition is then combined with an AND to a condition where the property is named RuleType and a value of “messages”.

More examples of business rule group queries are available in the appendix.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example5
{
static Formatter out = new Formatter();
static public String executeExample5()
{
  try
  {
    out.clear();

    // Retrieve business rule groups based on three conditions where
    // two of the conditions are combined in an OR node
    // Create PropertyQueryNodes for each condition for the OR node
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"General");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("MissingProperty",
    QueryOperator.EQUAL, "SomeValue");
    // Combine the two PropertyQueryNodes with an OR node
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1, propertyNode2);
    // Create the third PropertyQueryNode
      PropertyQueryNode propertyNode3 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
    QueryOperator.EQUAL,"messages");
```

The left-hand side of the condition is combined to the right-hand with an AND node. The `AndNode` is the root of the query tree.

```
    // Combine OR node with third PropertyQueryNode with
    AndNode AndNode andNode =
    QueryNodeFactory.createAndNode(propertyNode3, orNode);

    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(andNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Iterate through the list of business rule groups
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Output attributes for each business rule group
      out.printlnBold("Business Rule Group");
      out.println("Name: " + brg.getName());
```

```
      out.println("Namespace: " +
        brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());
      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Output property names and values
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
        out.println("\t Property Name: " +
        prop.getName());
        out.println("\t Property Value: " +
        prop.getValue());
      }
    }
   }   catch (BusinessRuleManagementException e)
  {
  e.printStackTrace();
  out.println(e.getMessage());
  }
 return out.toString();
 }
}
```

Example

Web browser output for example 5.

```
Executing example5

Business Rule Group
Name: ConfigurationValues
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Display Name: ConfigurationValues
Description: null
Presentation Time zone: LOCAL
Save Date: Sun Jan 06 17:56:51 CST 2008
Property Name: IBMSystemVersion
Property Value: 6.2.0
Property Name: Department
Property Value: General
Property Name: RuleType
Property Value: messages
Property Name: IBMSystemTargetNameSpace
Property Value: http://BRSamples/com/ibm/websphere/sample/brules
Property Name: IBMSystemName
Property Value: ConfigurationValues
Property Name: IBMSystemDisplayName
Property Value: ConfigurationValues
```

Example 6: Update a business rule group property and publish

In this example, a property in a business rule group is updated and then the business rule group is published.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
```

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example6
{
static Formatter out = new Formatter();

static public String executeExample6()
{
 try
 {
   out.clear();
   out.printlnBold("Business Rule Group before publish:");
   // Retrieve business rule groups by a single property value
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsBySingleProperty("Department",
     QueryOperator.EQUAL,"General", 0, 0);

   if (brgList.size() > 0)
   {
     // Get the first business rule group from the list
     BusinessRuleGroup brg = brgList.get(0);
     // Retrieve the property from the business rule group
     UserDefinedProperty userDefinedProperty =
     (UserDefinedProperty) brg
       .getProperty("Department");

     out.println("Business Rule Group: " + brg.getName());
     out.println("Department Property value: "
     + brg.getProperty("Department").getValue());
```

The `getProperty` method returns a property by reference and changes made to the property are directly made to the business rule group.

```
     // Modify the property value in the brg
     // This updates the property value directly in the
     brg object
     userDefinedProperty.setValue("GeneralConfig");
     // Use the original list or create a new list
     // of business rule groups
     List<BusinessRuleGroup> publishList = new
     ArrayList<BusinessRuleGroup>();
     // Add the changed business rule group to the list
     publishList.add(brg);
```

The `BusinessRuleManager` class is used to publish the changes made to a business rule group. To publish the change, a list is passed to the `BusinessRuleManager` publish method even if there is only one item is being published.

```
     // Publish the list with the updated business rule group
     BusinessRuleManager.publish(publishList, true);

     out.println("");

     // Retrieve the business rule group again to verify the
     // changes were published
     out.printlnBold("Business Rule Group after publish:");
     brgList = BusinessRuleManager
     .getBRGsBySingleProperty("Department",
     QueryOperator.EQUAL, "GeneralConfig", 0, 0);

     brg = brgList.get(0);

     out.println("Business Rule Group: " + brg.getName());
     // Display the property value to show the change
```

```
     out.println("Department Property value: "
     + brg.getProperty("Department").getValue());
   }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
return out.toString();
}
}
```

Example

Web browser output for example 6.

```
Executing example6

Business Rule Group before publish:
Business Rule Group: ConfigurationValues
Department Property value: General

Business Rule Group after publish:
Business Rule Group: ConfigurationValues
Department Property value: GeneralConfig
```

Example 7: Update properties in multiple business rule groups and publish

In this example, properties in multiple business rule groups are updated before publish.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example7
{
static Formatter out = new Formatter();

static public String executeExample7()
{
  try
  {
    out.clear();
    out.printlnBold("Business Rule Group before publish:");
    // Retrieve business rule groups by a single property value
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsBySingleProperty("Department",
      QueryOperator.EQUAL, "Accounting", 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;

    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();
```

```
    // Iterate through all of the business rule groups and
    // modify the property
    while (iterator.hasNext())
    {
      // Retrieve the property from the business rule group
      brg = iterator.next();

      out.println("Business Rule Group: " + brg.getName());

      // Retrieve the property from the business rule group
      UserDefinedProperty prop = (UserDefinedProperty) brg
        .getProperty("Department");
      out.println("Department Property value: "
      +
      brg.getProperty("Department").getValue())
      ;

      // Modify the property value in the brg
      // This updates the property value directly in the
      brg object
      prop.setValue("Finance");
```

Each changed business rule group is added to the list.

```
      // Add the changed business rule group to the list
      publishList.add(brg);
      }

      // Publish the list with the updated business rule
      group
      BusinessRuleManager.publish(publishList, true);

      out.println("");

      // Retrieve the business rule groups again to verify the
      // changes were published
      out.printlnBold("Business Rule Group after
      publish:");

      brgList = BusinessRuleManager
        .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL,
        "Finance", 0, 0);
      iterator = brgList.iterator();

      while (iterator.hasNext())
      {
        brg = iterator.next();
        out.println("Business Rule Group: " +
        brg.getName());
        out.println("Department Property value: "
         +
         brg.getProperty("Department").getVa
         lue());
      }
} catch (BusinessRuleManagementException e)
{
  e.printStackTrace();
  out.println(e.getMessage());
}
return out.toString();
}
}
```

Example

Web browser output for example 7.

```
Executing example7

Business Rule Group before publish:
Business Rule Group: ApprovalValues
Department Property value: Accounting
Business Rule Group: DiscountRules
Department Property value: Accounting

Business Rule Group after publish:
Business Rule Group: ApprovalValues
Department Property value: Finance
Business Rule Group: DiscountRules
Department Property value: Finance
```

Example 8: Change the default business rule for a business rule group

In this example, the default business rule is changed with another business rule that is part of the available targets list for a specific operation.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example8
{
static Formatter out = new Formatter();

static public String executeExample8()
{
  try
  {
    out.clear();

  // Retrieve a business rule group by target namespace and name
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

  if (brgList.size() > 0)
  {
    out.printlnBold("Business Rule Group before publish:");
    // Get the first business rule group from the list
    // This should be the only business rule group in the list as
    // the combination of target namespace and name are unique
    BusinessRuleGroup brg = brgList.get(0);

    out.print("Business Rule Group: ");
    out.println(brg.getName());

    // Get the operation of the business rule group that
    // will have its default business rule updated
    Operation op =
    brg.getOperation("calculateShippingDiscount");
```

The default business rule is retrieved before it is updated with another rule that is part of the available target list for the operation. Rule sets and decision tables are specific to operations and only those business rule artifacts that are for an operation can be set to be default or be scheduled for another time on the operation.

```
// Retrieve the default business rule for the operation
BusinessRule defaultRule =
op.getDefaultBusinessRule();
out.print("Default Rule: ");
out.println(defaultRule.getName());

// Get the list of available business rules for this
operation
List<BusinessRule> ruleList =
op.getAvailableTargets();

Iterator<BusinessRule> iterator =
ruleList.iterator();
BusinessRule rule = null;

// Find a business rule that is different from the
current
// default
// business rule
while (iterator.hasNext())
{
  rule = iterator.next();
  if
  (!defaultRule.getName().equals(rule.getName()))
  {
```

The default business rule is set on the operation object. Setting the default business rule to null will remove any default business rule from the operation, however it is recommended that every operation have a default business rule specified.

```
    // Set the default business rule to be a
    // different business rule
    // This change is to the operation object
    // directly
    op.setDefaultBusinessRule(rule);
    break;
  }
}
// Use the original list or create a new list
// of business rule groups
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();
// Add the changed business rule group to the list
publishList.add(brg);
// Publish the list with the updated business rule
group
BusinessRuleManager.publish(publishList, true);

out.println("");

// Retrieve the business rule groups again to verify the
// changes were published

out.printlnBold("Business Rule Group after publish:");
brgList = BusinessRuleManager
.getBRGsByTNSAndName(
  "http://BRSamples/com/ibm/websphere/sample/brules",
  QueryOperator.EQUAL, "DiscountRules",
  QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
```

```
    out.println("Business Rule Group: " + brg.getName());
    op = brg.getOperation("calculateShippingDiscount");

    // Retrieve the default business rule for the operation
    defaultRule = op.getDefaultBusinessRule();
    out.print("Default Rule: ");
    out.println(defaultRule.getName());
   }
 } catch (BusinessRuleManagementException e)
 {
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Example

Web browser output for example 8.

```
Executing example8

Business Rule Group before publish:
Business Rule Group: DiscountRules
Default Rule: calculateShippingDiscount

Business Rule Group after publish:
Business Rule Group: DiscountRules
Default Rule: calculateShippingDiscountHoliday
```

Example 9: Schedule another rule for an operation in a business rule group

In this example, a business rule is scheduled to be active for 1 hour from the time of publish for a specific operation.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example9 {
static Formatter out = new Formatter();

static public String executeExample9()
{
 try
 {
   out.clear();

   // Retrieve a business rule group by target namespace and name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
```

```
      QueryOperator.EQUAL,
      "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

    if (brgList.size() > 0)
    {
      out.println("");
      out.printlnBold("Business Rule Group before publish:");
      // Get the first business rule group from the list
      // This should be the only business rule group in the
      list as
      // the combination of target namespace and name are unique
      BusinessRuleGroup brg = brgList.get(0);

      // Get the operation of the business rule group that
      // will have a new business rule scheduled
      Operation op =
      brg.getOperation("calculateShippingDiscount");

      printOperationSelectionRecord(op);
      // Get the list of available business rules for this operation
      List<BusinessRule> ruleList =
      op.getAvailableTargets();

      // Get the first rule in the list as this will be scheduled
      // for the operation
      BusinessRule rule = ruleList.get(0);

      // Get the list of scheduled business rules
      OperationSelectionRecordList opList = op
        .getOperationSelectionRecordList();
      // Create an end date in the future for the business rule
      Date future = new Date();
      long futureTime = future.getTime() + 3600000;
```

For a new scheduled rule, a start date and end date can be specified along with the rule. If the start date is set to null, this indicates that the rule will be active immediate upon publish. If an end date is set to null, the rule will not have an end date. An overlap of schedules is not allowed and can be checked by calling the validate method on the operation.

```
      // Create the new scheduled business rule with the current
      // date which means this rule will become active immediately
      // upon
      // publish and the future date.
      newOperationSelectionRecord(new Date(),
        new Date(futureTime), rule);
      // Add the new scheduled business rule to the list of
      // scheduled rule
      opList.addOperationSelectionRecord(newRecord);
```

Validate operation to insure that an overlap does not exist.

```
      // Validate the list to insure there isn't an overlap
      List<Problem> problems = op.validate();
      if (problems.size() == 0)
      {
        // Use the original list or create a new list
        // of business rule groups
        List<BusinessRuleGroup> publishList = new
        ArrayList<BusinessRuleGroup>();
        // Add the changed business rule group to the list
        publishList.add(brg);
        // Publish the list with the updated business
        rule group
        BusinessRuleManager.publish(publishList, true);
        out.println("");
```

```
        // Retrieve the business rule groups again to
        verify the
        // changes were published
        out.printlnBold("Business Rule Group after
        publish:");

        brgList =
        BusinessRuleManager.getBRGsByTNSAndName(
          "http://BRSamples/com/ibm/websphere
          /sample/brules",
          QueryOperator.EQUAL,
          "DiscountRules",
          QueryOperator.EQUAL, 0, 0);
          brg = brgList.get(0);

          op =
          brg.getOperation("calculateShippingDiscount");

          printOperationSelectionRecord(op);
        }
        // else handle the validation error
      }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
}
/*
Method to print the operation selection record for an operation. The
start date and end date are printed as well as the name of the rule
artifact for the scheduled time.
*/
private static void printOperationSelectionRecord(Operation op)
{
 OperationSelectionRecordList opSelectionRecordList = op
   .getOperationSelectionRecordList();
 Iterator<OperationSelectionRecord> opSelRecordIterator =
 opSelectionRecordList
   .iterator();
 OperationSelectionRecord record = null;
 while (opSelRecordIterator.hasNext())
 {
   out.printlnBold("Scheduled Destination:");
   record = opSelRecordIterator.next();
   out.println("Start Date: " + record.getStartDate()
   + " - End Date: " + record.getEndDate());
   BusinessRule ruleArtifact = record.getBusinessRuleTarget();
   out.println("Rule: " + ruleArtifact.getName());
   }
 }
}
```

Example

Web browser output for example 9.

```
Executing example9

Business Rule Group before publish:
Scheduled Destination:
Start Date: Thu Dec 01 00:00:00 CST 2005 - End Date: Sun Dec 25 00:00:00 CST 2005
Rule: calculateShippingDiscountHoliday

Business Rule Group after publish:
```

```
Scheduled Destination:
Start Date: Thu Dec 01 00:00:00 CST 2005 - End Date: Sun Dec 25 00:00:00 CST 2005
Rule: calculateShippingDiscountHoliday
Scheduled Destination:
Start Date: Mon Jan 07 21:08:31 CST 2008 - End Date: Mon Jan 07 22:08:31 CST 2008
Rule: calculateShippingDiscount
```

Example 10: Modify a parameter value in a template in a rule set

In this example a rule instance defined with a template is modified by changing a parameter value and then publish.

```
package com.ibm.websphere.sample.brules.mgmt;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;

public class Example10
{
static Formatter out = new Formatter();

static public String executeExample10()
{
  try
  {
    out.clear();

    // Retrieve a business rule group by target namespace and
    name
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "ApprovalValues",
        QueryOperator.EQUAL, 0, 0);
    if (brgList.size() > 0)
    {
      // Get the first business rule group from the list
      // This should be the only business rule group in the
      list as
      // the combination of target namespace and name are
      unique
      BusinessRuleGroup brg = brgList.get(0);
      // Get the operation of the business rule group that
      // has the business rule that will be modified as
      // the business rules are associated with a specific
      // operation
      Operation op = brg.getOperation("getApprover");

      // Get the business rule on the operation that will
      be modified
      List<BusinessRule> ruleList =
      op.getBusinessRulesByName(
        "getApprover", QueryOperator.EQUAL, 0,
        0);
```

```
if (ruleList.size() > 0)
{
  out.println("");
  out.printlnBold("Rule set before publish:");
  // Get the rule to be modified. Rules are
  unique by
  // target namespace and name, but for this
  example
  // there is only one business rule named
  "getApprover"
  RuleSet ruleSet = (RuleSet) ruleList.get(0);
  out.print(RuleArtifactUtility.printRuleSet(rule
  Set));
```

All of the rules in a rule set are in a rule block. Only one rule block is supported and the `getFirstRuleBlock` method should be used to retrieve the rule block.

```
// A rule set has all of the rules defined in a
rule block
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

// Iterate through the rules in the rule block
to find the
// rule instance called "LargeOrderApprover"
while (ruleIterator.hasNext())
{
  RuleSetRule rule = ruleIterator.next();
```

If a rule is not defined with a rule template, it only has a Web presentation that can be retrieved. No updates can be made to a rule that is not defined with a template. It is best to check if a rule has been defined with a template if the name of the rule is unknown.

```
// The rule must have been defined with a
template
// in order for it to be changed. Check
if the current
// rule is even based on a template.
if (rule instanceof
RuleSetTemplateInstanceRule)
{
```

Use the TemplateInstance object to create the rule.

```
  // Get the rule template instance
  RuleSetTemplateInstanceRule
  templateInstance =
  (RuleSetTemplateInstanceRule) rule;

  // Check for the rule instance
  which matches
  // the rule to modify
  if
  (templateInstance.getName().equals(
    "LargeOrderApprover"))
  {
```

For the template instance, only parameter values can be modified. The parameters are modified by retrieving the `ParameterValue` and setting it to the appropriate value. Because the `ParameterValue` is passed by reference, the update is made directly on the rule, rule set, and business rule group.

```
          // Get the parameter from the
          rule instance
          ParameterValue parameter =
          templateInstance
            .getParameterValue("par
            am2");

          // Modify the value of the
          parameter
          parameter.setValue("superviso
          r");
          break;
        }
      }
    }
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the list
    publishList.add(brg);

    // Publish the list with the updated business rule
     group
     BusinessRuleManager.publish(publishList, true);

    out.println("");
    // Retrieve the business rule groups again to verify
    the
    // changes were published
    out.printlnBold("Rule set after publish:");

    brgList = BusinessRuleManager
      .getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere/sample/brules",
        QueryOperator.EQUAL, "ApprovalValues",
        QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    op = brg.getOperation("getApprover");
    ruleList = op.getBusinessRulesByName(
      "getApprover", QueryOperator.EQUAL, 0,0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(ruleSet));
    }
  }
  } catch (BusinessRuleManagementException e)
  {
    e.printStackTrace();
    out.println(e.getMessage());
  }
  return out.toString();
  }
}
```

Example

Web browser output for example 10.

```
Executing example10

Rule set before publish:
Rule Set
Name: getApprover
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
```

```
Rule: LargeOrderApprover
Display Name: LargeOrderApprover
Description: null
Expanded User Presentation: If the number of items order is above 10 and
the order is above $5000, then it requires the approval of manager
User Presentation: If the number of items order is above {0} and
the order is above ${1}, then it requires the approval of {2}
Parameter Name: param0
Parameter Value: 10
Parameter Name: param1
Parameter Value: 5000
Parameter Name: param2
Parameter Value: manager
Rule: DefaultApprover
Display Name: DefaultApprover
Description: null
Expanded User Presentation: approver = peer
User Presentation: approver = {0}
Parameter Name: param0
Parameter Value: peer

Rule set after publish:
Rule Set
Name: getApprover
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Rule: LargeOrderApprover
Display Name: LargeOrderApprover
Description: null
Expanded User Presentation: If the number of items order is above 10 and
the order is above $5000, then it requires the approval of supervisor
User Presentation: If the number of items order is above {0} and the order
is above ${1}, then it requires the approval of {2}
Parameter Name: param0
Parameter Value: 10
Parameter Name: param1
Parameter Value: 5000
Parameter Name: param2
Parameter Value: supervisor
Rule: DefaultApprover
Display Name: DefaultApprover
Description: null
Expanded User Presentation: approver = peer
User Presentation: approver = {0}
Parameter Name: param0
Parameter Value: peer
```

Example 11: Add a new rule from a template to a rule set

In this example, a new rule is added from a template to a rule set. Before the new rule instance is created, parameters for the new rule instance are created.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
```

```
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example11
{
static Formatter out = new Formatter();

static public String executeExample11()
{
 try
 {
  out.clear();
  // Retrieve a business rule group by target namespace and
  name
  List<BusinessRuleGroup> brgList = BusinessRuleManager
   .getBRGsByTNSAndName(
   "http://BRSamples/com/ibm/websphere
   /sample/brules",
   QueryOperator.EQUAL,
   "ApprovalValues",
   QueryOperator.EQUAL, 0, 0);

  if (brgList.size() > 0)
  {
   // Get the first business rule group from the list
   // This should be the only business rule group in the
   list as
   // the combination of target namespace and name are
   unique
   BusinessRuleGroup brg = brgList.get(0);
   // Get the operation of the business rule group that
   // has the business rule that will be modified as
   // the business rules are associated with a specific
   // operation
   Operation op = brg.getOperation("getApprover");

   // Get the business rule on the operation that will
   be modified
   List<BusinessRule> ruleList =
   op.getBusinessRulesByName(
    "getApprover", QueryOperator.EQUAL, 0,0);

   if (ruleList.size() > 0)
   {
    out.println("");
    out.printlnBold("Rule set before publish:");
    // Get the rule to be modified. Rules are unique by
    // target namespace and name, but for this example
    // there is only one business rule named
    "getApprover"
    RuleSet ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(rule
    Set));
```

In order to add a new rule to the rule set, the appropriate template must be located in the rule set and an instance created from the template. The template can be located by name.

```
    // Get the list of rule templates
    ListRuleSetRuleTemplate> ruleTemplates =
    ruleSet
     .getRuleTemplates();

    Iterator<RuleSetRuleTemplate> templateIterator
    = ruleTemplates
     .iterator();
```

```
while (templateIterator.hasNext())
{
 RuleSetRuleTemplate template =
 templateIterator.next();

 // Locate the template to use to create a
 new rule
 if
 (template.getName().equals("Template_Larg
 eOrder"))
 {
```

For a template instance, a list of parameters must be created.

```
// Create a list for the parameters
for this template
// rule instance
List<ParameterValue> paramList =
new ArrayList<ParameterValue>();

// From the template definition,
get a specific parameter
// and set the value
Parameter param =
template.getParameter("param0");
ParameterValue paramValue = param
 .createParameterValue("
 20");

// Add parameter to the list
paramList.add(paramValue);

// Get the next parameter and set
the value
param = template.getParameter("param1");
paramValue =
param.createParameterValue("7500");

// Add parameter to the list
paramList.add(paramValue);

// Get the next parameter and set
the value
param =
template.getParameter("param2");
 paramValue = param
 .createParameterValue("
 2nd-line manager");

// Add parameter to the list
paramList.add(paramValue);
```

With the parameters created, the template instance can be created.

```
// Create the template rule
instance with the parameter
// list
RuleSetTemplateInstanceRule
 templateInstance = template
 .createRuleFromTemplate
 ("ExtraLargeOrder",
 paramList);
// Get the ruleblock for the rule
 set
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();
```

Once the template instance is created, it can be added to the ruleblock. Once it is added to the rule block it can be ordered among other template rule instances.

```
     // Add the template rule to the
     ruleblock
     ruleBlock.addRule(templateInstance)
     ;

     break;
     }
    }

    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the
    list
    publishList.add(brg);

    // Publish the list with the updated business
    rule group
    BusinessRuleManager.publish(publishList, true);

    out.println("");

    // Retrieve the business rule groups again to
    verify the
    // changes were published
    out.printlnBold("Rule set after publish:");

    brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "ApprovalValues",
      QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    op = brg.getOperation("getApprover");
    ruleList = op.getBusinessRulesByName(
         "getApprover", QueryOperator.EQUAL,
         0, 0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(rule
    Set));
    }
  }
 } catch (BusinessRuleManagementException e)
 {
  e.printStackTrace();
  out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Example

Web browser output for example 11.

```
Executing example11

Rule set before publish:
```

```
Rule Set
Name: getApprover
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Rule: LargeOrderApprover
Display Name: LargeOrderApprover
Description: null
Expanded User Presentation: If the number of items order is above 10 and
the order is above $5000, then it requires the approval of supervisor
User Presentation: If the number of items order is above {0} and the order
is above ${1}, then it requires the approval of {2}
Parameter Name: param0
Parameter Value: 10
Parameter Name: param1
Parameter Value: 5000
Parameter Name: param2
Parameter Value: supervisor
Rule: DefaultApprover
Display Name: DefaultApprover
Description: null
Expanded User Presentation: approver = peer
User Presentation: approver = {0}
Parameter Name: param0
Parameter Value: peer

Rule set after publish:
Rule Set
Name: getApprover
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
Rule: LargeOrderApprover
Display Name: LargeOrderApprover
Description: null
Expanded User Presentation: If the number of items order is above 10 and
the order is above $5000, then it requires the approval of supervisor
User Presentation: If the number of items order is above {0} and the order
is above ${1}, then it requires the approval of {2}
Parameter Name: param0
Parameter Value: 10
Parameter Name: param1
Parameter Value: 5000
Parameter Name: param2
Parameter Value: supervisor
Rule: DefaultApprover
Display Name: DefaultApprover
Description: null
Expanded User Presentation: approver = peer
User Presentation: approver = {0}
Parameter Name: param0
Parameter Value: peer
Rule: ExtraLargeOrder
Display Name:
Description: null
Expanded User Presentation: If the number of items order is above 20 and
the order is above $7500, then it requires the approval of 2nd-line manager
User Presentation: If the number of items order is above {0} and the order
is above ${1}, then it requires the approval of {2}
Parameter Name: param0
Parameter Value: 20
Parameter Name: param1
Parameter Value: 7500
Parameter Name: param2
Parameter Value: 2nd-line manager
```

Example 12: Modify a template in a decision table by changing a parameter value and then publish

In this example, a condition and action, both defined with templates, are modified in a decision table by changing the parameter values before it is published.

The easiest way to modify conditions and actions in a decision table is to use unique names for the templates at each condition level and for each action. The unique names can be searched for and then changes can be made to template instances defined with the template. When changes are made to a template instance of a particular template, all of the condition values defined with that template at that level will be updated. For action expressions, each instance is unique and a change to one does not change others.

For this example, there are a number of additional methods that were created to simplify the locating of a specific case edge for update, finding the specific parameter value, and finding the action expression defined with a specific template.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example12 {
static Formatter out = new Formatter();

static public String executeExample12()
{
 try
 {
   out.clear();
   // Retrieve a business rule group by target namespace and
   name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere
    /sample/brules",
    QueryOperator.EQUAL,
    "ConfigurationValues",
    QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
   {
    // Get the first business rule group from the list
    // This should be the only business rule group in the
    list as
    // the combination of target namespace and name are
    unique
    BusinessRuleGroup brg = brgList.get(0);
```

```
// Get the operation of the business rule group that
// has the business rule that will be modified as
// the business rules are associated with a specific
// operation
Operation op = brg.getOperation("getMessages");

// Get all business rules available for this
operation
List<BusinessRule> ruleList =
op.getAvailableTargets();

// For this operation there is only 1 business rule
and
// it is the business that we want to update
DecisionTable decisionTable = (DecisionTable)
ruleList.get(0);
out.println("");
out.printlnBold("Decision table before publish:");
out
 .print(RuleArtifactUtility
 .printDecisionTable(decisionT
  able));
```

The init rule and condition and actions are contained in a tree block. With the tree block, the root node can be retrieved.

```
// Get the tree block that contains all of the
conditions
// and actions for the decision table
TreeBlock treeBlock = decisionTable.getTreeBlock();
// From the tree block, get the tree node which is
the
// starting point for navigating through the decision
table
TreeNode treeNode = treeBlock.getRootNode();
```

The condition to be updated was defined with a template by the name of "Condition Value Template 2.1". The method `getCaseEdge` will recursively search from the TreeNode down to the appropriate case edge to find the case edge where the template is defined. The method expects that the level at which the template is defined is known as well as the current level. This method can be used to find the case edge with a template by a specific name in case the same name is used at multiple case edges.

```
// Find the case edge at level 1 below the root with
// specific template with a parameter value that has
// a specific name. Since we are starting at the top,
// the current depth is 0
CaseEdge caseEdge = getCaseEdge(treeNode, "param0",
"Condition Value Template 2.1", 1, 0);
```

With the case edge found, the `ConditionValueTemplateInstance` for the condition can be retrieved.

```
if (caseEdge != null)
{
 // Case edge was found. Get the value
 definition of the
 // case edge
 TreeConditionValueDefinition condition =
  caseEdge
  .getValueDefinition();
 // Get the condition expression defined with a
  template
```

```
TemplateInstanceExpression conditionExpression
= condition
 .getConditionValueTemplateInstance(
);
```

With the ConditionValueTemplateInstance , the appropriate parameter value can be retrieve and then updated with the getParameterValue method.

```
// Get the template for the expression
Template conditionTemplate =
conditionExpression
 .getTemplate();

 // Check that template is correct as it is
 possible to have
 // multiple templates for a condition value,
 but only one
 // applied
 if (conditionTemplate.getName().equals(
  "Condition Value Template 2.1"))
 {
  // Get the parameter value
  ParameterValue parameterValue =
  getParameterValue("param0",
   conditionExpression);

  // Set the new parameter value
  parameterValue.setValue("info");
 }
```

The different action expressions defined with templates that need to be updated can then be retrieved. The getActionExpressions method will return all actions that are defined with the template by name Action Value Template 1.

```
ConditionNode conditionNode = (ConditionNode)
treeNode;

// Get the case edges tree node
ListCaseEdge> caseEdges =
conditionNode.getCaseEdges();

// Create a list to hold all of the action
expressions that
// also need to be updated. Because every
action is
// independent of other action even though the
template is
// shared, all must be updated.
List<TemplateInstanceExpression> expressions =
new Vector<TemplateInstanceExpression>();

// Retrieve all of the expressions
for (CaseEdge edge : caseEdges)
{
  getActionExpressions("Action Value
  Template 1", edge,
  expressions);
}
```

With the list of action expressions, each item can be updated. For action expressions defined with templates the correct parameter value can be updated.

```
// Update the correct parameter in each
expression
for (TemplateInstanceExpression expression
expressions)
{
```

```
    for (ParameterValue parameterValue :
    expression
     .getParameterValues())
    {
     // Check for correct parameter
     although there is
     // only one paramater in our
     template
     if
      (parameterValue.getParameter().getN
      ame().equals("param0")) {
       String value =
       parameterValue.getValue();
       parameterValue.setValue("Info
       "
       +
       value.substring(value.
       indexOf(":"),
       value.length()));
    }
  }
}
// With the condition value and actions
updated, the
// business rule group can be published.
// Use the original list or create a new list
// of business rule groups
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Add the changed business rule group to the
list
publishList.add(brg);

// Publish the list with the updated business
rule group
BusinessRuleManager.publish(publishList, true);

out.println("");

// Retrieve the business rule groups again to
verify the
// changes were published
out.printlnBold("Decision table after
publish:");

brgList =
BusinessRuleManager.getBRGsByTNSAndName(
 "http://BRSamples/com/ibm/websphere
 /sample/brules",
 QueryOperator.EQUAL,
 "ConfigurationValues",
 QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
op = brg.getOperation("getMessages");
ruleList = op.getAvailableTargets();

decisionTable = (DecisionTable)
ruleList.get(0);
out.print(RuleArtifactUtility
 .printDecisionTable(decisionTable))
 ;
 }
}
} catch (BusinessRuleManagementException e)
{
```

```
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
}

/*
Method to recursively navigate through a decision table and locate a
case
edge that has a template with a specific name and contains a specific
parameter to change. This method assumes that the level(depth) in the
decesion table of the value that is to be changed is known and the
current level(currentDepth) is tracked *
*/
static private CaseEdge getCaseEdge(TreeNode node, String pName,
  String templateName, int depth, int currentDepth)
{
 // Check if the current node is an action. This is an indication
 // that this branch of the decision table has been exhausted
 // looking for the case edge
 if (node instanceof ActionNode)
 {
  return null;
 }

 // Get the case edges for this node
 List<CaseEdge> caseEdges = ((ConditionNode) node).getCaseEdges();
 for (CaseEdge caseEdge : caseEdges)
 {

 // Check if the correct level has been reached
 if (currentDepth < depth)
  {
   // Move down one level and then call getCaseEdge
   again
   // to process that level
   currentDepth++;
   return getCaseEdge(caseEdge.getChildNode(), pName,
    templateName, depth, currentDepth);
  } else
  {
   // The correct level has been reached. Get the
   condition in
   // order to check the templates on that condition on
   whether
   // they match the template sought
   TreeConditionValueDefinition condition = caseEdge
    .getValueDefinition();

   // Get the expression for the condition which has
   been defined
   // with a template
   TemplateInstanceExpression expression = condition
    .getConditionValueTemplateInstance();
   // Get the template from the expression
      Template template = expression.getTemplate();

   // Check if this is the template sought
    if (template.getName().equals(templateName))
    {
     // The template is found to match
      return caseEdge;
    } else
     caseEdge = null;
   }
 }
 return null;
```

```
}

/*
This method will check the different parameter values for an expression
and if the correct one is found, return that parameter value.
*/
private static ParameterValue getParameterValue(String pName,
  TemplateInstanceExpression expression)
{
 // Check that the expression is not null as null would indicate
 // that the expression that was passed in was probably not
 defined
 // with a template and does not have any parameters to check.
 if (expression != null) {
  // Get the parameter values for the expression
  List<ParameterValue> parameterValues = expression
    .getParameterValues();

  for (ParameterValue parameterValue : parameterValues)
  {
   // For the different parameters, check that it
   matches the
   // parameter value sought

   if
    (parameterValue.getParameter().getName().equals(pName
   ))
   {
    // Return the parameter value that matched
      return parameterValue;
   }
  }
 }
return null;
}
/*
This method finds all of the action expressions that are
defined with a specific template. It recursively works through
a case edge and adds action expressions that match to the
expressions parameter.
*/

private static void getActionExpressions(String templateName,
CaseEdge next, List<TemplateInstanceExpression>
expressions)
{
 ActionNode actionNode = null;
 TreeNode treeNode = next.getChildNode();

 // Check if the current node is at the action node level
 if (treeNode instanceof ConditionNode)
 {
  List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
   .getCaseEdges();

  Iterator<CaseEdge> caseEdgesIterator =
  caseEdges.iterator();

  // Work through all case edges to find the action
  // expressions
  while (caseEdgesIterator.hasNext())
   {
    getActionExpressions(templateName,
    caseEdgesIterator.next(),
     expressions);
   }
 } else {
```

```
    // ActionNode found
    actionNode = (ActionNode) treeNode;

    List<TreeAction> treeActions = actionNode.getTreeActions();
    // Check that there is at least one treeAction specified
    for
    // the expression and work through the expressions checking
    // if the expressions have been created with the specific
    // template.
    if (!treeActions.isEmpty())
    {

     Iterator<TreeAction> iterator =
     treeActions.iterator();

    while (iterator.hasNext())
     {
      TreeAction treeAction = iterator.next();
      TemplateInstanceExpression expression =
      treeAction
       .getValueTemplateInstance();

      Template template = expression.getTemplate();

      if (template.getName().equals(templateName))
      {
       // Expression found with matching
       template
       expressions.add(expression);
      }
     }
    }
   }
  }
}
```

Example

Web browser output for example 12.

```
Executing example12

Rule set before publish:
Decision Table
Name: getMessages
Namespace: http://BRSamples/com/ibm/websphere/sample/brules

Decision table after publish:
Decision Table
Name: getMessages
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
```

Example 13: Add a condition value and actions to a decision table

In this example, a condition value and action are added to a decision table. Condition values can be added to a decision table through the use of a template.

When adding a condition value to a condition node, you are adding a case edge. The new case edge is added at the end of the list of case edges. For the condition value, you must specify a template instance expression that has the appropriate parameter values set. In order to specify the template instance expression you will have to use a specific template. It is recommended to give the template names at each condition node level unique names in order to retrieve the correct templates

for that type of condition. If a single template definition is used, it may make it difficult to determine at which level the condition is being added.

When setting the condition value in a condition node, this will add the condition value with the same template instance to all condition nodes at the same level. This is done as the decision table is balanced. Also as part of the adding a new condition value, new action nodes will be added. These action nodes have tree actions that have null values specified for the user presentation and template instance expression. Because the condition value can be added to a condition node that does not have an action node as a child node, the addition of a condition node may result in a large number of action nodes. The number of action nodes is based upon the level the condition node is added and the number of condition nodes at that level and the number of condition nodes at each child level.

In order to find the action nodes that have been created, a search of action nodes with tree actions that have null user presentation and template instance expression may be performed. A `TreeActionValueTemplate` can be used to create an expression that can be set into the `TreeAction`. This pattern would need to be repeated for all new action nodes.

For this example two methods were provided to assist in setting up the new tree actions. `getEmptyActionNode` recursively looks for an empty action node from the current condition node and getParameterValue returns the value of a parameter that was specified by name.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example13
{
 static Formatter out = new Formatter();

 static public String executeExample13()
 {
  try
  {
   out.clear();
```

```
// Retrieve a business rule group by target namespace
// and name
List<BusinessRuleGroup> brgList = BusinessRuleManager
 .getBRGsByTNSAndName(
"http://BRSamples/com/ibm/websphere/sample/brules",
  QueryOperator.EQUAL,"ConfigurationValues",
  QueryOperator.EQUAL, 0, 0);

if (brgList.size() > 0)
 {
  // Get the first business rule group from the
  // list. This should be the only business
  // rule group in the list as the combination
  // of target namespace and name are unique
  BusinessRuleGroup brg = brgList.get(0);

  // Get the operation of the business rule
  // group that has the business rule that will
  // be modified as the business rules are
  // associated with a specific operation
  Operation op = brg.getOperation("getMessages");

  // Get all business rules available for
  // this operation
  List<BusinessRule> ruleList =
   op.getAvailableTargets();

  // For this operation there is only 1 business
  // rule and it is the business that we want
  // to update

  DecisionTable decisionTable = (DecisionTable)
   ruleList.get(0);
   out.printlnBold("Decision table before
   publish:");
   out.print(RuleArtifactUtility
   .printDecisionTable(decisionTable));
```

The level at which the condition value is going to be added needs to be located. This is typically passed as parameter as the user interface or application using the classes knows where to add the condition.

```
  // Get the tree block that contains all of
  // the conditions and actions for the decision
  // table
  TreeBlock treeBlock =
   decisionTable.getTreeBlock();

  // From the tree block, get the tree node which
  // is the starting point for navigating through
  // the decision table
  ConditionNode conditionNode = (ConditionNode)
   treeBlock.getRootNode();

  // Get the case edges for this node which is
  // the first level of conditions
  List<CaseEdge> caseEdges =
   conditionNode.getCaseEdges();

  // Get the case edge which will have the new
  // condition added
  CaseEdge caseEdge = caseEdges.get(0);

  // For the case edge get the condition node in
  // order to retrieve the templates for the
  // condition
```

```
conditionNode = (ConditionNode)
    caseEdge.getChildNode();

// Get the templates for the condition
List<TreeConditionValueTemplate>
 treeValueConditionTemplates = conditionNode
 .getAvailableValueTemplates();

Iterator<TreeConditionValueTemplate>
 treeValueConditionTemplateIterator =
 treeValueConditionTemplates.iterator();

TreeConditionValueTemplate conditionTemplate =
 null;
```

By using unique template names at each condition node level in the decision table, you can more easily insure the condition value is being added at the correct condition node value.

```
// Find the template that should be used
 while
(treeValueConditionTemplateIterator.hasNext())
{
 conditionTemplate =
  treeValueConditionTemplateIterator
   .next();
 if (conditionTemplate.getName().equals(
  "Condition Value Template
  2.1"))
  {
   // Template found
   break;
  }
  conditionTemplate = null;
}
if (conditionTemplate != null)
{
```

With the correct template found, an instance can be created and the appropriate parameter value set before it is added to the condition node.

```
// Get the parameter definition from the
// template
Parameter conditionParameter =
conditionTemplate.getParameter("param0");

// Create a parameter value instance to
// be used in a new condition template
// instance
ParameterValue conditionParameterValue =
  conditionParameter
  .createParameterValue("fatal");

List<ParameterValue>
  conditionParameterValues = new
  ArrayList<ParameterValue>();

// Add the parameter value to a list

conditionParameterValues
 .add(conditionParameterValue);

// Create a new condition template
// instance with the parameter value
TemplateInstanceExpression
 newConditionValue =
 conditionTemplate
```

```
  .createTemplateInstanceExpression(c
 onditionParameterValues);
// Add the condition template instance to
// this condition node
conditionNode

.addConditionValueToThisLevel(newConditionValue);
// When a condition node is added there
// are new action nodes that are created
// and empty. These must be filled with
// action template instances. By
// searching for each empty action
// node from the parent level, all of the
// new empty action nodes can be found.
conditionNode = (ConditionNode)
conditionNode.getParentNode();
```

With the condition value added to the condition node, the tree actions in the new action nodes must be set with a `TreeActionValueTemplate`. First locate the empty action node for the case edges. Use the parent condition node to insure that as you iterate through the condition nodes, you will pick up all action nodes.

```
// Get the case edges for the parent node
caseEdges = conditionNode.getCaseEdges();

Iterator<CaseEdge> caseEdgesIterator =
caseEdges.iterator();

while (caseEdgesIterator.hasNext())
{
 // For each case edge, retrieve an
 // empty action node if it exists
 ActionNode actionNode =
 getEmptyActionNode(caseEdgesIterator
 .next());

 // Check if all actions are filled
 if (actionNode != null)
 {
```

When an action node is found with empty tree actions, the tree action must be set with a `TreeActionValueTemplate`. First locate the template and then specify the parameters before creating a template instance. With the template instance created, the tree action can be updated. For this example the parameter was set with a value from another tree action in another action node under the same condition node. For other decision tables where another tree action might not have a value that may be used to create the new parameter values, the value will have to be passed as a parameter from the application.

```
  // Get the list of tree
  // actions. These
  // are not the actual
  // actions, but the
  // placeholders for the
  // actions
  List<TreeAction>
  treeActionList = actionNode
   .getTreeActions();

  List<TreeActionTermDefinition>
  treeActionTermDefinitions =
   treeBlock
  .getTreeActionTermDefinitions();

  List<TreeActionValueTemplate>
   treeActionValueTemplates =
```

```
 treeActionTermDefinitions
 .get(0).getValueTemplates();

TreeActionValueTemplate
 actionTemplate = null;

for (TreeActionValueTemplate
  tempActionTemplate :
  treeActionValueTemplates)
  {

  if
  (tempActionTemplate.get
   Name().equals(
   "Action Value
   Template 1"))
   {
    actionTemplate =
    tempActionTemplate;
    break;
   }
  }

  if (actionTemplate != null)
  {
   // Get another action
   // that is under
   // the parent condition
   // node in order
   // to use the value as
   // the basis for
   // the error message in
   // the new
   // action node. Move up
   // to the
   // parent condition
   // node first
   ConditionNode
   parentNode =
   (ConditionNode)
   actionNode
    .getParentNode();

   // Get the first case
   // edge of the
   // parent node as this
   // action will
   // always be filled in
   // as new actions
   // are added to the end
   // of the case
   // edge list.
   CaseEdge caseE =
    parentNode.getCas
    eEdges().get(
    0);

   // The child node is an
   // action node
   // and at the same
   // level as the new
   // action node.
   ActionNode aNode =
   (ActionNode) caseE
    .getChildNode();

   // Get the list of tree
```

```
// actions
TreeAction
 existingTreeAction =
 aNode
 .getTreeActions()
 .get(0);

// Get the template
// instance
// expression for the
// tree action
// from which you can
// retrieve the
// parameter

TemplateInstanceExpression
existingExpression =
 existingTreeAction
 .getValueTemplateInstance();

ParameterValue
existingParameterValue =
getParameterValue(
  "param0",
existingExpression);

String actionValue =
existingParameterValue
 .getValue();

// Create the new
// message from the
// message of the
// existing
// tree action
actionValue = "Fatal"
  +
actionValue.substring(actionValue
 .indexOf(":"), actionValue
 .length());
 Parameter
 actionParameter =
 actionTemplate
 .getParameter("param0");

 // Get the parameter
 // from the template
 ParameterValue
 actionParameterValue =
  actionParameter
 .createParameterValue(actionValue);

 // Add the parameter to
 // a list of templates
 List<ParameterValue>
 actionParameterValues = new
 ArrayList<ParameterValue>();

actionParameterValues.add(actionParameterValue);

 // Create a new tree
 // action instance

TemplateInstanceExpression
 treeAction = actionTemplate
.createTemplateInstanceExpression(actionParameterValues);
```

```
        // Set the tree action
        // in the action node
        // by setting it in the
        // tree action list
```

Here the tree action in the action node is updated.

```
        treeActionList.get(0)
        .setValueTemplateInstance(
        treeAction);
        }
       }
      }
    }
    // With the condition value and actions
    // updated, the business rule group can be
    // published.
    // Use the original list or create a new list
    // of business rule groups
     List<BusinessRuleGroup> publishList = new
      ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the
    // list
     publishList.add(brg);

    // Publish the list with the updated business
    // rule group

    BusinessRuleManager.publish(publishList, true);

    brgList =
     BusinessRuleManager.getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere/sample/brules",
     QueryOperator.EQUAL, "ConfigurationValues",
     QueryOperator.EQUAL, 0, 0);
     brg = brgList.get(0);
     op = brg.getOperation("getMessages");
     ruleList = op.getAvailableTargets();
     decisionTable = (DecisionTable)
      ruleList.get(0);
     out.printlnBold("Decision table after
      publish:");
     out
      .print(RuleArtifactUtility
       .printDecisionTable(decisionTable));
    }
  } catch (ValidationException e)
  {
   List<Problem> problems = e.getProblems();

   out.println("Problem = " +
   problems.get(0).getErrorType().name());

   e.printStackTrace();
   out.println(e.getMessage());
  } catch (BusinessRuleManagementException e)
  {
   e.printStackTrace();
   out.println(e.getMessage());
  }
  return out.toString();
 }

 /*
 * This method searches from the current case edge for any
 * action nodes that have empty tree actions. An empty
```

```
* action node is found by looking at the end of the list
* of case edges and checking if the action node has tree
* actions that have both a null user presentation and
* TemplateInstanceExpression.
*/
private static ActionNode getEmptyActionNode(CaseEdge next)
{
 ActionNode actionNode = null;
 TreeNode treeNode = next.getChildNode();

if (treeNode instanceof ConditionNode)
{
 List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
  .getCaseEdges();

 if (caseEdges.size() > 1)
 {
  // Get right-most case-edge as the new
  // condition and thus empty actions are at the
  // right-end of the case edges
  actionNode = getEmptyActionNode(caseEdges
   .get(caseEdges.size() - 1));

  if (actionNode != null)
  {
   return actionNode;
  }
}
} else
{
 actionNode = (ActionNode) treeNode;

 List<TreeAction> treeActions =
 actionNode.getTreeActions();

 if (!treeActions.isEmpty())
 {
 if
 ((treeActions.get(0).getValueUserPresentation() == null)
  &&
 (treeActions.get(0).getValueTemplateInstance() == null))
  {
   return actionNode;
  }
 }
  `actionNode = null;
 }
 return actionNode;
}
/*
* This method will check the different parameter values for an
* expression and if the correct one is found, return that
* parameter value.
*/
private static ParameterValue getParameterValue(String pName,
  TemplateInstanceExpression expression)
{
 ParameterValue parameterValue = null;

 // Check that the expression is not null as null would
 // indicate that the expression that was passed in was
 // probably not defined with a template and does not have
 // any parameters to check.
 if (expression != null)
 {
  // Get the parameter vlues for the expression
  List<ParameterValue> parameterValues = expression
```

```
    .getParameterValues();
   Iterator<ParameterValue> parameterIterator =
    parameterValues
    .iterator();

   // For the different parameters, check that it
   // matches the parameter value sought
   while (parameterIterator.hasNext())
   {
    parameterValue = parameterIterator.next();

    if
    (parameterValue.getParameter().getName().equals(pName))
    {
    // Return the parameter value that
    // matched
    return parameterValue;
    }
  }
 }
 return parameterValue;
}
}
```

Example

Web browser output for example 13.

```
Executing example13

Decision table before publish:
Decision Table
Name: getMessages
Namespace: http://BRSamples/com/ibm/websphere/sample/brules

Decision table after publish:
Decision Table
Name: getMessages
Namespace: http://BRSamples/com/ibm/websphere/sample/brules
```

Example 14: Handle errors in a rule set

This example focuses on how to catch problems in a rule set and find out what problem has occurred such that the appropriate message can be displayed or action can be taken to correct the situation.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.problem.ValidationError;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
```

```
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example14 {
static Formatter out = new Formatter();

static public String executeExample14() {
 try {
   out.clear();

   // Retrieve a business rule group by target namespace and
   name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
     QueryOperator.EQUAL,
     "ApprovalValues",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0) {
     // Get the first business rule group from the list
     // This should be the only business rule group in the
     list as
     // the combination of target namespace and name are
     unique
     BusinessRuleGroup brg = brgList.get(0);
     out.println("Business Rule Group retrieved");

     // Get the operation of the business rule group that
     // has the business rule that will be modified as
     // the business rules are associated with a specific
     // operation
     Operation op = brg.getOperation("getApprover");

     // Retrieve specific rule by name
     List<BusinessRule> ruleList =
     op.getBusinessRulesByName(
       "getApprover", QueryOperator.EQUAL, 0,
       0);

     // Get the specific rule
     RuleSet ruleSet = (RuleSet) ruleList.get(0);
     out.println("Rule Set retrieved");

     RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

     Iterator<RuleSetRule> ruleIterator =
     ruleBlock.iterator();

     // Search through the rules to find the rule to
     change
     while (ruleIterator.hasNext()) {
       RuleSetRule rule = ruleIterator.next();

       // Check that the rule was defined with a
       template
       // as it can be changed.
       if (rule instanceof
       RuleSetTemplateInstanceRule) {
        // Get the template rule instance
        RuleSetTemplateInstanceRule
        templateInstance =
        (RuleSetTemplateInstanceRule) rule;
        // Check for the correct template rule
        instance
        if (templateInstance.getName().equals(
          "LargeOrderApprover")) {
```

To cause a problem, this example sets a parameter to a value that is not compatible for the expression. The parameter is expecting an integer, but a string is passed in.

```
          // Get the parameter from the
          template instance
          ParameterValue parameter =
          templateInstance
            .getParameterValue("par
            am1");

          // Set an incorrect value for this
          parameter
          // This will cause a validation
          error
          parameter.setValue("$3500");
          out.println("Incorrect parameter
          value set");
          break;
        }
      }
    }
    // This code should never be reached because of the
    error
    // introduced
    // above

    // With the condition value and actions updated, the
    business
    // rule
    // group can be published.
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the list
    publishList.add(brg);

    // Publish the list with the updated business rule
    group
    BusinessRuleManager.publish(publishList, true);
  }
```

A `ValidationException` can be caught and from the exception, the problems can be retrieved. For each problem, the error can be checked to determine which error has occurred. A message can be printed out or the appropriate action can be taken.

```
  } catch (ValidationException e) {
  out.println("Validation Error");

  List<Problem> problems = e.getProblems();

  Iterator<Problem> problemIterator = problems.iterator();

  // Check the list of problems for the appropriate error and
  // perform the appropriate action (report error, correct
  // error, etc.)
  while (problemIterator.hasNext()) {
    Problem problem = problemIterator.next();
    ValidationError error = problem.getErrorType();

    // Check for specific error value
    if (error == ValidationError.TYPE_CONVERSION_ERROR) {
      // Handle this error by reporting the problem
      out
        .println("Problem: Incorrect value
        entered for a parameter");
```

```
      return out.toString();
    }
    // else if....
    // Checks can be done for other errors and the
    // appropriate error message or action can be
    performed
    // correct the problem
  }
} catch (BusinessRuleManagementException e) {
  out.println("Error occurred.");
  e.printStackTrace();
}
return out.toString();
}
}
```

Example

Web browser output for example 14.

```
Executing example14

Business Rule Group retrieved
Rule Set retrieved
Validation Error
Problem: Incorrect value entered for a parameter
```

Example 15: Handle errors in a business rule group

This example is similar to example 14 as it shows how to handle problems that occur when a business rule group is published. It shows how the problem can be determined and the correct message can be printed or action performed.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example15
{
static Formatter out = new Formatter();

static public String executeExample15()
{
 try
 {
  out.clear();
```

```
// Retrieve a business rule group by target namespace and
name
List<BusinessRuleGroup> brgList = BusinessRuleManager
 .getBRGsByTNSAndName(
 "http://BRSamples/com/ibm/websphere
 /sample/brules",
 QueryOperator.EQUAL,
 "ApprovalValues",
 QueryOperator.EQUAL, 0, 0);
 if (brgList.size() > 0)
 {
  // Get the first business rule group from the list
  // This should be the only business rule group in the
  list as
  // the combination of target namespace and name are
  unique
  BusinessRuleGroup brg = brgList.get(0);
  out.println("Business Rule Group retrieved");

  // Get the operation of the business rule group that
  // has the business rule that will be modified as
  // the business rules are associated with a specific
  // operation
  Operation op = brg.getOperation("getApprover");

  // Retrieve specific rule by name
  List<BusinessRule> ruleList =
  op.getBusinessRulesByName(
   "getApprover", QueryOperator.EQUAL, 0,
   0);

  // Get the specific rule
  RuleSet ruleSet = (RuleSet) ruleList.get(0);
   out.println("Rule Set retrieved");

  RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

  Iterator<RuleSetRule> ruleIterator =
   ruleBlock.iterator();

  // Search through the rules to find the rule to
  change
  while (ruleIterator.hasNext())
  {
   RuleSetRule rule = ruleIterator.next();

   // Check that the rule was defined with a
   template
   // as it can be changed.
   if (rule instanceof
   RuleSetTemplateInstanceRule)
   {
    // Get the template rule instance
    RuleSetTemplateInstanceRule
    templateInstance =
    (RuleSetTemplateInstanceRule) rule;

    // Check for the correct template rule
    instance
    if (templateInstance.getName().equals(
      "LargeOrderApprover"))
    {
     // Get the parameter from the
     template instance
     ParameterValue parameter =
     templateInstance
```

```
 .getParameterValue("par
 am1");

// Set the value for this parameter
// This value is in the correct
format and will
// not cause a validation error
parameter.setValue("4000");
out.println("Rule set parameter
value on set correctly");
break;
}
}
}
```

To insure a rule set is correct, the validate method can be called. The validate method is available on all objects and will return a list of problems that can be checked to determine the problem. When calling validate on an object, the validate method is called on all contained objects as well.

```
// Validate the changes made the rule set
List<Problem> problems = ruleSet.validate();
out.println("Rule set validated");

// No errors should occur for this test case, however,
// check if there are problems and then
// perform the correct action to recover or report
// the error
if (problems != null)
{
 Iterator<Problem> problemIterator =
 problems.iterator();

 while (problemIterator.hasNext())
 {
  Problem problem = problemIterator.next();

  if (problem instanceof
  ProblemStartDateAfterEndDate)
  {
   out
   .println("Incorrect
    value entered for a
     parameter");
   return out.toString();
  }
 }
} else
{
 out.println("No problems found for the rule
 set");
}
// Get the list of available rule targets
List<BusinessRule> ruleList2 =
op.getAvailableTargets();

// Get the first rule that will be scheduled
incorrectly
BusinessRule rule = ruleList2.get(0);

// The error condition will be to set the end time
for a
// scheduled rule to be 1 hour before the start time
// This will cause a validation error
Date future = new Date();
long futureTime = future.getTime() - 360000;
```

```
      // Get the operation selection list to add the
      incorrectly
      // scheduled item
      OperationSelectionRecordList opList = op
      .getOperationSelectionRecordList();

      // Create a new scheduled rule instance
      // No error is thrown until validated or a publish
      // occurs as more changes might be made
      OperationSelectionRecord newRecord = opList
       .newOperationSelectionRecord(new Date(),
       new Date(
       futureTime), rule);
```

When the record is added with an incorrect set of dates, this does not cause an error. It is possible overlaps might occur or no selection records are set for the operation as things are in the process of being changed. The error will be found when the business rule group with the operation selection record is published. The validate method is called before the objects are published and exceptions will be thrown if any errors exists.

```
      // Add the scheduled rule instance to the operation
      // No error here either
      opList.addOperationSelectionRecord(newRecord);
      out.println("New selection record added with
      incorrect schedule");

      // With the condition value and actions updated, the
      business
      // rule
      // group can be published.
      // Use the original list or create a new list
      // of business rule groups
      List<BusinessRuleGroup> publishList = new
      ArrayList<BusinessRuleGroup>();

      // Add the changed business rule group to the list
      publishList.add(brg);

      // Publish the list with the updated business rule
      group
      BusinessRuleManager.publish(publishList, true);
     }
   } catch (ValidationException e) {
   out.println("Validation Error");

   List<Problem> problems = e.getProblems();

   Iterator<Problem> problemIterator = problems.iterator();
   // There might be multiple problems
   // Go through the problems and handle each one or
   // report the problem
   while (problemIterator.hasNext())
   {
    Problem problem = problemIterator.next();

    // Each problem is a different type that can be
    compared
    if (problem instanceof ProblemStartDateAfterEndDate)
    {
     out
     .println("Rule schedule is
      incorrect. Start date is after end
      date.");
     return out.toString();
     }
```

```
    // else if....
    // Checks can be done for other errors and the
    // appropriate error message or action can be
    performed
    // correct the problem
  }
 }catch (BusinessRuleManagementException e)
 {
  out.println("Error occurred.");
  e.printStackTrace();
 }
 return out.toString();
 }
}
```

Example

Web browser output for example 15.

```
Executing example15

Business Rule Group retrieved
Rule Set retrieved
Rule set parameter value on set correctly
Rule set validated
Validation Error
Rule schedule is incorrect. Start date is after end date.
```

Additional Query Examples

The following examples are not included with the application containing examples 1-15, however they provide more examples on creating queries to retrieve business rule groups.

In these examples different properties and wildcard values ('_', '%') are used with different operators (AND, OR, LIKE, NOT_LIKE, EQUAL and NOT_EQUAL).

Example

For the examples, queries will be performed that return between different combinations of 4 business rule groups. It is important to understand the different attributes and properties of the business rule groups as these are used in the queries.

```
Name: BRG1
Targetnamespace : http://BRG1/com/ibm/br/rulegroup
Properties:
organization, 8JAA
department, claims
ID, 00000567
region: SouthCentralRegion
manager: Joe Bean

Name: BRG2
Targetnamespace : http://BRG2/com/ibm/br/rulegroup
Properties:
organization, 7GAA
department, accounting
ID, 0000047
ID_cert45, ABC
region: NorthRegion

Name: BRG3
Targetnamespace : http://BRG3/com/ibm/br/rulegroup
Properties:
organization, 7FAB
```

```
department, finance
ID, 0000053
ID_app45, DEF
region: NorthCentralRegion

Name: BRG4
Targetnamespace : http://BRG4/com/ibm/br/rulegroup
Properties:
organization, 7HAA
department, shipping
ID, 0000023
ID_app45, GHI
region: SouthRegion
```

Query by a single property:

This is an example of a query by a single property.

```
List<BusinessRuleGroup> brgList = null;

brgList = BusinessRuleManager.getBRGsBySingleProperty(
  "department", QueryOperator.EQUAL,
  "accounting", 0, 0);
// Returns BRG2
```

Query business rule groups by properties and wildcard (%) at the beginning and at the end of the value:

This is an example of a query of business rule groups by properties and wildcard (%) at the beginning and at the end of the value.

```
// Query Prop AND Prop
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "region", QueryOperator.LIKE,
  "%Region");

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode(
  "ID", QueryOperator.LIKE,
  "000005%");

QueryNode queryNode =
QueryNodeFactory.createAndNode(leftNode,
  rightNode);

brgList =
BusinessRuleManager.getBRGsByProperties(queryNode, 0, 0);
// Returns BRG1 and BRG3
```

Query business rule groups by properties and wildcard ('_'):

This is an example of a query of business rule groups by properties and wildcard (%).

```
brgList = BusinessRuleManager.getBRGsBySingleProperty("ID",
QueryOperator.LIKE, "00000_3", 0, 0);

// Returns BRG3 and BRG4
```

Query business rule group by properties with multiple wildcards ('_' and '%'):

This is an example of a query of business rule group by properties with multiple wildcards ('_' and '%').

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("region",
QueryOperator.LIKE, "__uth%Region",
  0, 0);

// Returns BRG1 and BRG4
```

Query business rule groups by NOT_LIKE operator and wildcard ('_'):

This is an example of a query of business rule group by NOT_LIKE operator and wildcard ('_').

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7__A", 0, 0);

// Returns BRG1 and BRG3
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7%", 0, 0);

// Returns BRG1
```

Query business rule groups by NOT_EQUAL operator:

This is an example of a query of business rule group by the NOT_EQUAL operator.

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("department",
QueryOperator.NOT_EQUAL,
  "claims", 0, 0);
// Returns BRG1
```

Query business rule groups by PropertyIsDefined:

This is an example of a query of business rule groups by PropertyIsDefined.

```
PropertyIsDefinedQueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

brgList = BusinessRuleManager.getBRGsByProperties(node, 0,
0);

// Returns BRG1
```

Query business rule groups by NOT PropertyIsDefined:

This is an example of a query of business rule groups by NOT PropertyIsDefined.

```
// NOT Prop
QueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

NotNode notNode = QueryNodeFactory.createNotNode(node);

brgList = BusinessRuleManager.getBRGsByProperties(notNode,
0, 0);

// Returns BRG1
```

Query business rule groups by multiple properties with a single NOT node:

This is an example of a query of business rule groups by multiple properties with a single NOT node.

```
// Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
  QueryOperator.LIKE, "00000%");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
  notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Query business rule groups by multiple properties with multiple NOT nodes combined with AND operator:

This is an example of a query business rule groups by multiple properties with multiple NOT nodes combined with AND operator.

```
// NOT Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "cla%");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

AndNode andNode = QueryNodeFactory.createAndNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG1 and BRG2
```

Query business rule groups by multiple properties with multiple NOT nodes combined with OR operator:

This is an example of a query business rule groups by multiple properties with multiple NOT nodes combined with OR operator.

```
// NOT Prop OR NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);
```

```
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department", QueryOperator.EQUAL,
  "claims");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

Query business rule groups by multiple properties combined with multiple AND operators:

This is an example of a query business rule groups by multiple properties combined with multiple AND operators.

```
// (Prop AND Prop) AND (Prop AND Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode,rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE,"000004_");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
QueryOperator.EQUAL,
  "NorthRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft,andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Query business rule groups by multiple properties combined with AND and OR operators:

This is an example of a query business rule groups by multiple properties combined with AND and OR operators.

```
// (Prop AND Prop) OR (Prop AND NOT Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
```

```
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode, rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "8JAA");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE, "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, notNode);

OrNode orNode = QueryNodeFactory.createOrNode(andNodeLeft,
andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG2 and BRG3
```

Query business rule groups by multiple properties combined with AND and NOT operators:

This is an example of a query business rule groups multiple properties combined with AND and NOT operators.

```
// Prop AND NOT (Prop AND Prop)
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE, "000005%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.EQUAL,
  "8JAA");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG3
```

Query business rule groups by multiple properties combined with NOT and OR operators:

This is an example of a query business rule groups by multiple properties combined with NOT and OR operators.

```
// NOT (Prop AND Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8_A_");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
rightNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

Query business rule groups by multiple properties combined with nested AND operators:

This is an example of a query business rule groups by multiple properties combined with nested AND operators.

```
// Prop AND (Prop AND (Prop AND Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

PropertyIsDefinedQueryNode node2 =
QueryNodeFactory.createPropertyIsDefinedQueryNode("ID_cert4
5");

AndNode andNode = QueryNodeFactory.createAndNode(node2,
andNodeLeft);
```

```
brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);
// Returns BRG2
```

Query business rule groups by multiple properties combined with nested AND operators:

This is an example of a query business rule groups by multiple properties combined with nested AND operators.

```
// (Prop AND (Prop AND Prop)) AND Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
"___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.LIKE,
"7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE,
"%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_app45",QueryOp
erator.LIKE, "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

Query business rule groups by multiple properties combined with nested AND operators and a NOT node:

This is an example of a query business rule groups by multiple properties combined with nested AND operators and a NOT node.

```
// Prop AND (Prop AND (Prop AND NOT Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
```

```
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,notNode);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "AB_");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Query business rule groups by multiple properties combined with nested AND operators:

This is an example of a query business rule groups by multiple properties combined with nested AND operators.

```
// (Prop AND (Prop AND Prop)) AND Prop - Return empty
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

//Returns no BRGs
```

Query business rule groups by multiple properties combined with nested OR operators:

This is an example of a query business rule groups by multiple properties combined with nested OR operators.

```
// (Prop OR (Prop OR Prop)) OR Prop

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,rightNode2);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG1
```

Query business rule groups by multiple properties combined with nested OR operators:

This is an example of a query business rule groups by multiple properties combined with nested OR operators.

```
// (Prop OR (Prop OR NOT Prop)) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,notNode);
```

```
OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

Query business rule groups by multiple properties combined with nested OR operators and a NOT node:

This is an example of a query business rule groups by multiple properties combined with nested OR operators and a NOT node.

```
// Prop OR NOT(Prop OR Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,
  rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft = QueryNodeFactory.createOrNode(leftNode,
notNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG3
```

Query business rule groups by multiple properties combined with nested OR operators and a NOT node:

This is an example of a query business rule groups by multiple properties combined with nested OR operators and a NOT node.

```
// NOT(Prop OR Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");
```

```
QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(notNode,leftNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG2 and BRG4
```

Query business rule groups by a list of nodes that are combined with an AND operator:

This is an example of a query business rule groups by a list of nodes that are combined with an AND operator.

```
// AND list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7H%");

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);
```

```
brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

Query business rule groups by a list of nodes and NOT node combined with an AND operator:

This is an example of a query business rule groups by a list of nodes and NOT node combined with an AND operator.

```
// AND list with a notNode
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

NotNode notNode =

QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Return BRG4
```

Query business rule groups by a list of nodes that are combined with an OR operator:

This is an example of a query business rule groups by a list of nodes that are combined with an OR operator.

```
// OR list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);
```

```
QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG3
```

Query business rule groups by a list of nodes and Not node combined with an OR operator:

This is an example of a query business rule groups by a list of nodes and Not node combined with an OR operator.

```
// OR list with Not node
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

list.add(leftNode2);

OrNode orNode = QueryNodeFactory.createOrNode(list);
```

```
brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

Common operations classes

This section contains additional classes that were used in the examples to simplify common operations.

Formatter class

This class provides different methods to help with displaying the different examples. It adds different HTML tags to format the output.

```
package com.ibm.websphere.sample.brules.mgmt;

public class Formatter {

private StringBuffer buffer;

public Formatter()
{
  buffer = new StringBuffer();
}

public void println(Object o)
{
  buffer.append(o);
  buffer.append("<br>\n");
  }

  public void print(Object o)
  {
    buffer.append(o);
  }

  public void printlnBold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b><brbr>\n");
  }

  public void printBold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b>");
  }

  public String toString()
  {
    return buffer.toString();
  }

  public void clear()
  {
    buffer = new StringBuffer();
  }
}
```

RuleArtifactUtility class

This utility class has two public methods. The first public method is for printing out a decision table. This method makes use of a private method that uses recursion to print out the conditions and actions for the decision table. The second public method is for printing out a rule set.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.RuleTemplate;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableRule;
import
com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionTermDefinition;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class RuleArtifactUtility
{
  static Formatter out = new Formatter();

  /*
  Method to print out a decision table with the conditions and
  actions printed out in a HTML tabular format. The conditions
  and actions are printed out with a separate method that
  recursively works through the case edges of the decision
  tables.
  */

  public static String printDecisionTable(BusinessRule
  ruleArtifact)
  {
    out.clear();
    out.printlnBold("Decision Table");
    DecisionTable decisionTable = (DecisionTable)
    ruleArtifact;
    out.println("Name: " +
    decisionTable.getName());
    out.println("Namespace: " +
    decisionTable.getTargetNameSpace());

    // Output the init rule for the decision table
    before
    // working through the table of conditions and
    actions
    DecisionTableRule initRule =
    decisionTable.getInitRule();
    if (initRule != null)
    {
      out.printBold("Init Rule: ");
      out.println(initRule.getName());
```

```
out.println("Display Name: " +
initRule.getDisplayName());
out.println("Description: " +
initRule.getDescription());
// The expanded user presentation
will automatically populate the
// presentation with the parameter
values and can be used for
// display if the init rule was
defined with a template. If no
// template was defined the
expanded user presentation
// is the same as the regular
presentation.
out.println("Extended User
Presentation: "
  +
initRule.getExpandedUse
rPresentation());
// The regular user presentation
will have placeholders in the
// string where the
// parameter can be substituted if
the init rule was defined with a
// template
// If the rule was not defined with
a template, the user
// presentation will only
// be a string without
placeholders. The placeholders are
of a
// format of {n} where
// n is the index (zero-based) of
the parameter in the template. This
// value
// can be used to create an
interface for editing where there
are
// fields with
// the parameter values available
for editing
out.println("User Presentation: " +
initRule.getUserPresentation());
// Init rules might be defined with
or without a template
// Check to make sure a template
was used before trying
// to access the parameters
if (initRule instanceof
DecisionTableTemplateInstanceRule)
{
  DecisionTableTemplateIn
  stanceRule
  templateInstance =
  (DecisionTableTemplateI
  nstanceRule) initRule;

  RuleTemplate template =
  templateInstance.getRul
  eTemplate();

  List<Parameter>
  parameters =
  template.getParameters(
  );
  Iterator<Parameter>
  paramIterator =
```

```
            parameters.iterator();

            Parameter parameter =
            null;

            while
            (paramIterator.hasNext(
            )) {
            parameter =
            paramIterator.next();

             out.println("Parameter
            Name: " +
            parameter.getName());
            out.println("Parameter
            Value: "
            +
            templateInstance.getPar
            ameterValue(parameter
            .getName()));
            }
          }
        }
        // For the rest of the decision table, start at
        the root and
        // recursively work through the different case
        edges and
        // actions
        TreeBlock treeBlock =
        decisionTable.getTreeBlock();
        TreeNode treeNode = treeBlock.getRootNode();

        printDecisionTableConditionsAndActions(treeNode
        , 0);
        out.println("");
        return out.toString();
      }
      /*Method to recursively work through the case edges and print
      out the conditions and actions.
    */
      static private void printDecisionTableConditionsAndActions(
        TreeNode treeNode, int indent)
      {
        out.print("<table border=\"1\">");
        if (treeNode instanceof ConditionNode)
        {
          // Get the case edges for the
          current TreeNode
          // and for each case edge print out
          the conditions
          ConditionNode conditionNode =
          (ConditionNode) treeNode;

          List<CaseEdge> caseEdges =
          conditionNode.getCaseEdges();
          Iterator<CaseEdge> caseEdgeIterator
          = caseEdges.iterator();

          CaseEdge caseEdge = null;

          while (caseEdgeIterator.hasNext())
          {
            out.print("<tr>");
            // If this is the start
            of the conditions for the
            // condition node,
            print out the condition term
```

```
if (indent == 0)
{
out.print("<td>");

TreeConditionTermDefinition
termDefinition =
conditionNode
.getTermDefinition();

out.print(termDefinitio
n.getUserPresentation()
);
out.print("</td>");
indent++;
} else {
// After the condition
term has been printed
for a
// case edge skip for
the rest of the case
edges
out.print("<td></td>");
}

caseEdge =
caseEdgeIterator.next()
;

out.print("<td>");

// Check if the
caseEdge is defined by
a template
if
(caseEdge.getValueDefin
ition() != null)
{
TemplateInstanceExpress
ion templateInstance =
caseEdge
.getValueTemplateInstan
ce();

out.println(templateIns
tance.getExpandedUserPr
esentation());

TreeConditionValueDefin
ition valueDef =
caseEdge
.getValueDefinition();

out.println(valueDef.ge
tUserPresentation());

Template template =
templateInstance.getTem
plate();

// Get the parameters
for the template
definition and
// print out the
parameter names and
values
List<Parameter>
parameters =
```

```
template.getParameters(
);
Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =
templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
(paramIterator.hasNext(
) &&
paramValues.hasNext())
{
parameter =
paramIterator.next();
parameterValue =
paramValues.next();

out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
  +
parameterValue.getValue
());
}
}

out.print("</td><td>");
// Print the child node
for the caseEdge
printDecisionTableCondi
tionsAndActions(caseEdg
e.getChildNode(),
0);

out.print("</td></tr>")
;
}


// Add Otherwise condition if it
exists
TreeNode otherwise =
conditionNode.getOtherwiseCase();

if (otherwise != null)
{
  out.print("<tr><td></td>
  <td>Otherwise</td><td>
  ");
  // Print the Otherwise
  ConditionNode
  printDecisionTableCondi
```

```
      tionsAndActions(otherwi
      se, 0);
      out.print("</td></td>")
      ;
    }
    out.print("</table>");
  } else {
    // ActionNode has been found and
    different logic is needed
    // to print out the TreeActions
    ActionNode actionNode =
    (ActionNode) treeNode;
    List<TreeAction> treeActions =
    actionNode.getTreeActions();

    Iterator<TreeAction>
    treeActionIterator =
    treeActions.iterator();

    TreeAction treeAction = null;

    // The ActionNode can contain
    multiple TreeActions to
    // print out
    while
    (treeActionIterator.hasNext())
    {
      out.print("<tr>");
      treeAction =
      treeActionIterator.next
      ();

      TreeActionTermDefinitio
      n treeActionTerm =
      treeAction
      .getTermDefinition();

      if (indent == 0) {
      out.print("<td>");
      out.print(treeActionTer
      m.getUserPresentation()
      );
      out.print("</td>");
      }
      out.print("<td>");
      TemplateInstanceExpress
      ion templateInstance =
      treeAction
      .getValueTemplateInstan
      ce();

      // Check that a
      template was specified
      for
      // the TreeAction
      before working with the
      // parameter name and
      values
      if (templateInstance !=
      null) {
      out.println(templateIns
      tance.getExpandedUserPr
      esentation());

      Template template =
      templateInstance.getTem
      plate();
```

```
        List<Parameter>
        parameters =
        template.getParameters(
        );

        Iterator<Parameter>
        paramIterator =
        parameters.iterator();

        List<ParameterValue>
        parameterValues =
        templateInstance
        .getParameterValues();
        Iterator<ParameterValue
        > paramValues =
        parameterValues
        .iterator();

        Parameter parameter =
        null;
        ParameterValue
        parameterValue = null;

        while
        (paramIterator.hasNext(
        ) &&
        paramValues.hasNext())
        {
        {parameter =
        paramIterator.next();
        parameterValue =
        paramValues.next();

        out.println(" Parameter
        Name: " +
        parameter.getName());
        out.println(" Parameter
        Value: "
        +
        parameterValue.getValue
        ());

        }
        } else
        {
        // If a template was
        not used, the only item
        that is
        // available is the
        UserPresentation if it
        was
        // specified when the
        rule was created
        out.print(treeAction.ge
        tValueUserPresentation(
        ));
        }

        out.print("</td></tr>")
        ;
      }
      out.print("</table>");
    }
  }
  /*
    Method to print out a rule set
```

```
  */
public static String printRuleSet(BusinessRule
ruleArtifact)
{
   out.clear();
   out.printlnBold("Rule Set");
   RuleSet ruleSet = (RuleSet) ruleArtifact;
   out.println("Name: " + ruleSet.getName());
   out.println("Namespace: " +
   ruleSet.getTargetNameSpace());

   // The rules in a rule set are contained in a
   rule block
   RuleBlock ruleBlock =
   ruleSet.getFirstRuleBlock();

   Iterator<RuleSetRule> ruleIterator =
   ruleBlock.iterator();

   RuleSetRule rule = null;

   // Iterate through the rules in the rule block.
   while (ruleIterator.hasNext())
   {
     rule = ruleIterator.next();
     out.printBold("Rule: ");
     out.println(rule.getName());
     out.println("Display Name: " +
     rule.getDisplayName());
     out.println("Description: " +
     rule.getDescription());
     // The expanded user presentation
     will automatically populate the
     // presentation with the parameter
     values and can be used for
     // display if the rule was defined
     with a template. If no
     // template was defined the
     expanded user presentation
     // is the same as the regular
     presentation.
     out.println("Expanded User
     Presentation: "
       +
     rule.getExpandedUserPre
     sentation());
     // The regular user presentation
     will have placeholders in the
     // string where the parameter can
     be substituted if the rule
     // was defined with a template. If
     the rule was not defined with
     // a template, the user
     presentation will only be a string
     // without placeholders. The
     placeholders are of a format of {n}
     // where n is the index (zerobased)
     of the parameter in the
     // template. This value can be used
     to create an interface for
     // editing where there are fields
     with the parameter values
     // available for editing
     out.println("User Presentation: " +
     rule.getUserPresentation());

     // Check if the rule was defined
```

```
        with a template
        if (rule instanceof
        RuleSetTemplateInstanceRule) {
          RuleSetTemplateInstance
          Rule templateInstance =
          (RuleSetTemplateInstanc
          eRule) rule;

          RuleSetRuleTemplate
          template =
          templateInstance
          .getRuleSetRuleTemplate
          ();

          List<Parameter>
          parameters =
          template.getParameters(
          );
          Iterator<Parameter>
          paramIterator =
          parameters.iterator();

          Parameter parameter =
          null;

          // Retrieve all of the
          parameters and output
          the name and value
          while
          (paramIterator.hasNext(
          ))
          {
          parameter =
          paramIterator.next();

          out.println("Parameter
          Name: " +
          parameter.getName());
          out.println("Parameter
          Value: "
            +
          templateInstance.getPar
          ameterValue(
          parameter.getName()).ge
          tValue());
          }
        }
      }
      out.println("");
      return out.toString();
    }
}
```

Chapter 4. Developing client applications for business processes and tasks

You can use a modeling tool to build and deploy business processes and tasks. These processes and tasks are interacted with at runtime, for example, a process is started, or tasks are claimed and completed. You can use Business Process Choreographer Explorer to interact with processes and tasks, or the Business Process Choreographer APIs to develop customized clients for these interactions.

About this task

These clients can be Enterprise JavaBeans™ (EJB) clients, Web service clients, or Web clients that exploit the Business Process Choreographer Explorer JavaServer Faces (JSF) components. Business Process Choreographer provides Enterprise JavaBeans (EJB) APIs and interfaces for Web services for you to develop these clients. The EJB API can be accessed by any Java application, including another EJB application. The interfaces for Web services can be accessed from either Java environments or Microsoft® .Net environments.

Comparison of the programming interfaces for interacting with business processes and human tasks

Enterprise JavaBeans (EJB), Web service, and Java Message Service (JMS), and Representational State Transfer Services (REST) generic programming interfaces are available for building client applications that interact with business processes and human tasks. Each of these interfaces has different characteristics.

The programming interface that you choose depends on several factors, including the functionality that your client application must provide, whether you have an existing end-user client infrastructure, whether you want to handle human workflows. To help you decide which interface to use, the following table compares the characteristics of the EJB, Web service, JMS, and REST programming interfaces.

	EJB interface	Web service interface	JMS message interface	REST interface
Functionality	This interface is available for both business processes and human tasks. Use this interface to build clients that work generically with processes and tasks.	This interface is available for both business processes and human tasks. Use this interface to build clients for a known set of processes and tasks.	This interface is available for business processes only. Use this interface to build messaging clients for a known set of processes.	This interface is available for both business processes and human tasks. Use this interface to build Web 2.0-style clients for a known set of processes and tasks.

© IBM Corporation 2005, 2006 © IBM 2005, 2009

	EJB interface	Web service interface	JMS message interface	REST interface
Data handling	Supports remote artifact loading of schemas for accessing business object metadata. If the EJB client application is running in the same cell as the WebSphere Process Server that it connects to, the schemas that are needed for the business objects of the processes and tasks do not have to be available on the client, they can be loaded from the server using the remote artifact loader (RAL). RAL can also be used cross-cell if the client application runs in a full WebSphere Process Server server installation. However, RAL cannot be used in a cross-cell setup where the client application runs in a WebSphere Process Server client installation.	Schema artifacts for input data, output data, and variables, must be available in an appropriate format on the client.	Schema artifacts for input data, output data, and variables, must be available in an appropriate format on the client.	Schema artifacts for input data, output data, and variables, must be available in an appropriate format on the client.
Client environment	A WebSphere Process Server installation or a WebSphere Process Server client installation.	Any runtime environment that supports Web service calls, including Microsoft .NET environments.	Any runtime environment that supports JMS clients, including SCA modules that use SCA JMS imports.	Any runtime environment that supports REST clients.
Security	Java 2, Enterprise Edition (J2EE) security.	Web services security.	Java 2, Enterprise Edition (J2EE) security for the WebSphere Process Server installation. You can also secure the queues where the JMS client application puts the API messages, for example, using WebSphere MQ security mechanisms.	Client application that call the REST methods must use an appropriate HTTP authentication mechanism.

An operation can be exposed by multiple protocols. Observe the following general considerations if you use the same operation in different protocols.

- In Web service and REST interfaces, all object identifiers, such as PIID, AIID, and TKIID are represented by a string type. Only the EJB API interface expects a type-safe object ID.
- Operation overloading is only used for EJB methods and not for WSDL operations. In some cases, multiple WSDL operations exist, in other cases, only one WSDL operation exists that allows all of the parameter variations either by omission (`minOccurs="0"`), or null values (`nillable="true"`).
- In some EJB methods, XML namespaces and local names are passed as separate parameters. Most WSDL operations use the QName XML schema type to pass these parameters.
- Asynchronous interactions with long-running WSDL request-response operations, such as the callWithReplyContext operation in the EJB interface or the callAsync operation in the WSDL interface, are represented by the call operation in the JMS interface.

- The EJB interface returns a set of API objects, which expose getter and setter methods for the contained fields. Web service and REST interfaces return complex-typed (XML or JSON) documents to the client.
- Some Human Task Manager services operating on human tasks are also available as Business Flow Manager services operating on activities that call a human task.
-

Related tasks

Developing EJB client applications for business processes and human tasks
The EJB APIs provide a set of generic methods for developing EJB client applications for working with the business processes and human tasks that are installed on a WebSphere Process Server.

Developing Web service API client applications
You can develop client applications that access business process applications and human task applications through Web services APIs.

Developing client applications using the Business Process Choreographer JMS API
You can develop client applications that access business process applications asynchronously through the Java Messaging Service (JMS) API.

Queries on business process and task data

Instance data for long-running business processes and human tasks are stored persistently in the database and are accessible by queries. Also, template data for business process templates and human task templates can be accessed using a query interface.

The EJB query interfaces, query API, and query table API, are available with Business Process Choreographer.

Depending on the clients that access process or task related data, one or more of the interfaces can be the right choice. REST and Web services APIs are available in Business Process Choreographer for querying task and process list data. However, for high volume process list and task list queries, use the Business Process Choreographer EJB query table API or REST query table API for performance reasons.

Comparison of the programming interfaces for retrieving process and task data

Business Process Choreographer provides a query table API and a query API for retrieving process and task data. Each of these interfaces has different characteristics.

The query interface that you choose depends on several factors, including the functionality that your client application must provide, whether you have an existing end-user client infrastructure, and performance considerations. To help you decide which interface to use, the following table compares the characteristics of the query table and the query programming interfaces.

Characteristic	query table API	query API
Availability	The query table API is available for the Business Flow Manager EJB interface and the REST programming interface.	The query API is available for EJB, Web service, JMS, and REST programming interfaces.

Characteristic	query table API	query API
Methods for content retrieval	The API provides the following methods: • queryEntities • queryEntityCount • queryRows • queryRowCount	The API provides the following methods: • query • queryAll
Methods for meta data retrieval	The API provides the following methods: • getQueryTableMetaData • findQueryTableMetaData • queryProcessTemplates • queryTaskTemplates	
Query table name	Specifies the query table on which the query table API is runs. Only one query table can be queried at any one time. For example, `queryEntities("CUST.TASKS", ...)`.	The SELECT clause specifies the columns and predefined database views on which the query runs. This specification is similar to an SQL select clause. For example, `query("TASK.TKIID, TASK.STATE, WORK_ITEM.REASON", ...)`.
SELECT clause and selected attributes	Use the filter options of the query table API to specify the attributes that the query is to return. Because the query is run against one query table, the attributes are uniquely identifiable by their names.	Use the SELECT clause to specify attributes. The syntax of the attribute name is: *view_name.attribute_name*. For example, to search for task states, specify TASK.STATE in your query.
WHERE clause and filters	Use the queryCondition property on the query table API to further filter the result of queries. Query tables provide pre-filtered content if primary query table filters, authorization filters, or query table filters have been specified on the query table definition.	Use the WHERE clause to filter queries.
WHERE clause and selection criteria	The WHERE clause of the query API is not needed in this form on the query table API. Use the queryCondition property on the query table API for additional filtering. Selection criteria in the query table definition select a particular property of the attached query table. This is achieved in addition to the filtering by the WHERE clause on the query API.	Selection criteria are not available for the query API. However, selection criteria are similar to the part of the WHERE clause that defines, for example, the name or locale of QUERY_PROPERTY, or TASK_CPROP, or TASK_DESC. For example, a WHERE clause of `QUERY_PROPERTY.NAME='xyz'` is the same as specifying `NAME='xyz'` as a selection criterion on the query table definition for the QUERY_PROPERTY attached query table.
Work items and authorization	Use the WORK_ITEM query table to access work items. You can customize the use of work items on the query table definition when the query table is developed and on the query table API, using the AuthorizationOptions object or the AdminAuthorizationOptions object. For example, to exclude everybody work items when querying the TASK query table, specify a queryCondition property `WI.EVERYBODY=0` or specify `setUseEverbody(Boolean.FALSE)` on the AuthorizationOptions property.	Use the WORK_ITEM view to access work items. All four types of work items are considered for the query result: everybody, individual, groups, and inherited work items. To filter the work items for a specific type of work item, customize the WHERE clause. For example, to exclude the everybody work items, specify `WORK_ITEM.EVERYBODY=0`, in the WHERE clause.
Parameters	You can use parameters in filters and selection criteria for composite query tables.	Parameters are not available for the query API unless stored queries are used.

Characteristic	query table API	query API
Stored queries and query tables	The difference between a stored query and a query table is that stored queries are defined for one particular query, while a query table is defined for a particular set of queries. For example, the query table definition does not allow the specification of an order-by clause because this information is typically available only when the query is run.	You can use stored queries to run query that contains a predefined set of options.
Materialized views	Materialized views are not available for the query table API.	Materialized views use database technologies to provide performance improvements for queries.
Custom tables	Supplemental query tables offer the same functionality as custom tables.	Custom tables are used to include data in queries that is external to the Business Process Choreographer database schema.
queryAll and authorization options	The queryAll functionality is provided by the AdminAuthorizationOptions object, which can be passed to the query table API instead of the AuthorizationOptions object. The caller must be in the BPESystemAdministrator J2EE role.	The queryAll method which can be used by users that have the BPESystemAdministrator J2EE role to return all of the objects in the query result without being restricted by work items for a particular user or group.
Internationalization	For attributes of query tables and for the query table, localized display names and descriptions are available when query tables are used.	Names of the columns of the selected views, as they appear in the database, are returned.

Query tables in Business Process Choreographer

Query tables support task and process list queries on data that is contained in the Business Process Choreographer database schema. This includes human task data and business process data that is managed by Business Process Choreographer, and external business data. Query tables provide an abstraction on the data of Business Process Choreographer that can be used by client applications. In this way, client applications become independent of the actual implementation of the query table. Query table definitions are deployed on Business Process Choreographer containers, and are accessible using the query table API.

There are three types of query tables: predefined query tables, supplemental query tables, and composite query tables.

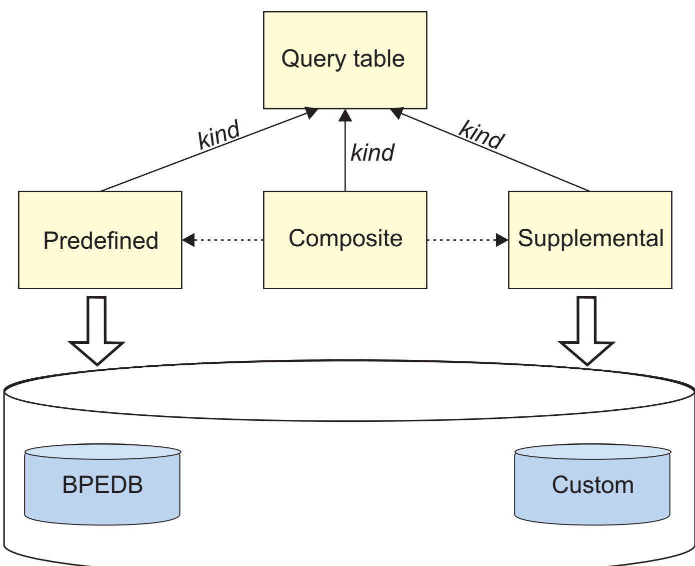


Figure 23. Query tables in Business Process Choreographer

Query tables are represented using similar models in the query table runtime, and can be queried using the query table API. While predefined and supplemental query tables point directly to tables or views in the database, composite query tables compose parts of this data and make it available in a single query table.

Query tables enhance the predefined database views and the existing query interfaces of Business Process Choreographer. Query tables:
- Are optimized for running process and task list queries, using performance optimized access patterns.
- Simplify and consolidate access to the information needed.
- Allow for the fine-grained configuration of authorization and filter options.

Query tables can be customized, for example, configuration options can determine that a query table contains only those tasks or process instances that are relevant in a particular scenario. Where performance is important, such as with high volume process list and task list queries, use query tables.

The Query Table Builder is provided as an Eclipse plug-in to:
- Develop composite and supplemental query tables
- Import and export query table definitions in XML format

The Query Table Builder can be downloaded on the WebSphere Business Process Management SupportPacs site. Look for PA71 WebSphere Process Server - Query Table Builder. To access the link, see the related references section of this topic.

Related tasks

Administering query tables
Use the wsadmin script, manageQueryTable.py script to administer query tables in Business Process Choreographer, which were developed using the Query Table Builder. Unlike predefined query tables, which are available out-of-the-box, you must deploy composite and supplemental query tables on WebSphere Process Server before you can use them with the query table API.

Deploying query tables
Use the manageQueryTable.py script to deploy supplemental and composite query tables in Business Process Choreographer. Query tables are deployed on a stand-alone server that is running or in a cluster with at least one member running. The undeployment of supplemental and composite query tables is performed also on running servers. For supplemental query tables, the related physical database objects, typically a database view or database table, must be created if they do not exist prior to the usage of the query table.

Related reference

Database views for Business Process Choreographer
This reference information describes the columns in the predefined database views.

Predefined query tables

Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Predefined query tables use the same underlying physical data and therefore have the same structure as predefined database views. However, predefined query tables

enhance the functionality and performance of predefined database views because they are optimized for running process and task list queries.

The predefined query tables can be queried directly using the query table API. When you access the tables using the query table API, you are offered more options for configuration than when you use the query API. You can develop a composite query table that contains all of the information to be retrieved when the query is run, not just the information from a single table.

Authorization is enabled for all work items, that is, everybody, individual, group, and inherited work items. On predefined query tables with instance data, unless specified, the query table API defaults to everybody, individual, and group work items.

Properties

Predefined query tables have the following properties:

Table 6. Properties of predefined query tables

Property	Description
Name	The query table name is the name of one of the predefined database views, in uppercase, for example TASK.
Attributes	Attributes of predefined query tables define the pieces of information that are available for queries. These attributes are the names of columns, in uppercase, that are specified by the predefined database views. The attributes are defined with a name and a type. The type is one of the following: • **Boolean**: A boolean value • **Decimal**: A floating point number • **ID**: An object ID, such as TKIID of the TASK query table TASK • **Number**: An integer, short, or long • **String**: A string • **Timestamp**: A timestamp
Authorization	Predefined query tables use either instance-based or role-based authorization. • Predefined query tables with instance data require instance-based authorization. This means that only objects with a work item for the user who performs the query are returned. However, using the AdminAuthorizationOptions object, this verification can be reduced to a verification of the existence of a work item of any user. The user must have the BPESystemAdministrator J2EE role for those queries. • Predefined query tables with template data require role-based authorization, which means that only users in the BPESystemAdministrator J2EE role can access the contents of those query tables.

Predefined query tables with instance data

The following table shows the predefined query tables that contain instance data. These query tables:

- Can be used as the primary query of a composite query table.

- Use instance-based authorization if queried directly. This is accomplished with a join (SQL-) with the view that stores authorization information, that is, the predefined WORK_ITEM view or query table.
- Contain instance data, for example data of task instances or process instances.

Table 7. Predefined query tables containing instance data

Instance data	Query table name
Information about activities of a process instance.	ACTIVITY
	ACTIVITY_ATTRIBUTE
	ACTIVITY_SERVICE
Information about escalations belonging to human tasks.	ESCALATION
	ESCALATION_CPROP
	ESCALATION_DESC
Information about process instances.	PROCESS_ATTRIBUTE
	PROCESS_INSTANCE
	QUERY_PROPERTY
Information about human tasks.	TASK
	TASK_CPROP
	TASK_DESC

The WORK_ITEM query table also contains instance data, but this is not available as the primary query table or an attached query table. Work item information is available implicitly when querying query tables that use instance-based authorization. That is, attributes of the WORK_ITEM query table can be used when querying a query table with instance-based authorization, even though the attributes are not explicitly specified by the query table.

Predefined query tables with template data

Predefined query tables with template data require role-based authorization. They can be queried only by administrators using the AdminAuthorizationOptions object.

The following table shows the predefined query tables that contain template data. These query tables:

- Can be used as the primary query table of a composite query table.
- Use role-based authorization if queried directly. This means that the caller must be in the BPESystemAdministrator J2EE role, and AdminAuthorizationOptions must be used.
- Contain template data, for example, the template data of task templates or process templates.

Table 8. Predefined query tables containing template data

Template data	Query table name
Information about application components.	APPLICATION_COMP
Information about escalation templates.	ESC_TEMPL
	ESC_TEMPL_CPROP
	ESC_TEMPL_DESC

Table 8. Predefined query tables containing template data (continued)

Template data	Query table name
Information about process templates.	PROCESS_TEMPLATE
	PROCESS_TEMPL_ATTR
Information about task templates.	TASK_TEMPL
	TASK_TEMPL_CPROP
	TASK_TEMPL_DESC

Related concepts

Supplemental query tables
Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Composite query tables
Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Query table development
Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

Query table queries
Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

Authorization for query tables
Instance-based authorization, role-based authorization, or no authorization can be used when queries are run on query tables.

Related reference

Database views for Business Process Choreographer
This reference information describes the columns in the predefined database views.

Supplemental query tables

Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Supplemental query tables relate to database tables or database views in the Business Process Choreographer database. They are query tables that contain supplemental business data that is maintained by customer applications. Supplemental query tables provide information in a composite query table in addition to information that is contained in a predefined query table.

Supplemental query tables have the following properties:

Table 9. Properties of supplemental query tables

Property	Description
Name	The query table name must be unique in a Business Process Choreographer installation. When the query is run, this name is used to identify the query table that is queried. A query table is uniquely identified using its name, which is defined as *prefix.name*. The maximum length of *prefix.name* is 28 characters. The prefix must be different to the reserved prefix `'IBM'`, for example, `'COMPANY.BUS_DATA'`.
Database name	The name of the related table or view in the database. Only uppercase letters may be used.
Database schema	The schema of the related table or view in the database. Only uppercase letters can be used. The database schema must be different to the database schema of the Business Process Choreographer database. Nevertheless, the table or view must be accessible with the same JDBC data source that is used to access the Business Process Choreographer database.
Attributes	Attributes of supplemental query tables define the pieces of information that are available for queries. These attributes must match the related name of the columns in the related database table or view. The attributes are defined with a name and a type. The name is defined in uppercase. The type is one of the following: • **Boolean**: A boolean value • **Decimal**: A floating point number • **ID**: An object ID of 16 bytes in length, such as TKIID of the TASK query table • **Number**: An integer, short, or long • **String**: A string • **Timestamp**: A timestamp
Join	Joins must be defined on supplemental query tables if they are attached in composite query tables. A join defines which attributes are used to correlate information in the supplemental query table with the information in the primary query table. When a join is defined, the source attribute and the target attribute must be of the same type.
Authorization	No authorization is specified for supplemental query tables, therefore, all authenticated users can see the contents.

Related concepts

Predefined query tables
Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Composite query tables
Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Query table development
Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

Query table queries
Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

Authorization for query tables
Instance-based authorization, role-based authorization, or no authorization can be used when queries are run on query tables.

Composite query tables

Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Composite query tables allow for a fine-grained configuration of filters and authorization options for optimized data access when the query is run. They do not have a specific representation of data in the database; they access the database contents of the related predefined and supplemental query tables. Composite query tables are realized with SQL, which is optimized for task and process list queries.

Composite query tables are designed by client developers. They are suggested for use in production scenarios in favor of the standard Business Process Choreographer query APIs, because they provide an abstraction over the actual implementation of the query and thus enable query optimizations. Furthermore, composite query tables allow changes at runtime without redeployment of the client that accesses the query table.

The following figure provides an overview of the content of composite query tables:

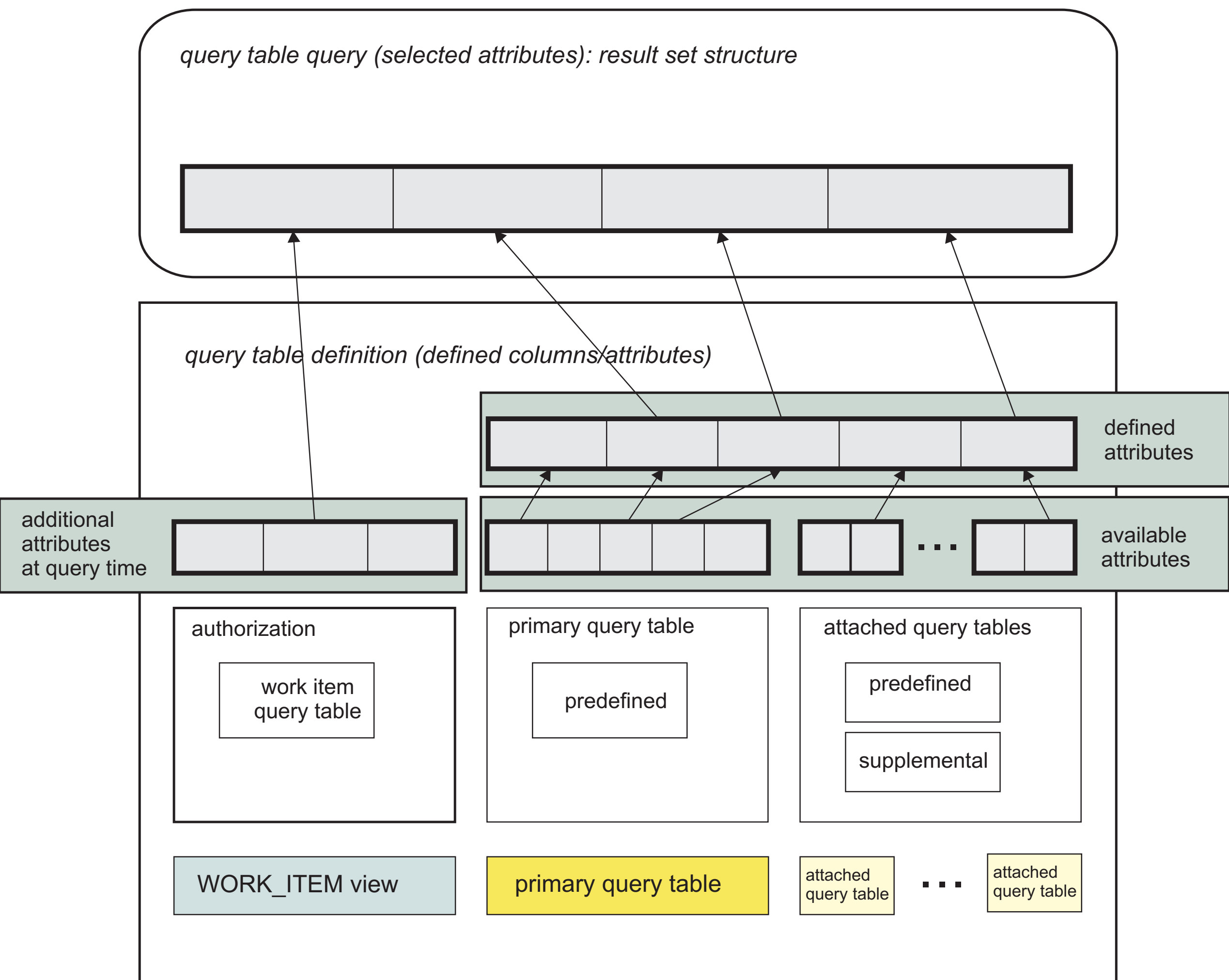


Figure 24. Composite query table content

All composite query tables are defined with one primary query table and zero or more attached query tables.

Primary query tables:

- Constitute the main information that is contained in a composite query table.
- Must be one of the predefined query tables.
- Uniquely identify each object in the composite query table by the primary key. For example, for the TASK predefined query table, this is the task ID TKIID.
- Authorize the contents of a query table using work items which are contained in the WORK_ITEM query table, if instance-based authorization is used.
- Determine the list of objects that are returned as rows of a table when querying the composite query table.

Attached query tables:

- Can be predefined query tables and supplemental query tables, which are already deployed on the system.
- Are available to provide information in addition to the information that is provided by the primary query table. For example, if TASK is the primary query

table, the description of the task provided in the TASK_DESC query table can be added to the contents of the composite query table.

Typically, the primary query table is chosen based on the purpose of the composite query table.

- If the composite query table describes a task list, the TASK query table is the primary query table.
- If the composite query table describes a process list, the PROCESS_INSTANCE query table is the primary query table.
- Lists of activities are retrieved using the ACTIVITY primary query table.
- Lists of human task escalations are retrieved using the ESCALATION primary query table.

The relationship between primary and attached query tables

A maximum of one row of the attached query table must correspond to a row in the primary query table, which is referred to as a one-to-one or one-to-zero relationship. If the one-to-one or one-to-zero relationship is violated, a runtime exception occurs when the query is run.

Primary query tables and attached query tables are correlated using a join attribute that is defined on the attached query table. This join attribute cannot be changed for predefined query tables, because it describes the relationship between the data in the various query tables of Business Process Choreographer. This join attribute is usually sufficient to maintain the one-to-one or one-to-zero relationship. For example, the CONTAINMENT_CTX_ID attribute is used on the TASK query table to attach the related process instance information that is identified by the PIID attribute on the PROCESS_INSTANCE query table. However, when a one-to many relationship exists, an additional criterion must be specified. This is called the selection criterion.

Selection criteria are specified during query table development using the Query Table Builder. They are used in the query table definition to choose one piece of information in a one-to-many relationship. For example, this could be `"LOCALE='en_US'"`. A task can have several descriptions that are identified using different locales for a single task.

Example 1:

The following figure provides a sample visualization of the selection criteria that is specified on attached query tables:

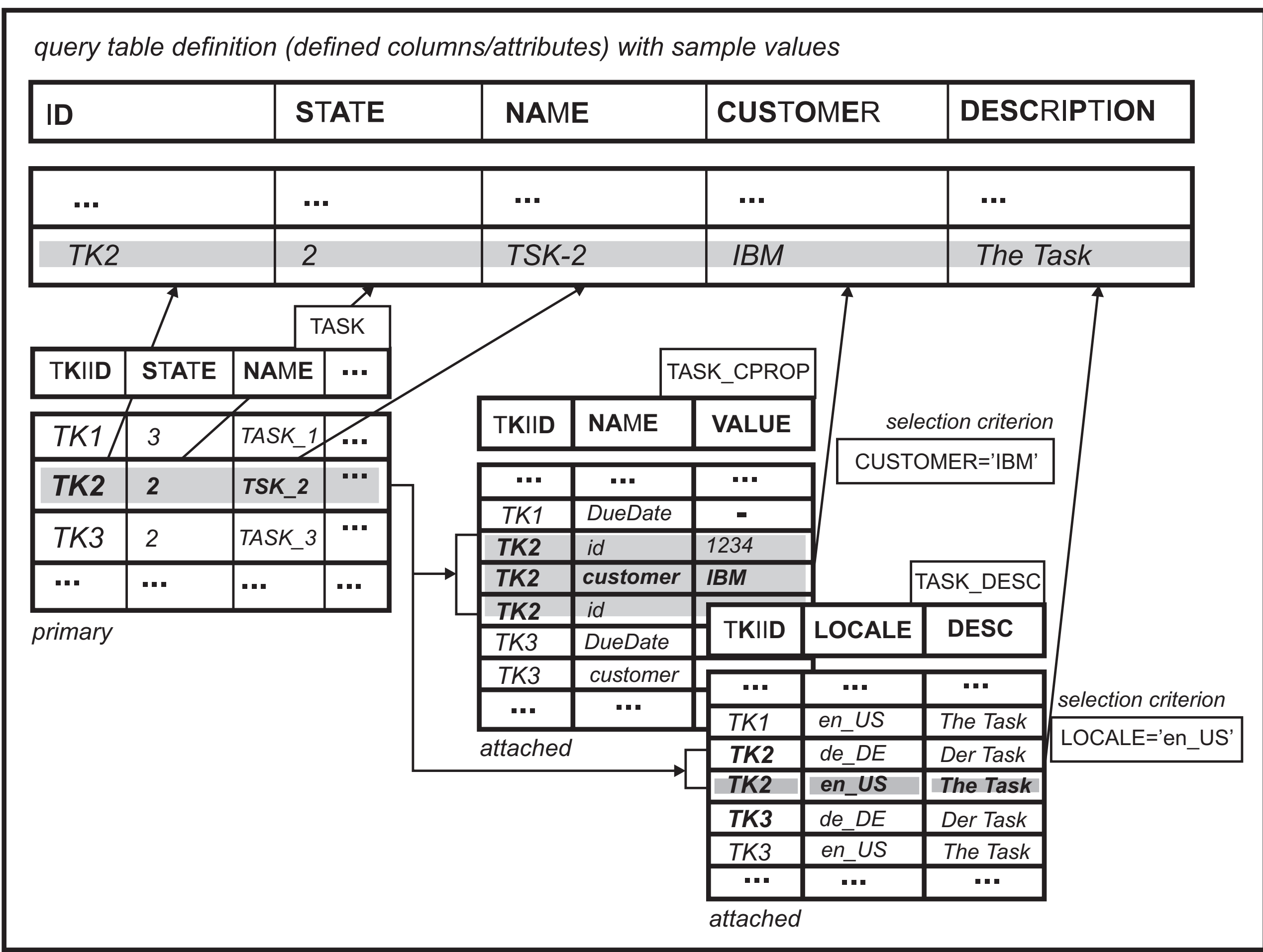


Figure 25. Composite query table with selection criteria

The composite query table contains the ID, STATE, NAME, CUSTOMER, and DESCRIPTION attributes.

- ID, STATE, and NAME are provided by the TASK primary query table.
- CUSTOMER is a custom property on TASK. Custom properties are stored in the TASK_CPROP query table. For a particular task, a custom property is uniquely identified using its name. This is reflected in the selection criterion `"CUSTOMER='IBM'"`.
- DESCRIPTION is the description of the task, which is stored in TASK_DESC query table. For each task instance, the task description for a particular task is uniquely identified by its locale. This is reflected in the selection criterion `"LOCALE='en_US'"`.

Example 2:

If TASK is the primary query table and TASK_DESC is attached to it, a particular locale must be chosen, which is the LOCALE attribute of the TASK_DESC query table. The focus of this example is on the relationship between the primary and the attached query tables, using TASK as the primary query table and TASK_DESC as the attached query table. The table shows sample contents of a composite query table with a valid selection criterion for the TASK_DESC attached query table.

Table 10. Valid contents of a composite query table

TASK primary query table information	TASK_DESC attached query table information	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_two	en_US	This is a description.
...	...	...

The following table shows hypothetical invalid contents (in **bold** type) if the selection criterion is set incorrectly, which means that the one-to-one or one-to-zero relationship is violated.

Table 11. Invalid contents of a composite query table

Information from TASK (primary query table)	Information from TASK_DESC (attached query table)	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_one	**de_DE**	**Das ist eine Beschreibung.**
...	...	...

Properties

Composite query tables have the following properties:

Table 12. Properties of composite query tables

Property	Description
Name	The query table name must be unique within a Business Process Choreographer installation. When the query is run, this query table name is used to identify the query table that is queried. A query table is uniquely identified using its name, which is defined as *prefix.name* for composite query tables. The maximum length of the *prefix.name* is 28 characters. The prefix must be different from the reserved prefix 'IBM', for example, 'COMPANY.TODO_TASK_LIST'.

Table 12. Properties of composite query tables (continued)

Property	Description
Attributes	Attributes of composite query tables define the pieces of information that are available for queries. The attributes are defined with a name, in uppercase. The type is inherited from the referenced attribute, which is one of the following: • **Boolean**: A boolean value • **Decimal**: A floating point number • **ID**: An object ID, such as TKIID of query table TASK • **Number**: An integer, short, or long • **String**: A string • **Timestamp**: A timestamp Attributes of composite query tables are defined using a reference to attributes of the primary query table or the attached query tables. The attributes of the composite query tables inherit the types and constants of referenced attributes. In addition to the attributes that are part of the query table definition, work item information can be queried at runtime. This is possible if the primary query table contains instance data, such as TASK or PROCESS_INSTANCE, and if instance-based authorization is used on the composite query table. For example, the query can be defined to return only human tasks of which the user is a potential owner.
Authorization	Each composite query table defines if instance-based, role-based, or no authorization is used when queries are run on it. If instance-based authorization is defined, only objects with a work item for the user who performs the query are returned. However, using AdminAuthorizationOptions this verification can be reduced to a verification of the existence of a work item of any user. The user must be in the BPESystemAdministrator J2EE role for those queries, and AdminAuthorizationOptions must be passed to the query table API. If role-based authorization is defined, the user must be in the BPESystemAdministrator J2EE role for those queries, and AdminAuthorizationOptions must be passed to the query table API. If no authorization is defined, the query is run without checks against the existence of work items of the related objects in the query table. All authenticated users can see the contents of the query table. Instance-based authorization can be defined if the primary query table contains instance data; role-based authorization can be defined if the primary query table contains template data. No authorization can be defined on composite query tables regardless of which primary query table is used.

Filters

Filters are used to limit the number of objects, or rows, that are contained in a composite query table.

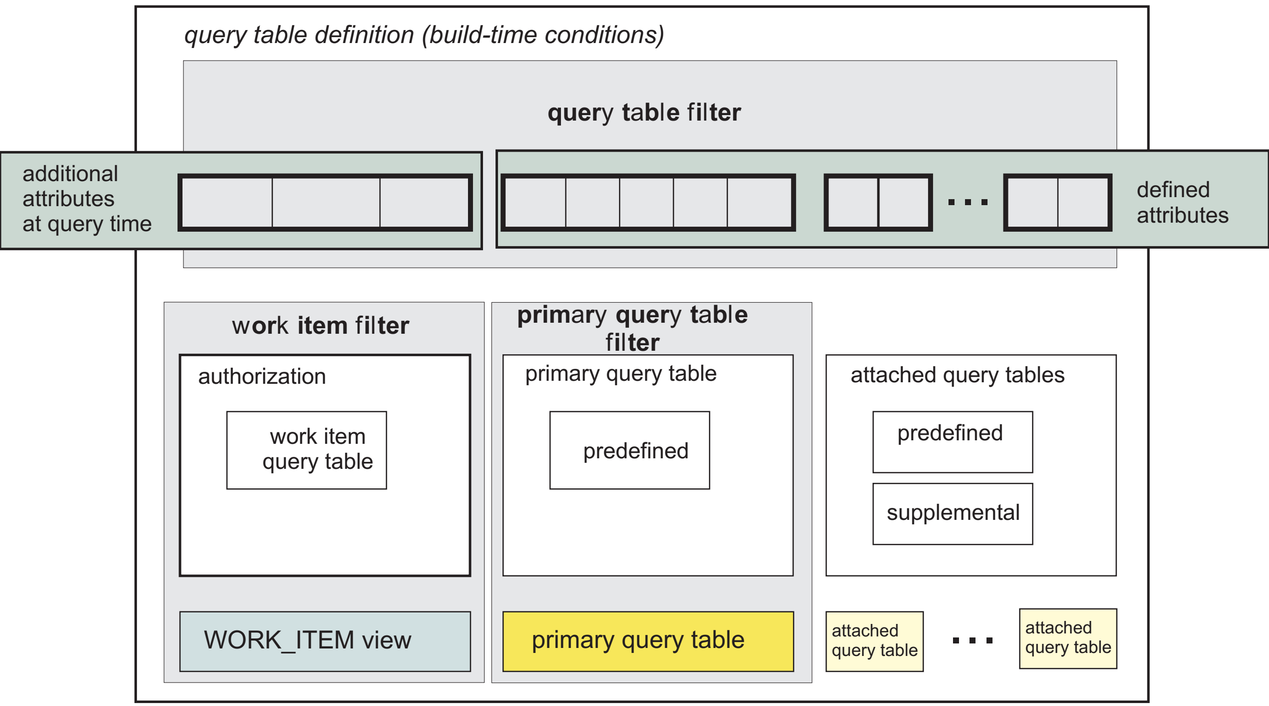


Figure 26. Filters in composite query tables

Filters in composite query tables can be defined during development on the:

- Primary query table, as the primary query table filter.
- Implicitly available WORK_ITEM query table which is responsible for authorization if the primary query table contains instance data. This filter is called the authorization filter, and is available only if the composite query table is configured to use instance-based authorization.
- Composite query table, as the query table filter.

Filters are defined during query table development. For example, a composite query table with the TASK primary query table can filter on tasks that are in the ready state (`"STATE=STATE_READY"` as the primary query table filter).

Authorization

Authorization for accessing the contents of a composite query table with a primary query table is similar to the authorization that is used to access the primary query table. The difference is that composite query tables can be configured to be more restrictive.

- If instance-based authorization is configured for use, the data contained in the composite query table is verified for existing work items in the WORK_ITEM query table. This verification is made against the primary query table. Everybody, individual, group, and inherited work items are used for the verification, depending on the configuration of the composite query table. If inherited work items are specified, objects that have a process instance as parent, such as a participating human task, with a related everybody, individual, or group work item as configured, are contained in the composite query table. Typically, inherited work items are useful only for administrators.

- Composite query tables with a primary query table that contains template data must not be set to use instance-based authorization. If role-based authorization is used, queries can be run only by users that are in the BPESystemAdministrator J2EE role, and the AdminAuthorizationOptions object must be used.

Related concepts

Predefined query tables
Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Supplemental query tables
Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Query table development
Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

Query table queries
Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

Authorization for query tables
Instance-based authorization, role-based authorization, or no authorization can be used when queries are run on query tables.

Query table development

Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

The Query Table Builder is available as an Eclipse plug-in and can be downloaded on the WebSphere Business Process Management SupportPacs site. Look for PA71 WebSphere Process Server - Query Table Builder. To access the link, see the related references section of this topic.

Query tables impact on the way applications are developed and deployed. The following steps describe the roles involved when you design and develop a Business Process Choreographer application that uses query tables.

Table 13. Query table development steps

Step	Who	Description
1. Analysis	Business analyst, client developer	Analyze which query tables are needed in the client application. Questions to be answered are: • How many task or process lists are provided to the user? Are there task or process lists that can share the same query table? • What kind of authorization is used? Instance-based authorization, role-based authorization, or none? • Are there other query tables already defined in the system that can be reused? • Must the query tables provide the content in multiple languages? If so, the selection criteria on attached query tables should be `LOCALE=$LOCALE`.
2. Query table development	Client developer, business analyst	Develop the query tables that are used in the client application. Try to specify the definition of the query tables such that the best performance is achieved with query table queries.
3. Query table deployment	Administrator	Query tables must be deployed to the runtime before they can be used. This step is done using the manageQueryTable.py wsadmin command.
4. Query table queries	Client developer	To run queries against query tables is the last step of query table development. The client developer must know the name of the query table and its attributes.

The following is sample code, which uses the query table API to query a query table. Examples 1 and 2 are provided to query the predefined query table TASK for simplicity reasons. Examples 3 and 4 are provided to query a composite query table, which is assumed to be deployed on the system. In application development, composite query tables should be used rather than directly querying the predefined query tables.

Example 1

```
// get the naming context and lookup the Business
// Flow Manager EJB home; note that the Business Flow
// Manager EJB home should be cached for performance
// reasons; also, it is assumed that there's an EJB
// reference to the local business flow manager EJB
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
 (LocalBusinessFlowManagerHome)
 ctx.lookup("java:comp/env/ejb/BFM");

// create the business flow manager client-side stub
LocalBusinessFlowManager bfm = home.create();

// ************************************************
// ****************** example 1 ******************
// ************************************************
```

```
// execute a query against the TASK predefined query
// table; this relates to a simple My ToDo's task list
EntityResultSet ers = null;
ers = bfm.queryEntities("TASK", null, null, null);

// print the result to STDOUT
EntityInfo entityInfo = ers.getEntityInfo();
List attList = entityInfo.getAttributeInfo();
int attSize = attList.size();

Iterator iter = ers.getEntities().iterator();
while (iter.hasNext()) {
 System.out.print("Entity: ");
 Entity entity = (Entity) iter.next();
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    entity.getAttributeValue(ai.getName()));
 }
 System.out.println();
}
```

Example 2

```
// *************************************************
// ****************** example 2 ******************
// *************************************************

// same example as example 1, but using the row-based
// query approach
RowResultSet rrs = null;
rrs = bfm.queryRows("TASK", null, null, null);

attList = rrs.getAttributeInfo();
attSize = attList.size();

// print the result to STDOUT
while (rrs.next()) {
 System.out.print("Row: ");
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    rrs.getAttributeValue(ai.getName()));
 }
 System.out.println();
}
```

Example 3

```
// *************************************************
// ****************** example 3 ******************
// *************************************************

// execute a query against a composite query table
// that has been deployed on the system before;
// the name is assumed to be COMPANY.TASK_LIST
ers = bfm.queryEntities(
  "COMPANY.TASK_LIST", null, null, null);
^
//  print the result to STDOUT ...
```

Example 4

```
// *************************************************
// ****************** example 4 ******************
// *************************************************
```

```
// query against the same query table as in example 3,
// but with customized options
FilterOptions fo = new FilterOptions();

// return only objects which are in state ready
fo.setQueryCondition("STATE=STATE_READY");

// sort by the id of the object
fo.setSortAttributes("ID");

// limit the number of entities to 50
fo.setThreshold(50);

// only get a sub-set of the defined attributes
// on the query table
fo.setSelectedAttributes("ID, STATE, DESCRIPTION");

AuthorizationOptions ao = new AuthorizationOptions();

// do not return objects that everybody is allowed
// to see
ao.setEverybodyUsed(Boolean.FALSE);

ers = bfm.queryEntities(
  "COMPANY.TASK_LIST", fo, ao, null);

//  print the result to STDOUT ...
```

Related concepts

Predefined query tables
Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Supplemental query tables
Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Composite query tables
Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Query table queries
Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

Filters and selection criteria of query tables
Filters and selection criteria are defined during query table development using the Query Table Builder, which uses a syntax similar to SQL WHERE clauses. Use these clearly defined filters and selection criteria to specify conditions that are based on attributes of query tables.

Related tasks

Administering query tables
Use the wsadmin script, manageQueryTable.py script to administer query tables in Business Process Choreographer, which were developed using the Query Table Builder. Unlike predefined query tables, which are available out-of-the-box, you must deploy composite and supplemental query tables on WebSphere Process Server before you can use them with the query table API.

Deploying query tables
Use the manageQueryTable.py script to deploy supplemental and composite query tables in Business Process Choreographer. Query tables are deployed on a stand-alone server that is running or in a cluster with at least one member running. The undeployment of supplemental and composite query tables is performed also on running servers. For supplemental query tables, the related physical database objects, typically a database view or database table, must be created if they do not exist prior to the usage of the query table.

Filters and selection criteria of query tables

Filters and selection criteria are defined during query table development using the Query Table Builder, which uses a syntax similar to SQL WHERE clauses. Use these clearly defined filters and selection criteria to specify conditions that are based on attributes of query tables.

For information on installing the Query Table Builder, see the WebSphere Business Process Management SupportPacs site. Look for PA71 WebSphere Process Server - Query Table Builder. To access the link, see the related references section of this topic.

Attributes

Attributes in an expression refer to attributes of query tables. Depending on the location of the expression, different attributes are available. For the client developer, query filters passed to the query table API are the only location where expressions can be used. For developers of composite query tables, various other locations exist where expressions can be used. The following table describes the attributes that are available at the different locations.

Table 14. Attributes for query table expressions

<table>
<tr><th>Where</th><th>Expression</th><th>Available attributes</th></tr>
<tr><td>Query table API</td><td>Query filter</td><td rowspan="2">• All attributes defined on the query table.
• If instance-based authorization is used, all attributes defined on the WORK_ITEM query tables, prefixed with `'WI.'` .

Examples:
• `STATE=STATE_READY`, if the query table contains a STATE attribute and if a STATE_READY constant is defined for this attribute
• `STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER`, if the query table contains a STATE attribute and the query table uses instance-based authorization</td></tr>
<tr><td rowspan="4">Composite query table</td><td>Query table filter</td></tr>
<tr><td>Primary query table filter</td><td>• All attributes defined for the primary query table.

Example:
• `STATE=STATE_READY`, if the query table contains a STATE attribute and a STATE_READY constant is defined for this attribute</td></tr>
<tr><td>Authorization filter</td><td>• All attributes defined on the WORK_ITEM predefined query table, prefixed with `'WI.'` .

Example:
• `WI.REASON=REASON_POTENTIAL_OWNER`</td></tr>
<tr><td>Selection criterion</td><td>• All attributes defined on the related attached query table.

Example:
• `LOCALE='en_US'`, if the attached query table contains a LOCALE attribute, such as the TASK_DESC query table</td></tr>
</table>

The following figure shows the various locations of filters and selection criteria in expressions, and includes examples:

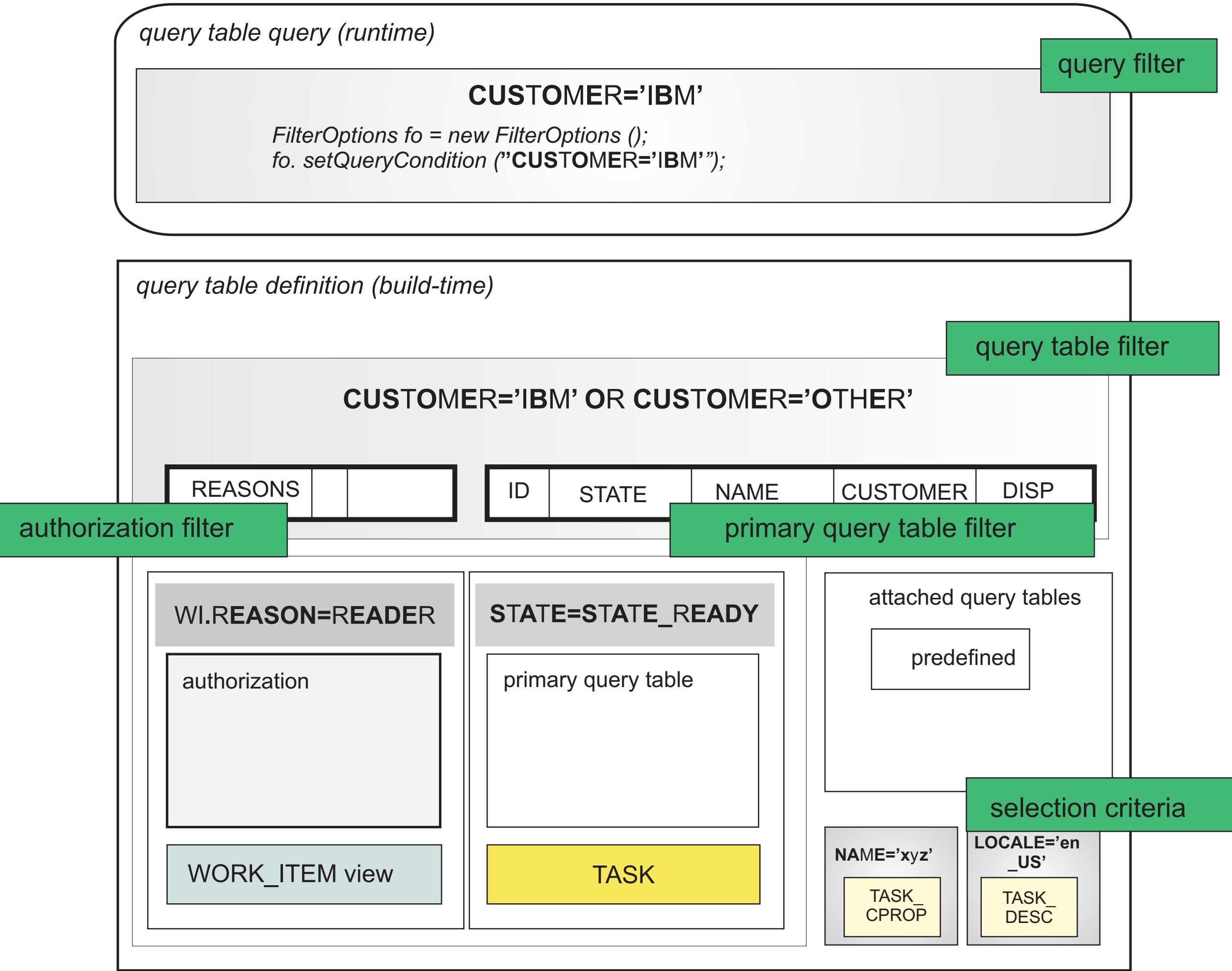


Figure 27. Filters and selection criteria in expressions

Expressions

Expressions have the following syntax:

```
expression  ::=  attribute binary_op value |
                 attribute unary_op |
                 attribute list_op list |
                 (expression) |
                 expression AND expression |
                 expression> OR expression
```

The following rules apply:

- AND takes precedence over OR. Subexpressions are connected using AND and OR.
- Brackets can be used to group expressions and must be balanced.

Examples:

- `STATE = STATE_READY`
- `NAME IS NOT NULL`
- `STATE IN (2, 5, STATE_FINISHED)`
- `((PRIORITY=1) OR (WI.REASON=2)) AND (STATE=2)`

An expression is executed in a certain scope which determines the attributes that are valid for the expression. Selection criteria, or query filters, are run in the scope of the query table on which the query is run.

The following example is for a query that is run on the predefined TASK query table:

```
'(STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER)
OR (WI.REASON=REASON_OWNER)'
```

Binary operators

The following binary operators are available:

binary_op `::= = | < | > | <> | <= | >= | LIKE | NOT LIKE`

The following rules apply:

- The left-side operand of a binary operator must reference an attribute of a query table.
- The right-side operand of a binary operator must be a literal value, constant value, or parameter.
- The `LIKE` and `NOT LIKE` operators are only valid for attributes of attribute type `STRING`.
- The left-side operand and the right-side operand must be of compatible attribute types.
- User parameters must be compatible to the attribute type of the left-side attribute.

Examples:

- `STATE > 2`
- `NAME LIKE 'start%'`
- `STATE <> PARAM(theState)`

Unary operators

The following unary operators are available:

unary_op `::= IS NULL | IS NOT NULL`

The following rules apply:

- The left-side operand of a unary operator must reference an attribute of a query table. Valid attributes depend on the location of the filter or selection criterion.
- All attributes can be checked for null values, for example: `CUSTOMER IS NOT NULL`.

Example:

```
DESCRIPTION IS NOT NULL
```

List operators

The following list operators are available:

list_op `::= IN | NOT IN`

The following rules apply:

- The right-side of a list operator must not be replaced by a user parameter.

- User parameters can be used within the list on the right-side operand.

Example:

```
STATE IN (STATE_READY, STATE_RUNNING, PARAM(st), 1)
```

Lists are represented as follows:

list ::= *value* [, *list*]

The following rules apply:
- The right-side of a list operator must not be replaced by a user parameter.
- User parameters can be used within the list on the right-side operand.

Examples:
- `(2, 5, 8)`
- `(STATE_READY, STATE_CLAIMED)`

Values

In expressions, a value is one of the following:
- **Constant**: A constant value, which is defined for the attribute of a predefined query table. For example, `STATE_READY` is defined for the STATE attribute of the TASK query table.
- **Literal**: Any hardcoded value.
- **Parameter**: A parameter is replaced when the query is run with a specific value.

Constants are available for some attributes of predefined query tables. For information on constants that are available on attributes of predefined query tables, refer to the information on predefined views. Only constants that define integer values are exposed with query tables. Also, instead of constants, related literal values, or parameters can be used.

Examples:
- `STATE_READY` on the STATE attribute of the TASK query table can be used in a filter to check whether the task is in the ready state.
- `REASON_POTENTIAL_OWNER` on the REASON attribute of the WORK_ITEM query table can be used in a filter to check whether the user who runs the query against a query table is a potential owner.
- Query filter `STATE=STATE_READY` is the same as `STATE=2`, if the query is run on the TASK query table.

Literals can also be used in expressions. A special syntax must be used for timestamps and for IDs.

Examples:
- `STATE=1`
- `NAME='theName'`
- `CREATED > TS ('2008-11-26 T12:00:00')`
- `TKTID=ID('_TKT:801a011e.9d57c52.ab886df6.1fcc0000')`

Parameters in expressions allow for a dynamicity of composite query tables. There are user parameters and system parameters:

- User parameters are specified using PARAM (*name*). This parameter must be provided when the query is run. It is passed as an instance of the com.ibm.bpe.api.Parameter class into the query table API.
- System parameters are parameters that are provided by the query table runtime, without being specified when the query is run. The system parameters `$USER` and `$LOCALE` are available.
  - `$USER`, which is a string, contains the value of the user who runs the query.
  - `$LOCALE`, which is a string, contains the value of the locale that is used when the query is run. An example for the value of `$LOCALE` is `'en_US'`.

You can specify a parameter in the selection criteria of an attached query table which selects on a specific locale. For example, if the primary query table is TASK in a composite query table and an attached query table is TASK_DESC. The following are examples of parameters:
- `STATE=PARAM(theState)`
- `LOCALE=$LOCALE`
- `OWNER=$USER`

Related concepts

Query table development
Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

Query table queries
Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

Authorization for query tables

Instance-based authorization, role-based authorization, or no authorization can be used when queries are run on query tables.

The authorization type used when a query is run on a query table is defined on the query table.
- Instance-based authorization indicates that objects in the query table are authorized using a work item. This is done by verifying if a suitable work item exists.
- Role-based authorization is based on J2EE roles. It indicates that the caller must be in the BPESystemAdministrator J2EE role to see the contents of the query table. It is available for predefined query tables with template data and for composite query tables with a primary query table that contains template data. Objects in those query tables do not have related work items.
- When no authorization is specified, all authenticated users can see all contents of the query table, after filters are applied.

The type of authorization on predefined query tables and the type of authorization that can be configured on composite and supplemental query tables is outlined in the following table.

Table 15. Types of authorization for query tables

Query table	Instance-based authorization	Role-based authorization	No authorization
Predefined	Required for predefined query tables with instance data.	Required for predefined query tables with template data.	N/A
Composite	Can be turned off which means that no authorization is used and the security constraints are overridden. That is, every authenticated user can use the query table to retrieve data, independently of whether they are authorized for the respective objects. Composite query tables with a primary query table that contains template data must not bet set to use instance-based authorization.	Can be turned off, for example for composite query tables with a primary query table that contains template data. This means that no authorization is used and the security constraints are overridden. That is, every authenticated user can use the query table to retrieve data, independently of whether they are authorized for the respective objects. Composite query tables with a primary query table that contains instance data must not bet set to use role-based authorization.	All authenticated users can see all contents of the query table, after filters are applied.
Supplemental	Supplemental query tables must not be set to use instance-based authorization because they are not managed by Business Process Choreographer, and therefore it has no authorization information for the contents of these tables.	Supplemental query tables must not be set to use role-based authorization.	All authenticated users can see all contents of the query table, after filters are applied.

The following figure provides an overview of the available options for the authorization types, depending on the type of query table. Also, it outlines the different behaviors and the query table API and its authorization options.

Composite query table	primary query table with instance data	all	primary query table with template data
Predefined query tables	instance data	*n/a*	template data
Supplemental query tables	*n/a*	business data	*n/a*
Authorization	Instance-based authorization	None	Role-based authorization
Query with AuthorizationOptions	**(A)** Query result contains objects with work items related to the caller.	**(C)** Query result contains all objects that are in this query table.	*n/a*
Query with AdminAuthorization Options*	**(B)** Query result contains all objects that are in this query table.		**(D)** Query result contains all objects that are in this query table.

Figure 28. Instance-based authorization for query tables

*) If the onBehalfUser is set, (A) applies

Instance-based authorization for objects in the query result using work items depend on the authorization parameter that is passed to the query table API and on the setting of the instance-based authorization flag of the query table.

- (A) Queries on predefined or composite query tables using the AuthorizationOptions object return entities that correlate with a related work item for this particular user. This is also the case if the AdminAuthorizationOptions object is used and onBehalfUser is set. Standard clients which present task or process lists to users usually use this combination of query tables and query table API parameters.

- (B) The full content of a query table consists of the entities that have a related work item, as configured with the instance-based authorization of the query table. Instance-based authorization considers four types of work items: everybody, individual, group, and inherited. The caller must be in the BPESystemAdministrator J2EE role. This combination of query tables and query table API parameters is intended for use in administrative scenarios where the full list of available tasks or processes must be shown or searched.
- (C) Queries on query tables that do not use instance-based or row-based authorization return the same result if AdminAuthorizationOptions or AuthorizationOptions is passed into the query table API. This is available for supplemental and composite query tables. There is no check on work items or J2EE roles, therefore all authenticated users see the full content. Clients that do not want to restrict object visibility by applying the instance-based or role-based authorization constraints that are provided by Business Process Choreographer can turn off authorization checks when query table definitions are developed. When using claim and complete, however, users must have related work items.
- (D) Template data in predefined query tables or composite query tables with role-based authorization configured can be accessed only with role-based authorization. This requires the caller to be in the BPESystemAdministrator J2EE role. The query table API can be used to access template information instead of the query API.

Work items and instance-based authorization

Instance-based authorization provided by Business Process Choreographer is based on work items. Each work item describes who has which rights on what object. This information is accessible using the WORK_ITEM query table, if instance-based authorization is used.

The table describes the different types of work items that are considered if instance-based authorization is used when a query is run against a query table:

Table 16. Work item types

Work item type	Description
everybody	Everybody work items allow all users to access a specific object, such as a task or a process instance. In this case, the EVERYBODY attribute of the related work item is set to TRUE.
individual	Individual work items are work items that are created for particular users. The OWNER_ID attribute of the related work item is set to a specific user. Multiple work items which differ in the OWNER_ID attribute can exist for an object, such as a task.
group	Group work items are work items that are created for users of a particular group. The GROUP_NAME attribute of the related work item is set to a specific group.
inherited	Readers and administrators of process instances are also allowed to inherit the access to the human tasks which belong to these process instances, including escalations. Checks for an inherited work item in task queries are performed with complex SQL joins at runtime, which impacts on performance.

Work items are created by Business Process Choreographer in different situations. For example, at task creation, work items are created for the different roles, such as reader and potential owner, if related people assignment criteria were specified.

The following table describes the types of work items that are created, depending on the people assignment criteria that are defined, if instance-based authorization is used when a query is run on a query table. Inherited work items do not appear in the table because they reflect a relationship that is not explicitly modeled during process application development.

Table 17. Work items and people assignment criteria

Work item type	Related people assignment criteria
everybody	Everybody
individual	All people assignment criteria except verbs *Nobody*, *Everybody*, and *Group*
group	Group

Authorization filter on composite query tables

On composite query tables, an authorization filter can be specified if instance-based authorization is used. This filter restricts the work items which are used for authorization, based on certain attributes of work items. For example, the authorization filter `"WI.REASON=REASON_POTENTIAL_OWNER"` on a composite query table with the TASK primary query table restricts the tasks that are returned when a person runs a query. The result contains only tasks that represent a to-do for that person, that is, the result is restricted to those tasks the person is authorized to claim. This filter can also be specified as the query table filter or as the query filter. Nevertheless, for query performance reasons, it is beneficial to specify those filters as the authorization filter.

Related concepts

Predefined query tables
Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Supplemental query tables
Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Composite query tables
Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Authorization options for the query table API
When you run a query on a query table in Business Process Choreographer, authorization options can be passed as input parameters to the methods of the query table API.

Attribute types for query tables

Attribute types are needed in Business Process Choreographer when query tables are defined, when literal values are used in queries, and when values of a query result are accessed. Rules and mappings are available for each of the attribute types.

A subset of the types that are available in the Java programming language and databases is used to define the type of an attribute of a query table. Attribute types are an abstraction of the concrete Java type or database type. For supplemental query tables, you must use a valid database type to attribute type mapping.

The following table describes the attribute types:

Table 18. Attribute types

Attribute type	Description
ID	The ID which is used to identify a human task (TKIID), a process instance (PIID), or other objects. For example, IDs are used to claim or complete a particular human task, which is identified with the specified TKIID.
STRING	Task descriptions or query properties can be represented as a string.
NUMBER	Numbers are used for attributes, such as the priority on a task.
TIMESTAMP	Timestamps describe a point in time, such as the time when a human task is created, or a process instance is finished.
DECIMAL	Decimals can be used as the type for query properties, for example when defining a query property with a variable of XSD type double.

Table 18. Attribute types (continued)

Attribute type	Description
BOOLEAN	Booleans can have one of two values, true or false. For example, human tasks provide an attribute, autoClaim, which identifies whether the task is claimed automatically if only a single user exists as the potential owner for this task.

Database type to attribute type mapping:

Use attribute types to define query tables in Business Process Choreographer, when you run queries on the query tables, and to access values of a query result.

The following table describes the database types and their mapping to attribute types:

Table 19. Database type to attribute type mapping

Database type	Attribute type
A binary type with 16 bytes. This is the type used for IDs such as TKIID on TASK of the Business Process Choreographer tables.	ID
A character based type. The length depends on the column in the database table that is referenced by the attribute of the query table.	STRING
An integer database type, such as integer, short, or long.	NUMBER
A timestamp database type.	TIMESTAMP
A decimal type, such as float or double.	DECIMAL
A type that is convertible to a Boolean value, such as a number. 1 is interpreted as *true*, and all other numbers as *false*.	BOOLEAN

Example:

Consider a table in a DB2 environment, CUSTOM.ADDITIONAL_INFO, which is to be represented in Business Process Choreographer as a supplemental query table. The following SQL statement creates the database table:

```
CREATE TABLE CUSTOM.ADDITIONAL_INFO
(
  PIID       CHAR(16) FOR BIT DATA,
  INFO       VARCHAR(220),
  COUNT      INTEGER
);
```

The following mapping of database column types to query table attribute types is used for a supplemental query table for the CUSTOM.ADDITIONAL_INFO table.

Table 20. Database types to attribute types mapping example

Database column and type	Query table attribute and type
PIID CHAR(16) FOR BIT DATA	PIID (ID)
INFO VARCHAR(220)	INFO (STRING)
COUNT INTEGER	COUNT (NUMBER)

Supplemental query tables typically refer to existing database tables and views, such that table or view creation is not necessary.

Attribute type to literal representation mapping:

Attribute types are used when query tables are defined in Business Process Choreographer, when queries are run on the query tables, and when values of a query result are accessed. Use this topic for information on attribute type to literal representation mapping.

Literal values can be used in expressions to define filter and selection criteria, such as in filters of composite query tables, and in filters that are passed to the query table API.

The following table describes the attribute types and their mapping to literal values. Placeholders are marked *italic*. Note that the attribute types ID and TIMESTAMP, which can be passed to the query table API, use a special syntax, which is also used by the query API.

Table 21. Attribute type to literal values mapping

Attribute type	Syntax and usage as literal value in expressions
ID	`ID ('string representation of an ID' )`
	When developing client applications, IDs are represented either as a string or as an instance of the com.ibm.bpe.api.OID interface. The string representation can be obtained from an instance of the com.ibm.bpe.api.OID interface using the toString method. The string must be enclosed in single quotation marks.
STRING	`'the string'`
	The string must be enclosed in quotes.
NUMBER	`number`
	The number as text, and no quotation marks. Constants are defined for some number attributes on predefined query tables, and can be used.
TIMESTAMP	`TS ('YYYY-MM-DDThh:mm:ss')`
	The timestamp must be specified as: • `YYYY` is the 4-digit year • `MM` is the 2-digit month of the year • `DD` is the 2-digit day of the month • `hh` is the 2-digit hour of the day (24-hour) • `mm` is the 2-digit minutes of the hour • `ss` is the 2-digit seconds of the minute The timestamp is interpreted as defined in the user's time zone.
DECIMAL	`number.fraction`
	The decimal number as text and no quotation marks; the `.fraction` part is optional.
BOOLEAN	`true`, `false`
	The Boolean value as text.

Examples:

- `filterOptions.setQueryCondition("STATE=2");`

- `filterOptions.setQueryCondition("STATE=STATE_READY");`
- `a selection criterion on an attached query table TASK_DESC: "LOCALE='en_US'"`
- `filterOptions.setQueryCondition( "PTID=ID('_PT:8001011e.1dee8e51.247d6df6.29a60000')");`

Attribute type to parameter mapping:

Use attribute types when you define query tables in Business Process Choreographer, when you run queries on the query tables, and to access values of a query result.

The following table describes the attribute types and their mapping to parameter values that can be used in expressions to define filter and selection criteria, such as in filters of composite query tables, and in filters passed to the query table API.

Table 22. Attribute type to user parameter values mapping

Attribute type	Usage as parameter value in expressions
ID	`PARAM`*(name)* When developing client applications, IDs are represented either as a string or as an instance of the com.ibm.bpe.api.OID interface. As a parameter, both representations are valid. An array of bytes reflecting a valid OID can also be used (byte).
STRING	`PARAM`*(name)* The string representation of the object that is passed to the query table API at runtime by the toString method.
NUMBER	`PARAM`*(name)* A java.lang.Long, java.lang.Integer, java.lang.Short, or a java.lang.String representation of the number is passed to the query table API. Names of constants, as defined on some attributes of predefined query tables, can be also passed.
TIMESTAMP	`PARAM`*(name)* The following representations are valid: • A java.lang.String representation of the timestamp • Instances of com.ibm.bpe.api.UTCDate • Instances of java.util.Calendar
DECIMAL	`PARAM`*(name)* A java.lang.Long, java.lang.Integer, java.lang.Short, java.lang.Double, java.lang.Float, or a java.lang.String representation of the decimal is passed to the query table API.
BOOLEAN	`PARAM`*(name)* Valid values are: • A java.lang.String representation of the boolean • A java.lang.Short, java.lang.Integer, java.lang.Long with appropriate values; `0` (for false), or `1` (for true) • A `java.lang.Boolean` object

Example:

```
...
// this example shows a query against a composite query table
// COMP.TASKS with a parameter "customer"
java.util.List params = new java.util.ArrayList();

list.add(new com.ibm.bpe.api.Parameter("customer", "IBM");
bfm.queryEntities("COMP.TASKS", null, null, params);
...
```

Attribute type to Java object type mapping:

Attribute types are used when query tables are defined in Business Process Choreographer, when queries are run on the query tables, and when values of a query result are accessed. Use this topic for information on attribute type to Java object type mapping.

The following table describes the attribute types and their mapping to Java object types in query result sets.

Table 23. Attribute type to Java object type mapping

Attribute type	Related Java object type
ID	com.ibm.bpe.api.OID
STRING	java.lang.String
NUMBER	java.lang.Long
TIMESTAMP	java.util.Calendar
DECIMAL	java.lang.Double
BOOLEAN	java.lang.Boolean

Example:

```
...
// the follwing example shows a query against a composite query table
// COMP.TA; attribute "STATE" is of attribute type NUMBER
...
// run the query
EntityResultSet rs = bfm.queryEntities("COMP.TA",null,null,params);

// get the entities and iterate over it
List entities = rs.getEntities();
for (int i = 0 ; i < entities.size(); i++) {

      // work on a particular entity
      Entity en = (Entity) entities.get(i);

      // note that the following code could be written
      // more generalized using the attribute info objects
      // contained in ei.getAttributeInfo()

      // get attribute STATE
      Long state = (Long) en.getAttributeValue("STATE");
      ...
}
...
```

Attribute type compatibility:

Use attribute types when you define query tables in Business Process Choreographer, when you run queries on the query tables, and to access values of a query result.

The following table shows the attribute types and their compatible attribute types, which can be used to define filters and selection criteria in query tables. Compatible attribute types are marked with **X**.

Table 24. Attribute type compatibility

Attribute type	ID	STRING	NUMBER	TIMESTAMP	DECIMAL	BOOLEAN
ID	X					
STRING		X				
NUMBER			X		X	
TIMESTAMP				X		
DECIMAL			X		X	
BOOLEAN						X

In query table expressions that specify filter and condition criteria, types of attributes or values that are compared must be compatible. For example, `WI.OWNER_ID=1` is an invalid filter because the left-side operand is of type STRING, and the right-side operand is of type NUMBER.

Query table queries

Queries are run on query tables in Business Process Choreographer using the query table API, which is available on the Business Flow Manager EJB and the REST API.

A query is run on one query table only. Entity-based API methods and row-based API methods are used to retrieve content from query tables. Input parameters are passed into the methods of the query table API.

Related concepts

Predefined query tables
Predefined query tables provide access to the data in the Business Process Choreographer database. They are the query table representation of the corresponding predefined Business Process Choreographer database views, such as the TASK view or the PROCESS_INSTANCE view.

Supplemental query tables
Supplemental query tables in Business Process Choreographer expose to the query table API business data that is not managed by Business Process Choreographer. With supplemental query tables, this external data can be used with data from the predefined query tables when retrieving business process instance information or human task information.

Composite query tables
Composite query tables in Business Process Choreographer comprise predefined query tables and supplemental query tables. They combine data from existing tables or views. Use a composite query table to retrieve the information for a process instance list or task list, such as My To Dos.

Query table development
Supplemental and composite query tables in Business Process Choreographer are developed during application development using the Query Table Builder. Predefined query tables cannot be developed or deployed. They are available when Business Process Choreographer is installed and provide a simple view on the artifacts in the Business Process Choreographer database schema.

Filters and selection criteria of query tables
Filters and selection criteria are defined during query table development using the Query Table Builder, which uses a syntax similar to SQL WHERE clauses. Use these clearly defined filters and selection criteria to specify conditions that are based on attributes of query tables.

Query table API methods:

Queries are run on query tables in Business Process Choreographer using the query table API. Entity-based API methods and row-based API methods are available to retrieve content from query tables.

The following entity-based methods and row-based methods are provided to run queries on query tables in Business Process Choreographer using the query table API:

Table 25. Methods for queries run on query tables

Purpose	Methods
Query contents	• queryEntities • queryRows Both methods return contents of the query table. The queryEntities method returns content based on entities and queryRows returns content based on rows.
Query the number of objects	• queryEntityCount • queryRowCount Both methods return the number of objects in the query table, while the actual number can depend on whether the entity-based or the row-based approach is taken.

Entity-based queries, using the queryEntities method and the queryEntityCount method, assume that a query table contains uniquely identifiable entities, as defined by the primary key on the primary query table.

Row-based queries, using the queryRows method and the queryRowCount method, return a result set like JDBC, which is row-based, and provides first and next methods for navigating in it. The result set that is returned when you run a query on a query table using the query table API can be compared to QueryResultSet that is returned by the query API. In general, the number of rows is greater than the number of entities that are contained in a query table. The same entity, for example, a human task which is identified by its task ID, such as TKIID, might occur multiple times in the row result set.

A specific instance that is contained in any predefined query table exists only once in a Business Process Choreographer environment. Examples of instances are human tasks and business processes. Those instances are uniquely identified using an ID or a set of IDs. This is the TKIID for instances of human tasks and the PIID for process instances.

Composite query tables are composed of a primary query table and zero or more attached query tables. Objects that are contained in composite query tables are uniquely identified by the unique ID of the objects that are contained in the primary query table. The primary query table of a composite query table determines its entity type. For example, a composite query table with the TASK primary query table contains entities of the TASK type. The one-to-one or one-to-zero relationship between the primary and attached query tables ensures that the attached query tables do not result in duplicate entities.

Entity-based queries exploit the uniquely identifiable entities of a query table, as defined by the primary key on the primary query table. A client application programmer for user interfaces is typically interested in unique instances without duplicates, for example, to display a human task once only on the user interface. Unique instances are returned if the entity-based query table API is used.

Row-based queries can return duplicate rows of the primary query table if instance-based authorization is used.

- Information from the WORK_ITEM query table is retrieved with the query. For example, if the `WI.REASON` attribute is retrieved in addition to the attributes that are defined on the query table, multiple rows qualify for the result. This is because there can be multiple reasons why a user can access an entity, such as, a task or a process instance.
- Instance-based authorization is used, and distinct is not specified. Even though work item information is not retrieved, multiple rows may be returned if instance-based authorization is used.

If the entity-based query table API is used:

- Entity-based queries are always run with the SQL distinct operator.
- Entity-based queries return a result which allows array values for work-item-related information.

Query table API parameters:

You use query table API methods to retrieve content when you run queries against a query table in Business Process Choreographer.

The following input parameters are passed to the methods of the query table API:

Table 26. Parameters of the query table API

Parameter	Optional	Type and description
Query table name	No	java.lang.String
		The unique name of the query table.
Filter options	Yes	com.ibm.bpe.api.FilterOptions
		Options which can be used to define the query. For example, a query threshold is set on this parameter to limit the number of results returned.
Authorization options	Yes	com.ibm.bpe.api.AuthorizationOptions or com.ibm.bpe.api.AdminAuthorizationOptions
		Authorization can be further constrained if instance-based authorization is used. For query tables which require role-based authorization, an instance of AdminAuthorizationOptions must be passed.
Parameters	Yes	A java.util.List of com.ibm.bpe.api.Parameter
		This parameter is used to pass user parameters, which are specified in a filter or selection criterion on a composite query table.

A query is run on one specific query table only. The relationship between multiple query tables is defined with composite query tables. In terms of the query API (as distinct from the query table API), this corresponds to database views.

Filters and selection criteria are specified in expressions during query table development using the Query Table Builder. For more information, refer to the information center topic on composite query tables and the topic on filter and search criteria of query tables. For information on the Query Table Builder, see the WebSphere Business Process Management SupportPacs site. Look for PA71 WebSphere Process Server - Query Table Builder. To access the link, see the related references section of this topic.

Query table name:

When you run a query on a query table in Business Process Choreographer, the query table name is passed as an input parameter to the methods of the query table API.

The query table name is the name of the query table on which the query is run.

- For predefined query tables, this is the name of the predefined query table.
- For composite and supplemental query tables, this is the name of the respective query table that is specified while modeling the query table. The name of a composite or supplemental query table follows the `prefix.name` naming convention, and `prefix` may not be `'IBM'`.

Both the query table name and prefix must be in uppercase. The maximum length of the query table name is 28 characters.

Filter options:

When you run a query on a query table in Business Process Choreographer, filter options can be passed as input parameters to the methods of the query table API.

An instance of the **com.ibm.bpe.api.FilterOptions** class can be passed to the query table API. The filter options allow a configuration of the query using:

- A threshold and offset (skipCount)
- Sort attributes (similar to the ORDER BY clause in an SQL query)
- An additional query filter
- The set of attributes returned, including work item information
- Other

The result set that can be obtained from a query table is specified by the definition of the query table. However, you might want to specify additional options when the query is run. The following table describes the options that can be specified as filter options using the **com.ibm.bpe.api.FilterOptions** object.

Table 27. Query table API parameters: Filter options

Option	Type	Description
Selected attributes	java.lang.String	• A comma separated list of attributes of the query table that must be returned in the result set. • If instance-based authorization is used, work item information can be retrieved by specifying attributes of the WORK_ITEM query table, prefixed with `'WI.'`. An example is `WI.REASON`. • If null is specified, all attributes of the query table are returned, without work item information.
Query filter	java.lang.String	The query filter, which filters in addition to the filters and selection criteria that are defined on the query table.
Sort attributes	java.lang.String	A comma separated list of attributes of the query table, optionally followed by ASC or DESC, for ascending or descending, respectively. This list is similar to the SQL ORDER BY clause: *sortAttributes* ::= *attribute* [ASC|DESC] [, *sortAttributes*]. If ASC or DESC is not specified, ASC is assumed. Sorting occurs in the sequence of the sort attributes. This example sorts tasks in query table TASK in descending order by state, and within the groups of the same STATE by NAME, in ascending order: `"STATE DESC, NAME ASC"`.

Table 27. Query table API parameters: Filter options (continued)

Option	Type	Description
Threshold	java.lang.Integer	Defines the maximum: • Number of rows returned if queryRows is used. • Number of entities returned if queryEntities is used. The actual number of available entities in the respective query table may exceed the threshold number of entities for the query even if the entity result set does not contain as many entities as the threshold number. This is due to technical reasons if work item information is selected. • Count returned if queryRowCount or queryEntityCount is used. The default is null which means that no threshold is set.
Skip count	java.lang.Integer	Defines the number of rows (row-based queries) or the number of entities (entity-based queries) that are skipped. As with the threshold parameter, skipCount may not be accurate for entity-based queries. Skip count is used to allow paging over a large result set. The default is null which means that no skipCount is set.
Time zone	java.util.TimeZone	The time zone that is used when converting timestamps. An example is CREATED on the predefined query table TASK. If not specified (null), the time zone on the server is used.
Locale	java.util.Locale	The locale which is used to calculate the value of the `$LOCALE` system parameter. An example usage of `$LOCALE` in a selection criterion is: `'LOCALE=$LOCALE'`.
Distinct rows	java.lang.Boolean	Used for row-based queries only. If set to true, row-based queries return distinct rows. This does not imply that unique rows are returned due to the possible multiplicity of work item information.
Query condition	setQueryCondition	This performs additional filtering on the result set. Attributes that are defined on the query table, can be referenced if the authorization is set to be required. Columns that are defined on the WORK_ITEM query table can be referenced also using the prefix `'WI.'`, for example, `WI.REASON=REASON_POTENTIAL_OWNER`.

Authorization options for the query table API:

When you run a query on a query table in Business Process Choreographer, authorization options can be passed as input parameters to the methods of the query table API.

Use an instance of the com.ibm.bpe.api.AuthorizationOptions class or the com.ibm.bpe.api.AdminAuthorizationOptions class to specify additional authorization options when the query is run.

If instance-based authorization is used, instances of the com.ibm.bpe.api.AuthorizationOptions class allow the specification of the type of work items used to identify eligible instances that are returned by the query.

An instance of the com.ibm.bpe.api.AuthorizationOptions class can be passed to the query table API if the query is run on a predefined query table that contains instance data. It can also be passed if the query is run on a composite query table with a primary query table that contains instance data and instance-based authorization is configured to be used. If the query is run on a predefined query table with template data or a composite query table with role-based authorization configured, an `EngineNotAuthorizedException` exception is thrown. In all other cases, the authorization options passed to the query table API are ignored.

Composite query tables can restrict the types of work items that are considered when identifying objects (or entities) that are contained in it. For example, if the authorization options that are passed to the query table API are configured to use everybody work items, this is only taken into account if everybody work items are defined for use on the definition of the composite query table. As a simple rule, a work item type that is not specified to be considered on the query table definition cannot be overwritten to be considered by the query table API, but a work item type that is specified to be considered on the query table definition can be overwritten not to be used. Also, the authorization type of a composite or predefined query table cannot be overwritten by the query table API.

Depending on the type of query table that is queried, different authorization option defaults apply if the authorization object is not specified or if the related attributes (everybody, individual, group, or inherited) are set to null, which is the default.

The following table shows the authorization option defaults for instance-based authorization for the query table type and work item type used.

Table 28. Query table API parameters: Authorization option defaults for instance-based authorization

Query table type	Everybody work item	Individual work item	Group work item	Inherited work item
Predefined with instance data	TRUE	TRUE	TRUE	FALSE
Predefined with template data	N/A	N/A	N/A	N/A
Composite with a primary query table with instance data	TRUE	TRUE	TRUE	TRUE

Table 28. Query table API parameters: Authorization option defaults for instance-based authorization (continued)

Query table type	Everybody work item	Individual work item	Group work item	Inherited work item
Composite with a primary query table with template data	N/A	N/A	N/A	N/A
Supplemental	N/A	N/A	N/A	N/A

N/A means that instance-based authorization is not used and, therefore, any setting on the authorization object with respect to work items is ignored.

If TRUE is specified, the resulting query will only consider the specific work item type if the query table is defined to use this type of work item. This is true for all predefined query tables with instance data, but might not be true for a composite query table. For the group work item, the latter must also be enabled on the human task container. An example of the inherited work item set to TRUE is that the administrator of a process instance may see participating human task instances that are created for that process instance.

Specify an instance of the com.ibm.bpe.api.AdminAuthorizationOptions class instead of an instance of the com.ibm.bpe.api.AuthorizationOptions class if:

- A query is run on a query table with role-based authorization. Predefined query tables with template data require role-based authorization, and composite query tables with a primary query table with template data can be configured to require role-based authorization.
- A query is run on a query table with instance data or on a composite query table with a primary query table that contains instance data. It should return the content of that query table, regardless of restrictions due to authorization for a particular user. This behavior is equivalent to using the queryAll method on the query API (as distinct from the query table API).
- A query should be executed on behalf of another user.

The following table describes how the various behaviors above are accomplished:

Table 29. Query table API parameters: AdminAuthorizationOptions

Situation	Description
onBehalfUser set to null	• If the query is run on a query table with role-based authorization, all contents of that query table are returned. • If the query is run on a query table which uses instance-based authorization, the particular objects contained in the query table are not checked for work items for a particular user. All objects that are contained in the query table are returned.
onBehalfUser set to a particular user	The query is run with the authority of the specified user, and the objects in the query table are checked against the work items for this user, if instance-based authorization is used.

If you specify com.ibm.bpe.api.AdminAuthorizationOptions, the caller must be in the BPESystemAdministrator J2EE role.

Related concepts

Authorization for query tables
Instance-based authorization, role-based authorization, or no authorization can be used when queries are run on query tables.

Parameters:

When you run a query on a query table in Business Process Choreographer, you can pass user parameters as input parameters to the methods of the query table API. In query table definitions, you can specify parameters in filters on the primary query table, on the authorization, and on the query table. Parameters can also be specified in selection criteria on attached query tables.

The system parameters, `$USER` and `$LOCALE`, are replaced at runtime in filters and selection criteria, and are not required to be passed into the query table API. The input value for the calculation of the `$LOCALE` system parameter is provided by setting the locale in the filter options.

User parameters must be passed into the query table API when the query is run. This is accomplished by passing a list of instances of the com.ibm.bpe.api.Parameter class.

The following properties must be specified on a parameter object:

Table 30. User parameters for the query table API

Property	Description
Name	The name of the parameter as used in the query table definition. The name is case sensitive.
Value	The value of the parameter. The type of the parameter must be compatible with the type of the left-hand operand of all filters and selection criteria where this parameter is used. Constants that are defined on some attributes of predefined query tables can be passed as a string, for example `STATE_READY`.

The following is an example of parameters:

```
// get the naming context and look up the Business
// Flow Manager EJB home; note that the Business Flow
// Manager EJB home should be cached for performance
// reasons; also, it is assumed that there is an EJB
// reference to the local Business Flow Manager EJB
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
(LocalBusinessFlowManagerHome)
ctx.lookup("java:comp/env/ejb/BFM");

// create the Business Flow Manager client-side stub
LocalBusinessFlowManager bfm = home.create();

// execute a query against a composite query
// table CUST.CPM with the primary query table filter
// set to 'STATE=PARAM(theState)'
EntityResultSet ers = null;
List parameterList = new ArrayList();
parameterList.add(new Parameter
("theState", new Integer(2)));
```

```
ers = bfm.queryEntities
("CUST.CPM", null, null, parameterList);

// work on the result set
// ...
```

Results of query table queries:

You use query table API methods when you run queries on a query table in Business Process Choreographer. The result of a queryEntityCount method or queryRowCount method query is a number. The queryEntities and the queryRows methods return result sets.

EntityResultSet

An instance of the com.ibm.bpe.api.EntityResultSet class is returned by the method queryEntities. An entity result set has the following properties:

Table 31. Entity result set properties of a query table API entity

Property	Description
queryTableName	Name of the query table on which the query was run.
entityTypeName	• If the query was run on a composite query table, this is the name of the primary query table. • If the query was run on a predefined query table or on a supplemental query table, this is the name of the query table, that is, the same value as the *queryTableName* property.
entityInfo	This property contains the meta information of the entities that are contained in the entity result set. A java.util.List list of the com.ibm.bpe.api.AttributeInfo objects can be retrieved on this object. This list contains the attribute names and attribute types of the information contained in the entities of this result set. Meta information about the attributes which constitute the key for these entities is also contained.
entities	A java.util.List list of entity objects.
locale	The locale that is calculated for the `$LOCALE` system parameter.

Instances of the com.ibm.bpe.api.Entity class contain the information that is retrieved from the query table query. An entity represents a uniquely identifiable object such as a task, a process instance, an activity, or an escalation. The following properties are available for entities:

Table 32. Entity properties of a query table API entity

Property	Description
entityInfo	The entityInfo object which is also contained in the entity result set.
attributeValue (*attributeName*)	The value of the specified attribute that is retrieved for this entity. The type is contained in the related AttributeInfo object of this attribute.

Table 32. Entity properties of a query table API entity (continued)

Property	Description
attributeValuesOfArray (*attributeName*)	An array of values. Use this property if the attribute info property *array* is set to true which is currently the case only if the attribute refers to work item information.

The number of entities in the entity result set is retrieved using the size() method on the list of entities.

Example: Entity-based query table API:

```
...
// the following example shows a query against
// predefined query table TASK, using the entity-based API

...
// run the query
EntityResultSet rs = bfm.queryEntities("TASK", null, null, null);

// get the entities meta information
EntityInfo ei = rs.getEntityInfo();
List atts = ei.getAttributeInfo();

// get the entities and iterate over it
Iterator entitiesIter = rs.getEntities().iterator();
while (entitiesIter.hasNext()) {

      // work on a particular entity
      Entity en = (Entity) entitiesIter.next();

      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = en.getAttributeValue(ai.getName()) ;

            // process...
      }
}
...
```

RowResultSet

An instance of the com.ibm.bpe.api.RowResultSet class is returned by the queryRows method. This type of result set is similar to a JDBC result set. A row result set has the following properties:

Table 33. Row result set properties of a query table API row

Property	Description
primaryQueryTableName	• If the query was run on a composite query table, this is the name of the primary query table. • If the query was run on a predefined query table or on a supplemental query table, this is the name of the query table, that is, the same value as property *queryTableName*.
attributeInfo	This property contains a list of the com.ibm.bpe.api.AttributeInfo objects that describe the meta information for this result set. AttributeInfo objects contain the attribute names and attribute types of the information. Meta data about keys is not contained because row result sets do not have a key.

Table 33. Row result set properties of a query table API row (continued)

Property	Description
attributeValue	The value of the specified attribute that was retrieved for this row. The type is contained in the related AttributeInfo object of this attribute.
next, first, last, previous	The row result set is navigated using these methods. Compare its usage to iterators, enumerations, or JDBC result sets.

The number of rows in the row result set is retrieved using the size() method on the list of rows.

Example: Row-based query table API:

```
...
// the following example shows a query against
// predefined query table TASK, using the entity-based API
...
// run the query
RowResultSet rs = bfm.queryRows("TASK", null, null, null);

// get the entities meta information
List atts = rs.getAttributeInfo();

// get the entities and iterate over it
while (rs.next()) {

      // work on a particular row
      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = rs.getAttributeValue(ai.getName()) ;

            // process...
      }
}
...
```

Query table queries for meta data retrieval

Queries are run on query tables in Business Process Choreographer using the query table API. Methods are available to retrieve meta data from query tables.

The following methods are provided to retrieve meta data when you run queries on query tables in Business Process Choreographer using the query table API:

Table 34. Methods for meta data retrieval on query tables

Purpose	Method
Return the meta data of a specific query table	getQueryTableMetaData
Return a list of query table meta data with specific properties	findQueryTableMetaData
Return contents of a query table, based on entities, and a subset of the meta data for the selected attributes	queryEntities
Return contents of a query table, based on rows, and a subset of the meta data for the selected attributes	queryRows

Meta data of query tables consists of data that relates to structure and data that relates to internationalization.

The following table shows the meta data that is related to the structure of a query table.

Table 35. Meta data related to query table structure

Meta data	Description	Returned by getQuery-TableMetaData	Returned by findQuery-TableMetaData	Returned by queryEntities	Returned by queryRows
Query table name	The name of the query table	Yes	Yes	Yes	Yes
Primary query table name	For supplemental and predefined query tables, name of the query table; for composite query tables the name of the primary query table	Yes	Yes	Yes	Yes
Kind	The type of query table: predefined, composite, or supplemental	Yes	Yes	No	No
Authorization	The authorization that is defined on the query table: • Use of work items • Instance-based, role-based, or no authorization	Yes	Yes	No	No
Defined attributes	Meta data of the attributes that are defined on the query table	Yes	Yes	No. Meta data of the selected attributes is returned.	No. Meta data of the selected attributes is returned.
Key attributes	Key attributes of the query table	Yes	Yes	Yes	No. Not applicable to row-based queries.

The following table shows the meta data that is related to the internationalization of a query table.

Table 36. Meta data related to query table internationalization

Meta data	Description	Returned by getQuery-TableMetaData	Returned by findQuery-TableMetaData	Returned by queryEntities	Returned by queryRows
locales[]	Locales for which display names and descriptions of the query table and attributes are defined.	Yes	Yes	No	No
Locale	Value of the `$LOCALE` system parameter which results from the locale that is passed to the API.	Yes	Yes	Yes	Yes
Display name and description of the query table	Display names and descriptions for the query table, which are provided for all defined locales.	Yes	Yes	No	No

Table 36. Meta data related to query table internationalization (continued)

Meta data	Description	Returned by getQuery-TableMetaData	Returned by findQuery-TableMetaData	Returned by queryEntities	Returned by queryRows
Display names and descriptions of the attributes	Display names and descriptions for the attributes, which are provided for all defined locales.	Yes	Yes	No	No

All EJB query table API methods which return query table meta data accept a locale parameter, such as `FilterOptions.setLocale` and `MetaDataOptions.setLocale`. This parameter should be set to the Java locale that the client uses to present information to the user. This locale parameter is used to calculate the value of the `$LOCALE` system parameter, which can be used in filters and selection criteria. The locale that is returned contains the actual Java locale that is used for `$LOCALE`.

If the display names and descriptions of a specific query table are retrieved, pass getLocale to the related methods to get the display names and descriptions in the same locale as the descriptions of the tasks. For example, these descriptions are attached using a selection criterion of `'LOCALE=$LOCALE'`.

Example:

```
// the following example shows how meta data for a particular
// composite query table can be retrieved

...
// run the query
MetaDataOptions mdo = new MetaDataOptions("TASK", null, false, new Locale("en_US"));
List list = bfm.findQueryTableMetaData(mdo);

// to get the meta data of a specific query table
// use bfm.getQueryTableMetaData(...)

// iterate through the list of query tables that have TASK as primary query table
// => at least one query table is returned: the predefined query table TASK

Iterator iter = list.iterator();
while (iter.hasNext()) {
 QueryTableMetaData md = (QueryTableMetaData) iter.next();
 Locale effectiveLocale = md.getLocale();
 String queryTableDisplayName = md.getDisplayName(effectiveLocale);
 System.out.println("found query table: " + queryTableDisplayName);
 List attributesList = md.getAttributeMetaData();
 Iterator attrIter = attributesList.iterator();
 while (attrIter.hasNext()) {
  AttributeMetaData amd = (AttributeMetaData) attrIter.next();
  String attributeDisplayName = amd.getDisplayName(effectiveLocale);
  System.out.println("\tattribute:" + attributeDisplayName);
 }
}
```

Internationalization for query table meta data

Internationalization is supported for query table meta data.

Display names and descriptions can be provided for composite query tables in different locales. For example, a composite query table can define a display name for the query table in the `en_US` locale, the `de` locale, and in the default locale. This is done when the query table is developed using the Query Table Builder. To

deploy query tables with localized display names and descriptions, the `-deploy jarFile` option must be used when the query table is deployed on the Business Process Choreographer container.

In terms of locale handling, the behavior of the query table API methods, queryEntities and queryRows, and the meta data methods of the query table API, getQueryTableMetaData and findQueryTableMetaData, is similar to that provided by Java resource bundles.

To make the display names and descriptions of the query table meta data consistent with the contents of the query table, the value of the `$LOCALE` system parameter depends on the locales for which display names and descriptions are specified on the query table.

Example:

Consider the following scenario of a client which displays task lists or process lists and creates a request to query a query table.

- The client did not specify the locale it uses to present information to the user. It is likely that the application is not enabled for different languages.
  - A default locale is specified on the query table for display names and descriptions. This is the case for all composite and supplemental query tables that are built with the current version of the Query Table Builder. Therefore, the value of `$LOCALE` is set to `default`.
  - The query table does not specify display names or descriptions on the query table for the default locale. This is the case for all predefined query tables and for all query tables that are deployed using the `-deploy qtdFile` option. The value of `$LOCALE` is based on the Java resource bundle method.
- The client specified the locale to use to present information to the user. For example, this is the case when the REST API for query tables is used.
  - Display names and descriptions are specified on the query table. The Java resource bundle method is used to calculate the value of `$LOCALE`, based on the locale that is passed in by the client.
  - Display names and descriptions are not specified on the query table. The value of `$LOCALE` is set to the value that is passed in by the client.

Query tables and query performance

Query tables introduce a clean programming model for developing client applications that retrieve lists of human tasks and business processes in Business Process Choreographer. Query tables have a positive effect on query performance. Options for query tables are described, as well as the query table API parameters that impact on query performance. Information is also provided on other factors that impact on performance.

Query response times on query tables depend mainly on the authorization options, filters, and selection criteria that are used. The following are some general performance tips to consider.

- Authorization options have considerable performance impact. Enable authorization using as few options as is possible, such as individual and group work items. Avoid using inherited work items. The authorization options can be further restricted when the query is run. Also, if not needed, specify that authorization using work items is not required.

- If authorization using work items is required, specify an authorization filter. For example, to allow only objects in the query table with a potential owner work item, use WI.REASON=REASON_POTENTIAL_OWNER.
- Filtering on the primary query table is efficient, for example, to allow only tasks in the ready state in the query table where TASK is the primary query table.
- Filters on the query table, as well as query filters, which are filters that are passed when the query is run, are less efficient as primary filters in terms of performance.
- Avoid, where possible, using parameters in filters and selection criteria.
- Avoid using `LIKE` operators in filters and selection criteria.

Composite query table definition

The following table provides information about the query performance impact of options that are defined on composite query tables. It also provides information other topics related to composite query table definitions. The impact given in column Performance Impact is an average performance impact, actual impact observations may vary.

Table 37. Query performance impact of composite query table options

Object or topic	Performance impact	Description
Query table filter	Negative	Filters on query tables are the filters with the highest negative impact on query performance. These filters typically cannot use any defined indexes in the database.
Primary query table filter	Positive	A filter on the primary query table provides high performance filtering at a very early stage of the query result set calculation. It is suggested to restrict the contents of the query table using a primary query table filter.
Authorization filter	Positive	A filter on authorization can improve the performance of the query, such as how the primary query table filter improves it. If possible, an authorization filter should be applied. For example, if reader work items should not be considered, specify `WI.REASON=REASON_READER`.
Selection criteria	None	Some primary query table to attached query table relationships require the definition of a selection criterion in order to meet the one-to-one or one-to-zero relationship. A selection criterion typically has low performance impact because it is evaluated for a small numbers of rows only.
Parameters	None	Currently, using parameters in query tables has no negative performance impact. Nevertheless, parameters should be used only if needed.
Instance-based authorization	Negative	If instance-based authorization is used, each object in the query table must be checked against the existence of a work item. Work items are represented as entries in the WORK_ITEM query table. This verification affects performance.

Table 37. Query performance impact of composite query table options (continued)

Object or topic	Performance impact	Description
Instance-based authorization: • everybody • individuals • groups • inherited	Negative	Each type of work item that is specified for use in the query table has a performance impact. Applications with high volume queries should only use individual and group work items, or only one of those. Inherited work items are usually not required, in particular when defining task lists that return human tasks representing to-dos. They should be used only when it is clear that they are needed, for example, to return lists of tasks that belong to a business process where a person might have read access based on the authorization for the enclosing business process.
Role-based authorization or no authorization	None	If role-based authorization or no authorization is used, checks against work items are not made.
Number of defined attributes	Currently none	Currently, the number of attributes contained in a query table has no impact on performance. Nevertheless, only those attributes that are needed should be part of a query table.

Query table API

The following table provides information about the query performance impact of options that are specified on the query table API. The impact given in the Performance impact column is an average performance impact; actual impact observations may vary.

Table 38. Query performance impact of query table API options

Option	Performance impact	Description
Selected attributes	Negative (less is better)	The number of attributes that are selected when a query is run on a query table impacts on the number that need to be processed both by the database and by the Business Process Choreographer query table runtime. Also, for composite query tables, information from attached query tables need be retrieved only if those are either specified by the selected attributes or referenced by the query table filter or by the query filter.
Query filter	Negative	If specified, the query filter currently has the same performance impact as the query table filter. However, it is a good practice if filters are specified on query tables rather than passed into the query table API.
Sort attributes	Negative	The sorting of query result sets is an expensive operation, and database optimizations are restricted if sorting is used. If not needed, sorting should be avoided. Most applications require sorting, however.
Threshold	Positive	The specification of a threshold can greatly improve the performance of queries. It is a best practice to always specify a threshold.
Skip count	Negative	Skipping a particular number of objects in the query result set is expensive and should be done only if required, for example when paging over a query result.

Table 38. Query performance impact of query table API options (continued)

Option	Performance impact	Description
Time zone	None	The time zone setting has no performance impact.
Locale	None	The locale setting has no performance impact.
Distinct rows	Negative	Using distinct in queries has some performance impact but might be necessary in order to retrieve non-duplicate rows. This option impacts only on row based queries and is ignored otherwise.
Count queries	Positive	If only the total number of entities or the number of rows for a particular query is needed, that is, the contents are not needed for all entries of the query table, the method queryEntityCount or queryRowCount should be used. The Business Process Choreographer runtime can apply optimizations that are valid only for count queries.

Other considerations

Other factors to consider with regard to performance are:

Table 39. Query table performance: Other considerations

Item	Description
Number of query tables on the system	The number of query tables which are deployed on a Business Process Choreographer container does not influence the performance of query table queries. Also, currently, it does not influence the navigation of business process instances, nor does it have impact on claim or complete operations on human tasks. Due to maintainability, keep the number of query tables at a reasonable level. Typically, one query table represents one task list or process list which is displayed on the user interface.
Database tuning	Although optimized SQL is used to access the contents of a query table, database tuning best practices need still to be implemented on a Business Process Choreographer database: • Database memory should be set to a maximum, taking into account other processes that are running on the database server, as well as hardware constraints. • Statistics on the database must be up-to-date, and should be updated on a regular basis. Typically, those procedures are already implemented in large topologies. For example, collect database statistics for the optimizer once per week in order to reflect changes of the data in the database. • Database systems provide tools to reorganize (or defragment) the data containers. The physical layout of the data in a database can also influence query performance and access paths of queries. • Optimal indexes are the key for good query performance. Business Process Choreographer comes with predefined indexes which are optimized for both process navigation and query performance of typical scenarios. In customized environments, additional indexes may be necessary in order to support high volume task or process list queries. Use tools provided by the database in order to support the queries which are run on a query table.

Business Process Choreographer EJB query API

Use the query method or the queryAll method of the service API to retrieve stored information about business processes and tasks.

The query method can be called by all users, and it returns the properties of the objects for which work items exist. The queryAll method can be called only by users who have one of the following J2EE roles: BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor, or TaskSystemMonitor. This method returns the properties of all the objects that are stored in the database.

All API queries are mapped to SQL queries. The form of the resulting SQL query depends on the following aspects:

- Whether the query was invoked by someone with one of the J2EE roles.
- The objects that are queried. Predefined database views are provided for you to query the object properties.
- The insertion of a from clause, join conditions, and user-specific conditions for access control.

You can include both custom properties and variable properties in queries. If you include several custom properties or variable properties in your query, this results in self-joins on the corresponding database table. Depending on your database system, these `query()` calls might have performance implications.

You can also store queries in the Business Process Choreographer database using the createStoredQuery method. You provide the query criteria when you define the stored query. The criteria are applied dynamically when the stored query runs, that is, the data is assembled at runtime. If the stored query contains parameters, these are also resolved when the query runs.

For more information on the Business Process Choreographer APIs, see the Javadoc in the com.ibm.bpe.api package for process-related methods and in the com.ibm.task.api package for task-related methods.

Related reference

Database views for Business Process Choreographer
This reference information describes the columns in the predefined database views.

Syntax of the API query method

The syntax of the Business Process Choreographer API queries is similar to SQL queries. A query can include a select clause, a where clause, an order-by clause, a skip-tuples parameter, a threshold parameter and a time-zone parameter.

The syntax of the query depends on the object type. The following table shows the syntax for each of the different object types.

Table 40.

Object	Syntax
Process template	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`

Table 40. (continued)

Object	Syntax
Task template	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Business-process and task-related data	`QueryResultSet query (java.lang.String selectClause, java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer skipTuples java.lang.Integer threshold, java.util.TimeZone timezone);`

Select clause:

The select clause in the query function identifies the object properties that are to be returned by a query.

The select clause describes the query result. It specifies a list of names that identify the object properties (columns of the result) to return. Its syntax is similar to the syntax of an SQL SELECT clause; use commas to separate parts of the clause. Each part of the clause must specify a column from one of the predefined views. The columns must be fully specified by view name and column name. The columns returned in the QueryResultSet object appear in the same order as the columns specified in the select clause.

The select clause does not support SQL aggregation functions, such as AVG(), SUM(), MIN(), or MAX().

To select the properties of multiple name-value pairs, such as custom properties and properties of variables that can be queried, add a one-digit counter to the view name. This counter can take the values 1 through 9.

Examples of select clauses

- ″WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON″

  Gets the object types of the associated objects and the assignment reasons for the work items.
- ″DISTINCT WORK_ITEM.OBJECT_ID″

  Gets all of the IDs of objects, without duplicates, for which the caller has a work item.
- ″ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON″

  Gets the names of the activities the caller has work items for and their assignment reasons.
- ″ACTIVITY.STATE, PROCESS_INSTANCE.STARTER″

  Gets the states of the activities and the starters of their associated process instances.
- ″DISTINCT TASK.TKIID, TASK.NAME″

  Gets all of the IDs and names of tasks, without duplicates, for which the caller has a work item.
- ″TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE″

  Gets the values of the custom properties that are specified further in the where clause.

- ″QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE

  Gets the values of the properties of variables that can be queried. These parts are specified further in the where clause.
- ″COUNT(DISTINCT TASK.TKIID)″

  Counts the number of work items for unique tasks that satisfy the where clause.

Where clause:

The where clause in the query function describes the filter criteria to apply to the query domain.

The syntax of a where clause is similar to the syntax of an SQL WHERE clause. You do not need to explicitly add an SQL from clause or join predicates to the API where clause, these constructs are added automatically when the query runs. If you do not want to apply filter criteria, you must specify `null` for the where clause.

The where-clause syntax supports:
- Keywords: AND, OR, NOT
- Comparison operators: =, <=, <, <>, >,>=, LIKE

  The LIKE operation supports the wildcard characters that are defined for the queried database.
- Set operation: IN

The following rules also apply:
- Specify object ID constants as `ID('string-rep-of-oid')`.
- Specify binary constants as `BIN('UTF-8 string')`.
- Use symbolic constants instead of integer enumerations. For example, instead of specifying an activity state expression `ACTIVITY.STATE=2`, specify `ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY`.
- If the value of the property in the comparison statement contains single quotation marks (′), double the quotation marks, for example, `"TASK_CPROP.STRING_VALUE='d''automatisation'"`.
- Refer to properties of multiple name-value pairs, such as custom properties, by adding a one-digit suffix to the view name. For example: `"TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'"`
- Specify time-stamp constants as `TS('yyyy-mm-ddThh:mm:ss')`. To refer to the current date, specify `CURRENT_DATE` as the timestamp.

  You must specify at least a date or a time value in the timestamp:
  - If you specify a date only, the time value is set to zero.
  - If you specify a time only, the date is set to the current date.
  - If you specify a date, the year must consist of four digits; the month and day values are optional. Missing month and day values are set to 01. For example, `TS('2003')` is the same as `TS('2003-01-01T00:00:00')`.
  - If you specify a time, these values are expressed in the 24-hour system. For example, if the current date is 1 January 2003, `TS('T16:04')` or `TS('16:04')` is the same as `TS('2003-01-01T16:04:00')`.

Examples of where clauses
- Comparing an object ID with an existing ID

  `"WORK_ITEM.WIID = ID('_WI:800c00ed.df8d7e7c.feffff80.38')"`

This type of where clause is usually created dynamically with an existing object ID from a previous call. If this object ID is stored in a *wiid1* variable, the clause can be constructed as:

```
"WORK_ITEM.WIID = ID('" + wiid1.toString() + "')"
```

- Using time stamps

```
"ACTIVITY.STARTED >= TS('2002-06-1T16.00.00')"
```

- Using symbolic constants

```
"WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER"
```

- Using Boolean values true and false

```
"ACTIVITY.BUSINESS_RELEVANCE = TRUE"
```

- Using custom properties

```
"TASK_CPROP1.NAME = 'prop1' AND " TASK_CPROP1.STRING_VALUE = 'v1' AND
 TASK_CPROP2.NAME = 'prop2' AND " TASK_CPROP2.STRING_VALUE = 'v2'"
```

Order-by clause:

The order-by clause in the query function specifies the sort criteria for the query result set.

You can specify a list of columns from the views by which the result is sorted. These columns must be fully qualified by the name of the view and the column.

The order-by clause syntax is similar to the syntax of an SQL order-by clause; use commas to separate each part of the clause. You can also specify `ASC` to sort the columns in ascending order, and `DESC` to sort the columns in descending order. If you do not want to sort the query result set, you must specify `null` for the order-by clause.

Sort criteria are applied on the server, that is, the locale of the server is used for sorting. If you specify more than one column, the query result set is ordered by the values of the first column, then by the values of the second column, and so on. You cannot specify the columns in the order-by clause by position as you can with an SQL query.

Examples of order-by clauses

- ″PROCESS_TEMPLATE.NAME″

  Sorts the query result alphabetically by the process-template name.
- ″PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC″

  Sorts the query result by the creation date and, for a specific date, sorts the results alphabetically by the process-instance name in reverse order.
- ″ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE″

  Sorts the query result by the activity owner, then the activity-template name, and then the state of the activity.

Skip-tuples parameter:

The skip-tuples parameter specifies the number of query-result-set tuples from the beginning of the query result set that are to be ignored and not to be returned to the caller in the query result set.

Use this parameter with the threshold parameter to implement paging in a client application, for example, to retrieve the first 20 items, then the next 20 items, and so on.

If this parameter is set to `null` and the threshold parameter is not set, all of the qualifying tuples are returned.

Example of a skip-tuples parameter

- new Integer(5)

  Specifies that the first five qualifying tuples are not to be returned.

Threshold parameter:

The threshold parameter in the query function restricts the number of objects returned from the server to the client in the query result set.

Because query result sets in production scenarios can contain thousands or even millions of items, specify a value for the threshold parameter. If you set the threshold parameter accordingly, the database query is faster and less data needs to transfer from the server to the client. The threshold parameter can be useful, for example, in a graphical user interface where only a small number of items should be displayed at one time.

If this parameter is set to `null` and the skip-tuples parameter is not set, all of the qualifying objects are returned.

Example of a threshold parameter

- new Integer(50)

  Specifies that 50 qualifying tuples are to be returned.

Timezone parameter:

The time-zone parameter in the query function defines the time zone for time-stamp constants in the query.

Time zones can differ between the client that starts the query and the server that processes the query. Use the time-zone parameter to specify the time zone of the time-stamp constants used in the where clause, for example, to specify local times. The dates returned in the query result set have the same time zone that is specified in the query.

If the parameter is set to `null`, the timestamp constants are assumed to be Coordinated Universal Time (UTC) times.

Examples of time-zone parameters

- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null,
                 (Integer)null,
                 java.util.TimeZone.getDefault() );
  ```

  Returns object IDs for activities that started later than 17:40 local time on 1 January 2005.
- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null, (Integer)null, (TimeZone)null);
  ```

  Return object IDs for activities that started later than 17:40 UTC on 1 January 2005. This specification is, for example, 6 hours earlier in Eastern Standard Time.

Parameters in stored queries:

A stored query is a query that is stored in the database and identified by a name. The qualifying tuples are assembled dynamically when the query is run. To make stored queries reusable, you can use parameters in the query definition that are resolved at runtime.

For example, you have defined custom properties to store customer names. You can define queries to return the tasks that are associated with a particular customer, ACME Co. To query this information, the where clause in your query might look similar to the following example:

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = 'ACME Co.'";
```

To make this query reusable so that you can also search for the customer, BCME Ltd, you can use parameters for the values of the custom property. If you add parameters to the task query, it might look similar to the following example:

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = '@param1'";
```

The `@param1` parameter is resolved at runtime from the list of parameters that is passed to the query method. The following rules apply to the use of parameters in queries:

- Parameters can only be used in the where clause.
- Parameters are strings.
- Parameters are replaced at runtime using string replacement. If you need special characters you must specify these in the where clause or passed-in at runtime as part of the parameter.
- Parameter names consist of the string `@param` concatenated with an integer number. The lowest number is 1, which points to the first item in the list of parameters that is passed to the query API at runtime.
- A parameter can be used multiple times within a where clause; all occurrences of the parameter are replaced by the same value.

Related tasks

Managing stored queries
Stored queries provide a way to save queries that are run often. The stored query can be either a query that is available to all users (public query), or a query that belongs to a specific user (private query).

Query results:

A query result set contains the results of a Business Process Choreographer API query.

The elements of the result set are properties of the objects that satisfy the where clause given by the caller, and that the caller is authorized to see. You can read elements in a relative fashion using the API next method or in an absolute fashion using the first and last methods. Because the implicit cursor of a query result set is initially positioned before the first element, you must call either the first or next methods before reading an element. You can use the size method to determine the number of elements in the set.

An element of the query result set comprises the selected attributes of work items and their associated referenced objects, such as activity instances and process instances. The first attribute (column) of a QueryResultSet element specifies the value of the first attribute specified in the select clause of the query request. The second attribute (column) of a QueryResultSet element specifies the value of the second attribute specified in the select clause of the query request, and so on.

You can retrieve the values of the attributes by calling a method that is compatible with the attribute type and by specifying the appropriate column index. The numbering of the column indexes starts with 1.

Attribute type	Method
String	getString
OID	getOID
Timestamp	getTimestamp getString getTimestampAsLong
Integer	getInteger getShort getLong getString getBoolean
Boolean	getBoolean getShort getInteger getLong getString
byte[]	getBinary

Example:

The following query is run:

```
QueryResultSet resultSet = process.query("ACTIVITY.STARTED,
                                          ACTIVITY.TEMPLATE_NAME AS NAME,
                                          WORK_ITEM.WIID, WORK_ITEM.REASON",
                                          (String)null, (String)null,
                                          (Integer)null, (TimeZone)null);
```

The returned query result set has four columns:
- Column 1 is a time stamp
- Column 2 is a string
- Column 3 is an object ID
- Column 4 is an integer

You can use the following methods to retrieve the attribute values:

```
while (resultSet.next())
{
 java.util.Calendar activityStarted = resultSet.getTimestamp(1);
 String templateName = resultSet.getString(2);
 WIID wiid = (WIID) resultSet.getOID(3);
 Integer reason = resultSet.getInteger(4);
}
```

You can use the display names of the result set, for example, as headings for a printed table. These names are the column names of the view or the name defined by the AS clause in the query. You can use the following method to retrieve the display names in the example:

```
resultSet.getColumnDisplayName(1) returns "STARTED"
resultSet.getColumnDisplayName(2) returns "NAME"
resultSet.getColumnDisplayName(3) returns "WIID"
resultSet.getColumnDisplayName(4) returns "REASON"
```

User-specific access conditions

User-specific access conditions are added when the SQL SELECT statement is generated from the API query. These conditions guarantee that only those objects are returned to the caller that satisfy the condition specified by the caller and to which the caller is authorized.

The access condition that is added depends on whether the user is a system administrator.

Queries invoked by users who are not system administrators

The generated SQL WHERE clause combines the API where clause with an access control condition that is specific to the user. The query retrieves only those objects that the user is authorized to access, that is, only those objects for which the user has a work item. A work item represents the assignment of a user or user group to an authorization role of a business object, such as a task or process. If, for example, the user, John Smith, is a member of the potential owners role of a given task, a work item object exists that represents this relationship.

For example, if a user, who is not a system administrator, queries tasks, the following access condition is added to the WHERE clause if group work items are not enabled:

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'user'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

So, if John Smith wants to get a list of tasks for which he is the potential owner, the API where clause might look as follows:

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

This API where clause results in the following access condition in the SQL statement:

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JohnSmith'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

This also means that if John Smith wants to see the activities and tasks for which he is a process reader or a process administrator and for which he does not have a work item, then a property from the PROCESS_INSTANCE view must be added to the select, where, or order-by clause of the query, for example, PROCESS_INSTANCE.PIID.

If group work items are enabled, an additional access condition is added to the WHERE clause that allows a user to access objects that the group has access to.

Queries invoked by system administrators

System administrators can invoke the query method to retrieve objects that have associated work items. In this case, a join with the WORK_ITEM view is added to the generated SQL query, but no access control condition for the WORK_ITEM.OWNER_ID.

In this case, the SQL query for tasks contains the following:

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
```

queryAll queries

This type of query can be invoked only by system administrators or system monitors. Neither conditions for access control nor a join to the WORK_ITEM view are added. This type of query returns all of the data for all of the objects.

Examples of the query and queryAll methods

These examples show the syntax of various typical API queries and the associated SQL statements that are generated when the query is processed.

Example: Querying tasks in the ready state:

This example shows how to use the query method to retrieve tasks that the logged-on user can work with.

John Smith wants to get a list of the tasks that have been assigned to him. For a user to be able to work on a task, the task must be in the ready state. The logged-on user must also have a potential owner work item for the task. The following code snippet shows the query method call for this query:

```
query( "DISTINCT TASK.TKIID",
          "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )
            AND " +
          "TASK.STATE = TASK.STATE.STATE_READY AND " +
          "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

The following actions are taken when the SQL SELECT statement is generated:

- A condition for access control is added to the where clause. This example assumes that group work items are not enabled.
- Constants, such as `TASK.STATE.STATE_READY`, are replaced by their numeric values.
- A FROM clause and join conditions are added.

The following code snippet shows the SQL statement that is generated from the API query:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND    WI.REASON = 1
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

To restrict the API query to tasks for a specific process, for example, sampleProcess, the query looks as follows:

```
query( "DISTINCT TASK.TKIID",
          "PROCESS_TEMPLATE.NAME = 'sampleProcess' AND "+
          "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )
            AND " +
          "TASK.STATE = TASK.STATE.STATE_READY AND " +
          "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Example: Querying tasks in the claimed state:

This example shows how to use the query method to retrieve tasks that the logged-on user has claimed.

The user, John Smith, wants to search for tasks that he has claimed and are still in the claimed state. The condition that specifies ″claimed by John Smith″ is `TASK.OWNER = 'JohnSmith'`. The following code snippet shows the query method call for the query:

```
query( "DISTINCT TASK.TKIID",
          "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
          "TASK.OWNER = 'JohnSmith'",
          (String)null, (String)null, (Integer)null, (TimeZone)null )
```

The following code snippet shows the SQL statement that is generated from the API query:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   TA.OWNER = 'JohnSmith'
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

When a task is claimed, work items are created for the owner of the task. So, an alternative way of forming the query for John Smith's claimed tasks is to add the following condition to the query instead of using `TASK.OWNER = 'JohnSmith'`:

```
WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER
```

The query then looks like the following code snippet:

```
query( "DISTINCT TASK.TKIID",
          "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
          "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER",
          (String)null, (String)null, (Integer)null, (TimeZone)null )
```

The following actions are taken when the SQL SELECT statement is generated:

- A condition for access control is added to the where clause. This example assumes that group work items are not enabled.
- Constants, such as `TASK.STATE.STATE_READY`, are replaced by their numeric values.
- A FROM clause and join conditions are added.

The following code snippet shows the SQL statement that is generated from the API query:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   AND    WI.REASON = 4
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

John is about to go on vacation so his team lead, Anne Grant, wants to check on his current work load. Anne has system administrator rights. The query she invokes is the same as the one John invoked. However, the SQL statement that is generated is different because Anne is an administrator. The following code snippet shows the generated SQL statement:

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  TA.TKIID = WI.OBJECT_ID =
   AND    TA.STATE = 8
   AND   TA.OWNER = 'JohnSmith')
```

Because Anne is an administrator, an access control condition is not added to the WHERE clause.

Example: Querying escalations:

This example shows how to use the query method to retrieve escalations for the logged-on user.

When a task is escalated, and escalation receiver work item is created. The user, Mary Jones wants to see a list of tasks that have been escalated to her. The following code snippet shows the query method call for the query:

```
query( "DISTINCT ESCALATION.ESIID, ESCALATION.TKIID",
         "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_ESCALATION_RECEIVER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

The following actions are taken when the SQL SELECT statement is generated:

- A condition for access control is added to the where clause. This example assumes that group work items are not enabled.
- Constants, such as TASK.STATE.STATE_READY, are replaced by their numeric values.
- A FROM clause and join conditions are added.

The following code snippet shows the SQL statement that is generated from the API query:

```
 SELECT DISTINCT ESCALATION.ESIID, ESCALATION.TKIID
   FROM   ESCALATION ESC, WORK_ITEM WI
   WHERE  ESC.ESIID = WI.OBJECT_ID
   AND    WI.REASON  = 10
   AND
  ( WI.OWNER_ID = 'MaryJones' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Example: Using the queryAll method:

This example shows how to use the queryAll method to retrieve all of the activities that belong to a process template.

The queryAll method is available only to users with system administrator or system monitor rights. The following code snippet shows the queryAll method call for the query to retrieve all of the activities that belong to the process template, sampleProcess:

```
queryAll( "DISTINCT ACTIVITY.AIID",
       "PROCESS_TEMPLATE.NAME = 'sampleProcess'",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

The following code snippet shows the SQL query that is generated from the API query:

```
SELECT DISTINCT ACTIVITY.AIID
  FROM   ACTIVITY AI, PROCESS_TEMPLATE PT
  WHERE  AI.PTID = PT.PTID
  AND    PT.NAME = 'sampleProcess'
```

Because the call is invoked by an administrator, an access control condition is not added to the generated SQL statement. A join with the WORK_ITEM view is also not added. This means that the query retrieves all of the activities for the process template, including those activities without work items.

Example: Including query properties in a query:

This example shows how to use the query method to retrieve tasks that belong to a business process. The process has query properties defined for it that you want to include in the search.

For example, you want to search for all of the human tasks in the ready state that belong to a business process. The process has a query property, **customerID**, with the value CID_12345, and a namespace. The following code snippet shows the query method call for the query:

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                   PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY.NAME = 'customerID' AND " +
          " QUERY_PROPERTY.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY.NAMESPACE =
             'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

If you now want to add a second query property to the query, for example, **Priority**, with a given namespace, the query method call for the query looks as follows:

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                   PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY1.NAME = 'customerID' AND " +
          " QUERY_PROPERTY1.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY1.NAMESPACE =
                'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " QUERY_PROPERTY2.NAME = 'Priority' AND " +
          " QUERY_PROPERTY2.NAMESPACE =
                'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

If you add more than one query property to the query, you must number each of the properties that you add as shown in the code snippet. However, querying custom properties affects performance; performance decreases with the number of custom properties in the query.

Example: Including custom properties in a query:

This example shows how to use the query method to retrieve tasks that have custom properties.

For example, you want to search for all of the human tasks in the ready state that have a custom property, **customerID**, with the value `CID_12345`. The following code snippet shows the query method call for the query:

```
query ( " DISTINCT TASK.TKIID ",
          " TASK_CPROP.NAME = 'customerID' AND " +
          " TASK_CPROP.STRING_VALUE = 'CID_12345' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

If you now want to retrieve the tasks and their custom properties, the query method call for the query looks as follows:

```
query ( " DISTINCT TASK.TKIID, TASK_CPROP.NAME, TASK_CPROP.STRING_VALUE",
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

The SQL statement that is generated from this API query is shown in the following code snippet:

```
SELECT DISTINCT TA.TKIID , TACP.NAME , TACP.STRING_VALUE
   FROM  TASK TA LEFT JOIN TASK_CPROP TACP ON (TA.TKIID = TACP.TKIID),
         WORK_ITEM WI
   WHERE WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND  (WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID IS NULL AND WI.EVERYBODY = 1 )
```

This SQL statement contains an outer join between the TASK view and the TASK_CPROP view. This means that tasks that satisfy the WHERE clause are retrieved even if they do not have any custom properties.

Developing EJB client applications for business processes and human tasks

The EJB APIs provide a set of generic methods for developing EJB client applications for working with the business processes and human tasks that are installed on a WebSphere Process Server.

About this task

With these Enterprise JavaBeans (EJB) APIs, you can create client applications to do the following:

- Manage the life cycle of processes and tasks from starting them through to deleting them when they complete
- Repair activities and processes
- Manage and distribute the workload over members of a work group

The EJB APIs are provided as two stateless session enterprise beans:

- BusinessFlowManagerService interface provides the methods for business process applications
- HumanTaskManagerService interface provides the methods for task-based applications

For more information on the EJB APIs, see the Javadoc in the com.ibm.bpe.api package and the com.ibm.task.api package.

The following steps provide an overview of the actions you need to take to develop an EJB client application.

Procedure

1. Decide on the functionality that the application is to provide.
2. Decide which of the session beans that you are going to use.

   Depending on the scenarios that you want to implement with your application, you can use one, or both, of the session beans.
3. Determine the authorization authorities needed by users of the application.

   The users of your application must be assigned the appropriate authorization roles to call the methods that you include in your application, and to view the objects and the attributes of these objects that these methods return. When an instance of the appropriate session bean is created, WebSphere Application Server associates a context with the instance. The context contains information about the caller's principal ID, group membership list, and roles. This information is used to check the caller's authorization for each call.

   The Javadoc contains authorization information for each of the methods.
4. Decide how to render the application.

   The EJB APIs can be called locally or remotely.
5. Develop the application.
   a. Access the EJB API.
   b. Use the EJB API to interact with processes or tasks.
      - Query the data.
      - Work with the data.

Related concepts

Comparison of the programming interfaces for interacting with business processes and human tasks
Enterprise JavaBeans (EJB), Web service, and Java Message Service (JMS), and Representational State Transfer Services (REST) generic programming interfaces are available for building client applications that interact with business processes and human tasks. Each of these interfaces has different characteristics.

Related reference

Database views for Business Process Choreographer
This reference information describes the columns in the predefined database views.

Accessing the EJB APIs

The Enterprise JavaBeans (EJB) APIs are provided as two stateless session enterprise beans. Business process applications and task applications access the appropriate session enterprise bean through the home interface of the bean.

About this task

The BusinessFlowManagerService interface provides the methods for business process applications, and the HumanTaskManagerService interface provides the methods for task-based applications. The application can be any Java application, including another Enterprise JavaBeans (EJB) application.

Accessing the remote interface of the session bean

An EJB client application for business processes or human tasks accesses the remote interface of the session bean through the remote home interface of the bean.

About this task

The session bean can be either the BusinessFlowManager session bean for process applications or the HumanTaskManager session bean for task applications.

Procedure

1. Add a reference to the remote interface of the session bean to the application deployment descriptor. Add the reference to one of the following files:
   - The application-client.xml file, for a Java 2 Platform, Enterprise Edition (J2EE) client application
   - The web.xml file, for a Web application
   - The ejb-jar.xml file, for an Enterprise JavaBeans (EJB) application

   The reference to the remote home interface for process applications is shown in the following example:

   ```
   <ejb-ref>
    <ejb-ref-name>ejb/BusinessFlowManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
    <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
   </ejb-ref>
   ```

   The reference to the remote home interface for task applications is shown in the following example:

   ```
   <ejb-ref>
    <ejb-ref-name>ejb/HumanTaskManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.task.api.HumanTaskManagerHome</home>
    <remote>com.ibm.task.api.HumanTaskManager</remote>
   </ejb-ref>
   ```

   If you use WebSphere Integration Developer to add the EJB reference to the deployment descriptor, the binding for the EJB reference is automatically created when the application is deployed. For more information on adding EJB references, refer to the WebSphere Integration Developer documentation.
2. Package the generated stubs with your application.
   a. For process applications, package the <*install_root*>/ProcessChoreographer/client/bpe137650.jar file with the enterprise archive (EAR) file of your application.
   b. For task applications, package the <*install_root*>/ProcessChoreographer/client/task137650.jar file with the EAR file of your application.
   c. Set the class path parameter in the manifest file of the application module to include the JAR file.

      The application module can be a J2EE application, a Web application, or an EJB application.
3. Decide on how you are going to provide definitions of business objects.

   To work with business objects in a remote client application, you need to have access to the corresponding schemas for the business objects (XSD or WSDL files) that are used to interact with a process or task. Access to these files can be provided in one of the following ways:
   - If the client application does not run in a J2EE managed environment, package the files with the client application's EAR file.

- If the client application is a Web application or an EJB client in a managed J2EE environment, either package the files with the client application's EAR file or leverage remote artifact loading.
  a. Use the Business Process Choreographer EJB API createMessage and the ClientObjectWrapper.getObject methods to load the remote business object definitions from the corresponding application on the server transparently.
  b. Use the Service Data Object Programming API to create or read a business object as part of an already instantiated business object. Do this by using the commonj.sdo.DataObject.createDataObject or getDataObject methods on the DataObject interface.
  c. When you want to create a business object as the value for a business object's property that is typed using the XML schema any or anyType, use the Business Object services to create or read your business object. To do this, you must set the remote artifact loader context to point to the application that the schemas will be loaded from. Then you can use the appropriate Business Object services.

     For example, create a business object, where ″ApplicationName″ is the name of the application that contains your business object definitions.

```
BOFactory bofactory = (BOFactory) new
   ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");

com.ibm.wsspi.al.ALContext.setContext
   ("RALTemplateName", "ApplicationName");
try {
     DataObject dataObject = bofactory.create("uriName", "typeName" );
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

     For example, read XML input, where ″ApplicationName″ is the name of the application that contains your business object definitions.

```
BOXMLSerializer serializerService =
  (BOXMLSerializer) new ServiceManager().locateService
    ("com/ibm/websphere/bo/BOXMLSerializer");
ByteArrayInputStream input = new ByteArrayInputStream(<?xml?>..);

com.ibm.wsspi.al.ALContext.setContext
          ("RALTemplateName", "ApplicationName");
try {
     BOXMLDocument document = serializerService.readXMLDocument(input);
     DataObject dataObject = document.getDataObject();
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

4. Locate the remote home interface of the session bean through the Java Naming and Directory Interface (JNDI).

   The following example shows this step for a process application:

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the BusinessFlowManager bean
Object result =
       initialContext.lookup("java:comp/env/ejb/BusinessFlowManagerHome");

// Convert the lookup result to the proper type
   BusinessFlowManagerHome processHome =
          (BusinessFlowManagerHome)javax.rmi.PortableRemoteObject.narrow
          (result,BusinessFlowManagerHome.class);
```

The remote home interface of the session bean contains a create method for EJB objects. The method returns the remote interface of the session bean.

5. Access the remote interface of the session bean.

   The following example shows this step for a process application:

   ```
   BusinessFlowManager process = processHome.create();
   ```

   Access to the session bean does not guarantee that the caller can perform all of the actions provided by the bean; the caller must also be authorized for these actions. When an instance of the session bean is created, a context is associated with the instance of the session bean. The context contains the caller's principal ID, group membership list, and indicates whether the caller has one of the Business Process Choreographer J2EE roles. The context is used to check the caller's authorization for each call, even when administrative security is not set. If administrative security is not set, the caller's principal ID has the value `UNAUTHENTICATED`.

6. Call the business functions exposed by the service interface.

   The following example shows this step for a process application:

   ```
   process.initiate("MyProcessModel",input);
   ```

   Calls from applications are run as transactions. A transaction is established and ended in one of the following ways:

   - Automatically by WebSphere Application Server (the deployment descriptor specifies TX_REQUIRED).
   - Explicitly by the application. You can bundle application calls into one transaction:

     ```
     // Obtain user transaction interface
     UserTransaction transaction=
           (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

     // Begin a transaction
     transaction.begin();

     // Applications calls ...

     // On successful return, commit the transaction
     transaction.commit();
     ```

   Tip: To prevent database lock conflicts, avoid running statements similar to the following in parallel:

   ```
   // Obtain user transaction interface
   UserTransaction transaction=
         (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

   transaction.begin();

   //read the activity instance
   process.getActivityInstance(aiid);
   //claim the activity instance
   process.claim(aiid);

   transaction.commit();
   ```

   The getActivityInstance method and other read operations set a read lock. In this example, a read lock on the activity instance is upgraded to an update lock on the activity instance. This can result in a database deadlock when these transactions are run in parallel.

Example

Here is an example of how steps 3 through 5 might look for a task application.

```
//Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the remote home interface of the HumanTaskManager bean
Object result =
        initialContext.lookup("java:comp/env/ejb/HumanTaskManagerHome");

//Convert the lookup result to the proper type
HumanTaskManagerHome taskHome =
          (HumanTaskManagerHome)javax.rmi.PortableRemoteObject.narrow
          (result,HumanTaskManagerHome.class);

...
//Access the remote interface of the session bean.
HumanTaskManager task = taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

Accessing the local interface of the session bean

An EJB client application for business processes or human tasks accesses the local interface of the session bean through the local home interface of the bean.

About this task

The session bean can be either the BusinessFlowManager session bean for process applications or the HumanTaskManager session bean for human task applications.

Procedure

1. Add a reference to the local interface of the session bean to the application deployment descriptor. Add the reference to one of the following files:
   - The application-client.xml file, for a Java 2 Platform, Enterprise Edition (J2EE) client application
   - The web.xml file, for a Web application
   - The ejb-jar.xml file, for an Enterprise JavaBeans (EJB) application

   The reference to the local home interface for process applications is shown in the following example:

   ```
   <ejb-local-ref>
    <ejb-ref-name>ejb/LocalBusinessFlowManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
    <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
   </ejb-local-ref>
   ```

   The reference to the local home interface for task applications is shown in the following example:

   ```
   <ejb-local-ref>
    <ejb-ref-name>ejb/LocalHumanTaskManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
    <local>com.ibm.task.api.LocalHumanTaskManager</local>
   </ejb-local-ref>
   ```

   If you use WebSphere Integration Developer to add the EJB reference to the deployment descriptor, the binding for the EJB reference is automatically created when the application is deployed. For more information on adding EJB references, refer to the WebSphere Integration Developer documentation.

2. Locate the local home interface of the session bean through the Java Naming and Directory Interface (JNDI).

   The following example shows this step for a process application:

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the local home interface of the BusinessFlowManager bean

 LocalBusinessFlowManagerHome processHome =
      (LocalBusinessFlowManagerHome)initialContext.lookup
      ("java:comp/env/ejb/LocalBusinessFlowManagerHome");
```

   The local home interface of the session bean contains a create method for EJB objects. The method returns the local interface of the session bean.
3. Access the local interface of the session bean.

   The following example shows this step for a process application:

```
LocalBusinessFlowManager process = processHome.create();
```

   Access to the session bean does not guarantee that the caller can perform all of the actions provided by the bean; the caller must also be authorized for these actions. When an instance of the session bean is created, a context is associated with the instance of the session bean. The context contains the caller's principal ID, group membership list, and indicates whether the caller has one of the Business Process Choreographer J2EE roles. The context is used to check the caller's authorization for each call, even when administrative security is not set. If administrative security is not set, the caller's principal ID has the value `UNAUTHENTICATED`.
4. Call the business functions exposed by the service interface.

   The following example shows this step for a process application:

```
process.initiate("MyProcessModel",input);
```

   Calls from applications are run as transactions. A transaction is established and ended in one of the following ways:
   - Automatically by WebSphere Application Server (the deployment descriptor specifies TX_REQUIRED).
   - Explicitly by the application. You can bundle application calls into one transaction:

```
// Obtain user transaction interface
UserTransaction transaction=
      (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

// Begin a transaction
transaction.begin();

// Applications calls ...

// On successful return, commit the transaction
transaction.commit();
```

   Tip: To prevent database deadlocks, avoid running statements similar to the following in parallel:

```
// Obtain user transaction interface
UserTransaction transaction=
      (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

transaction.begin();

//read the activity instance
process.getActivityInstance(aiid);
```

```
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

The getActivityInstance method and other read operations set a read lock. In this example, a read lock on the activity instance is upgraded to an update lock on the activity instance. This can result in a database deadlock when these transactions are run in parallel

Example

Here is an example of how steps 2 through 4 might look for a task application.

```
//Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the local home interface of the HumanTaskManager bean
LocalHumanTaskManagerHome taskHome =
        (LocalHumanTaskManagerHome)initialContext.lookup
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Access the local interface of the session bean
LocalHumanTaskManager task = taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

Querying business-process and task-related objects

The client applications work with business-process and task-related objects. You can query business-process and task-related objects in the database to retrieve specific properties of these objects.

About this task

During the configuration of Business Process Choreographer, a relational database is associated with both the business process container and the task container. The database stores all of the template (model) and instance (runtime) data for managing business processes and tasks. You use SQL-like syntax to query this data.

You can perform a one-off query to retrieve a specific property of an object. You can also save queries that you use often and include these stored queries in your application.

Related reference

Database views for Business Process Choreographer
This reference information describes the columns in the predefined database views.

Filtering data using variables in queries

A query result returns the objects that match the query criteria. You might want to filter these results on the values of variables.

About this task

You can define variables that are used by a process at runtime in its process model. For these variables, you declare which parts can be queried.

For example, John Smith, calls his insurance company's service number to find out the progress of his insurance claim for his damaged car. The claims administrator uses the customer ID to the find the claim.

Procedure

1. Optional: List the properties of the variables in a process that can be queried.

   Use the process template ID to identify the process. You can skip this step if you know which variables can be queried.

   ```
   List variableProperties = process.getQueryProperties(ptid);
   for (int i = 0; i < variableProperties.size(); i++)
   {
      QueryProperty queryData = (QueryProperty)variableProperties.get(i);
      String variableName = queryData.getVariableName();
      String name         = queryData.getName();
      int mappedType      = queryData.getMappedType();
      ...
   }
   ```

2. List the process instances with variables that match the filter criteria.

   For this process, the customer ID is modeled as part of the variable customerClaim that can be queried. You can therefore use the customer's ID to find the claim.

   ```
   QueryResultSet result = process.query
       ("PROCESS_INSTANCE.NAME, QUERY_PROPERTY.STRING_VALUE",
        "QUERY_PROPERTY.VARIABLE_NAME = 'customerClaim' AND " +
        "QUERY_PROPERTY.NAME = 'customerID' AND " +
        "QUERY_PROPERTY.STRING_VALUE like 'Smith%'",
         (String)null, (Integer)null,
         (Integer)null, (TimeZone)null );
   ```

   This action returns a query result set that contains the process instance names and the values of the customer IDs for customers whose IDs start with Smith.

Managing stored queries

Stored queries provide a way to save queries that are run often. The stored query can be either a query that is available to all users (public query), or a query that belongs to a specific user (private query).

About this task

A stored query is a query that is stored in the database and identified by a name. A private and a public stored query can have the same name; private stored queries from different owners can also have the same name.

You can have stored queries for business process objects, task objects, or a combination of these two object types.

Related concepts

Parameters in stored queries
A stored query is a query that is stored in the database and identified by a name. The qualifying tuples are assembled dynamically when the query is run. To make stored queries reusable, you can use parameters in the query definition that are resolved at runtime.

Managing public stored queries:

Public stored queries are created by the system administrator. These queries are available to all users.

About this task

As the system administrator, you can create, view, and delete public stored queries. If you do not specify a user ID in the API call, it is assumed that the stored query is a public stored query.

Procedure

1. Create a public stored query.

   For example, the following code snippet creates a stored query for process instances and saves it with the name CustomerOrdersStartingWithA.

   ```
   process.createStoredQuery("CustomerOrdersStartingWithA",
               "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
               "PROCESS_INSTANCE.NAME LIKE 'A%'",
               "PROCESS_INSTANCE.NAME",
                (Integer)null, (TimeZone)null);
   ```

   The result of the stored query is a sorted list of all the process-instance names that begin with the letter A and their associated process instance IDs (PIID).
2. Run the query defined by the stored query.

   ```
   QueryResultSet result = process.query("CustomerOrdersStartingWithA",
                 new Integer(0), null);
   ```

   This action returns the objects that fulfill the criteria. In this case, all of the customer orders that begin with A.
3. List the names of the available public stored queries.

   The following code snippet shows how to limit the list of returned queries to just the public queries.

   ```
   String[] storedQuery = process.getStoredQueryNames(StoredQueryData.KIND_PUBLIC);
   ```
4. Optional: Check the query that is defined by a specific stored query.

   A stored private query can have the same name as a stored public query. If these names are the same, the private stored query is returned. The following code snippet shows how to return only the public query with the specified name. If you use the Human Task Manager API to retrieve information about a stored query, use StoredQuery for the returned object instead of StoredQueryData.

   ```
   StoredQueryData storedQuery = process.getStoredQuery
      (StoredQueryData.KIND_PUBLIC, "CustomerOrdersStartingWithA");
   String selectClause = storedQuery.getSelectClause();
   String whereClause = storedQuery.getWhereClause();
   String orderByClause = storedQuery.getOrderByClause();
   Integer threshold = storedQuery.getThreshold();
   String owner = storedQuery.getOwner();
   ```
5. Delete a public stored query.

   The following code snippet shows how to delete the stored query that you created in step 1.

   ```
   process.deleteStoredQuery("CustomerOrdersStartingWithA");
   ```

Managing private stored queries for other users:

Private queries can be created by any user. These queries are available only to the owner of the query and the system administrator.

About this task

As the system administrator, you can manage private stored queries that belong to a specific user.

Procedure

1. Create a private stored query for the user ID Smith.

   For example, the following code snippet creates a stored query for process instances and saves it with the name CustomerOrdersStartingWithA for the user ID Smith.

   ```
   process.createStoredQuery("Smith", "CustomerOrdersStartingWithA",
                "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
                "PROCESS_INSTANCE.NAME LIKE 'A%'",
                "PROCESS_INSTANCE.NAME",
                 (Integer)null, (TimeZone)null,
                 (List)null, (String)null);
   ```

   The result of the stored query is a sorted list of all the process-instance names that begin with the letter A and their associated process instance IDs (PIID).

2. Run the query defined by the stored query.

   ```
   QueryResultSet result = process.query
                         ("Smith", "CustomerOrdersStartingWithA",
                           (Integer)null, (Integer)null, (List)null);
   new Integer(0));
   ```

   This action returns the objects that fulfill the criteria. In this case, all of the customer orders that begin with A.

3. Get a list of the names of the private queries that belong to a specific user.

   For example, the following code snippet shows how to get a list of private queries that belongs to the user Smith.

   ```
   String[] storedQuery = process.getStoredQueryNames("Smith");
   ```

4. View the details of a specific query.

   The following code snippet shows how to view the details of the CustomerOrdersStartingWithA query that is owned by the user Smith.

   ```
   StoredQueryData storedQuery = process.getStoredQuery
      ("Smith", "CustomerOrdersStartingWithA");
   String selectClause = storedQuery.getSelectClause();
   String whereClause = storedQuery.getWhereClause();
   String orderByClause = storedQuery.getOrderByClause();
   Integer threshold = storedQuery.getThreshold();
   String owner = storedQuery.getOwner();
   ```

   If you use the Human Task Manager API to retrieve information about a stored query, use StoredQuery for the returned object instead of StoredQueryData.

5. Delete a private stored query.

   The following code snippet shows how to delete a private query that is owned by the user Smith.

   ```
   process.deleteStoredQuery("Smith", "CustomerOrdersStartingWithA");
   ```

Working with your private stored queries:

If you are not a system administrator, you can create, run, and delete your own private stored queries. You can also use the public stored queries that the system administrator created.

Procedure

1. Create a private stored query.

   For example, the following code snippet creates a stored query for process instances and saves it with a specific name. If a user ID is not specified, it is assumed that the stored query is a private stored query for the logged-on user.

```
process.createStoredQuery("CustomerOrdersStartingWithA",
            "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
            "PROCESS_INSTANCE.NAME LIKE 'A%'",
            "PROCESS_INSTANCE.NAME",
             (Integer)null, (TimeZone)null);
```

This query returns a sorted list of all the process-instance names that begin with the letter A and their associated process instance IDs (PIID).

2. Run the query defined by the stored query.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",
              new Integer(0));
```

This action returns the objects that fulfill the criteria. In this case, all of the customer orders that begin with A.

3. Get a list of the names of the stored queries that the logged-on user can access.

The following code snippet shows how to get both the public and the private stored queries that the user can access.

```
String[] storedQuery = process.getStoredQueryNames();
```

4. View the details of a specific query.

The following code snippet shows how to view the details of the CustomerOrdersStartingWithA query that is owned by the user Smith.

```
StoredQueryData storedQuery = process.getStoredQuery
   ("CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();
```

If you use the Human Task Manager API to retrieve information about a stored query, use StoredQuery for the returned object instead of StoredQueryData.

5. Delete a private stored query.

The following code snippet shows how to delete a private stored query.

```
process.deleteStoredQuery("CustomerOrdersStartingWithA");
```

Developing applications for business processes

A business process is a set of business-related activities that are invoked in a specific sequence to achieve a business goal. Examples are provided that show how you might develop applications for typical actions on processes.

About this task

A business process can be either a microflow or a long-running process:

- Microflows are short running business processes that are executed synchronously. After a very short time, the result is returned to the caller.
- Long-running, interruptible processes are executed as a sequence of activities that are chained together. The use of certain constructs in a process causes interruptions in the process flow, for example, invoking a human task, invoking a service using an synchronous binding, or using timer-driven activities.

  Parallel branches of the process are usually navigated asynchronously, that is, activities in parallel branches are executed concurrently. Depending on the type and the transaction setting of the activity, an activity can be run in its own transaction.

Required roles for actions on process instances

Access to the BusinessFlowManager interface does not guarantee that the caller can perform all of the actions on a process. The caller must be logged on to the client application with a role that is authorized to perform the action.

The following table shows the actions on a process instance that a specific role can take.

Action	Caller's principal role		
	Reader	**Starter**	**Administrator**
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspend			x
transferWorkItem			x

Required roles for actions on business-process activities

Access to the BusinessFlowManager interface does not guarantee that the caller can perform all of the actions on an activity. The caller must be logged on to the client application with a role that is authorized to perform the action.

The following table shows the actions on an activity instance that a specific role can take.

Action	Caller's principal role				
	Reader	Editor	Potential owner	Owner	Administrator
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x
getCustomProperties	x	x	x	x	x
getCustomProperty	x	x	x	x	x
getCustomPropertyNames	x	x	x	x	x
getFaultMessage	x	x	x	x	x
getFaultNames	x	x	x	x	x
getInputMessage	x	x	x	x	x
getOutputMessage	x	x	x	x	x
getVariable	x	x	x	x	x
getVariableNames	x	x	x	x	x
getInputVariableNames	x	x	x	x	x
getOutputVariableNames	x	x	x	x	x
getWorkItems	x	x	x	x	x
setCustomProperty		x		x	x
setFaultMessage		x		x	x
setOutputMessage		x		x	x
setVariable					x
transferWorkItem				x To potential owners or administrators only	x

Managing the life cycle of a business process

A process instance comes into existence when a Business Process Choreographer API method that can start a process is invoked. The navigation of the process instance continues until all of its activities are in an end state. Various actions can be taken on the process instance to manage its life cycle.

About this task

Examples are provided that show how you might develop applications for the following typical life-cycle actions on processes.

Starting business processes:

The way in which a business process is started depends on whether the process is a microflow or a long-running process. The service that starts the process is also important to the way in which a process is started; the process can have either a unique starting service or several starting services.

About this task

Examples are provided that show how you might develop applications for typical scenarios for starting microflows and long-running processes.

Running a microflow that contains a unique starting service:

A microflow can be started by a receive activity or a pick activity. The starting service is unique if the microflow starts with a receive activity or when the pick activity has only one onMessage definition.

About this task

If the microflow implements a request-response operation, that is, the process contains a reply, you can use the call method to run the process passing the process template name as a parameter in the call.

If the microflow is a one-way operation, use the sendMessage method to run the process. This method is not covered in this example.

Procedure

1. Optional: List the process templates to find the name of the process you want to run.

   This step is optional if you already know the name of the process.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   The results are sorted by name. The query returns an array containing the first 50 sorted templates that can be started by the call method.
2. Start the process with an input message of the appropriate type.

   When you create the message, you must specify its message type name so that the message definition is contained.

   ```
   ProcessTemplateData template = processTemplates[0];
   //create a message for the single starting receive activity
   ClientObjectWrapper input = process.createMessage
                              (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }

   //run the process
   ClientObjectWrapper output = process.call(template.getName(), input);
   DataObject myOutput = null;
   ```

```
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

This action creates an instance of the process template, CustomerTemplate, and passes some customer data. The operation returns only when the process is complete. The result of the process, OrderNo, is returned to the caller.

Running a microflow that contains a non-unique starting service:

A microflow can be started by a receive activity or a pick activity. The starting service is not unique if the microflow starts with a pick activity that has multiple onMessage definitions.

About this task

If the microflow implements a request-response operation, that is, the process contains a reply, you can use the call method to run the process passing the ID of the starting service in the call.

If the microflow is a one-way operation, use the sendMessage method to run the process. This method is not covered in this example.

Procedure

1. Optional: List the process templates to find the name of the process you want to run.

   This step is optional if you already know the name of the process.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   The results are sorted by name. The query returns an array containing the first 50 sorted templates that can be started as microflows.

2. Determine the starting service to be called.

   This example uses the first template that is found.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
           process.getStartActivities(template.getID());
   ```

3. Start the process with an input message of the appropriate type.

   When you create the message, you must specify its message type name so that the message definition is contained.

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input =
         process.createMessage(activity.getServiceTemplateID(),
                               activity.getActivityTemplateID(),
                               activity.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //run the process
   ```

```
ClientObjectWrapper output = process.call(activity.getServiceTemplateID(),
                                          activity.getActivityTemplateID(),
                                          input);
//check the output of the process, for example, an order number
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

This action creates an instance of the process template, CustomerTemplate, and passes some customer data. The operation returns only when the process is complete. The result of the process, OrderNo, is returned to the caller.

Starting a long-running process that contains a unique starting service:

If the starting service is unique, you can use the initiate method and pass the process template name as a parameter. This is the case when the long-running process starts with either a single receive or pick activity and when the single pick activity has only one onMessage definition.

Procedure

1. Optional: List the process templates to find the name of the process you want to start.

   This step is optional if you already know the name of the process.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

   The results are sorted by name. The query returns an array containing the first 50 sorted templates that can be started by the initiate method.

2. Start the process with an input message of the appropriate type.

   When you create the message, you must specify its message type name so that the message definition is contained. If you specify a process-instance name, it must not start with an underscore. If a process-instance name is not specified, the process instance ID (PIID) in String format is used as the name.

   ```
   ProcessTemplateData template = processTemplates[0];
   //create a message for the single starting receive activity
   ClientObjectWrapper input = process.createMessage
                               (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //start the process
   PIID piid = process.initiate(template.getName(), "CustomerOrder", input);
   ```

   This action creates an instance, CustomerOrder, and passes some customer data. When the process starts, the operation returns the object ID of the new process instance to the caller.

   The starter of the process instance is set to the caller of the request. This person receives a work item for the process instance. The process administrators, readers, and editors of the process instance are determined and receive work

items for the process instance. The follow-on activity instances are determined. These are started automatically or, if they are human task, receive, or pick activities, work items are created for the potential owners.

Starting a long-running process that contains a non-unique starting service:

A long-running process can be started through multiple initiating receive or pick activities. You can use the initiate method to start the process. If the starting service is not unique, for example, the process starts with multiple receive or pick activities, or a pick activity that has multiple onMessage definitions, then you must identify the service to be called.

Procedure

1. Optional: List the process templates to find the name of the process you want to start.

   This step is optional if you already know the name of the process.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

   The results are sorted by name. The query returns an array containing the first 50 sorted templates that can be started as long-running processes.

2. Determine the starting service to be called.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
           process.getStartActivities(template.getID());
   ```

3. Start the process with an input message of the appropriate type.

   When you create the message, you must specify its message type name so that the message definition is contained. If you specify a process-instance name, it must not start with an underscore. If a process-instance name is not specified, the process instance ID (PIID) in String format is used as the name.

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input = process.createMessage
                               (activity.getServiceTemplateID(),
                                activity.getActivityTemplateID(),
                                activity.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //start the process
   PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                                activity.getActivityTemplateID(),
                                input);
   ```

   This action creates an instance and passes some customer data. When the process starts, the operation returns the object ID of the new process instance to the caller.

   The starter of the process instance is set to the caller of the request and receives a work item for the process instance. The process administrators, readers, and editors of the process instance are determined and receive work items for the process instance. The follow-on activity instances are determined. These are

started automatically or, if they are human task, receive, or pick activities, work items are created for the potential owners.

Suspending and resuming a business process:

You can suspend long-running, top-level process instance while it is running and resume it again to complete it.

Before you begin

The caller must be an administrator of the process instance or a business process administrator. To suspend a process instance, it must be in the running or failing state.

About this task

You might want to suspend a process instance, for example, so that you can configure access to a back-end system that is used later in the process. When the prerequisites for the process are met, you can resume the process instance. You might also want to suspend a process to fix a problem that is causing the process instance to fail, and then resume it again when the problem is fixed.

Procedure

1. Get the running process, CustomerOrder, that you want to suspend.
```
ProcessInstanceData processInstance =
                    process.getProcessInstance("CustomerOrder");
```
2. Suspend the process instance.
```
PIID piid = processInstance.getID();
process.suspend( piid );
```
This action suspends the specified top-level process instance. The process instance is put into the suspended state. Subprocesses with the autonomy attribute set to `child` are also suspended if they are in the running, failing, terminating, or compensating state. Inline tasks that are associated with this process instance are also suspended; stand-alone tasks associated with this process instance are not suspended.

   In this state, activities that are started can still be finished but no new activities are activated, for example, a human task activity in the claimed state can be completed.
3. Resume the process instance.
```
process.resume( piid );
```
This action puts the process instance and its subprocesses into the states they had before they were suspended.

Restarting a business process:

You can restart a process instance that is in the finished, terminated, failed, or compensated state.

Before you begin

The caller must be an administrator of the process instance or a business process administrator.

About this task

Restarting a process instance is similar to starting a process instance for the first time. However, when a process instance is restarted, the process instance ID is known and the input message for the instance is available.

If the process has more than one receive activity or pick activity (also known as a receive choice activity) that can create the process instance, all of the messages that belong to these activities are used to restart the process instance. If any of these activities implement a request-response operation, the response is sent again when the associated reply activity is navigated.

Procedure

1. Get the process that you want to restart.
   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```
2. Restart the process instance.
   ```
   PIID piid = processInstance.getID();
   process.restart( piid );
   ```
   This action restarts the specified process instance.

Terminating a process instance:

Sometimes, it is necessary for someone with process administrator authorization to terminate a top-level process instance that is known to be in an unrecoverable state. Because a process instance terminates immediately, without waiting for any outstanding subprocesses or activities, you should terminate a process instance only in exceptional situations.

Procedure

1. Retrieve the process instance that is to be terminated.
   ```
   ProcessInstanceData processInstance =
          process.getProcessInstance("CustomerOrder");
   ```
2. Terminate the process instance.

   If you terminate a process instance, you can terminate the process instance with or without compensation.

   To terminate the process instance with compensation:
   ```
   PIID piid = processInstance.getID();
   process.forceTerminate(piid, CompensationBehaviour.INVOKE_COMPENSATION);
   ```
   To terminate the process instance without compensation:
   ```
   PIID piid = processInstance.getID();
   process.forceTerminate(piid);
   ```
   If you terminate the process instance with compensation, the compensation of the process is run as if a fault had occurred on the top-level scope. If you terminate the process instance without compensation, the process instance is terminated immediately without waiting for activities, to-do tasks, or inline invocation tasks to end normally.

   Applications that are started by the process and standalone tasks that are related to the process are not terminated by the force terminate request. If these applications are to be terminated, you must add statements to your process application that explicitly terminate the applications started by the process.

Deleting process instances:

Completed process instances are automatically deleted from the Business Process Choreographer database if the corresponding property is set for the process template in the process model. You might want to keep process instances in your database, for example, to query data from process instances that are not written to the audit log. However, stored process instance data does not only impact disk space and performance but also prevents process instances that use the same correlation set values from being created. Therefore, you should regularly delete process instance data from the database.

About this task

To delete a process instance, you need process administrator rights and the process instance must be a top-level process instance.

The following example shows how to delete all of the finished process instances.

Procedure

1. List the process instances that are finished.

   ```
   QueryResultSet result =
        process.query("DISTINCT PROCESS_INSTANCE.PIID",
                      "PROCESS_INSTANCE.STATE =
                              PROCESS_INSTANCE.STATE.STATE_FINISHED",
                       (String)null, (Integer)null, (TimeZone)null);
   ```

   This action returns a query result set that lists process instances that are finished.
2. Delete the process instances that are finished.

   ```
   while (result.next() )
   {
      PIID piid = (PIID) result.getOID(1);
      process.delete(piid);
   }
   ```

   This action deletes the selected process instance and its inline tasks from the database.

Processing human task activities

Human task activities in business processes are assigned to various people in your organization through work items. When a process is started, work items are created for the potential owners.

About this task

When a human task activity is activated, both an activity instance and an associated to-do task are created. Handling of the human task activity and the work item management is delegated to Human Task Manager. Any state change of the activity instance is reflected in the task instance and vice versa.

A potential owner claims the activity. This person is responsible for providing the relevant information and completing the activity.

Procedure

1. List the activities belonging to a logged-on person that are ready to be worked on:

   ```
   QueryResultSet result =
        process.query("ACTIVITY.AIID",
                      "ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
   ```

```
                    ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
                    WORK_ITEM.REASON =
                         WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                    (String)null, (Integer)null, (TimeZone)null);
```

This action returns a query result set that contains the activities that can be worked on by the logged-on person.

2. Claim the activity to be worked on:

```
if (result.size() > 0)
{
 result.first();
 AIID aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
    activityInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

When the activity is claimed, the input message of the activity is returned.

3. When work on the activity is finished, complete the activity. The activity can be completed either successfully or with a fault message. If the activity is successful, an output message is passed. If the activity is unsuccessful, the activity is put into the failed or stopped state and a fault message is passed. You must create the appropriate messages for these actions. When you create the message, you must specify the message type name so that the message definition is contained.

   a. To complete the activity successfully, create an output message.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the activity
process.complete(aiid, output);
```

   This action sets an output message that contains the order number.

   b. To complete the activity when a fault occurs, create a fault message.

```
//retrieve the faults modeled for the human task activity
List faultNames = process.getFaultNames(aiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      process.createMessage(aiid, faultNames.get(0) );

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
```

```
}

process.complete(aiid, myFault,(String)faultNames.get(0) );
```

This action sets the activity in either the failed or the stopped state. If the **continueOnError** parameter for the activity in the process model is set to true, the activity is put into the failed state and the navigation continues. If the **continueOnError** parameter is set to false and the fault is not caught on the surrounding scope, the activity is put into the stopped state. In this state the activity can be repaired using force complete or force retry.

Processing a single person workflow

Some workflows are performed by only one person, for example, ordering books from an online bookstore. This type of workflow has no parallel paths. The completeAndClaimSuccessor API supports the processing of this type of workflow.

About this task

In an online bookstore, the purchaser completes a sequence of actions to order a book. This sequence of actions can be implemented as a series of human task activities (to-do tasks). If the purchaser decides to order several books, this is equivalent to claiming the next human task activity. This type of workflow is also known as *page flow* because user interface definitions are associated with the activities to control the flow of the dialogs in the user interface.

The completeAndClaimSuccessor API completes a human task activity and claims the next one in the same process instance for the logged-on person. It returns information about the next claimed activity, including the input message to be worked on. Because the next activity is made available within the same transaction of the activity that completed, the transactional behavior of all the human task activities in the process model must be set to `participates`.

Compare this example with the example that uses both the Business Flow Manager API and the Human Task Manager API.

Procedure

1. Claim the first activity in the sequence of activities.

```
//
//Query the list of activities that can be claimed by the logged-on user
//
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                  "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
                   ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
                   ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
                   WORK_ITEM.REASON =
                        WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                   (String)null, (Integer)null, (TimeZone)null);
...
//
//Claim the first activity
//
if (result.size() > 0)
{
 result.first();
 AIID aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
```

```
    activityInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

When the activity is claimed, the input message of the activity is returned.

2. When work on the activity is finished, complete the activity, and claim the next activity.

   To complete the activity, an output message is passed. When you create the output message, you must specify the message type name so that the message definition is contained.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the activity and claim the next one
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   This action sets an output message that contains the order number and claims the next activity in the sequence. If `AutoClaim` is set for successor activities and if there are multiple paths that can be followed, all of the successor activities are claimed and a random activity is returned as the next activity. If there are no more successor activities that can be assigned to this user, `Null` is returned.

   If the process contains parallel paths that can be followed and these paths contain human task activities for which the logged-on user is a potential owner of more than one of these activities, a random activity is claimed automatically and returned as the next activity.

3. Work on the next activity.

```
String name = successor.getActivityName();

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

4. Continue with step 2 to complete the activity.

Related tasks

Processing a single person workflow that includes human tasks
Some workflows are performed by only one person, for example, ordering books from an online bookstore. This example shows how to implement the sequence of actions for ordering the book as a series of human task activities (to-do tasks). Both the Business Flow Manager and the Human Task Manager APIs are used to process the workflow.

Sending a message to a waiting activity

You can use inbound message activities (receive activities, onMessage in pick activities, onEvent in event handlers) to synchronize a running process with events from the "outside world". For example, the receipt of an e-mail from a customer in response to a request for information might be such an event.

About this task

You can use originating tasks to send the message to the activity.

Procedure

1. List the activity service templates that are waiting for a message from the logged-on user in a process instance with a specific process instance ID.

   ```
   ActivityServiceTemplateData[] services = process.getWaitingActivities(piid);
   ```

2. Send a message to the first waiting service.

   It is assumed that the first service is the one that you want serve. The caller must be a potential starter of the activity that receives the message, or an administrator of the process instance.

   ```
   VTID vtid = services[0].getServiceTemplateID();
   ATID atid = services[0].getActivityTemplateID();
   String inputType = services[0].getInputMessageTypeName();

   // create a message for the service to be called
     ClientObjectWrapper message =
           process.createMessage(vtid,atid,inputMessageTypeName);
     DataObject myMessage = null;
     if ( message.getObject()!= null && message.getObject() instanceof DataObject )
     {
       myMessage = (DataObject)message.getObject();
       //set the strings in the message, for example, chocolate is to be ordered
       myMessage.setString("Order", "chocolate");
     }

     // send the message to the waiting activity
     process.sendMessage(vtid, atid, message);
   }
   ```

   This action sends the specified message to the waiting activity service and passes some order data.

   You can also specify the process instance ID to ensure that the message is sent to the specified process instance. If the process instance ID is not specified, the message is sent to the activity service, and the process instance that is identified by the correlation values in the message. If the process instance ID is specified, the process instance that is found using the correlation values is checked to ensure that it has the specified process instance ID.

Handling events

An entire business process and each of its scopes can be associated with event handlers that are invoked if the associated event occurs. Event handlers are similar to receive or pick activities in that a process can provide Web service operations using event handlers.

About this task

You can invoke an event handler any number of times as long as the corresponding scope is running. In addition, multiple instances of an event handler can be activated concurrently.

The following code snippet shows how to get the active event handlers for a given process instance and how to send an input message.

Procedure

1. Determine the data of the process instance ID and list the active event handlers for the process.

```
ProcessInstanceData processInstance =
       process.getProcessInstance( "CustomerOrder2711");
EventHandlerTemplateData[] events = process.getActiveEventHandlers(
                                      processInstance.getID() );
```

2. Send the input message.

   This example uses the first event handler that is found.

```
EventHandlerTemplateData event = null;
if ( events.length > 0 )
{
    event = events[0];

    // create a message for the service to be called
    ClientObjectWrapper input = process.createMessage(
    event.getID(), event.getInputMessageTypeName());

    if (input.getObject() != null && input.getObject() instanceof DataObject )
    {
       DataObject inputMessage = (DataObject)input.getObject();
       // set content of the message, for example, a customer name, order number
       inputMessage.setString("CustomerName", "Smith");
       inputMessage.setString("OrderNo", "2711");

        // send the message
       process.sendMessage( event.getProcessTemplateName(),
                            event.getPortTypeNamespace(),
                            event.getPortTypeName(),
                            event.getOperationName(),
     input );
     }
 }
```

   This action sends the specified message to the active event handler for the process.

Analyzing the results of a process

A process can expose Web services operations that are modeled as Web Services Description Language (WSDL) one-way or request-response operations. The results of long-running processes with one-way interfaces cannot be retrieved using the getOutputMessage method, because the process has no output. However, you can query the contents of variables, instead.

About this task

The results of the process are stored in the database only if the process template from which the process instance was derived does not specify automatic deletion of the derived process instances.

Procedure

Analyze the results of the process, for example, check the order number.

```
QueryResultSet result = process.query
                       ("PROCESS_INSTANCE.PIID",
                        "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
                         PROCESS_INSTANCE.STATE =
                                 PROCESS_INSTANCE.STATE.STATE_FINISHED",
                         (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  PIID piid = (PIID) result.getOID(1);
  ClientObjectWrapper output = process.getOutputMessage(piid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject() instanceof DataObject )
  {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
  }
}
```

Repairing activities

A long-running process can contain activities that are also long running. These activities might encounter uncaught errors and go into the stopped state. An activity in the running state might also appear to be not responding. In both of these cases, a process administrator can act on the activity in a number of ways so that the navigation of the process can continue.

About this task

The Business Process Choreographer API provides the forceRetry and forceComplete methods for repairing activities. Examples are provided that show how you might add repair actions for activities to your applications.

Forcing the completion of an activity:

Activities in long-running processes can sometimes encounter faults. If these faults are not caught by a fault handler in the enclosing scope and the associated activity template specifies that the activity stops when an error occurs, the activity is put into the stopped state so that it can be repaired. In this state, you can force the completion of the activity.

About this task

You can also force the completion of activities in the running state if, for example, an activity is not responding.

Additional requirements exist for certain types of activities.

Human task activities
: You can pass parameters in the force-complete call, such as the message that should have been sent or the fault that should have been raised.

Script activities

You cannot pass parameters in the force-complete call. However, you must set the variables that need to be repaired.

Invoke activities

You can also force the completion of invoke activities that call an asynchronous service that is not a subprocess if these activities are in the running state. You might want to do this, for example, if the asynchronous service is called and it does not respond.

Procedure

1. List the stopped activities in the stopped state.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

   This action returns the stopped activities for the CustomerOrder process instance.

2. Complete the activity, for example, a stopped human task activity.

   In this example, an output message is passed.

```
if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);
   ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper output =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( output.getObject()!= null && output.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)output.getObject();
     //set the parts in your message, for example, an order number
     myMessage.setInt("OrderNo", 4711);
   }

   boolean continueOnError = true;
   process.forceComplete(aiid, output, continueOnError);
}
```

   This action completes the activity. If an error occurs, the **continueOnError** parameter determines the action to be taken if a fault is provided with the forceComplete request.

   In the example, **continueOnError** is true. This value means that if a fault is provided, the activity is put into the failed state. The fault is propagated to the enclosing scopes of the activity until it is either handled or the process scope is reached. The process is then put into the failing state and it eventually reaches the failed state.

Retrying the execution of a stopped activity:

If an activity in a long-running process encounters an uncaught fault in the enclosing scope and if the associated activity template specifies that the activity stops when an error occurs, the activity is put into the stopped state so that it can be repaired. You can retry the execution of the activity.

About this task

You can set variables that are used by the activity. With the exception of script activities, you can also pass parameters in the force-retry call, such as the message that was expected by the activity.

Procedure

1. List the stopped activities.
```
QueryResultSet result =
      process.query("DISTINCT ACTIVITY.AIID",
                    "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                     PROCESS_INSTANCE.NAME='CustomerOrder'",
                     (String)null, (Integer)null, (TimeZone)null);
```
   This action returns the stopped activities for the CustomerOrder process instance.
2. Retry the execution of the activity, for example, a stopped human task activity.
```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper input =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
    {
      myMessage = (DataObject)input.getObject();
      //set the strings in your message, for example, chocolate is to be ordered
      myMessage.setString("OrderNo", "chocolate");
    }

   boolean continueOnError = true;
   process.forceRetry(aiid, input, continueOnError);
}
```
   This action retries the activity. If an error occurs, the **continueOnError** parameter determines the action to be taken if an error occurs during processing of the forceRetry request.

   In the example, **continueOnError** is true. This means that if an error occurs during processing of the forceRetry request, the activity is put into the failed state. The fault is propagated to the enclosing scopes of the activity until it is either handled or the process scope is reached. The process is then put into the failing state and a fault handler on the process level is run before the process state ends in the failed state.

BusinessFlowManagerService interface

The BusinessFlowManagerService interface exposes business-process functions that can be called by a client application.

The methods that can be called by the BusinessFlowManagerService interface depend on the state of the process or the activity and the authorization of the person that uses the application containing the method. The main methods for manipulating business process objects are listed here. For more information about these methods and the other methods that are available in the BusinessFlowManagerService interface, see the Javadoc in the com.ibm.bpe.api package.

Process templates

A process template is a versioned, deployed, and installed process model that contains the specification of a business process. It can be instantiated and started by issuing appropriate requests, for example, sendMessage(). The execution of the process instance is driven automatically by the server.

Table 41. API methods for process templates

Method	Description
getProcessTemplate	Retrieves the specified process template.
queryProcessTemplates	Retrieves process templates that are stored in the database.

Process instances

The following API methods are related to starting process instances.

Table 42. API methods are related to starting process instances

Method	Description
call	Creates and runs a microflow.
callWithReplyContext	Creates and runs a microflow with a unique starting service or a long-running process with a unique starting service from the specified process template. The call waits asynchronously for the result.
callWithUISettings	Creates and runs a microflow and returns the output message and the client user interface (UI) settings.
initiate	Creates a process instance and initiates processing of the process instance. Use this method for long-running processes. You can also use this method for microflows that you want to fire and forget.
sendMessage	Sends the specified message to the specified activity service and process instance. If a process instance with the same correlation set values does not exist, it is created. The process can have either unique or non-unique starting services.
getStartActivities	Returns information about the activities that can start a process instance from the specified process template.
getActivityServiceTemplate	Retrieves the specified activity service template.

Table 43. API methods for controlling the life cycle of process instances

Method	Description
suspend	Suspends the execution of a long-running, top-level process instance that is in the running or failing state.

Table 43. API methods for controlling the life cycle of process instances (continued)

Method	Description
resume	Resumes the execution of a long-running, top-level process instance that is in the suspended state.
restart	Restarts a long-running, top-level process instance that is in the finished, failed, or terminated state.
forceTerminate	Terminates the specified top-level process instance, its subprocesses with child autonomy, and its running, claimed, or waiting activities.
delete	Deletes the specified top-level process instance and its subprocesses with child autonomy.
query	Retrieves the properties from the database that match the search criteria.

Activities

For invoke activities, you can specify in the process model that these activities continue in error situations. If the continueOnError flag is set to false and an unhandled error occurs, the activity is put into the stopped state. A process administrator can then repair the activity. The continueOnError flag and the associated repair functions can, for example, be used in a long-running process where an invoke activity fails occasionally, but the effort required to model compensation and fault handling is too high.

The following methods are available for working with and repairing activities.

Table 44. API methods for controlling the life cycle of activity instances

Method	Description
claim	Claims a ready activity instance for a user to work on the activity.
cancelClaim	Cancels the claim of the activity instance.
complete	Completes the activity instance.
completeAndClaimSuccessor	Completes the activity instance and claims the next one in the same process instance for the logged-on person.
forceComplete	Forces the completion of the following: • An activity instance that is in the running or stopped state. • A human task activity that is in the state ready or claimed. • A wait activity in state waiting.
forceRetry	Forces the repetition of the following: • An activity instance that is in the running or stopped state. • A human task activity that is in the state ready or claimed.

Table 44. API methods for controlling the life cycle of activity instances (continued)

Method	Description
query	Retrieves the properties from the database that match the search criteria.

Variables and custom properties

The interface provides a get and a set method to retrieve and set values for variables. You can also associate named properties with, and retrieve named properties from, process and activity instances. Custom property names and values must be of the java.lang.String type.

Table 45. API methods for variables and custom properties

Method	Description
getVariable	Retrieves the specified variable.
setVariable	Sets the specified variable.
getCustomProperty	Retrieves the named custom property of the specified activity or process instance.
getCustomProperties	Retrieves the custom properties of the specified activity or process instance.
getCustomPropertyNames	Retrieves the names of the custom properties for the specified activity or process instance.
setCustomProperty	Stores custom-specific values for the specified activity or process instance.

Developing applications for human tasks

A task is the means by which components invoke humans as services or by which humans invoke services. Examples of typical applications for human tasks are provided.

About this task

For more information on the Human Task Manager API, see the Javadoc in the com.ibm.task.api package.

Starting an invocation task that invokes a synchronous interface

An invocation task is associated with a Service Component Architecture (SCA) component. When the task is started, it invokes the SCA component. Start an invocation task synchronously only if the associated SCA component can be called synchronously.

About this task

Such an SCA component can, for example, be implemented as a microflow or as a simple Java class.

This scenario creates an instance of a task template and passes some customer data. The task remains in the running state until the two-way operation returns. The result of the task, OrderNo, is returned to the caller.

Procedure

1. Optional: List the task templates to find the name of the invocation task you want to run.

   This step is optional if you already know the name of the task.

   ```
   TaskTemplate[] taskTemplates = task.queryTaskTemplates
   ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
    "TASK_TEMPL.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   The results are sorted by name. The query returns an array containing the first 50 sorted originating templates.
2. Create an input message of the appropriate type.

   ```
   TaskTemplate template = taskTemplates[0];

   // create a message for the selected task
   ClientObjectWrapper input = task.createInputMessage( template.getID());
   DataObject myMessage = null ;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the parts in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   ```
3. Create the task and run the task synchronously.

   For a task to run synchronously, it must be a two-way operation. The example uses the createAndCallTask method to create and run the task.

   ```
   ClientObjectWrapper output = task.createAndCallTask( template.getName(),
                                                        template.getNamespace(),
                                                        input);
   ```
4. Analyze the result of the task.

   ```
   DataObject myOutput = null;
   if ( output.getObject() != null && output.getObject() instanceof DataObject )
   {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
   }
   ```

Starting an invocation task that invokes an asynchronous interface

An invocation task is associated with a Service Component Architecture (SCA) component. When the task is started, it invokes the SCA component. Start an invocation task asynchronously only if the associated SCA component can be called asynchronously.

About this task

Such an SCA component can, for example, be implemented as a long-running process or a one-way operation.

This scenario creates an instance of a task template and passes some customer data.

Procedure

1. Optional: List the task templates to find the name of the invocation task you want to run.

   This step is optional if you already know the name of the task.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
  ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
   "TASK_TEMPL.NAME",
    new Integer(50),
    (TimeZone)null);
```

The results are sorted by name. The query returns an array containing the first 50 sorted originating templates.

2. Create an input message of the appropriate type.

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. Create the task and run it asynchronously.

   The example uses the createAndStartTask method to create and run the task.

```
task.createAndStartTask( template.getName(),
                         template.getNamespace(),
                         input,
                         (ReplyHandlerWrapper)null);
```

Creating and starting a task instance

This scenario shows how to create an instance of a task template that defines a collaboration task (also known as a *human task* in the API) and start the task instance.

Procedure

1. Optional: List the task templates to find the name of the collaboration task you want to run.

   This step is optional if you already know the name of the task.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_HUMAN",
 "TASK_TEMPL.NAME",
  new Integer(50),
  (TimeZone)null);
```

   The results are sorted by name. The query returns an array containing the first 50 sorted task templates.

2. Create an input message of the appropriate type.

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. Create and start the collaboration task; a reply handler is not specified in this example.

   The example uses the createAndStartTask method to create and start the task.

```
TKIID tkiid = task.createAndStartTask( template.getName(),
                                       template.getNamespace(),
                                       input,
                                       (ReplyHandlerWrapper)null);
```

Work items are created for the people concerned with the task instance. For example, a potential owner can claim the new task instance.

4. Claim the task instance.

```
ClientObjectWrapper input2 = task.claim(tkiid);
DataObject taskInput = null ;
if ( input2.getObject()!= null && input2.getObject() instanceof DataObject )
{
  taskInput = (DataObject)input2.getObject();
  // read the values
  ...
}
```

When the task instance is claimed, the input message of the task is returned.

Processing to-do tasks or collaboration tasks

To-do tasks (also known as *participating tasks* in the API) or collaboration tasks (also known as *human tasks* in the API) are assigned to various people in your organization through work items. To-do tasks and their associated work items are created, for example, when a process navigates to a human task activity.

About this task

One of the potential owners claims the task associated with the work item. This person is responsible for providing the relevant information and completing the task.

Procedure

1. List the tasks belonging to a logged-on person that are ready to be worked on.

```
QueryResultSet result =
     task.query("TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_READY AND
                (TASK.KIND = TASK.KIND.KIND_PARTICIPATING OR
                 TASK.KIND = TASK.KIND.KIND_HUMAN)AND
                 WORK_ITEM.REASON =
                   WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                 (String)null, (Integer)null, (TimeZone)null);
```

This action returns a query result set that contains the tasks that can be worked on by the logged-on person.

2. Claim the task to be worked on.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper input = task.claim(tkiid);
  DataObject taskInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
    taskInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

When the task is claimed, the input message of the task is returned.

3. When work on the task is finished, complete the task.

The task can be completed either successfully or with a fault message. If the task is successful, an output message is passed. If the task is unsuccessful, a fault message is passed. You must create the appropriate messages for these actions.

a. To complete the task successfully, create an output message.

```
ClientObjectWrapper output =
      task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the task
task.complete(tkiid, output);
```

This action sets an output message that contains the order number. The task is put into the finished state.

b. To complete the task when a fault occurs, create a fault message.

```
//retrieve the faults modeled for the task
List faultNames = task.getFaultNames(tkiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      task.createFaultMessage(tkiid, (String)faultNames.get(0));

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

task.complete(tkiid, (String)faultNames.get(0), myFault);
```

This action sets a fault message that contains the error code. The task is put into the failed state.

Suspending and resuming a task instance

You can suspend collaboration task instances (also known as *human tasks* in the API) or to-do task instances (also known as *participating tasks* in the API).

Before you begin

The task instance can be in the ready or claimed state. It can be escalated. The caller must be the owner, originator, or administrator of the task instance.

About this task

You can suspend a task instance while it is running. You might want to do this, for example, so that you can gather information that is needed to complete the task. When the information is available, you can resume the task instance.

Procedure

1. Get a list of tasks that are claimed by the logged-on user.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                   "TASK.STATE = TASK.STATE.STATE_CLAIMED",
                                   (String)null,
                                   (Integer)null,
                                   (TimeZone)null);
```

This action returns a query result set that contains a list of the tasks that are claimed by the logged-on user.

2. Suspend the task instance.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.suspend(tkiid);
}
```

This action suspends the specified task instance. The task instance is put into the suspended state.

3. Resume the process instance.

```
task.resume( tkiid );
```

This action puts the task instance into the state it had before it was suspended.

Analyzing the results of a task

A to-do task (also known as a *participating* task in the API) or a collaboration task (also known as a *human task* in the API) runs asynchronously. If a reply handler is specified when the task starts, the output message is automatically returned when the task completes. If a reply handler is not specified, the message must be retrieved explicitly.

About this task

The results of the task are stored in the database only if the task template from which the task instance was derived does not specify automatic deletion of the derived task instances.

Procedure

Analyze the results of the task.
The example shows how to check the order number of a successfully completed task.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                   "TASK.NAME = 'CustomerOrder' AND
                                    TASK.STATE = TASK.STATE.STATE_FINISHED",
                                    (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper output = task.getOutputMessage(tkiid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject() instanceof DataObject)
  {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
  }
}
```

Terminating a task instance

Sometimes it is necessary for someone with administrator rights to terminate a task instance that is known to be in an unrecoverable state. Because the task instance is terminated immediately, you should terminate a task instance only in exceptional situations.

Procedure

1. Retrieve the task instance to be terminated.
```
Task taskInstance = task.getTask(tkiid);
```
2. Terminate the task instance.
```
TKIID tkiid = taskInstance.getID();
task.terminate(tkiid);
```
The task instance is terminated immediately without waiting for any outstanding tasks.

Deleting task instances

Task instances are only automatically deleted when they complete if this is specified in the associated task template from which the instances are derived. This example shows how to delete all of the task instances that are finished and are not automatically deleted.

Procedure

1. List the task instances that are finished.
```
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_FINISHED",
                 (String)null, (Integer)null, (TimeZone)null);
```
This action returns a query result set that lists task instances that are finished.
2. Delete the task instances that are finished.
```
while (result.next() )
{
 TKIID tkiid = (TKIID) result.getOID(1);
 task.delete(tkiid);
}
```

Releasing a claimed task

When a potential owner claims a task, this person is responsible for completing the task. However, sometimes the claimed task must be released so that another potential owner can claim it.

About this task

Sometimes it is necessary for someone with administrator rights to release a claimed task. This situation might occur, for example, when a task must be completed but the owner of the task is absent. The owner of the task can also release a claimed task.

Procedure

1. List the claimed tasks owned by a specific person, for example, Smith.
```
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_CLAIMED AND
                 TASK.OWNER = 'Smith'",
                 (String)null, (Integer)null, (TimeZone)null);
```

This action returns a query result set that lists the tasks claimed by the specified person, Smith.

2. Release the claimed task.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.cancelClaim(tkiid, true);
}
```

This action returns the task to the ready state so that it can be claimed by one of the other potential owners. Any output or fault data that is set by the original owner is kept.

Managing work items

During the lifetime of an activity instance or a task instance, the set of people associated with the object can change, for example, because a person is on vacation, new people are hired, or the workload needs to be distributed differently. To allow for these changes, you can develop applications to create, delete, or transfer work items.

About this task

A work item represents the assignment of an object to a user or group of users for a particular reason. The object is typically a human task activity instance, a process instance, or a task instance. The reasons are derived from the role that the user has for the object. An object can have multiple work items because a user can have different roles in association with the object, and a work item is created for each of these roles. For example, a to-do task instance can have an administrator, reader, editor, and owner work item at the same time.

The actions that can be taken to manage work items depend on the role that the user has, for example, an administrator can create, delete and transfer work items, but the task owner can transfer work items only.

- Create a work item.

```
// query the task instance for which an additional
// administrator is to be specified
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // create the work item
  task.createWorkItem((TKIID)(result.getOID(1)),
                        WorkItem.REASON_ADMINISTRATOR,"Smith");
}
```

  This action creates a work item for the user Smith who has the administrator role.

- Delete a work item.

```
// query the task instance for which a work item is to be deleted
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
```

```
  // delete the work item
  task.deleteWorkItem((TKIID)(result.getOID(1)),
                        WorkItem.REASON_READER,"Smith");
}
```

This action deletes the work item for the user Smith who has the reader role.

- Transfer a work item.

```
// query the task that is to be rescheduled
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.NAME='CustomerOrder' AND
                 TASK.STATE=TASK.STATE.STATE_READY AND
                 WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER AND
                 WORK_ITEM.OWNER_ID='Miller'",
                 (String)null, (Integer)null, (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // transfer the work item from user Miller to user Smith
  // so that Smith can work on the task
  task.transferWorkItem((TKIID)(result.getOID(1)),
                         WorkItem.REASON_POTENTIAL_OWNER,"Miller","Smith");
}
```

This action transfers the work item to the user Smith so that he can work on it.

Creating task templates and task instances at runtime

You usually use a modeling tool, such as WebSphere Integration Developer to build task templates. You then install the task templates in WebSphere Process Server and create instances from these templates, for example, using Business Process Choreographer Explorer. However, you can also create human or participating task instances or templates at runtime.

About this task

You might want to do this, for example, when the task definition is not available when the application is deployed, the tasks that are part of a workflow are not yet known, or you need a task to cover some ad-hoc collaboration between a group of people.

You can model ad-hoc To-do or Collaboration tasks by creating instances of the com.ibm.task.api.TaskModel class, and using them to either create a reusable task template, or directly create a run-once task instance. To create an instance of the TaskModel class, a set of factory methods is available in the com.ibm.task.api.ClientTaskFactory factory class. Modeling human tasks at runtime is based on the Eclipse Modeling Framework (EMF).

Procedure

1. Create an org.eclipse.emf.ecore.resource.ResourceSet using the createResourceSet factory method.
2. Optional: If you intend to use complex message types, you can either define them using the org.eclipse.xsd.XSDFactory that you can get using the factory method getXSDFactory(), or directly import an existing XML schema using the loadXSDSchema factory method .

   To make the complex types available to the WebSphere Process Server, deploy them as part of an enterprise application.
3. Create or import a Web Services Definition Language (WSDL) definition of the type javax.wsdl.Definition.

You can create a new WSDL definition using the createWSDLDefinition method. Then you can add it a port type and operation. You can also directly import an existing WSDL definition using the loadWSDLDefinition factory method.

4. Create the task definition using the createTTask factory method.

   If you want to add or manipulate more complex task elements, you can use the com.ibm.wbit.tel.TaskFactory class that you can retrieve using the getTaskFactory factory method .
5. Create the task model using the createTaskModel factory method, and pass it the resource bundle that you created in the step 1 and which aggregates all other artifacts you created in the meantime.
6. Optional: Validate the model using the TaskModel validate method.

Results

Use one of the Human Task Manager EJB API create methods that have a **TaskModel** parameter to either create a reusable task template, or a run-once task instance.

Creating runtime tasks that use simple Java types:

This example creates a runtime task that uses only simple Java types in its interface, for example, a String object.

About this task

The example runs only inside the context of the calling enterprise application, for which the resources are loaded.

Procedure

1. Access the ClientTaskFactory and create a resource set to contain the definitions of the new task model.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Create the WSDL definition and add the descriptions of your operations.

```
// create the WSDL interface
Definition definition = factory.createWSDLDefinition
         ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

// create a port type
PortType portType = factory.createPortType( definition, "doItPT" );

// create an operation; the input and output messages are of type String:
// a fault message is not specified
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
          new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
          (Map)null );
```

3. Create the EMF model of your new human task.

   If you are creating a task instance, a valid-from date (UTCDate) is not required.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
```

```
                                                "http://www.ibm.com/task/test/",
                                                portType,
                                                operation );
```

This step initializes the properties of the task model with default values.

4. Modify the properties of your human task model.

   ```
   // use the methods from the com.ibm.wbit.tel package, for example,
   humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

   // retrieve the task factory to create or modify composite task elements
   TaskFactory taskFactory = factory.getTaskFactory();

   // specify escalation settings
   TVerb verb = taskFactory.createTVerb();
   verb.setName("John");

   // create escalationReceiver and add verb
   TEscalationReceiver escalationReceiver =
                       taskFactory.createTEscalationReceiver();
   escalationReceiver.setVerb(verb);

   // create escalation and add escalation receiver
   TEscalation escalation = taskFactory.createTEscalation();
   escalation.setEscalationReceiver(escalationReceiver);
   ```
5. Create the task model that contains all the resource definitions.

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```
6. Validate the task model and correct any validation problems that are found.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```
7. Create the runtime task instance or template.

   Use the HumanTaskManagerService interface to create the task instance or the task template. Because the application uses simple Java types only, you do not need to specify an application name.
   - The following snippet creates a task instance:

     ```
     atask.createTask( taskModel, (String)null, "HTM" );
     ```
   - The following snippet creates a task template:

     ```
     task.createTaskTemplate( taskModel, (String)null );
     ```

Results

If a runtime task instance is created, it can now be started. If a runtime task template is created, you can now create task instances from the template.

Creating runtime tasks that use complex types:

This example creates a runtime task that uses complex types in its interface. The complex types are already defined, that is, the local file system on the client has XSD files that contain the description of the complex types.

About this task

The example runs only inside the context of the calling enterprise application, for which the resources are loaded.

Procedure

1. Access the ClientTaskFactory and create a resource set to contain the definitions of the new task model.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Add the XSD definitions of your complex types to the resource set so that they are available when you define your operations.

   The files are located relative to the location where the code is executed.

```
factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
```

3. Create the WSDL definition and add the descriptions of your operations.

```
// create the WSDL interface
Definition definition = factory.createWSDLDefinition
         ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

// create a port type
PortType portType = factory.createPortType( definition, "doItPT" );

// create an operation; the input message is an InputBO and
// the output message an OutputBO;
// a fault message is not specified
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://Input", "InputBO" ),
          new QName( "http://Output", "OutputBO" ),
          (Map)null );
```

4. Create the EMF model of your new human task.

   If you are creating a task instance, a valid-from date (UTCDate) is not required.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   This step initializes the properties of the task model with default values.

5. Modify the properties of your human task model.

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. Create the task model that contains all the resource definitions.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

7. Validate the task model and correct any validation problems that are found.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

8. Create the runtime task instance or template.

   Use the HumanTaskManagerService interface to create the task instance or the task template. You must provide an application name that contains the data

type definitions so that they can be accessed. The application must also contain a dummy task or process so that the application is loaded by Business Process Choreographer.

- The following snippet creates a task instance:
  ```
  task.createTask( taskModel, "BOapplication", "HTM" );
  ```
- The following snippet creates a task template:
  ```
  task.createTaskTemplate( taskModel, "BOapplication" );
  ```

Results

If a runtime task instance is created, it can now be started. If a runtime task template is created, you can now create task instances from the template.

Creating runtime tasks that use an existing interface:

This example creates a runtime task that uses an interface that is already defined, that is, the local file system on the client has a file that contains the description of the interface.

About this task

The example runs only inside the context of the calling enterprise application, for which the resources are loaded.

Procedure

1. Access the ClientTaskFactory and create a resource set to contain the definitions of the new task model.
   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```
2. Access the WSDL definition and the descriptions of your operations.

   The interface description is located relative to the location where the code is executed.
   ```
   Definition definition = factory.loadWSDLDefinition(
                              resourceSet, "interface.wsdl"  );
   PortType portType = definition.getPortType(
                         new QName( definition.getTargetNamespace(), "doItPT" ) );
   Operation operation = portType.getOperation
                                   ("doIt", (String)null, (String)null);
   ```
3. Create the EMF model of your new human task.

   If you are creating a task instance, a valid-from date (UTCDate) is not required.
   ```
   TTask humanTask = factory.createTTask( resourceSet,
                                          TTaskKinds.HTASK_LITERAL,
                                          "TestTask",
                                          new UTCDate( "2005-01-01T00:00:00" ),
                                          "http://www.ibm.com/task/test/",
                                          portType,
                                          operation );
   ```
   This step initializes the properties of the task model with default values.
4. Modify the properties of your human task model.
   ```
   // use the methods from the com.ibm.wbit.tel package, for example,
   humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

   // retrieve the task factory to create or modify composite task elements
   TaskFactory taskFactory = factory.getTaskFactory();

   // specify escalation settings
   ```

```
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Create the task model that contains all the resource definitions.

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

6. Validate the task model and correct any validation problems that are found.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

7. Create the runtime task instance or template.

   Use the HumanTaskManagerService interface to create the task instance or the task template. You must provide an application name that contains the data type definitions so that they can be accessed. The application must also contain a dummy task or process so that the application is loaded by Business Process Choreographer.

   - The following snippet creates a task instance:

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - The following snippet creates a task template:

     ```
     task.createTaskTemplate( taskModel, "BOapplication" );
     ```

Results

If a runtime task instance is created, it can now be started. If a runtime task template is created, you can now create task instances from the template.

Creating runtime tasks that use an interface from the calling application:

This example creates a runtime task that uses an interface that is part of the calling application. For example, the runtime task is created in a Java snippet of a business process and uses an interface from the process application.

About this task

The example runs only inside the context of the calling enterprise application, for which the resources are loaded.

Procedure

1. Access the ClientTaskFactory and create a resource set to contain the definitions of the new task model.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();

   // specify the context class loader so that following resources are found
   ResourceSet resourceSet = factory.createResourceSet
                ( Thread.currentThread().getContextClassLoader() );
   ```

2. Access the WSDL definition and the descriptions of your operations.

   Specify the path within the containing package JAR file.

   ```
   Definition definition = factory.loadWSDLDefinition( resourceSet,
                             "com/ibm/workflow/metaflow/interface.wsdl" );
                         PortType portType = definition.getPortType(
   ```

```
                      new QName( definition.getTargetNamespace(), "doItPT" ) );
Operation operation = portType.getOperation
                                  ("doIt", (String)null, (String)null);
```

3. Create the EMF model of your new human task.

   If you are creating a task instance, a valid-from date (UTCDate) is not required.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   This step initializes the properties of the task model with default values.

4. Modify the properties of your human task model.

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Create the task model that contains all the resource definitions.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. Validate the task model and correct any validation problems that are found.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. Create the runtime task instance or template.

   Use the HumanTaskManagerService interface to create the task instance or the task template. You must provide an application name that contains the data type definitions so that they can be accessed.

   - The following snippet creates a task instance:

```
task.createTask( taskModel, "WorkflowApplication", "HTM" );
```

   - The following snippet creates a task template:

```
task.createTaskTemplate( taskModel, "WorkflowApplication" );
```

Results

If a runtime task instance is created, it can now be started. If a runtime task template is created, you can now create task instances from the template.

HumanTaskManagerService interface

The HumanTaskManagerService interface exposes task-related functions that can be called by a local or a remote client.

The methods that can be called depend on the state of the task and the authorization of the person that uses the application containing the method. The main methods for manipulating task objects are listed here. For more information

about these methods and the other methods that are available in the HumanTaskManagerService interface, see the Javadoc in the com.ibm.task.api package.

Task templates

The following methods are available to work with task templates.

Table 46. API methods for task templates

Method	Description
getTaskTemplate	Retrieves the specified task template.
createAndCallTask	Creates and runs a task instance from the specified task template and waits synchronously for the result.
createAndStartTask	Creates and starts a task instance from the specified task template.
createTask	Creates a task instance from the specified task template.
createInputMessage	Creates an input message for the specified task template. For example, create a message that can be used to start a task.
queryTaskTemplates	Retrieves task templates that are stored in the database.

Task instances

The following methods are available to work with task instances.

Table 47. API methods for task instances

Method	Description
getTask	Retrieves a task instance; the task instance can be in any state.
callTask	Starts an invocation task synchronously.
startTask	Starts a task that has already been created.
suspend	Suspends the collaboration or to-do task.
resume	Resumes the collaboration or to-do task.
terminate	Terminates the specified task instance. If an invocation task is terminated, this action has no impact on the invoked service.
delete	Deletes the specified task instance.
claim	Claims the task for processing.
update	Updates the task instance.
complete	Completes the task instance.
cancelClaim	Releases a claimed task instance so that it can be worked on by another potential owner.
createWorkItem	Creates a work item for the task instance.
transferWorkItem	Transfers the work item to a specified owner.

Table 47. API methods for task instances (continued)

Method	Description
deleteWorkItem	Deletes the work item.

Escalations

The following methods are available to work with escalations.

Table 48. API methods for working with escalations

Method	Description
getEscalation	Retrieves the specified escalation instance.

Custom properties

Tasks, task templates, and escalations can all have custom properties. The interface provides a get and a set method to retrieve and set values for custom properties. You can also associate named properties with, and retrieve named properties from task instances. Custom property names and values must be of the java.lang.String type. The following methods are valid for tasks, task templates, and escalations.

Table 49. API methods for variables and custom properties

Method	Description
getCustomProperty	Retrieves the named custom property of the specified task instance.
getCustomProperties	Retrieves the custom properties of the specified task instance.
getCustomPropertyNames	Retrieves the names of the custom properties for the task instance.
setCustomProperty	Stores custom-specific values for the specified task instance.

Allowed actions for tasks:

The actions that can be carried out on a task depend on whether the task is a to-do task, a collaboration task, an invocation task, or an administration task.

You cannot use all of the actions provided by the HumanTaskManager interface for all kinds of tasks. The following table shows the actions that you can carry out on each kind of task.

Action	Kind of task			
	To-do task	Collaboration task	Invocation task	Administration task
callTask			X	
cancelClaim	X	X[1]		
claim	X	X[1]		
complete	X	X[1]		X
completeWithFollowOnTask[4]	X	X[1]		
completeWithFollowOnTask[5]		X[3]	X[3]	

Action	Kind of task			
	To-do task	Collaboration task	Invocation task	Administration task
createFaultMessage	X	X	X	X
createInputMessage	X	X	X	X
createOutputMessage	X	X	X	X
createWorkItem	X	X[1]	X	X
delete	X[1]	X[1]	X	X[1]
deleteWorkItem	X	X[1]	X	X
getCustomProperty	X	X[1]	X	X
getDocumentation	X	X[1]	X	X
getFaultNames	X	X[1]		
getFaultMessage	X	X[1]	X	
getInputMessage	X	X[1]	X	
getOutputMessage	X	X[1]	X	
getUsersInRole	X	X[1]	X	X
getTask	X	X[1]	X	X
getUISettings	X	X[1]	X	X
resume	X	X[1]		
setCustomProperty	X	X[1]	X	X
setFaultMessage	X	X[1]		
setOutputMessage	X	X[1]		
startTask	X[1]	X[1]	X	X
startTaskAsSubtask[6]	X	X[1]		
startTaskAsSubtask[7]		X[3]	X[3]	
suspend	X	X[1]		
suspendWithCancelClaim	X	X[1]		
terminate	X[1]	X[1]	X[1]	
transferWorkItem	X	X[1]	X	X
update	X	X[1]	X	X

Notes:

1. For stand-alone tasks, ad-hoc tasks, and task templates only
2. For stand-alone tasks, inline tasks in business processes, and ad-hoc tasks only
3. For stand-alone tasks and ad-hoc tasks only
4. The tasks kinds that can have follow-on tasks
5. The task kinds that can be used as follow-on tasks
6. The tasks kinds that can have subtasks
7. The task kinds that can be used as subtasks

Developing applications for business processes and human tasks

People are involved in most business process scenarios. For example, a business process requires people interaction when the process is started or administered, or

when human task activities are performed. To support these scenarios, you need to use both the Business Flow Manager API and the Human Task Manager API.

About this task

To involve people in business process scenarios, you can include the following task kinds in the business process:

- An inline invocation task (also known as an *originating task* in the API).

  You can provide an invocation task for every receive activity, for each onMessage element of a pick activity, and for each onEvent element of an event handler. This task then controls who is authorized to start a process or communicate with a running process instance.
- An administration task.

  You can provide an administration task to specify who is authorized to administer the process or perform administrative operations on the failed activities of the process.
- A to-do task (also known as a *participating task* in the API).

  A to-do task implements a human task activity. This type of activity allows you to involve people in the process.

Human task activities in the business process represent the to-do tasks that people perform in the business process scenario. You can use both the Business Flow Manager API and the Human Task Manager API to realize these scenarios:

- The business process is the container for all of the activities that belong to the process, including the human task activities that are represented by to-do tasks. When a process instance is created, a unique object ID (PIID) is assigned.
- When a human task activity is activated during the execution of the process instance, an activity instance is created, which is identified by its unique object ID (AIID). At the same time, an inline to-do task instance is also created, which is identified by its object ID (TKIID). The relationship of the human task activity to the task instance is achieved by using the object IDs:
  - The to-do task ID of the activity instance is set to the TKIID of the associated to-do task.
  - The containment context ID of the task instance is set to the PIID of the process instance that contains the associated activity instance.
  - The parent context ID of the task instance is set to the AIID of the associated activity instance.
- The life cycles of all inline to-do task instances are managed by the process instance. When the process instance is deleted, then the task instances are also deleted. In other words, all of the tasks that have the containment context ID set to the PIID of the process instance are automatically deleted.

Determining the process templates or activities that can be started

A business process can be started by invoking the call, initiate, or sendMessage methods of the Business Flow Manager API. If the process has only one starting activity, you can use the method signature that requires a process template name as a parameter. If the process has more than one starting activity, you must explicitly identify the starting activity.

About this task

When a business process is modeled, the modeler can decide that only a subset of users can create a process instance from the process template. This is done by associating an inline invocation task to a starting activity of the process and by specifying authorization restrictions on that task. Only the people that are potential starters or administrators of the task are allowed to create an instance of the task, and thus an instance of the process template.

If an inline invocation task is not associated with the starting activity, or if authorization restrictions are not specified for the task, everybody can create a process instance using the starting activity.

A process can have more than one starting activity, each with different people queries for potential starters or administrators. This means that a user can be authorized to start a process using activity A but not using activity B.

Procedure

1. Use the Business Flow Manager API to create a list of the current versions of process templates that are in the started state.

   Tip: The queryProcessTemplates method excludes only those process templates that are part of applications that are not yet started. So, if you use this method without filtering the results, the method returns all of the versions of the process templates regardless of which state they are in.

   ```
   // current timestamp in UTC format, converted to yyyy-mm-ddThh:mm:ss
   String now = (new UTCDate()).toXsdString();
   String whereClause = "PROCESS_TEMPLATE.STATE =
                         PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                         PROCESS_TEMPLATE.VALID_FROM =
                         (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                          WHERE NAME=PROCESS_TEMPLATE.NAME AND
                          VALID_FROM <= TS('" + now + "'))";

   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
                                          ( whereClause,
                                            "PROCESS_TEMPLATE.NAME",
                                            (Integer)null, (TimeZone)null);
   ```

   The results are sorted by process template name.
2. Create the list of process templates and the list of starting activities for which the user is authorized.

   The list of process templates contains those process templates that have a single starting activity. These activities are either not secured or the logged-on user is allowed to start them. Alternatively, you might want to gather the process templates that can be started by at least one of the starting activities.

   Tip: A process administrator can also start a process instance. To get a complete list of templates, you also need to read the administration task template that is associated with the process template, and check whether the logged-on user is an administrator.

   ```
   List authorizedProcessTemplates = new ArrayList();
   List authorizedActivityServiceTemplates = new ArrayList();
   ```
3. Determine the starting activities for each of the process templates.

```
for( int i=0; i<processTemplates.length; i++ )
{
  ProcessTemplateData template = processTemplates[i];
  ActivityServiceTemplateData[] startActivities =
                process.getStartActivities(template.getID());
```

4. For each starting activity, retrieve the ID of the associated inline invocation task template.

```
for( int j=0; j<startActivities.length; j++ )
{
  ActivityServiceTemplateData activity = startActivities[j];
  TKTID tktid = activity.getTaskTemplateID();
```

   a. If an invocation task template does not exist, the process template is not secured by this starting activity.

      In this case, everybody can create a process instance using this start activity.

```
boolean isAuthorized = false;
       if ( tktid == null )
    {
      isAuthorized = true;
      authorizedActivityServiceTemplates.add(activity);
    }
```

   b. If an invocation task template exists, use the Human Task Manager API to check the authorization for the logged-on user.

      In the example, the logged-on user is Smith. The logged-on user must be a potential starter of the invocation task or an administrator.

```
if ( tktid != null )
    {
      isAuthorized =
        task.isUserInRole
            (tkid, "Smith", WorkItem.REASON_POTENTIAL_STARTER) ||
        task.isUserInRole(tktid, "Smith", WorkItem.REASON_ADMINISTRATOR);

      if ( isAuthorized )
      {
        authorizedActivityServiceTemplates.add(activity);
      }
    }
```

      If the user has the specified role, or if people assignment criteria for the role are not specified, the isUserInRole method returns the value `true`.

5. Check whether the process can be started using only the process template name.

```
if ( isAuthorized && startActivities.length == 1 )
    {
      authorizedProcessTemplates.add(template);
    }
```

6. End the loops.

```
  } // end of loop for each activity service template
}   // end of loop for each process template
```

Processing a single person workflow that includes human tasks

Some workflows are performed by only one person, for example, ordering books from an online bookstore. This example shows how to implement the sequence of actions for ordering the book as a series of human task activities (to-do tasks). Both the Business Flow Manager and the Human Task Manager APIs are used to process the workflow.

About this task

In an online bookstore, the purchaser completes a sequence of actions to order a book. This sequence of actions can be implemented as a series of human task activities (to-do tasks). If the purchaser decides to order several books, this is equivalent to claiming the next human task activity. Information about the sequence of tasks is maintained by Business Flow Manager, while the tasks themselves are maintained by Human Task Manager.

Compare this example with the example that uses only the Business Flow Manager API.

Procedure

1. Use the Business Flow Manager API to get the process instance that you want to work on.

   In this example, an instance of the CustomerOrder process.

   ```
   ProcessInstanceData processInstance =
      process.getProcessInstance("CustomerOrder");
   String piid = processInstance.getID().toString();
   ```
2. Use the Human Task Manager API to query the ready to-do tasks (kind participating) that are part of the specified process instance.

   Use the containment context ID of the task to specify the containing process instance. For a single person workflow, the query returns the to-do task that is associated with the first human task activity in the sequence of human task activities.

   ```
   //
    // Query the list of to-do tasks that can be claimed by the logged-on user
    // for the specified process instance
   //
   QueryResultSet result =
        task.query("DISTINCT TASK.TKIID",
                   "TASK.CONTAINMENT_CTX_ID = ID('" + piid + "') AND
                    TASK.STATE = TASK.STATE.STATE_READY AND
                    TASK.KIND = TASK.KIND.KIND_PARTICIPATING AND
                    WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                   (String)null, (Integer)null, (TimeZone)null);
   ```
3. Claim the to-do task that is returned.

   ```
   if (result.size() > 0)
   {
    result.first();
    TKIID tkiid = (TKIID) result.getOID(1);
    ClientObjectWrapper input = task.claim(tkiid);
    DataObject activityInput = null ;
     if ( input.getObject()!= null && input.getObject() instanceof DataObject )
     {
       taskInput = (DataObject)input.getObject();
       // read the values
       ...
     }
   }
   ```

   When the task is claimed, the input message of the task is returned.
4. Determine the human task activity that is associated with the to-do task.

   You can use one of the following methods to correlate activities to their tasks.

   - The task.getActivityID method:

     ```
     AIID aiid = task.getActivityID(tkiid);
     ```
   - The parent context ID that is part of the task object:

```
AIID aiid = null;
Task taskInstance = task.getTask(tkiid);

OID oid = taskInstance.getParentContextID();
if ( oid != null and oid instanceof AIID )
{
  aiid = (AIID)oid;
}
```

5. When work on the task is finished, use the Business Flow Manager API to complete the task and its associated human task activity, and claim the next human task activity in the process instance.

   To complete the human task activity, an output message is passed. When you create the output message, you must specify the message type name so that the message definition is contained.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the human task activity and its associated to-do task,
// and claim the next human task activity
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   This action sets an output message that contains the order number and claims the next human task activity in the sequence. If `AutoClaim` is set for successor activities and if there are multiple paths that can be followed, all of the successor activities are claimed and a random activity is returned as the next activity. If there are no more successor activities that can be assigned to this user, `Null` is returned.

   If the process contains parallel paths that can be followed and these paths contain human task activities for which the logged-on user is a potential owner of more than one of these activities, a random activity is claimed automatically and returned as the next activity.

6. Work on the next human task activity.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

7. Continue with step 5 to complete the human task activity and to retrieve the next human task activity.

Related tasks

Processing a single person workflow
Some workflows are performed by only one person, for example, ordering books from an online bookstore. This type of workflow has no parallel paths. The completeAndClaimSuccessor API supports the processing of this type of workflow.

Handling exceptions and faults

A BPEL process might encounter a fault at different points in the process.

About this task

Business Process Execution Language (BPEL) faults originate from:

- Web service invocations (Web Services Description Language (WSDL) faults)
- Throw activities
- BPEL standard faults that are recognized by Business Process Choreographer

Mechanisms exist to handle these faults. Use one of the following mechanisms to handle faults that are generated by a process instance:

- Pass control to the corresponding fault handlers
- Compensate previous work in the process
- Stop the process and let someone repair the situation (force-retry, force-complete)

A BPEL process can also return faults to a caller of an operation provided by the process. You can model the fault in the process as a reply activity with a fault name and fault data. These faults are returned to the API caller as checked exceptions.

If a BPEL process does not handle a BPEL fault or if an API exception occurs, a runtime exception is returned to the API caller. An example for an API exception is when the process model from which an instance is to be created does not exist.

The handling of faults and exceptions is described in the following tasks.

Handling Business Process Choreographer EJB API exceptions

If a method in the BusinessFlowManagerService interface or the HumanTaskManagerService interface does not complete successfully, an exception is thrown that denotes the cause of the error. You can handle this exception specifically to provide guidance to the caller.

About this task

However, it is common practice to handle only a subset of the exceptions specifically and to provide general guidance for the other potential exceptions. All specific exceptions inherit from a generic ProcessException or TaskException. Catch generic exceptions with a final `catch(ProcessException)` or `catch(TaskException)` statement. This statement helps to ensure the upward compatibility of your application program because it takes account of all of the other exceptions that can occur.

Checking which fault is set for a human task activity

When a human task activity is processed, it can complete successfully. In this case, you can pass an output message. If the human task activity does not complete successfully, you can pass a fault message.

About this task

You can read the fault message to determine the cause of the error.

Procedure

1. List the task activities that are in a failed or stopped state.

   ```
   QueryResultSet result =
        process.query("ACTIVITY.AIID",
                      "(ACTIVITY.STATE = ACTIVITY.STATE.STATE_FAILED OR
                        ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED) AND
                        ACTIVITY.KIND=ACTIVITY.KIND.KIND_STAFF",
                        (String)null, (Integer)null, (TimeZone)null);
   ```

   This action returns a query result set that contains failed or stopped activities.
2. Read the name of the fault.

   ```
   if (result.size() > 0)
   {
     result.first();
     AIID aiid = (AIID) result.getOID(1);
     ClientObjectWrapper faultMessage = process.getFaultMessage(aiid);
     DataObject fault = null ;
     if ( faultMessage.getObject() != null && faultMessage.getObject()
          instanceof DataObject )
     {
       fault = (DataObject)faultMessage.getObject();
       Type type = fault.getType();
       String name = type.getName();
       String uri = type.getURI();
     }
   }
   ```

   This returns the fault name. You can also analyze the unhandled exception for a stopped activity instead of retrieving the fault name.

Checking which fault occurred for a stopped invoke activity

In a well-designed process, exceptions and faults are usually handled by fault handlers. You can retrieve information about the exception or fault that occurred for an invoke activity from the activity instance.

About this task

If an activity causes a fault to occur, the fault type determines the actions that you can take to repair the activity.

Procedure

1. List the human task activities that are in a stopped state.

   ```
   QueryResultSet result =
        process.query("ACTIVITY.AIID",
                      "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                       ACTIVITY.KIND=ACTIVITY.KIND.KIND_INVOKE",
                       (String)null, (Integer)null, (TimeZone)null);
   ```

   This action returns a query result set that contains stopped invoke activities.
2. Read the name of the fault.

   ```
   if (result.size() > 0)
   {
     result.first();
     AIID aiid = (AIID) result.getOID(1);
     ActivityInstanceData activity = process.getActivityInstance(aiid);
   ```

```
  ProcessException excp = activity.getUnhandledException();
  if ( excp instanceof ApplicationFaultException )
  {
   ApplicationFaultException fault = (ApplicationFaultException)excp;
   String faultName = fault.getFaultName();
  }
}
```

Checking which unhandled exception or fault occurred for a failed process instance

In a well-designed process, exceptions and faults are usually handled by a fault handler. If the process implements a two-way operation, you can retrieve information about a fault or handled exception from the fault name property of the process instance object. For faults, you can also retrieve the corresponding fault message using the getFaultMessage API.

About this task

If a process instance fails because of an exception that is not handled by any fault handler, you can retrieve information about the unhandled exception from the process instance object. By contrast, if a fault is caught by a fault handler, then information about the fault is not available. You can, however, retrieve the fault name and message and return to the caller by using a FaultReplyException exception.

Procedure

1. List the process instances that are in the failed state.

   ```
   QueryResultSet result =
        process.query("PROCESS_INSTANCE.PIID",
                     "PROCESS_INSTANCE.STATE =
                         PROCESS_INSTANCE.STATE.STATE_FAILED",
                      (String)null, (Integer)null, (TimeZone)null);
   ```

   This action returns a query result set that contains the failed process instances.

2. Read the information for the unhandled exception.

   ```
   if (result.size() > 0)
   {
     result.first();
     PIID piid = (PIID) result.getOID(1);
     ProcessInstanceData pInstance = process.getProcessInstance(piid);

     ProcessException excp = pInstance.getUnhandledException();
     if ( excp instanceof RuntimeFaultException )
     {
      RuntimeFaultException xcp = (RuntimeFaultException)excp;
      Throwable cause = xcp.getRootCause();
     }
     else if ( excp instanceof StandardFaultException )
     {
      StandardFaultException xcp = (StandardFaultException)excp;
      String faultName = xcp.getFaultName();
     }
     else if ( excp instanceof ApplicationFaultException )
     {
      ApplicationFaultException xcp = (ApplicationFaultException)excp;
      String faultName = xcp.getFaultName();
     }
   }
   ```

Results

Use this information to look up the fault name or the root cause of the problem.

Developing Web service API client applications

You can develop client applications that access business process applications and human task applications through Web services APIs.

About this task

Client applications can be developed in any Web service client environment, including Java Web services and Microsoft .NET.

Related concepts

Comparison of the programming interfaces for interacting with business processes and human tasks
Enterprise JavaBeans (EJB), Web service, and Java Message Service (JMS), and Representational State Transfer Services (REST) generic programming interfaces are available for building client applications that interact with business processes and human tasks. Each of these interfaces has different characteristics.

Web service components and sequence of control

A number of client-side and server-side components participate in the sequence of control that represents a Web service request and response.

A typical sequence of control is as follows.

1. On the client side:
   a. A client application (provided by the user) issues a request for a Web service.
   b. A proxy client (also provided by the user, but which can be automatically generated using client-side utilities) wraps the service request in a SOAP request envelope.
   c. The client-side development infrastructure forwards the request to a URL defined as the Web service's endpoint.
2. The network transmits the request to the Web service endpoint using HTTP or HTTPS.
3. On the server side:
   a. The generic Web services API receives and decodes the request.
   b. The request is either handled directly by the generic Business Flow Manager or Human Task Manager component, or forwarded to the specified business process or human task.
   c. The returned data is wrapped in a SOAP response envelope.
4. The network transmits the response to the client-side environment using HTTP or HTTPS.
5. Back on the client side:
   a. The client-side development infrastructure unwraps the SOAP response envelope.
   b. The proxy client extracts the data from the SOAP response and passes it to the client application.
   c. The client application processes the returned data as necessary.

Overview of the Web services APIs

Web services APIs allow you to develop client applications that use Web services to access business processes and human tasks running in the Business Process Choreographer environment.

The Business Process Choreographer Web services API provides two separate Web service interfaces (WSDL port types):

- The Business Flow Manager API. Allows client applications to interact with microflows and long-running processes, for example:
  - Create process templates and process instances
  - Claim existing processes
  - Query a process by its ID

  Refer to “Developing applications for business processes” on page 240 for a complete list of possible actions.
- The Human Task Manager API. Allows client applications to:
  - Create and start tasks
  - Claim existing tasks
  - Complete tasks
  - Query a task by its ID
  - Query a collection of tasks.

  Refer to “Developing applications for human tasks” on page 260 for a complete list of possible actions.

Client applications can use either or both of the Web service interfaces.

Example

The following is a possible outline for a client application that accesses the Human Task Manager Web services API to process a participating human task:

1. The client application issues a query Web service call to the WebSphere Process Server requesting a list of participating tasks to be worked on by a user.
2. The list of participating tasks is returned in a SOAP/HTTP response envelope.
3. The client application then issues a claim Web service call to claim one of the participating tasks.
4. The WebSphere Process Server returns the task’s input message.
5. The client application issues a complete Web service call to complete the task with an output or fault message.

Requirements for business processes and human tasks

Business processes and human tasks developed with the WebSphere Integration Developer to run on the Business Process Choreographer must conform to specific rules to be accessible through the Web services APIs.

The requirements are:

1. The interfaces of business processes and human tasks must be defined using the ″document/literal wrapped″ style defined in the Java API for XML-based RPC (JAX-RPC 1.1) specification. This is the default style for all business processes and human tasks developed with the WID.

2. Fault messages exposed by business processes and human tasks for Web service operations must comprise a single WSDL message part defined with an XML Schema element. For example:

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

Related information

Java API for XML based RPC (JAX-RPC) downloads page

Which style of WSDL should I use?

Developing client applications

The client application development process consists of a number of steps.

Procedure

1. Decide which Web services API your client application needs to use: the Business Flow Manager API, Human Task Manager API, or both.
2. Export the necessary files from the WebSphere Process Server environment. Alternatively, you can copy the files from the WebSphere Process Server client CD.
3. In your chosen client application development environment, generate a *proxy client* using the exported artifacts.
4. Optional: Generate *helper classes*. Helper classes are required if your client application interacts directly with concrete processes or tasks on the WebSphere server. They are not, however, necessary if your client application is only going to perform generic tasks such as issuing queries.
5. Develop the code for your client application.
6. Add any necessary security mechanisms to your client application.

Copying artifacts

A number of artifacts must be copied from the WebSphere environment to help in the creation of client applications.

There are two ways to obtain these artifacts:

- Publish and export them from the WebSphere Process Server environment.
- Copy files from the WebSphere Process Server client CD.

Publishing and exporting artifacts from the server environment

Before you can develop client applications to access the Web services APIs, you must publish and export a number of artifacts from the WebSphere server environment.

About this task

The artifacts to be exported are:

- Web Service Definition Language (WSDL) files describing the port types and operations that make up the Web services APIs.
- XML Schema Definition (XSD) files containing definitions of data types referenced by services and methods in the WSDL files.
- Additional WSDL and XSD files describing business objects. Business objects describe concrete business processes or human tasks running on the WebSphere server. These additional files are only required if your client application needs to interact directly with the concrete business processes or human tasks through

the Web services APIs. They are not necessary if your client application is only going to perform generic tasks, such as issuing queries.

After these artifacts are published, you need to copy them to your client programming environment, where they are used to generate a proxy client and helper classes.

Specifying the Web service endpoint address:

The Web service endpoint address is the URL that a client application must specify to access the Web services APIs. The endpoint address is written into the WSDL file that you export to generate a proxy client for your client application.

About this task

The Web service endpoint address to use depends on your WebSphere server configuration:

- Scenario 1. A single WebSphere server. The WebSphere endpoint address to specify is the host name and port number of the server, for example **host1:9080**.
- Scenario 2. A WebSphere cluster composed of several servers. The WebSphere endpoint address to specify is the host name and port of the server that is hosting the Web services APIs, for example, **host2:9081**.
- Scenario 3. A Web server is used as a front end. The WebSphere endpoint address to specify is the host name and port of the Web server, for example: **host:80**.

By default, the Web service endpoint address takes the form *protocol*`://`*host*:*port*/ *context_root*/*fixed_path*. Where:

- *protocol*. The communications protocol to be used between the client application and the WebSphere server. The default protocol is HTTP. You can instead choose to use the more secure HTTPS (HTTP over SSL) protocol. It is recommended to use HTTPS.
- *host:port*. The host name and port number used to access the system that is hosting the Web services APIs. These values vary depending on the WebSphere server configuration; for example, whether your client application is to access the application directly or through a Web server front end.
- *context_root*. You are free to choose any value for the context root. The value you choose must, however, be unique within each WebSphere cell. The default value uses a "node_server/cluster" suffix that eliminates the risk of naming conflicts.
- *fixed_path* is either `/sca/com/ibm/bpe/api/BFMWS` (for the Business Flow Manager API) or `/sca/com/ibm/task/api/HTMWS` (for the Human Task Manager API) and cannot be modified.

The Web service endpoint address is initially specified when configuring the business process container or human task container:

Procedure

1. Log on to the administrative console with a user ID with administrator rights.
2. Choose **Applications** → **SCA modules**.

   Note: You can also select **Applications** → **Enterprise applications** to display a list of all available enterprise applications.

3. Select **BPEContainer** (for the business process container) or **TaskContainer** (for the human task container) from the list of SCA modules or applications.
4. Choose **Provide HTTP endpoint URL information** from the list of **Additional properties**.
5. Select one of the default prefixes from the list, or enter a custom prefix. Use a prefix from the default prefix list if your client applications are to connect directly to the application server hosting the Web services API. Otherwise, specify a custom prefix.
6. Click **Apply** to copy the selected prefix to the SCA module.
7. Click **OK**. The URL information is saved to your workspace.

Results

You can view the current value in the administrative console (for example, for the business process container: **Enterprise Applications** → **BPEContainer** → **View Deployment Descriptor**).

In the exported WSDL file, the `location` attribute of the `soap:address` element contains the specified Web services endpoint address. For example:

```
<wsdl:service name="BFMWSService">
  <wsdl:port name="BFMWSPort" binding="this:BFMWSBinding">
   <soap:address location=
         "https://myserver:9080/WebServicesAPIs/sca/com/ibm/bpe/api/BFMWS"/>
```

Publishing WSDL files:

A Web Service Definition Language (WSDL) file contains a detailed description of all the operations available with a Web services API. Separate WSDL files are available for the Business Flow Manager and Human Task Manager Web services APIs. You must first publish these WSDL files then copy them from the WebSphere environment to your development environment, where they are used to generate a proxy client.

Before you begin

Before publishing the WSDL files, be sure to specify the correct Web services endpoint address. This is the URL that your client application uses to access the Web services APIs.

About this task

You only need to publish WSDL files once.

Note: If you have the WebSphere Process Server client CD, you can copy the files directly from there to your client programming environment instead.

Publishing the business process WSDL:

Use the administrative console to publish the WSDL file.

Procedure

1. Log on to the administrative console with a user ID with administrator rights.
2. Select **Applications** → **SCA modules**

Note: You can also select **Applications → Enterprise applications** to display a list of all available enterprise applications.

3. Choose the **BPEContainer** application from the list of SCA modules or applications.
4. Select **Publish WSDL files** from the list of **Additional properties**
5. Click on the zip file in the list.
6. On the File Download window that appears, click **Save**.
7. Browse to a local folder and click **Save**.

Results

The exported zip file is named BPEContainer_WSDLFiles.zip. The zip file contains a WSDL file that describes the Web services, and any XSD files referenced from within the WSDL file.

Publishing the human task WSDL:

Use the administrative console to publish the WSDL file.

Procedure

1. Log on to the administrative console with a user ID with administrator rights.
2. Select **Applications → SCA modules**

   Note: You can also select **Applications → Enterprise applications** to display a list of all available enterprise applications.
3. Choose the **TaskContainer** application from the list of SCA modules or applications.
4. Select **Publish WSDL files** from the list of **Additional properties**
5. Click on the zip file in the list.
6. On the File Download window that appears, click **Save**.
7. Browse to a local folder and click **Save**.

Results

The exported zip file is named TaskContainer_WSDLFiles.zip. The zip file contains a WSDL file that describes the Web services, and any XSD files referenced from within the WSDL file.

Exporting business objects:

Business processes and human tasks have well-defined interfaces that allow them to be accessed externally as Web services. If these interfaces reference business objects, you need to export the interface definitions and business objects to your client programming environment.

About this task

This procedure must be repeated for each business object that your client application needs to interact with.

In WebSphere Process Server, business objects define the format of request, response and fault messages that interact with business processes or human tasks. These messages can also contain definitions of complex data types.

For example, to create and start a human task, the following items of information must be passed to the task interface:

- The task template name
- The task template namespace
- An input message, containing formatted business data
- A response wrapper for returning the response message
- A fault message for returning faults and exceptions

These items are encapsulated within a single business object. All operations of the Web service interface are modeled as a ″document/literal wrapped″ operation. Input and output parameters for these operations are encapsulated in wrapper documents. Other business objects define the corresponding response and fault message formats.

In order to create and start the business process or human task through a Web service, these wrapper objects must be made available to the client application on the client side.

This is achieved by exporting the business objects from the WebSphere environment as Web Service Definition Language (WSDL) and XML Schema Definition (XSD) files, importing the data type definitions into your client programming environment, then converting them to helper classes for use by the client application.

Procedure

1. Launch the WebSphere Integration Developer Workspace if it is not already running.
2. Select the Library module containing the business objects to be exported. A Library module is a compressed file that contains the necessary business objects.
3. Export the Library module.
4. Copy the exported files to your client application development environment.

Example

Assume a business process exposes the following Web service operation:

```
<wsdl:operation name="updateCustomer">
    <wsdl:input message="tns:updateCustomerRequestMsg"
                name="updateCustomerRequest"/>
    <wsdl:output message="tns:updateCustomerResponseMsg"
                 name="updateCustomerResponse"/>
    <wsdl:fault message="tns:updateCustomerFaultMsg"
                name="updateCustomerFault"/>
  </wsdl:operation>
```

with the WSDL messages defined as:

```
<wsdl:message name="updateCustomerRequestMsg">
    <wsdl:part element="types:updateCustomer"
               name="updateCustomerParameters"/>
  </wsdl:message>
  <wsdl:message name="updateCustomerResponseMsg">
    <wsdl:part element="types:updateCustomerResponse"
               name="updateCustomerResult"/>
  </wsdl:message>
```

```
<wsdl:message name="updateCustomerFaultMsg">
  <wsdl:part element="types:updateCustomerFault"
            name="updateCustomerFault"/>
</wsdl:message>
```

The *concrete* customer-defined elements `types:updateCustomer`, `types:updateCustomerResponse`, and `types:updateCustomerFault` must be passed to and received from the Web services APIs using <xsd:any> parameters in all *generic* operations (call, sendMessage, and so on) performed by the client application. These customer-defined elements are created, serialized and deserialized on the client application side using helper classes that are generated using the exported XSD files.

Using files on the client CD

As an alternative to exporting artifacts from the WebSphere server environment, you can copy the files necessary for generating a client application from the WebSphere Process Server client CD.

In this case, you must manually modify the default Web services endpoint address of the Business Flow Manager API or Human Task Manager API.

If the client application is to access both APIs, you must edit the default endpoint address for both APIs.

Copying files from the client CD:

The files necessary to access the Web services APIs are available on the WebSphere Process Server client CD.

Procedure

1. Access the client CD and browse to the ProcessChoreographer\client directory.
2. Copy the necessary files to your client application development environment.

   For the Business Flow Manager API, copy:

   BFMWS.wsdl
      Describes the Web services available in the Business Flow Manager Web services API. This file contains the endpoint address.

   BFMIF.wsdl
      Describes the parameters and data type of each Web service in the Business Flow Manager Web services API.

   BFMIF.xsd
      Describes data types used in the Business Flow Manager Web services API.

   BPCGEN.xsd
      Contains data types that are common between the Business Flow Manager and Human Task Manager Web services APIs.

   For the Human Task Manager API, copy:

   HTMWS.wsdl
      Describes the Web services available in the Human Task Manager Web services API. This file contains the endpoint address.

   HTMIF.wsdl
      Describes the parameters and data type of each Web service in the Human Task Manager Web services API.

HTMIF.xsd
: Describes data types used in the Human Task Manager Web services API.

BPCGEN.xsd
: Contains data types that are common between the Business Flow Manager and Human Task Manager Web services APIs.

Note: The BPCGen.xsd file is common to both APIs.

What to do next

After you copy the files, you must manually change the Web services API endpoint address the BFMWS.wsdl or HTMWS.wsdl files to that of the WebSphere application server that is hosting the Web services APIs.

Manually changing the Web service endpoint address:

If you copy files from the client CD, you must change the default Web service endpoint address specified in WSDL files to that of the server that is hosting the Web services APIs.

About this task

You can use the administrative console to set the Web service endpoint address before exporting the WSDL files. If, however, you copy the WSDL files from the WebSphere Process Server client CD, you must modify the default Web service endpoint address manually.

The Web service endpoint address to use depends on your WebSphere server configuration:

- Scenario 1. There is a single WebSphere server. The WebSphere endpoint address to specify is the host name and port number of the server, for example **host1:9080**.
- Scenario 2. A WebSphere cluster composed of several servers. The WebSphere endpoint address to specify is the host name and port of the server that is hosting the Web services APIs, for example, **host2:9081**.
- Scenario 3. A Web server is used as a front end. The WebSphere endpoint address to specify is the host name and port of the Web server, for example: **host:80**.

Changing the Business Flow Manager API endpoint:

If you copy the Business Flow Manager API files from the WebSphere Process Server client CD, you must manually edit the default endpoint address.

Procedure

1. Navigate to the directory containing the files copied from the client CD.
2. Open the BFMWS.wsdl file in a text editor or XML editor.
3. Locate the `soap:address` element (towards the bottom of the file).
4. Modify the value of the `location` attribute with the HTTP URL of the server on which the Web service API is running. To do this:
   a. Optionally, replace `http` with `https` to use the more secure HTTPS protocol.

b. Replace *localhost* with the host name or IP address of the Web services APIs server endpoint address.
c. Replace *9080* with the port number of the application server.
d. Replace *BPEContainer_N1_server1* with the context root of the application running the Web services API. The default context root is composed of:
   - *BPEContainer*. The application name.
   - *N1*. The node name.
   - *server1*. The server name.
e. Do not modify the fixed portion of the URL (`/sca/com/ibm/bpe/api/BFMWS`) .

For example, if the application is running on the server **s1.n1.ibm.com** and the server is accepting SOAP/HTTP requests at port **9080**, modify the `soap:address` element as follows:

```
<soap:address location="http://si.n1.ibm.com:9080/
                BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

Changing the Human Task Manager API endpoint:

If you copy the Human Task Manager API files from the WebSphere Process Server client CD, you must manually edit the default endpoint address.

Procedure

1. Navigate to the directory containing the files copied from the client CD.
2. Open the HTMWS.wsdl file in a text editor or XML editor.
3. Locate the `soap:address` element (towards the bottom of the file).
4. Modify the value of the `location` attribute with the correct endpoint address. To do this:
   a. Optionally, replace `http` with `https` to use the more secure HTTPS protocol.
   b. Replace *localhost* with the host name or IP address of the Web services API server's endpoint address.
   c. Replace *9080* with the port number of the application server.
   d. Replace *HTMContainer_N1_server1* with the context root of the application running the Web services API. The default context root is composed of:
      - *HTMContainer*. The application name.
      - *N1*. The node name.
      - *server1*. The server name.
   e. Do not modify the fixed portion of the URL (`/sca/com/ibm/task/api/HTMWS`).

   For example, if the application is running on the server **s1.n1.ibm.com** and the server is accepting SOAP/HTTPS requests at port **9081**, modify the `soap:address` element as follows:

```
<soap:address location="https://si.n1.ibm.com:9081/
                HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

Developing client applications in the Java Web services environment

You can use any Java-based development environment compatible with Java Web services to develop client applications for the Web services APIs.

Generating a proxy client (Java Web services)

Java Web service client applications use a *proxy client* to interact with the Web services APIs.

About this task

A proxy client for Java Web services contains a number of Java Bean classes that the client application calls to perform Web service requests. The proxy client handles the assembly of service parameters into SOAP messages, sends SOAP messages to the Web service over HTTP, receives responses from the Web service, and passes any returned data to the client application.

Basically, therefore, a proxy client allows a client application to call a Web service as if it were a local function.

Note: You only need to generate a proxy client once. All client applications accessing the same Web services API can then use the same proxy client.

In the IBM Web services environment, there are two ways to generate a proxy client:

- Using Rational® Application Developer or WebSphere Integration Developer integrated development environments.
- Using the WSDL2Java command-line tool.

Other Java Web services development environments usually include either the WSDL2Java tool or proprietary client application generation facilities.

Using Rational Application Developer to generate a proxy client:

The Rational Application Developer integrated development environment allows you to generate a proxy client for your client application.

Before you begin

Before generating a proxy client, you must have previously exported the WSDL files that describe the business process or human task Web services interfaces from the WebSphere environment (or the WebSphere Process Server client CD) and copied them to your client programming environment.

Procedure

1. Add the appropriate WSDL file to your project:
   - For business processes:
     a. Unzip the exported file BPEContainer_*nodename_servername*_WSDLFiles.zip to a temporary directory.
     b. Import the subdirectory META-INF from the unzipped directory BPEContainer_*nodename_servername*.ear/b.jar.
   - For human tasks:
     a. Unzip the exported file TaskContainer_*nodename_servername*_WSDLFiles.zip to a temporary directory.
     b. Import the subdirectory META-INF from the unzipped directory TaskContainer_*nodename_servername*.ear/h.jar.

   A new directory wsdl and subdirectory structure are created in your project.
2. Modify the Web Service wizard properties:
   a. In Rational Application Developer, choose **Preferences** → **Web services** → **Code generation** → **IBM WebSphere runtime.**

b. Select the **Generate Java from WSDL using the no wrapped style** option.

Note: If you cannot select the **Web services** option in the **Preferences** menu, you must first enable the required capabilities as follows: **Window** → **Preferences** → **Workbench** → **Capabilities**. Click on **Web Service Developer** and click **OK**. Then reopen the Preferences window and change the **Code Generation** option.

3. Select the BFMWS.WSDL or HTMWS.WSDL file located in the newly-created wsdl directory.
4. Right-click and choose **Web services** → **Generate client**.

   Before continuing with the remaining steps, ensure that the server has started.
5. On the Web Services window, click **Next** to accept all defaults.
6. On the Web Service Selection window, click **Next** to accept all defaults.
7. On the Client Environment Configuration window:
   a. Click **Edit** and change the Web service runtime option to IBM WebSphere
   b. Change the J2EE Version option to 1.4.
   c. Click **OK**.
   d. Click **Next**.
8. This step is only necessary if you need to generate a Web Services client that includes both Business Process and Human Task Web Services APIs, as there are duplicate methods in both WSDL files.
   a. On the Web Service Proxy window, select Define custom mapping for namespace to package then click **OK**.
   b. On the Web Service Client namespace to package mapping window, add the following namespaces and package:

      For BFMWS.wsdl:

Namespace	Package
http://www.ibm.com/xmlns/prod/websphere/business-process/types/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0/Binding	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.bpe

For HTMWS.wsdl:

Namespace	Package
http://www.ibm.com/xmlns/prod/websphere/human-task/types/6.0	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0/Binding	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.task

If asked to confirm overwriting, click **YesToAll**.

9. Click **Finish**.

Results

A proxy client, made up of a number of proxy, locator and helper Java classes, is generated and added to your project. The deployment descriptor is also updated.

Using WSDL2Java to generate a proxy client:

WSDL2Java is a command-line tool that generates a proxy client. A proxy client make it easier to program client applications.

Before you begin

Before generating a proxy client, you must have previously exported the WSDL files that describe the business process or human task Web services APIs from the WebSphere environment (or the WebSphere Process Server client CD) and copied them to your client programming environment.

About this task

Procedure

1. Use the WSDL2Java tool to generate a proxy client: Type:

   wsdl2java *options WSDLfilepath*

   Where:

   - *options* include:

     -noWrappedOperations (-w)
     : Disables the detection of wrapped operations. Java beans for request and response messages are generated.

       Note: This is not the default value.

     -role (-r)
     : Specify the value **client** to generate files and binding files for client-side development.

     -container (-c)
     : The client-side container to use. Valid arguments include:

       client A client container

       ejb An Enterprise JavaBeans (EJB) container.

       none No container

       web A Web container

     -output (-o)
     : The folder in which to store the generated files.

     For a complete list of WSDL2Java parameters, use the **-help** command line switch, or refer to the online help for the WSDL2Java tool in the WID/RAD.

   - *WSDLfilepath* is the path and filename of the WSDL file that you exported from WebSphere environment or copied from the client CD.

   The following example generates a proxy client for the Human Task Activities Web services API:

   ```
   call wsdl2java.bat -r client -c client -noWrappedOperations
                      -output c:\ws\proxyClient  c:\ws\bin\HTMWS.wsdl
   ```

2. Include the generated class files in your project.

Creating helper classes for BPEL processes (Java Web services)

Business objects referenced in concrete API requests (for example, sendMessage, or call) require client applications to use ″document/literal wrapped″ style elements. Client applications require helper classes to help them generate the necessary wrapper elements.

Before you begin

To create helper classes, you must have exported the WSDL file of the Web services API from the WebSphere Process Server environment.

About this task

The call() and sendMessage() operations of the Web services APIs allow interaction with BPEL processes on the WebSphere Process Server. The input message of the call() operation expects the document/literal wrapper of the process input message to be provided.

There are a number of possible techniques for generating helper classes for a BPEL process or human task, including:

1. Use the SoapElement object.

   In the Rational Application Developer environment available in WebSphere Integration Developer, the Web service engine supports JAX-RPC 1.1. In JAX-RPC 1.1, the SoapElement object extends a Document Object Model (DOM) element, so it is possible to use the DOM API to create, read, load, and save SOAP messages.

   For example, assume the WSDL file contains the following input message for a workflow process or human task:

   ```
   <xsd:element name="operation1">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="input1" nillable="true" type="xsd:string"/>
             </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
   ```

   The WSDL file is created when you develop a process or human task module.

   To create the corresponding SOAP message in your client application using the DOM API:

   ```
   SOAPFactory soapfactoryinstance = SOAPFactory.newInstance();
   SOAPElement soapmessage = soapfactoryinstance.createElement
                             ("operation1", namespaceprefix, interfaceURI);
   SOAPElement inputelement = soapfactoryinstance.createElement("input1");
   inputelement.addTextNode( message value);
   soapmessage.addChildElement(outputelement);
   ```

   The following example shows how to create input parameters for the sendMessage operation in your client application:

   ```
   SendMessage inWsend = new SendMessage();
   inWsend.setProcessTemplateName(processtemplatename);
   inWsend.setPortType(porttype);
   inWsend.setOperation(operationname);
   inWsend.set_any(soapmessage);
   ```

2. Use the WebSphere Custom Data Binding feature.

   This technique is described in the following developerWorks articles:

   - How to choose a custom mapping technology for Web services

- Developing Web Services with EMF SDOs for complex XML schema

Interoperability With Patterns and Strategies for Document-Based Web Services

Web Services support for Schema/WSDL(s) containing optional JAX-RPC 1.0/1.1 XML Schema Types

Creating a client application (Java Web services)

A client application sends requests to and receives responses from the Web services APIs. By using a proxy client to manage communications and helper classes to format complex data types, a client application can invoke Web service methods as if they were local functions.

Before you begin

Before starting to create a client application, generate the proxy client and any necessary helper classes.

About this task

You can develop client applications using any Web services-compatible development tool, for example IBM Rational Application Developer (RAD). You can build any type of Web services application to call the Web services APIs.

Procedure

1. Create a new client application project.
2. Generate the proxy client and add the Java helper classes to your project.
3. Code your client application.
4. Build the project.
5. Run the client application.

Example

The following example shows how to use the Business Flow Manager Web service API.

```
// create the proxy
       BFMIFProxy proxy = new BFMIFProxy();
 // prepare the input data for the operation
      GetProcessTemplate iW = new GetProcessTemplate();
      iW.setIdentifier(your_process_template_name);

 // invoke the operation
      GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

 // process output of the operation
      ProcessTemplateType ptd = oW.getProcessTemplate();
      System.out.println("getName= " + ptd.getName());
      System.out.println("getPtid= " + ptd.getPtid());
```

Adding security (Java Web services)

You must secure Web service communications by implementing security mechanisms in your client application.

About this task

WebSphere Application Server currently supports the following security mechanisms for the Web services APIs:

- The user name token
- Lightweight Third Party Authentication (LTPA)

Implementing the user name token:

The user name token security mechanism provides user name and password credentials.

About this task

With the user name token security mechanism, you can choose to implement various *callback handlers*. Depending on your choice:

- You are prompted to supply a user name and password each time you run the client application.
- The user name and password are written into the deployment descriptor.

In either case, the supplied user name and password must match those of an authorized role in the corresponding business process container or human task container.

The user name and password are encapsulated in the request message envelope, and so appear ″in clear″ in the SOAP message header. It is therefore strongly recommended that you configure the client application to use the HTTPS (HTTP over SSL) communications protocol. All communications are then encrypted. You can select the HTTPS communications protocol when you specify the Web service API's endpoint URL address.

To define a user name token:

Procedure

1. Create a security token:
   a. Open the **Deployment Editor** of your module
   b. Click the **WS Extension** tab.
   c. Under **Service References**, the following Web Service References may be listed:
      - service/BFMWSService for business processes
      - service/HTMWSService for human tasks

      Which are listed depends on whether BFMWS.wsdl (for business process), HTMWS.wsdl (for human tasks), or both, were added when generating the proxy client.
   d. For both service references:
      1) Select one of the **Service References**.
      2) Expand the **Request Generator Configuration** section.
      3) Expand the **Security Token** subsection.
      4) Click **Add**. The Security Token window opens.
      5) In the **Name** field, type a name for the new security token: **UserNameTokenBFM** or **UserNameTokenHTM** .

6) In the **Token type** drop-down list, select **Username**. (The **Local name** field is automatically populated with a default value.)
7) Leave the **URI** field blank. No URI value is required for a user name token.
8) Click **OK**.

2. Create a token generator:
   a. Open the **Deployment Editor** of your module
   b. Click on the **WS Binding** tab
   c. Under **Service References**, the same Web Service References are listed as in the previous step:
      - service/BFMWSService for business processes
      - service/HTMWSService for human tasks
   d. For both service references:
      1) Select one of the **Service References**.
      2) Expand the **Security Request Generator Binding Configuration** section.
      3) Expand the **Token Generator** subsection.
      4) Click **Add**. The Token Generator window opens.
      5) In the **Name** field, type a name for the new token generator, such as ″UserNameTokenGeneratorBFM″ or ″UserNameTokenGeneratorHTM″.
      6) In the **Token generator class** field, ensure that the following token generator class is selected: **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**.
      7) In the **Security token** drop-down list, select the appropriate security token that you created earlier.
      8) Select the **Use value type** check box.
      9) In the **Value type** field, select **Username Token**. (The **Local name** field is automatically populated to reflect your choice of **Username Token**.)
      10) In the **Call back handler** field, type either ″com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler″ (which prompts for the user name and password when you run the client application) or ″com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler″.
      11) If you choose **NonPromptCallbackHandler**, you must specify a valid user name and password in the corresponding field of the deployment descriptor.
      12) Click **OK**.

Related information

IBM WebSphere Developer Technical Journal: Web services security with WebSphere Application Server V6

Implementing the LTPA security mechanism:

The Lightweight Third Party Authentication (LTPA) security mechanism can be used when the client application is running within a previously established security context.

About this task

The LTPA security mechanism is only available if your client application is running in a secure environment in which a security context has already been established. For example, if your client application is running in an Enterprise JavaBeans (EJB) container, then the EJB client must log in before being able to invoke the client application. A security context is then established. If the EJB client application then invokes a Web service, the LTPA callback handler retrieves the LTPA token from the security context and adds it to the SOAP request message. On the server side, the LTPA token is handled by the LTPA mechanism.

To implement the LTPA security mechanism:

Procedure

1. In the Rational Application Developer environment available in WebSphere Integration Developer, choose **WS Binding** → **Security Request Generator Binding Configuration** → **Token Generator**.
2. Create a security token:
   a. Open the **Deployment Editor** of your module
   b. Click the **WS Extension** tab.
   c. Under **Service References**, the following **Web Service References** may be listed:
      - service/BFMWSService for business processes
      - service/HTMWSService for human tasks

      Which are listed depends on whether BFMWS.wsdl (for business process), HTMWS.wsdl (for human tasks), or both, were added when generating the proxy client.
   d. For both service references:
      1) Select one of the **Service References**.
      2) Expand the **Request Generator Configuration** section.
      3) Expand the **Security Token** subsection.
      4) Click **Add**. The Security Token window opens.
      5) In the **Name** field, type a name for the new security token: **LTPATokenBFM** or **LTPATokenHTM** .
      6) In the **Token type** drop-down list, select **LTPAToken**. (The **URI** and **Local name** fields are automatically populated with default values.)
      7) Click **OK**.
3. Create a token generator:
   a. Open the **Deployment Editor** of your module
   b. Click on the **WS Binding** tab
   c. Under **Service References**, the same Web Service References are listed as in the previous step:
      - service/BFMWSService for business processes
      - service/HTMWSService for human tasks
   d. For both service references:
      1) Select one of the **Service References**.
      2) Expand the **Security Request Generator Binding Configuration** section.
      3) Expand the **Token Generator** subsection.

4) Click **Add**. The Token Generator window opens.
5) In the **Name** field, type a name for the new token generator, such as ″LTPATokenGeneratorBFM″ or ″LTPATokenGeneratorHTM″.
6) In the **Token generator class** field, ensure that the following token generator class is selected: **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**.
7) In the **Security token** drop-down list, select the appropriate security token that you created earlier.
8) Select the **Use value type** check box.
9) In the **Value type** field, select **LTPAToken**. (The **URI** and **Local name** fields are automatically populated to reflect your choice of **LTPA Token**.)
10) In the **Call back handler** field, type either ″com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler″.
11) Click **OK**.

Results

At runtime, the **LTPATokenCallbackHandler** retrieves the LTPA token from the existing security context and adds it to the SOAP request message.

Adding transaction support (Java Web services)

Java Web service client applications can be configured to allow server-side request processing to participate in the client's transaction, by passing a client application context as part of the service request. This atomic transaction support is defined in the Web Services-Atomic Transaction (WS-AT) specification.

About this task

WebSphere Application Server runs each Web services API request as a separate atomic transaction. Client applications can be configured to use transaction support in one of the following ways:

- Participate in the transaction. Server-side request processing is performed within the client application transaction context. Then, if the server encounters a problem while the Web services API request is running and rolls back, the client application's request is also rolled back.
- Not use transaction support. WebSphere Application Server still creates a new transaction in which to run the request, but server-side request processing is not performed with the client application transaction context.

Developing client applications in the .NET environment

Microsoft .NET offers a powerful development environment in which to connect applications through Web services.

Generating a proxy client (.NET)

.NET client applications use a *proxy client* to interact with the Web service APIs. A proxy client shields client applications from the complexity of the Web service messaging protocol.

Before you begin

To create a proxy client, you must first export a number of WSDL files from the WebSphere environment and copy them to your client programming environment.

Note: If you have the WebSphere Process Server client CD, you can copy the files from there instead.

About this task

A proxy client comprises a set of C# bean classes. Each class contains all the methods and objects exposed by a single Web service. The service methods handle the assembly of parameters into complete SOAP messages, send SOAP messages to the Web service over HTTP, receives responses from the Web service, and handle any returned data.

Note: You only need to generate a proxy client once. All client applications accessing the Web services APIs can then use the same proxy client.

Procedure

1. Use the WSDL command to generate a proxy client: Type:

   wsdl *options WSDLfilepath*

   Where:

   - *options* include:

     /language
     : Allows you to specify the language used to create the proxy class. The default is C#. You can also specify **VB** (Visual Basic), **JS** (JScript), or **VJS** (Visual J#) as the language argument.

     /output
     : The name of the output file, with the appropriate suffix. For example, proxy.cs

     /protocol
     : The protocol implemented in the proxy class. **SOAP** is the default setting.

     For a complete list of **WSDL.exe** parameters , use the **/?** command line switch, or refer to the online help for the WSDL tool in Visual Studio.
   - *WSDLfilepath* is the path and filename of the WSDL file that you exported from the WebSphere environment or copied from the client CD.

   The following example generates a proxy client for the Human Task Manager Web services API:

   ```
   wsdl /language:cs /output:proxyclient.cs c:\ws\bin\HTMWS.wsdl
   ```
2. Compile the proxy client as a Dynamic Link Library (DLL) file.

Creating helper classes for BPEL processes (.NET)

Certain Web services API operations require client applications to use ″document/literal″ style wrapped elements. Client applications require helper classes to help them generate the necessary wrapper elements.

Before you begin

To create helper classes, you must have exported the WSDL file of the Web services API from the WebSphere Process Server environment.

About this task

The call() and sendMessage() operations of the Web services APIs cause BPEL processes to be launched within WebSphere Process Server. The input message of

the call() operation expects the document/literal wrapper of the BPEL process input message to be provided. To generate the necessary beans and classes for the BPEL process, copy the `<wsdl:types>` element into a new XSD file, then use the xsd.exe tool to generate helper classes.

Procedure

1. If you have not already done so, export the WSDL file of the BPEL process interface from WebSphere Integration Developer.
2. Open the WSDL file in a text editor or XML editor.
3. Copy the contents of all child elements of the `<wsdl:types>` element and paste it into a new, skeleton, XSD file.
4. Run the `xsd.exe` tool on the XSD file:

   call xsd.exe file.xsd /classes /o

   Where:

   file.xsd
   The XML Schema Definition file to convert.

   /classes (/c)
   Generate helper classes that correspond to the contents of the specified XSD file or files.

   /output (/o)
   Specify the output directory for generated files. If this directory is omitted, the default is the current directory.

   For example:

   call xsd.exe ProcessCustomer.xsd /classes /output:c:\temp
5. Add the class file that is generated to your client application. If you are using Visual Studio, for example, you can do this using the **Project → Add Existing Item** menu option.

Example

If the ProcessCustomer.wsdl file contains the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                  xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  name="ProcessCustomer"
                  targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
  <wsdl:types>
    <xsd:schema targetNamespace="http://ProcessTypes/bpel/ProcessCustomer"
                xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
                schemaLocation="xsd-includes/http.com.ibm.bpe.unittest.sca.xsd"/>
      <xsd:element name="doit">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input1" nillable="true" type="bons1:Customer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="doitResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="output1" nillable="true" type="bons1:Customer"/>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
  <wsdl:message name="doitRequestMsg">
  <wsdl:part element="tns:doit" name="doitParameters"/>
</wsdl:message>
  <wsdl:message name="doitResponseMsg">
  <wsdl:part element="tns:doitResponse" name="doitResult"/>
</wsdl:message>
  <wsdl:portType name="ProcessCustomer">
  <wsdl:operation name="doit">
    <wsdl:input message="tns:doitRequestMsg" name="doitRequest"/>
    <wsdl:output message="tns:doitResponseMsg" name="doitResponse"/>
  </wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
```

The resulting XSD file contains:

```
<xsd:schema xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                      xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
 <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
              schemaLocation="Customer.xsd"/>
 <xsd:element name="doit">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="input1" type="bons1:Customer" nillable="true"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="doitResponse">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="output1" type="bons1:Customer" nillable="true"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Related information

Microsoft documentation for the XML Schema Definition Tool (XSD.EXE)

Creating a client application (.NET)

A client application sends requests to and receives responses from the Web services APIs. By using a proxy client to manage communications and helper classes to format complex data types, a client application can invoke Web service methods as if they were local functions.

Before you begin

Before starting to create a client application, generate the proxy client and any necessary helper classes.

About this task

You can develop .NET client applications using any .NET-compatible development tool, for example, Visual Studio .NET. You can build any type of .NET application to call the generic Web service APIs.

Procedure

1. Create a new client application project. For example, create a **WinFX Windows® Application** in Visual Studio.
2. In the project options, add a reference to the Dynamic Link Library (DLL) file of the proxy client. Add all of the helper classes that contain business object definitions to your project. In Visual Studio, for example, you can do this using the **Project → Add existing item** option.
3. Create a proxy client object. For example:

```
HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService service =
    new HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService();
```

4. Declare any business object data types used in messages to be sent to or received from the Web service. For example:

```
HTMClient.HTMReference.TKIID id = new HTMClient.HTMReference.TKIID();

ClipBG bg = new ClipBG();
Clip clip = new Clip();
```

5. Call specific Web service functions and specify any required parameters. For example, to create and start a human task:

```
HTMClient.HTMReference.createAndStartTask task =
   new HTMClient.HTMReference.createAndStartTask();
HTMClient.HTMReference.StartTask sTask = new HTMClient.HTMReference.StartTask();

sTask.taskName = "SimpleTask";
sTask.taskNamespace = "http://myProcess/com/acme/task";
sTask.inputMessage = bg;
task.inputTask = sTask;

id = service.createAndStartTask(task).outputTask;
```

6. Remote processes and tasks are identified with persistent IDs (`id` in the example in the previous step). For example, to claim a previously created human task:

```
HTMClient.HTMReference.claimTask claim = new HTMClient.HTMReference.claimTask();
claim.inputTask = id;
```

Adding security (.NET)

You can secure Web service communications by integrating security mechanisms into your client application.

About this task

These security mechanisms can include user name token (user name and password), or custom binary and XML-based security tokens.

Procedure

1. Download and install the Web Services Enhancements (WSE) 2.0 SP3 for Microsoft .NET. This is available from:

   http://www.microsoft.com/downloads/details.aspx?familyid=1ba1f631-c3e7-420a-bc1e-ef18bab66122&displaylang=en
2. Modify the generated proxy client code as follows.

   Change:

```
public class Export1_MyMicroflowHttpService : System.Web.Services.Protocols.SoapHttpClientProtocol {
```

   To:

```
public class Export1_MyMicroflowHttpService : Microsoft.Web.Services2.WebServicesClientProtocol {
```

Note: These modifications are lost if you regenerate the proxy client by running the WSDL.exe tool.

3. Modify the client application code by adding the following lines at the top of the file:

```
using System.Web.Services.Protocols;
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Security.Tokens;
...
```

4. Add code to implement the desired security mechanism. For example, the following code adds user name and password protection:

```
string user = "U1";
string pwd = "password";
UsernameToken token =
   new UsernameToken(user, pwd, PasswordOption.SendPlainText);

me._proxy.RequestSoapContext.Security.Tokens.Clear();
me._proxy.RequestSoapContext.Security.Tokens.Add(token);
```

Querying business-process and task-related objects

You can use the Web services APIs to query business-process and task-related objects in the Business Process Choreographer database to retrieve specific properties of these objects.

About this task

The Business Process Choreographer database stores template (model) and instance (runtime) data for managing business processes and tasks.

Through the Web services APIs, client applications can issue queries to retrieve information from the database about business processes and tasks.

Client applications can issue a one-off query to retrieve a specific property of an object. Queries that you use often can be saved. These stored queries can then be retrieved and used by your client application.

Queries on business-process and task-related objects using the Web services APIs

Use the query interface of the Web services APIs to obtain information about business processes and tasks.

Client applications use an SQL-like syntax to query the database.

Example for Java Web services

```
string processTemplateName = "ProcessCustomerLR";
query query1 = new query();
query1.selectClause = "DISTINCT PROCESS_INSTANCE.STARTED, PROCESS_INSTANCE.PIID";
query1.whereClause =
      "PROCESS_INSTANCE.TEMPLATE_NAME = '" + processTemplateName + "'";
query1.orderByClause = "PROCESS_INSTANCE.STARTED";
query1.threshold = null;
query1.timeZone = "UTC"; query1.skipTuples = null;
queryResponse queryResponse1 = proxy.query(query1);
```

Information retrieved from the database is returned through the Web services APIs as a *query result set*.

For example:

```
QueryResultSetType queryResultSet = queryResponse1.queryResultSet;
if (queryResultSet != null) {
   Console.WriteLine("--> QueryResultSetType");
   Console.WriteLine(" . size= " + queryResultSet.size);
   Console.WriteLine(" . numberColumns= " + queryResultSet.numberColumns);
   string indent = " . ";

   // -- the query column info
   QueryColumnInfoType[] queryColumnInfo = queryResultSet.QueryColumnInfo;
   if (queryColumnInfo.Length > 0) {
      Console.WriteLine();
      Console.WriteLine("= . QueryColumnInfoType size= " + queryColumnInfo.Length);
      Console.Write( " | tableName ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].tableName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | columnName ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].columnName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | data type ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         QueryColumnInfoTypeType tt = queryColumnInfo[i].type;
         Console.WriteLine( " | " + tt.ToString());
      }
         Console.WriteLine();
   }
   else {
      Console.WriteLine("--> queryColumnInfo= <null>");
   }

   // - the query result values
   string[][] result = queryResultSet.result;
   if (result !=null) {
      Console.WriteLine();
      Console.WriteLine("= . result size= " + result.Length);
      for (int i = 0; i &lt; result.Length; i++) {
        Console.Write(indent +i );
        string[] row = result[i];
        for (int j = 0; j &lt; row.Length; j++ ) {
           Console.Write(" | " + row[j]);
        }
        Console.WriteLine();
      }
   }
   else {
      Console.WriteLine("--> result= <null>");
   }
}
else {
   Console.WriteLine("--> QueryResultSetType= <null>");
}
```

The query function returns objects according to the caller's authorization. The query result set only contains the properties of those objects that the caller is authorized to see.

Predefined database views are provided for you to query the object properties. For process templates, the query function has the following syntax:

```
ProcessTemplateData[] queryProcessTemplates
                       (java.lang.String whereClause,
                        java.lang.String orderByClause,
                        java.lang.Integer threshold,
                        java.util.TimeZone timezone);
```

For task templates, the query function has the following syntax:

```
TaskTemplate[] queryTaskTemplates
                (java.lang.String whereClause,
                 java.lang.String orderByClause,
                 java.lang.Integer threshold,
                 java.util.TimeZone timezone);
```

For the other business-process and task-related objects, the query function has the following syntax:

```
QueryResultSet query (java.lang.String selectClause,
                      java.lang.String whereClause,
                      java.lang.String orderByClause,
                      java.lang.Integer skipTuples
                      java.lang.Integer threshold,
                      java.util.TimeZone timezone);
```

The query interface also contains a queryAll method. You can use this method to retrieve all of the relevant data about an object, for example, for monitoring purposes. The caller of the queryAll method must have one of the following Java 2 Platform, Enterprise Edition (J2EE) roles: BPESystemAdministrator, BPESystemMonitor, TaskSystemAdministrator, or TaskSystemMonitor. Authorization checking using the corresponding work item of the object is not applied.

Example for .NET

```
ProcessTemplateType[] templates = null;

   try {
        queryProcessTemplates iW = new queryProcessTemplates();
        iW.whereClause = "PROCESS_TEMPLATE.STATE=PROCESS_TEMPLATE.STATE.STATE_STARTED";
        iW.orderByClause = null;
        iW.threshold = null;
        iW.timeZone = null;

        Console.WriteLine("--> queryProcessTemplates ... ");
        Console.WriteLine("--> query: WHERE " + iW.whereClause + " ORDER BY " +
          iW.orderByClause + " THRESHOLD " + iW.threshold + " TIMEZONE" + iW.timeZone);

        templates = proxy.queryProcessTemplates(iW);

        if (templates.Length < 1) {
          Console.WriteLine("--> No templates found :-(");
        }
        else {
          for (int i = 0; i < templates.Length ; i++) {
            Console.Write("--> found template with ptid: " + templates[i].ptid);
            Console.WriteLine(" and name: " + templates[i].name);
           /*  ... other properties of ProcessTemplateType ... */
          }
        }
      }
      catch (Exception e) {
        Console.WriteLine("exception= " + e);
      }
```

Managing stored queries

Stored queries provide a way to save queries that are run often. The stored query can be either a query that is available to all users (public query), or a query that belongs to a specific user (private query).

About this task

A stored query is a query that is stored in the database and identified by a name. A private and a public stored query can have the same name; private stored queries from different owners can also have the same name.

You can have stored queries for business process objects, task objects, or a combination of these two object types.

Managing public stored queries
Public stored queries are created by the system administrator. These queries are available to all users.

Managing private stored queries for other users
Private queries can be created by any user. These queries are available only to the owner of the query and the system administrator.

Working with your private stored queries
If you are not a system administrator, you can create, run, and delete your own private stored queries. You can also use the public stored queries that the system administrator created.

Developing client applications using the Business Process Choreographer JMS API

You can develop client applications that access business process applications asynchronously through the Java Messaging Service (JMS) API.

About this task

JMS client applications exchange request and response messages with the JMS API. To create a request message, the client application fills a JMS TextMessage message body with an XML element representing the document/literal wrapper of the corresponding operation.

Related concepts

Comparison of the programming interfaces for interacting with business processes and human tasks
Enterprise JavaBeans (EJB), Web service, and Java Message Service (JMS), and Representational State Transfer Services (REST) generic programming interfaces are available for building client applications that interact with business processes and human tasks. Each of these interfaces has different characteristics.

Requirements for business processes

Business processes developed with the WebSphere Integration Developer to run on the Business Process Choreographer must conform to specific rules to be accessible through the JMS API.

The requirements are:

1. The interfaces of business processes must be defined using the ″document/literal wrapped″ style defined in the Java API for XML-based RPC

(JAX-RPC 1.1) specification. This is the default style for all business processes and human tasks developed with the WebSphere Integration Developer.

2. Fault messages exposed by business processes and human tasks for Web service operations must comprise a single WSDL message part defined with an XML Schema element. For example:

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

Related information

Java API for XML based RPC (JAX-RPC) downloads page

Which style of WSDL should I use?

Authorization for JMS renderings

To authorize use of the JMS interface, security settings must be enabled in WebSphere Application Server.

When the business process container is installed, the role **JMSAPIUser** must be mapped to a user ID. This user ID is used to issue all JMS API requests. For example, if **JMSAPIUser** is mapped to ″User A″, all JMS API requests appear to the process engine to originate from ″User A″.

The **JMSAPIUser** role must be assigned the following authorities:

Request	Required authorization
forceTerminate	Process administrator
sendEvent	Potential activity owner or process administrator

Note: For all other requests, no special authorizations are required.

Special authority is granted to a person with the role of business process administrator. A business process administrator is a special role; it is different from the process administrator of a process instance. A business process administrator has all privileges.

You cannot delete the user ID of the process starter from your user registry while the process instance exists. If you delete this user ID, the navigation of this process cannot continue. You receive the following exception in the system log file:

```
no unique ID for: <user ID>
```

Accessing the JMS interface

To send and receive messages through the JMS interface, an application must first create a connection to the BPC.cellname.Bus, create a session, then generate message producers and consumers.

About this task

The process server accepts Java Message Service (JMS) messages that follow the point-to-point paradigm. An application that sends or receives JMS messages must perform the following actions.

The following example assumes that the JMS client is executed in a managed environment (EJB, application client, or Web client container). If you want to

execute the JMS client in a J2SE environment, refer to "IBM Client for JMS on J2SE with IBM WebSphere Application Server" at http://www-1.ibm.com/support/docview.wss?uid=swg24012804.

Procedure

1. Create a connection to the BPC.*cellname*.Bus. No preconfigured connection factory exists for a client application's requests: a client application can either use the JMS API's ReplyConnectionFactory or create its own connection factory, in which case it can use Java Naming and Directory Interface (JNDI) lookup to retrieve the connection factory. The JNDI-lookup name must be the same as the name specified when configuring the Business Process Choreographer's external request queue. The following example assumes the client application creates its own connection factory named "jms/clientCF".

```
//Obtain the default initial JNDI context.
Context initialContext = new InitialContext();

// Look up the connection factory.
// Create a connection factory that connects to the BPC bus.
// Call it, for example, "jms/clientCF".
// Also configure an appropriate authentication alias.
ConnectionFactory connectionFactory =
            (ConnectionFactory)initialcontext.lookup("jms/clientCF");

// Create the connection.
Connection connection = connectionFactory.createConnection();
```

2. Create a session so that message producers and consumers can be created.

```
// Create a transaction session using auto-acknowledgement.
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. Create a message producer to send messages. The JNDI-lookup name must be the same as the name specified when configuring the Business Process Choreographer's external request queue.

```
// Look up the destination of the Business Process Choreographer input queue to
// send messages to.
Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");

// Create a message producer.
MessageProducer producer = session.createProducer(sendQueue);
```

4. Create a message consumer to receive replies. The JNDI-lookup name of the reply destination can specify a user-defined destination, but it can also specify the default (Business Process Choreographer-defined) reply destination `jms/BFMJMSReplyQueue`. In both cases, the reply destination must lie on the `BPC.<cellname>.Bus`.

```
// Look up the destination of the reply queue.
Queue  replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

// Create a message consumer.
MessageConsumer consumer = session.createConsumer(replyQueue);
```

5. Send a message.

```
// Start the connection.
connection.start();

// Create a message - see the task descriptions for examples - and send it.
// This method is defined elsewhere ...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Set mandatory JMS header.
// targetFunctionName is the operation name of JMS API
// (for example, getProcessTemplate, sendMessage)
```

```
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Set the reply queue; this is mandatory if the replyQueue
// is not the default queue (as it is in this example).
requestMessage.setJMSReplyTo(replyQueue);

// Send the message.
producer.send(requestMessage);

// Get the message ID.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. Receive the reply.

```
// Receive the reply message and analyse the reply.
TextMessage  replyMessage = (TextMessage) consumer.receive();

// Get the payload.
String payload = replyMessage.getText();

session.commit();
```

7. Close the connection and free the resources.

```
 // Final housekeeping; free the resources.
session.close();
connection.close();
```

Note: It is not necessary to close the connection after each transaction. Once a connection has been started, any number of request and response messages can be exchanged before the connection is closed. The example shows a simple case with a single call within a single business method.

Structure of a Business Process Choreographer JMS message

The header and body of each JMS message must have a predefined structure.

A Java Message Service (JMS) message consists of:

- A message header for message identification and routing information.
- The body (payload) of the message that holds the content.

The Business Process Choreographer supports text message formats only.

Message header

JMS allows clients to access a number of message header fields.

The following header fields can be set by a Business Process Choreographer JMS client:

- **JMSReplyTo**

  The destination to send a reply to the request. If this field is not specified in the request message, the reply is sent to the Export interface's default reply destination (an Export is a client interface rendering of a business process component). This destination can be obtained using `initialContext.lookup(″jms/BFMJMSReplyQueue″);`
- **TargetFunctionName**

  The name of the WSDL operation, for example, ″queryProcessTemplates″. This field must always be set. Note that the TargetFunctionName specifies the operation of the generic JMS message interface described here. This should not be confused with operations provided by concrete processes or tasks that can be invoked indirectly, for example, using the **call** or **sendMessage** operations.

A Business Process Choreographer client can also access the following header fields:

- **JMSMessageID**

  Uniquely identifies a message. Set by the JMS provider when the message is sent. If the client sets the JMSMessageID before sending the message, it is overwritten by the JMS provider. If the ID of the message is required for authentication purposes, the client can retrieve the JMSMessageID after sending the message.
- **JMSCorrelationID**

  Links messages. Do not set this field. A Business Process Choreographer reply message contains the JMSMessageID of the request message.

Each response message contains the following JMS header fields:

- **IsBusinessException**

  ″False″ for WSDL output messages, or ″true″ for WSDL fault messages.

ServiceRuntimeExceptions are not returned to asynchronous client applications. When a severe exception occurs during the processing of a JMS request message, it results in a runtime failure, causing the transaction that is processing this request message to roll back. The JMS request message is then delivered again. If the failure occurs early, during processing of the message as part of the SCA Export (for example, while deserializing the message), retries are attempted up to the maximum number of failed deliveries specified by the SCA Export's receive destination. After the maximum number of failed deliveries is reached, the request message is added to the system exception destination of the Business Process Choreographer bus. If, however, the failure occurs during actual processing of the request by the Business Flow Manager's SCA component, the failed request message is handled by the WebSphere Process Server's failed event management infrastructure, that is, it may end up in the failed event management database if retries do not resolve the exceptional situation.

Message body

The JMS message body is a String containing an XML document representing the document/literal wrapper element of the operation.

A simple example of a valid request message body is:

```
<?xml version="1.0" encoding="UTF-8"?>
<_6:queryProcessTemplates xmlns:_6="http://www.ibm.com/xmlns/prod/
      websphere/business-process/services/6.0">
<whereClause>PROCESS_TEMPLATE.STATE IN (1)</whereClause>
</_6:queryProcessTemplates>
```

Copying artifacts for JMS client applications

A number of artifacts can be copied from the WebSphere Process Server environment to help in the creation of JMS client applications.

About this task

These artifacts are mandatory only if you use the BOXMLSerializer to create the JMS message body. For the JMS API, these artifacts are:

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

wsa.xsd

You can obtain these artifacts in the following ways:

- Publish and export the artifacts from the WebSphere Process Server environment.

  These client artifacts are in the *install_root*\ProcessChoreographer\client directory.
- Copy files from the *install_root*\ProcessChoreographer\client directory on the WebSphere Process Server client CD.

Results

Checking the response message for business exceptions

JMS client applications must check the message header of all response messages for business exceptions.

About this task

A JMS client application must first check the **IsBusinessException** property in the response message's header.

For example:

Example

```
// receive response message
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject().receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse is a bussiness fault
  // any api can end w/a processFaultMsg
  // the call api also w/a businessFaultMsg
 }
else {
  // strResponse is the output message
}
```

Example: executing a long running process using the Business Process Choreographer JMS API

This example shows how to create a generic client application that uses the JMS API to work with long-running processes.

Procedure

1. Set up the JMS environment, as described in "Accessing the JMS interface" on page 313.
2. Obtain a list of installed process definitions.
   - Send `queryProcessTemplates`.
   - This returns a list of ProcessTemplate objects.
3. Obtain a list of start activities (receive or pick with `createInstance="yes"`).
   - Send `getStartActivities`.
   - This returns a list of InboundOperationTemplate objects.
4. Create an input message. This is environment-specific, and might require the use of predeployed, process-specific artifacts.

5. Create a process instance.
   - Issue a `sendMessage`.

   With the JMS API, you can also use the `call` operation for interacting with long-running, request-response operations provided by a business process. This operation returns the operation result or fault to the specified reply-to destination, even after a long period of time. Therefore, if you use the `call` operation, you do not need to use the `query` and `getOutputMessage` operations to obtain the process' output or fault message.
6. Optional: Obtain output messages from the process instances by repeating the following steps:
   a. Issue `query` to obtain the finished state of the process instance.
   b. Issue `getOutputMessage` .
7. Optional: Work with additional operations exposed by the process:
   a. Issue `getWaitingActivities` or `getActiveEventHandlers` to obtain a list of InboundOperationTemplate objects.
   b. Create input messages.
   c. Send messages with `sendMessage`.
8. Optional: Get and set custom properties that are defined on the process or contained activities with `getCustomProperties` and `setCustomProperties`.
9. Finish working with a process instance:
   a. Send `delete` and `terminate` to finish working with the long-running process.

Developing Web applications for business processes and human tasks, using JSF components

Business Process Choreographer provides several JavaServer Faces (JSF) components. You can extend and integrate these components to add business-process and human-task functionality to Web applications.

About this task

You can use WebSphere Integration Developer to build your Web application. For applications that include human tasks, you can generate a JSF custom client. For more information on generating a JSF client, go to the information center for WebSphere Integration Developer.

You can also develop your Web client using the JSF components provided by Business Process Choreographer.

Procedure

1. Create a dynamic project and change the Web Project Features properties to include the JSF base components.

   For more information on creating a Web project, go to the information center for WebSphere Integration Developer.
2. Add the prerequisite Business Process Choreographer Explorer Java archive (JAR files).

   Add the following files to the WEB-INF/lib directory of your project:
   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar

- bpcjsfcomponents.jar

If you are deploying your Web application on a remote server, also add the following files. These files are needed for remotely accessing the Business Process Choreographer APIs.

- bpe137650.jar
- task137650.jar

In WebSphere Process Server, all of these files are in the following directory:

- On Windows systems: *install_root*\ProcessChoreographer\client
- On UNIX®, Linux®, and i5/OS® systems: *install_root*/ProcessChoreographer/client

3. Add the EJB references that you need to the Web application deployment descriptor, web.xml file.

```
<ejb-ref id="EjbRef_1">
  <ejb-ref-name>ejb/BusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
  <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
<ejb-ref id="EjbRef_2">
  <ejb-ref-name>ejb/HumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.task.api.HumanTaskManagerHome</home>
  <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
<ejb-local-ref id="EjbLocalRef_1">
  <ejb-ref-name>ejb/LocalBusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
  <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
</ejb-local-ref>
<ejb-local-ref id="EjbLocalRef_2">
  <ejb-ref-name>ejb/LocalHumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
  <local>com.ibm.task.api.LocalHumanTaskManager</local>
</ejb-local-ref>
```

4. Add the Business Process Choreographer Explorer JSF components to the JSF application.

   a. Add the tag library references that you need for your applications to the JavaServer Pages (JSP) files. Typically, you need the JSF and HTML tag libraries, and the tag library required by the JSF components.

      - `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>`
      - `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`
      - `<%@ taglib uri="http://com.ibm.bpe.jsf/taglib" prefix="bpe" %>`

   b. Add an `<f:view>` tag to the body of the JSP page, and an `<h:form>` tag to the `<f:view>` tag.

   c. Add the JSF components to the JSP files.

      Depending on your application, add the List component, the Details component, the CommandBar component, or the Message component to the JSP files. You can add multiple instances of each component.

   d. Configure the managed beans in the JSF configuration file.

      By default, the configuration file is the faces-config.xml file. This file is in the WEB-INF directory of the Web application.

      Depending on the component that you add to your JSP file, you also need to add the references to the query and other wrapper objects to the JSF

configuration file. To ensure correct error handling, you also need to define both an error bean and a navigation target for the error page in the JSF configuration file. Ensure that you use BPCError for the name of the error bean and error for the name of the navigation target of the error page.

```
<faces-config>
...
<managed-bean>
  <managed-bean-name>BPCError</managed-bean-name>
  <managed-bean-class>com.ibm.bpc.clientcore.util.ErrorBeanImpl
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

...
<navigation-rule>
...
<navigation-case>
<description>
The general error page.
</description>
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

In error situations that trigger the error page, the exception is set on the error bean.

e. Implement the custom code that you need to support the JSF components.

5. Deploy the application.

If you are deploying the application in a network deployment environment, change the target resource Java Naming and Directory Interface (JNDI) names to values where the Business Flow Manager and Human Task Manager APIs can be found in your cell.

- If your business process containers are configured on another server in the same managed cell, the names have the following structure:

```
cell/nodes/nodename/servers/servername/com/ibm/bpe/api/BusinessManagerHome
cell/nodes/nodename/servers/servername/com/ibm/task/api/HumanTaskManagerHome
```

- If your business process containers are configured on a cluster in the same cell, the names have the following structure:

```
cell/clusters/clustername/com/ibm/bpe/api/BusinessFlowManagerHome
cell/clusters/clustername/com/ibm/task/api/HumanTaskManagerHome
```

Map the EJB references to the JNDI names or manually add the references to the ibm-web-bnd.xmi file.

The following table lists the reference bindings and their default mappings.

Table 50. Mapping of the reference bindings to JNDI names

Reference binding	JNDI name	Comments
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Remote session bean
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Local session bean
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Remote session bean
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Local session bean

Results

Your deployed Web application contains the functionality provided by the Business Process Choreographer Explorer components.

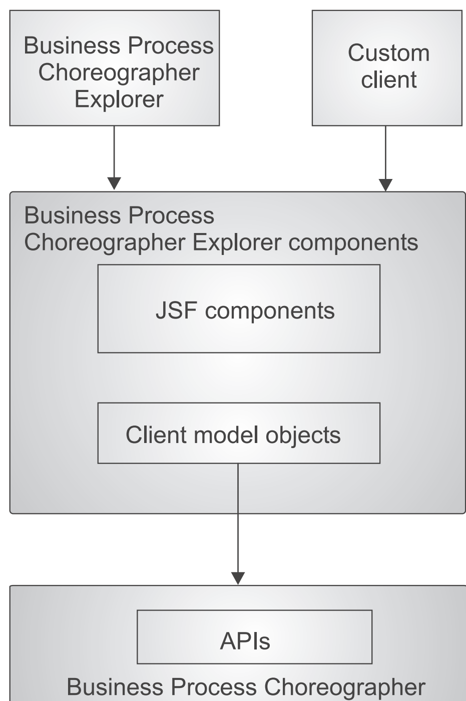
What to do next

If you are using custom JSPs for the process and task messages, you must map the Web modules that are used to deploy the JSPs to the same servers that the custom JSF client is mapped to.

Business Process Choreographer Explorer components

The Business Process Choreographer Explorer components are a set of configurable, reusable elements that are based on the JavaServer Faces (JSF) technology. You can imbed these elements in Web applications. The Web applications can then access installed business process and human task applications.

The components consist of a set of JSF components and a set of client model objects. The relationship of the components to Business Process Choreographer, Business Process Choreographer Explorer, and other custom clients is shown in the following figure.



JSF components

The Business Process Choreographer Explorer components include the following JSF components. You imbed these JSF components in your JavaServer Pages (JSP) files when you build Web applications for working with business processes and human tasks.

- List component

The List component displays a list of application objects in a table, for example, tasks, activities, process instances, process templates, work items, or escalations. This component has an associated list handler.

- Details component

  The Details component displays the properties of tasks, work items, activities, process instances, and process templates. This component has an associated details handler.
- CommandBar component

  The CommandBar component displays a bar with buttons. These buttons represent commands that operate on either the object in a details view or the selected objects in a list. These objects are provided by a list handler or a details handler.
- Message component

  The Message component displays a message that can contain either a Service Data Object (SDO) or a simple type.

Client model objects

The client model objects are used with the JSF components. The objects implement some of the interfaces of the underlying Business Process Choreographer API and wrap the original object. The client model objects provide national language support for labels and converters for some properties.

Error handling in JSF components

The JavaServer Faces (JSF) components exploit a predefined managed bean, BPCError, for error handling. In error situations that trigger the error page, the exception is set on the error bean.

This bean implements the com.ibm.bpc.clientcore.util.ErrorBean interface. The error page is displayed in the following situations:

- If an error occurs during the execution of a query that is defined for a list handler, and the error is generated as a ClientException error by the execute method of a command
- If a ClientException error is generated by the execute method of a command and this error is not an ErrorsInCommandException error nor does it implement the CommandBarMessage interface
- If an error message is displayed in the component, and you follow the hyperlink for the message

A default implementation of the com.ibm.bpc.clientcore.util.ErrorBeanImpl interface is available.

The interface is defined as follows:

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * This setter method call allows a locale and
     * the exception to be passed. This allows the
     * getExceptionMessage methods to return localized Strings
     *
     */
    public void setException(Exception ex, Locale locale);
```

```
    public Exception getException();
    public String getStack();
    public String getNestedExceptionMessage();
    public String getNestedExceptionStack();
    public String getRootExceptionMessage();
    public String getRootExceptionStack();

    /*
     * This method returns the exception message
     * concatenated recursively with the messages of all
     * the nested exceptions.
     */
    public String getAllExceptionMessages();

    /*
     * This method is returns the exception stack
     * concatenated recursively with the stacks of all
     * the nested exceptions.
     */
    public String getAllExceptionStacks();
}
```

Default converters and labels for client model objects

The client model objects implement the corresponding interfaces of the Business Process Choreographer API.

The List component and the Details component operate on any bean. You can display all of the properties of a bean. However, if you want to set the converters and labels that are used for the properties of a bean, you must use either the `column` tag for the List component, or the `property` tag for the Details component. Instead of setting the converters and labels, you can define default converter and labels for the properties by defining the following static methods. You can define the following static methods:

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

The following table shows the client model objects that implement the corresponding Business Flow Manager and Human Task Manager API classes and provide default labels and converter for their properties. This wrapping of the interfaces provides locale-sensitive labels and converters for a set of properties. The following table shows the mapping of the Business Process Choreographer interfaces to the corresponding client model objects.

Table 51. How Business Process Choreographer interfaces are mapped to client model objects

Business Process Choreographer interface	Client model object class
com.ibm.bpe.api.ActivityInstanceData	com.ibm.bpe.clientmodel.bean.ActivityInstanceBean
com.ibm.bpe.api.ActivityServiceTemplateData	com.ibm.bpe.clientmodel.bean.ActivityServiceTemplateBean
com.ibm.bpe.api.ProcessInstanceData	com.ibm.bpe.clientmodel.bean.ProcessInstanceBean
com.ibm.bpe.api.ProcessTemplateData	com.ibm.bpe.clientmodel.bean.ProcessTemplateBean
com.ibm.task.api.Escalation	com.ibm.task.clientmodel.bean.EscalationBean
com.ibm.task.api.Task	com.ibm.task.clientmodel.bean.TaskInstanceBean
com.ibm.task.api.TaskTemplate	com.ibm.task.clientmodel.bean.TaskTemplateBean

Adding the List component to a JSF application

Use the Business Process Choreographer Explorer List component to display a list of client model objects, for example, business process instances or task instances.

Procedure

1. Add the List component to the JavaServer Pages (JSP) file.

   Add the `bpe:list` tag to the `h:form` tag. The `bpe:list` tag must include a model attribute. Add `bpe:column` tags to the `bpe:list` tag to add the properties of the objects that are to appear in each of the rows in the list.

   The following example shows how to add a List component to display task instances.

   ```
   <h:form>

      <bpe:list model="#{TaskPool}">
         <bpe:column name="name" action="taskInstanceDetails" />
         <bpe:column name="state" />
         <bpe:column name="kind" />
         <bpe:column name="owner" />
         <bpe:column name="originator" />
      </bpe:list>

   </h:form>
   ```

   The model attribute refers to a managed bean, TaskPool. The managed bean provides the list of Java objects over which the list iterates and then displays in individual rows.

2. Configure the managed bean referred to in the `bpe:list` tag.

   For the List component, this managed bean must be an instance of the com.ibm.bpe.jsf.handler.BPCListHandler class.

   The following example shows how to add the TaskPool managed bean to the configuration file.

   ```
   <managed-bean>
   <managed-bean-name>TaskPool</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.jsf.handler.BPCListHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
      <managed-property>
         <property-name>query</property-name>
         <value>#{TaskPoolQuery}</value>
      </managed-property>
      <managed-property>
         <property-name>type</property-name>
         <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
      </managed-property>
   </managed-bean>

   <managed-bean>
   <managed-bean-name>TaskPoolQuery</managed-bean-name>
   <managed-bean-class>sample.TaskPoolQuery</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
      <managed-property>
        <property-name>type</property-name>
        <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
      </managed-property>
   </managed-bean>

   <managed-bean>
   <managed-bean-name>htmConnection</managed-bean-name>
   <managed-bean-class>com.ibm.task.clientmodel.HTMConnection</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
      <managed-property>
   ```

```
      <property-name>jndiName</property-name>
      <value>java:comp/env/ejb/LocalHumanTaskManagerEJB</value>
   </managed-property>
</managed-bean>
```

The example shows that TaskPool has two configurable properties: `query` and `type`. The value of the query property refers to another managed bean, TaskPoolQuery. The value of the type property specifies the bean class, the properties of which are shown in the columns of the displayed list. The associated query instance can also have a property type. If a property type is specified, it must be the same as the type specified for the list handler.

You can add any type of query logic to the JSF application as long as the result of the query can be represented as list of strongly-typed beans. For example, the TaskPoolQuery is implemented using a list of com.ibm.task.clientmodel.bean.TaskInstanceBean objects.

3. Add the custom code for the managed bean that is referred to by the list handler.

   The following example shows how to add custom code for the TaskPool managed bean.

```
public class TaskPoolQuery implements Query {

  public List execute throws ClientException {

    // Examine the faces-config file for a managed bean "htmConnection".
    //
    FacesContext ctx = FacesContext.getCurrentInstance();
    Application  app = ctx.getApplication();
    ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
    htmConnection = (HTMConnection) htmVb.getValue(ctx);
    HumanTaskManagerService taskService =
         htmConnection.getHumanTaskManagerService();

    // Then call the actual query method on the Human Task Manager service.
    //
    // Add the database columns for all of the properties you want to show
    // in your list to the select statement
    //
    QueryResultSet queryResult = taskService.query(
     "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND, TASK.STATE, TASK.TYPE,"
      + "TASK.STARTER, TASK.OWNER, TASK.STARTED, TASK.ACTIVATED, TASK.DUE,
         TASK.EXPIRES, TASK.PRIORITY",
      "TASK.KIND IN(101,102,105) AND TASK.STATE IN(2)
        AND WORK_ITEM.REASON IN (1)",
      (String)null,
      (Integer)null,
      (TimeZone)null);
    List applicationObjects = transformToTaskList ( queryResult );
    return applicationObjects ;
  }


  private List transformToTaskList(QueryResultSet result) {

ArrayList array = null;
int entries = result.size();
array = new ArrayList( entries );

// Transforms each row in the QueryResultSet to a task instance beans.
 for (int i = 0; i < entries; i++) {
   result.next();
   array.add( new TaskInstanceBean( result, connection ));
```

```
  }
  return array ;
   }
 }
```

The TaskPoolQuery bean queries the properties of the Java objects. This bean must implement the com.ibm.bpc.clientcore.Query interface. When the list handler refreshes its contents, it calls the execute method of the query. The call returns a list of Java objects. The getType method must return the class name of the returned Java objects.

Results

Your JSF application now contains a JavaServer page that displays the properties of the requested list of objects, for example, the state, kind, owner, and originator of the task instances that are available to you.

How lists are processed

Every instance of the List component is associated with an instance of the com.ibm.bpe.jsf.handler.BPCListHandler class.

This list handler tracks the selected items in the associated list and it provides a notification mechanism to associate the list entries with the details pages for the different kinds of items. The list handler is bound to the List component through the **model** attribute of the `bpe:list` tag.

The notification mechanism of the list handler is implemented using the com.ibm.bpe.jsf.handler.ItemListener interface. You can register implementations of this interface in the configuration file of your JavaServer Faces (JSF) application.

The notification is triggered when a link in the list is clicked. Links are rendered for all of the columns for which the **action** attribute is set. The value of the **action** attribute is either a JSF navigation target, or a JSF action method that returns a JSF navigation target.

The BPCListHandler class also provides a refreshList method. You can use this method in JSF method bindings to implement a user interface control for running the query again.

Query implementations

You can use the list handler to display all kinds of objects and their properties. The content of the list that is displayed depends on the list of objects that is returned by the implementation of the com.ibm.bpc.clientcore.Query interface that is configured for the list handler. You can set the query either programmatically using the setQuery method of the BPCListHandler class, or you can configure it in the JSF configuration files of the application.

You can run queries not only against the Business Process Choreographer APIs, but also against any other source of information that is accessible from your application, for example, a content management system or a database. The only requirement is that the result of the query is returned as a `java.util.List` of objects by the execute method.

The type of the objects returned must guarantee that the appropriate getter methods are available for all of the properties that are displayed in the columns of the list for which the query is defined. To ensure that the type of the object that is

returned fits the list definitions, you can set the value of the type property on the BPCListHandler instance that is defined in the faces configuration file to the fully qualified class name of the returned objects. You can return this name in the getType call of the query implementation. At runtime, the list handler checks that the object types conform to the definitions.

To map error messages to specific entries in a list, the objects returned by the query must implement a method with the signature `public Object getID()`.

Default converters and labels

The items returned by a query must be beans and their class must match the class specified as the type in the definition of the BPCListHandler class or com.ibm.bpc.clientcore.Query interface. In addition, the List component checks whether the item class or a superclass implements the following methods:

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

If these methods are defined for the beans, the List component uses the label as the default label for the list and the SimpleConverter as the default converter for the property. You can overwrite these settings with the **label** and **converterID** attributes of the `bpe:list` tag. For more information, see the Javadoc for the SimpleConverter interface and the ColumnTag class.

User-specific time zone information

The JavaServer Faces (JSF) components provide a utility for handling user-specific time zone information in the List component.

The BPCListHandler class uses the com.ibm.bpc.clientcore.util.User interface to get information about the time zone and locale of each user. The List component expects the implementation of the interface to be configured with **user** as the managed-bean name in your JavaServer Faces (JSF) configuration file. If this entry is missing from the configuration file, the time zone in which WebSphere Process Server is running is returned.

The com.ibm.bpc.clientcore.util.User interface is defined as follows:

```
public interface User {

    /**
     * The locale used by the client of the user.
     * @return Locale.
     */
    public Locale getLocale();
   /**
    * The time zone used by the client of the user.
    * @return TimeZone.
    */
    public TimeZone getTimeZone();

   /**
    * The name of the user.
    * @return name of the user.
    */
    public String getName();
}
```

Error handling in the List component

When you use the List component to display lists in your JSF application, you can take advantage of the error handling functions provided by the com.ibm.bpe.jsf.handler.BPCListHandler class.

Errors that occur when queries are run or commands are run

If an error occurs during the execution of a query, the BPCListHandler class distinguishes between errors that were caused by insufficient access rights and other exceptions. To catch errors due to insufficient access rights, the **rootCause** parameter of the ClientException that is thrown by the execute method of the query must be a com.ibm.bpe.api.EngineNotAuthorizedException or a com.ibm.task.api.NotAuthorizedException exception. The List component displays the error message instead of the result of the query.

If the error is not caused by insufficient access rights, the BPCListHandler class passes the exception object to the implementation of the com.ibm.bpc.clientcore.util.ErrorBean interface that is defined by the BPCError key in your JSF application configuration file. When the exception is set, the error navigation target is called.

Errors that occur when working with items that are displayed in a list

The BPCListHandler class implements the com.ibm.bpe.jsf.handler.ErrorHandler interface. You can provide information about these errors with the map parameter of type java.util.Map in the setErrors method. This map contains identifiers as keys and the exceptions as values. The identifiers must be the values returned by the getID method of the object that caused the error. If the map is set and any of the IDs match any of the items displayed in the list, the list handler automatically adds a column containing the error message to the list.

To avoid outdated error messages in the list, reset the errors map. In the following situations, the map is reset automatically:
- The refreshList method BPCListHandler class is called.
- A new query is set on the BPCListHandler class.
- The CommandBar component is used to trigger actions on items of the list. The CommandBar component uses this mechanism as one of the methods for error handling.

List component: Tag definitions

The Business Process Choreographer Explorer List component displays a list of objects in a table, for example, tasks, activities, process instances, process templates, work items, and escalations.

The List component consists of the JSF component tags: `bpe:list` and `bpe:column`. The `bpe:column` tag is a subelement of the `bpe:list` tag.

Component class

com.ibm.bpe.jsf.component.ListComponent

Example syntax

```
<bpe:list model="#{ProcessTemplateList}">
         rows="20"
         styleClass="list"
```

```
        headerStyleClass="listHeader"
        rowClasses="normal">

    <bpe:column name="name" action="processTemplateDetails"/>
    <bpe:column name="validFromTime"/>
    <bpe:column name="executionMode" label="Execution mode"/>
    <bpe:column name="state" converterID="my.state.converter"/>
    <bpe:column name="autoDelete"/>
    <bpe:column name="description"/>

</bpe:list>
```

Tag attributes

The body of the bpe:list tag can contain only bpe:column tags. When the table is rendered, the List component iterates over the list of application objects and renders all of the columns for each of the objects.

Table 52. bpe:list attributes

Attribute	Required	Description
buttonStyleClass	no	The cascading style sheet (CSS) style class for rendering the buttons in the footer area.
cellStyleClass	no	The CSS style class for rendering individual table cells.
checkbox	no	Determines whether the check box for selecting multiple items is rendered. The attribute has a value of either true or false. If the value is set to true, the check box column is rendered.
headerStyleClass	no	The CSS style class for rendering the table header.
model	yes	A value binding for a managed bean of the com.ibm.bpe.jsf.handler.BPCListHandler class.
rows	no	The number of rows that are shown on a page. If the number of items exceeds the number of rows, paging buttons are displayed at the end of the table. Value expressions are not supported for this attribute.
rowClasses	no	The CSS style class for rendering the rows in the table.
selectAll	no	If this attribute is set to true, all of the items in the list are selected by default.
styleClass	no	The CSS style class for rendering the overall table containing titles, rows, and paging buttons.

Table 53. bpe:column attributes

Attribute	Required	Description
action	no	If this attribute is specified, a link is rendered in the column. Either a JavaServer Faces action method or the Faces navigation target is triggered when this link is clicked. A JavaServer Faces action method has the following signature: String method().
converterID	no	The Faces converter ID that is used for converting the property value. If this attribute is not set, any Faces converter ID that is provided by the model for this property is used.
label	no	A literal or value binding expression that is used as a label for the header of the column or the cell of the table header row. If this attribute is not set, any label that is provided by the model for this property is used.
name	yes	The name of the property that is displayed in this column.

Adding the Details component to a JSF application

Use the Business Process Choreographer Explorer Details component to display the properties of tasks, work items, activities, process instances, and process templates.

Procedure

1. Add the Details component to the JavaServer Pages (JSP) file.

   Add the `bpe:details` tag to the `<h:form>` tag. The `bpe:details` tag must contain a **model** attribute. You can add properties to the Details component with the `bpe:property` tag.

   The following example shows how to add a Details component to display some of the properties for a task instance.

```
<h:form>

   <bpe:details model="#{TaskInstanceDetails}">
      <bpe:property name="displayName" />
      <bpe:property name="owner" />
      <bpe:property name="kind" />
      <bpe:property name="state" />
      <bpe:property name="escalated" />
      <bpe:property name="suspended" />
      <bpe:property name="originator" />
      <bpe:property name="activationTime" />
      <bpe:property name="expirationTime" />
   </bpe:details>

</h:form>
```

   The **model** attribute refers to a managed bean, TaskInstanceDetails. The bean provides the properties of the Java object.

2. Configure the managed bean referred to in the `bpe:details` tag.

   For the Details component, this managed bean must be an instance of the com.ibm.bpe.jsf.handler.BPCDetailsHandler class. This handler class wraps a Java object and exposes its public properties to the details component.

The following example shows how to add the TaskInstanceDetails managed bean to the configuration file.

```
<managed-bean>
   <managed-bean-name>TaskInstanceDetails</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.jsf.handler.BPCDetailsHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>
```

The example shows that the TaskInstanceDetails bean has a configurable `type` property. The value of the type property specifies the bean class (com.ibm.task.clientmodel.bean.TaskInstanceBean), the properties of which are shown in the rows of the displayed details. The bean class can be any JavaBeans class. If the bean provides default converter and property labels, the converter and the label are used for the rendering in the same way as for the List component.

Results

Your JSF application now contains a JavaServer page that displays the details of the specified object, for example, the details of a task instance.

Details component: Tag definitions

The Business Process Choreographer Explorer Details component displays the properties of tasks, work items, activities, process instances, and process templates.

The Details component consists of the JSF component tags: `bpe:details` and `bpe:property`. The `bpe:property` tag is a subelement of the `bpe:details` tag.

Component class

com.ibm.bpe.jsf.component.DetailsComponent

Example syntax

```
<bpe:details model="#{MyActivityDetails}">
    <bpe:property name="name"/>
    <bpe:property name="owner"/>
    <bpe:property name="activated"/>
</bpe:details>
```

```
<bpe:details model="#{MyActivityDetails}" style="style" styleClass="cssStyle">
            style="style"
            styleClass="cssStyle"
</bpe:details>
```

Tag attributes

Use `bpe:property` tags to specify both the subset of attributes that are shown and the order in which these attributes are shown. If the details tag does not contain any attribute tags, it renders all of the available attributes of the model object.

Table 54. bpe:details attributes

Attribute	Required	Description
columnClasses	no	A list of cascading style sheet style (CSS) style classes, separated by commas, for rendering columns.
id	no	The JavaServer Faces ID of the component.

Table 54. bpe:details attributes (continued)

Attribute	Required	Description
model	yes	A value binding for a managed bean of the com.ibm.bpe.jsf.handler.BPCDetailsHandler class.
rowClasses	no	A list of CSS style classes, separated by commas, for rendering rows.
styleClass	no	The CSS class that is used for rendering the HTML element.

Table 55. bpe:property attributes

Attribute	Required	Description
converterID	no	The ID used to register the converter in the JavaServer Faces (JSF) configuration file.
label	no	The label for the property. If this attribute is not set, a default label is provided by the client model class.
name	yes	The name of the property to be displayed. This name must correspond to a named property as defined in the corresponding client model class.

Adding the CommandBar component to a JSF application

Use the Business Process Choreographer Explorer CommandBar component to display a bar with buttons. These buttons represent commands that operate on the details view of an object or the selected objects in a list.

About this task

When the user clicks a button in the user interface, the corresponding command is run on the selected objects. You can add and extend the CommandBar component in your JavaServer Faces (JSF) application.

Procedure

1. Add the CommandBar component to the JavaServer Pages (JSP) file.

   Add the `bpe:commandbar` tag to the `<h:form>` tag. The `bpe:commandbar` tag must contain a model attribute.

   The following example shows how to add a CommandBar component that provides refresh and claim commands for a task instance list.

```
<h:form>

   <bpe:commandbar model="#{TaskInstanceList}">
      <bpe:command commandID="Refresh" >
                   action="#{TaskInstanceList.refreshList}"
                   label="Refresh"/>

      <bpe:command commandID="MyClaimCommand" >
                   label="Claim" >
                   commandClass="<customcode>"/>
   </bpe:commandbar>

</h:form>
```

The **model** attribute refers to a managed bean. This bean must implement the ItemProvider interface and provide the selected Java objects. The CommandBar component is usually used with either the List component or the Details component in the same JSP file. Generally, the model that is specified in the tag is the same as the model that is specified in the List component or Details component on the same page. So for the List component, for example, the command acts on the selected items in the list.

In this example, the **model** attribute refers to the TaskInstanceList managed bean. This bean provides the selected objects in the task instance list. The bean must implement the ItemProvider interface. This interface is implemented by the BPCListHandler class and the BPCDetailsHandler class.

2. Optional: Configure the managed bean that is referred to in the `bpe:commandbar` tag.

   If the CommandBar **model** attribute refers to a managed bean that is already configured, for example, for a list or details handler, no further configuration is required. If you use neither the BPCListHandler class nor the BPCDetailsHandler class for the model, you must refer to another object that has a class that implements the ItemProvider interface.

3. Add the code that implements the custom commands to the JSF application.

   The following code snippet shows how to write a command class that implements the Command interface. This command class (MyClaimCommand) is referred to by the `bpe:command` tag in the JSP file.

```
public class MyClaimCommand implements Command {

  public String execute(List selectedObjects) throws ClientException {
     if( selectedObjects != null && selectedObjects.size() > 0 ) {
        try {
         // Determine HumanTaskManagerService from an HTMConnection bean.
         // Configure the bean in the faces-config.xml for easy access
         // in the JSF application.
         FacesContext ctx = FacesContext.getCurrentInstance();
         ValueBinding vb =
           ctx.getApplication().createValueBinding("{htmConnection}");
         HTMConnection htmConnection = (HTMConnection) htmVB.getValue(ctx);
         HumanTaskManagerService htm =
            htmConnection.getHumanTaskManagerService();

         Iterator iter = selectedObjects.iterator() ;
         while( iter.hasNext() ) {
           try {
                TaskInstanceBean task = (TaskInstanceBean) iter.next() ;
                TKIID tiid = task.getID() ;

                htm.claim( tiid ) ;
                task.setState( new Integer(TaskInstanceBean.STATE_CLAIMED ) ) ;

           }
           catch( Exception e ) {
             ;   // Error while iterating or claiming task instance.
                  // Ignore for better understanding of the sample.
           }
         }
       }
       catch( Exception e ) {
         ;  // Configuration or communication error.
           // Ignore for better understanding of the sample
       }
     }
     return null;
  }
```

```
  // Default implementations
  public boolean isMultiSelectEnabled() { return false; }
  public boolean[] isApplicable(List itemsOnList) {return null; }
  public void setContext(Object targetModel) {; // Not used here }
}
```

The command is processed in the following way:

a. A command is invoked when a user clicks the corresponding button in the command bar. The CommandBar component retrieves the selected items from the item provider that is specified in the **model** attribute and passes the list of selected objects to the execute method of the commandClass instance.

b. Optional: The **commandClass** attribute refers to a custom command implementation that implements the Command interface. This means that the command must implement the public String execute(List selectedObjects) throws ClientException method. The command returns a result that is used to determine the next navigation rule for the JSF application.

c. Optional: After the command completes, the CommandBar component evaluates the **action** attribute. The **action** attribute can be a static string or a method binding to a JSF action method with the `public String Method()` signature. Use the **action** attribute to override the outcome of a command class or to explicitly specify an outcome for the navigation rules. The **action** attribute is not processed if the command generates an exception other than an ErrorsInCommandException exception.

d. If the **commandClass** attribute does not have a command class specified, the action is immediately called. For example, for the refresh command in the example, the JSF value expression `#{TaskInstanceList.refreshList}` is called instead of a command.

Results

Your JSF application now contains a JavaServer page that implements a customized command bar.

How commands are processed

Use the CommandBar component to add action buttons to your application. The component creates the buttons for the actions in the user interface and handles the events that are created when a button is clicked.

These buttons trigger functions that act on the objects that are returned by a com.ibm.bpe.jsf.handler.ItemProvider interface, such as the BPCListHandler class, or the BPCDetailsHandler class. The CommandBar component uses the item provider that is defined by the value of the **model** attribute in the `bpe:commandbar` tag.

When a button in the command-bar section of the application's user interface is clicked, the associated event is handled by the CommandBar component in the following way.

1. The CommandBar component identifies the implementation of the com.ibm.bpc.clientcore.Command interface that is specified for the button that generated the event.
2. If the model associated with the CommandBar component implements the com.ibm.bpe.jsf.handler.ErrorHandler interface, the clearErrorMap method is invoked to remove error messages from previous events.

3. The getSelectedItems method of the ItemProvider interface is called. The list of items that is returned is passed to the execute method of the command, and the command is invoked.
4. The CommandBar component determines the JavaServer Faces (JSF) navigation target. If an **action** attribute is not specified in the `bpe:commandbar` tag, the return value of the execute method specifies the navigation target. If the **action** attribute is set to a JSF method binding, the string returned by the method is interpreted as the navigation target. The **action** attribute can also specify an explicit navigation target.

CommandBar component: Tag definitions

The Business Process Choreographer Explorer CommandBar component displays a bar with buttons. These buttons operate on the object in a details view or the selected objects in a list.

The CommandBar component consists of the JSF component tags: `bpe:commandbar` and `bpe:command`. The `bpe:command` tag is a subelement of the `bpe:commandbar` tag.

Component class

com.ibm.bpe.jsf.component.CommandBarComponent

Example syntax

```
<bpe:commandbar model="#{TaskInstanceList}">

    <bpe:command
       commandID="Work on"
       label="Work on..."
       commandClass="com.ibm.bpc.explorer.command.WorkOnTaskCommand"
       context="#{TaskInstanceDetailsBean}"/>

    <bpe:command
       commandID="Cancel"
       label="Cancel"
       commandClass="com.ibm.task.clientmodel.command.CancelClaimTaskCommand"
       context="#{TaskInstanceList}"/>

</bpe:commandbar>
```

Tag attributes

Table 56. bpe:commandbar attributes

Attribute	Required	Description
buttonStyleClass	no	The cascading style sheet (CSS) style class that is used for rendering the buttons in the command bar.
id	no	The JavaServer Faces ID of the component.
model	yes	A value binding expression to a managed bean that implements the ItemProvider interface. This managed bean is usually the com.ibm.bpe.jsf.handler.BPCListHandler class or the com.ibm.bpe.jsf.handler.BPCDetailsHandler class that is used by the List component or Details component in the same JavaServer Pages (JSP) file as the CommandBar component.

Table 56. bpe:commandbar attributes (continued)

Attribute	Required	Description
styleClass	no	The CSS style class that is used for rendering the command bar.

Table 57. bpe:command attributes

Attribute	Required	Description
action	no	A JavaServer Faces action method or the Faces navigation target that is to be triggered by the command button. The navigation target that is returned by the action overwrites all other navigation rules. The action is called when either an exception is not thrown or an ErrorsInCommandException exception is thrown by the command.
commandClass	no	The name of the command class. An instance of the class is created by the CommandBar component and run if the command button is selected.
commandID	yes	The ID of the command.
context	no	An object that provides context for commands that are specified using the **commandClass** attribute. The context object is retrieved when the command bar is first accessed.
immediate	no	Specifies when the command is triggered. If the value of this attribute is `true`, the command is triggered before the input of the page is processed. The default is `false`.
label	yes	The label of the button that is rendered in the command bar.
rendered	no	Determines whether a button is rendered. The value of the attribute can be either a Boolean value or a value expression.
styleClass	no	The CSS style class that is used for rendering the button. This style overrides the button style defined for the command bar.

Adding the Message component to a JSF application

Use the Business Process Choreographer Explorer Message component to render data objects and primitive types in a JavaServer Faces (JSF) application.

About this task

If the message type is a primitive type, a label and an input field are rendered. If the message type is a data object, the component traverses the object and renders the elements within the object.

Procedure

1. Add the Message component to the JavaServer Pages (JSP) file.

   Add the `bpe:form` tag to the `<h:form>` tag. The `bpe:form` tag must include a model attribute.

The following example shows how to add a Message component.

```
<h:form>

   <h:outputText value="Input Message" />
   <bpe:form model="#{MyHandler.inputMessage}" readOnly="true" />

   <h:outputText value="Output Message" />
   <bpe:form model="#{MyHandler.outputMessage}" />

</h:form>
```

The **model** attribute of the Message component refers to a com.ibm.bpc.clientcore.MessageWrapper object. This wrapper object wraps either a Service Data Object (SDO) object or a Java primitive type, for example, `int` or `boolean`. In the example, the message is provided by a property of the MyHandler managed bean.

2. Configure the managed bean referred to in the `bpe:form` tag.

   The following example shows how to add the MyHandler managed bean to the configuration file.

```
<managed-bean>
<managed-bean-name>MyHandler</managed-bean-name>
<managed-bean-class>com.ibm.bpe.sample.jsf.MyHandler</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>

   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>

</managed-bean>
```

3. Add the custom code to the JSF application.

   The following example shows how to implement input and output messages.

```
public class MyHandler implements ItemListener {

  private TaskInstanceBean taskBean;
  private MessageWrapper inputMessage, outputMessage

  /* Listener method, e.g. when a task instance was selected in a list handler.
   * Ensure that the handler is registered in the faces-config.xml or manually.
   */
  public void itemChanged(Object item) {
    if( item instanceof TaskInstanceBean ) {
        taskBean = (TaskInstanceBean) item ;
    }
  }

  /* Get the input message wrapper
   */
  public MessageWrapper getInputMessage() {
    try{
        inputMessage = taskBean.getInputMessageWrapper() ;
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return inputMessage;
  }

  /* Get the output message wrapper
   */
  public MessageWrapper getOutputMessage() {
    // Retrieve the message from the bean. If there is no message, create
    // one if the task has been claimed by the user. Ensure that only
```

```
      // potential owners or owners can manipulate the output message.
      try{
          outputMessage = taskBean.getOutputMessageWrapper();
          if( outputMessage == null
           && taskBean.getState() == TaskInstanceBean.STATE_CLAIMED ) {
              HumanTaskManagerService htm = getHumanTaskManagerService();
              outputMessage = new MessageWrapperImpl();
              outputMessage.setMessage(
                 htm.createOutputMessage( taskBean.getID() ).getObject()
              );
           }
      }
      catch( Exception e ) {
          ;       //...ignore errors for simplicity
      }
      return outputMessage
    }
  }
```

The MyHandler managed bean implements the com.ibm.jsf.handler.ItemListener interface so that it can register itself as an item listener to list handlers. When the user clicks an item in the list, the MyHandler bean is notified in its itemChanged(Object item) method about the selected item. The handler checks the item type and then stores a reference to the associated TaskInstanceBean object. To use this interface, add an entry to the itemListener list in the appropriate list handler in the faces-config.xml file.

The MyHandler bean provides the getInputMessage and getOutputMessage methods. Both of these methods return a MessageWrapper object. The methods delegate the calls to the referenced task instance bean. If the task instance bean returns null, for example, because a message is not set, the handler creates and stores a new, empty message. The Message component displays the messages provided by the MyHandler bean.

Results

Your JSF application now contains a JavaServer page that can render data objects and primitive types.

Message component: Tag definitions

The Business Process Choreographer Explorer Message component renders `commonj.sdo.DataObject` objects and primitive types, such as integers and strings, in a JavaServer Faces (JSF) application.

The Message component consists of the JSF component tag: `bpe:form`.

Component class

com.ibm.bpe.jsf.component.MessageComponent

Example syntax

```
<bpe:form model="#{TaskInstanceDetailsBean.inputMessageWrapper}"
          simplification="true" readOnly="true"
          styleClass4table="messageData"
          styleClass4output="messageDataOutput">
</bpe:form>
```

Tag attributes

Table 58. bpe:form attributes

Attribute	Required	Description
id	no	The JavaServer Faces ID of the component.
model	yes	A value binding expression that refers to either a `commonj.sdo.DataObject` object or a `com.ibm.bpc.clientcore.MessageWrapper` object.
readOnly	no	If this attribute is set to `true`, a read-only form is rendered. By default, this attribute is set to `false`.
simplification	no	If this attribute is set to `true`, properties that contain simple types and have a cardinality of zero or one are shown. By default, this attribute is set to `true`.
style4validinput	no	The cascading style sheet (CSS) style for rendering input that is valid.
style4invalidinput	no	The CSS style for rendering input that is not valid.
styleClass4invalidInput	no	The CSS style class name for rendering input that is not valid.
styleClass4output	no	The CSS style class name for rendering the output elements.
styleClass4table	no	The class name of the CSS table style for rendering the tables rendered by the message component.
styleClass4validInput	no	The CSS style class name for rendering input that is valid.

Developing JSP pages for task and process messages

The Business Process Choreographer Explorer interface provides default input and output forms for displaying and entering business data. You can use JSP pages to provide customized input and output forms.

About this task

To include user-defined JavaServer Pages (JSP) pages in the Web client, you must specify them when you model a human task in WebSphere Integration Developer. For example, you can provide JSP pages for a specific task and its input and output messages, and for a specific user role or all user roles. At runtime, the user-defined JSP pages are included in the user interface to display output data and collect input data.

The customized forms are not self-contained Web pages; they are HTML fragments that Business Process Choreographer Explorer imbeds in an HTML form, for example, fragments for all of the labels and input fields of a message.

When a button is clicked on the page that contains the customized forms, the input is submitted and validated in Business Process Choreographer Explorer. The validation is based on the type of the properties provided and the locale used in the browser. If the input cannot be validated, the same page is shown again and

information about the validation errors is provided in the messageValidationErrors request attribute. The information is provided as a map that maps the XML Path Expression (XPath) of the properties that are not valid to the validation exceptions that occurred.

To add customized forms to Business Process Choreographer Explorer, complete the following steps using WebSphere Integration Developer.

Procedure

1. Create the customized forms.

   The user-defined JSP pages for the input and output forms used in the Web interface need access to the message data. Use Java snippets in a JSP or the JSP execution language to access the message data. Data in the forms is available through the request context.
2. Assign the JSP pages to a task.

   Open the human task in the human task editor. In the client settings, specify the location of the user-defined JSP pages and the role to which the customized form applies, for example, administrator. The client settings for Business Process Choreographer Explorer are stored in the task template. At runtime these settings are retrieved with the task template.
3. Package the user-defined JSP pages in a Web archive (WAR file).

   You can either include the WAR file in the enterprise archive with the module that contains the tasks or deploy the WAR file separately. If the JSPs are deployed separately, make the JSPs available on the server where the Business Process Choreographer Explorer or the custom client is deployed.

   If you are using custom JSPs for the process and task messages, you must map the Web modules that are used to deploy the JSPs to the same servers that the custom JSF client is mapped to.

Results

The customized forms are rendered in Business Process Choreographer Explorer at runtime.

User-defined JSP fragments

The user-defined JavaServer Pages (JSP) fragments are imbedded in an HTML form tag. At runtime, Business Process Choreographer Explorer includes these fragments in the rendered page.

The user-defined JSP fragment for the input message is imbedded before the JSP fragment for the output message.

```
<html....>
   ...
   <form...>
     Input JSP (display task input message)

     Output JSP (display task output message)

   </form>
   ...
</html>
```

Because the user-defined JSP fragments are embedded in an HTML form tag, you can add input elements. The name of the input element must match the XML Path

Language (XPath) expression of the data element. It is important to prefix the name of the input element with the provided prefix value:

```
<input id="address"
       type="text"
       name="${prefix}/selectPromotionalGiftResponse/address"
       value="${messageMap['/selectPromotionalGiftResponse/address"]}
       size="60"
       align="left" />
```

The prefix value is provided as a request attribute. The attribute ensures that the input name is unique in the enclosing form. The prefix is generated by Business Process Choreographer Explorer and it should not be changed:

```
String prefix = (String)request.getAttribute("prefix");
```

The prefix element is set only if the message can be edited in the given context. Output data can be displayed in different ways depending on the state of the human task. For example, if the task is in the claimed state, the output data can be modified. However, if the task is in the finished state, the data can be displayed only. In your JSP fragment, you can test whether the prefix element exists and render the message accordingly. The following JSTL statement shows how you might test whether the prefix element is set.

```
...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<c:choose>
  <c:when test="${not empty prefix}">
    <!--Read/write mode-->
  </c:when>
  <c:otherwise>
    <!--Read-only mode-->
  </c:otherwise>
</c:choose>
```

Creating plug-ins to customize human task functionality

Business Process Choreographer provides an event handling infrastructure for events that occur during the processing of human tasks. Plug-in points are also provided so that you can adapt the functionality to your needs. You can use the service provider interfaces (SPIs) to create customized plug-ins for handling events and the post processing of people query results.

About this task

You can create plug-ins for human task API events and escalation notification events. You can also create a plug-in that processes the results that are returned from people resolution. For example, at peak periods you might want to add users to the result list to help balance the workload.

Before you can use the plug-ins, you must install and register them. You can register the plug-in to post process people query results with the TaskContainer application. The plug-in is then available for all tasks.

Creating API event handlers

An API event occurs when an API method manipulates a human task. Use the API event handler plug-in service provider interface (SPI) to create plug-ins to handle the task events sent by the API or the internal events that have equivalent API events.

About this task

Complete the following steps to create an API event handler.

Procedure

1. Write a class that implements the APIEventHandlerPlugin3 interface or extends the APIEventHandler implementation class. This class can invoke the methods of other classes.
   - If you use the APIEventHandlerPlugin3 interface, you must implement all of the methods of the APIEventHandlerPlugin3 interface and the APIEventHandlerPlugin interface.
   - If you extend the APIEventHandler implementation class, overwrite the methods that you need.

   This class runs in the context of a Java 2 Enterprise Edition (J2EE) Enterprise application. Ensure that this class and its helper classes follow the EJB specification.

   Note: If you want to call the HumanTaskManagerService interface from this class, do not call a method that updates the task that produced the event. This action might result in inconsistent task data in the database.
2. Assemble the plug-in class and its helper classes into a JAR file.

   You can make the JAR file available in one of the following ways:
   - As a utility JAR file in the application EAR file.
   - As a shared library that is installed with the application EAR file.
   - As a shared library that is installed with the TaskContainer application. In this case, the plug-in is available for all tasks.
3. Create a service provider configuration file for the plug-in in the META-INF/services/ directory of your JAR file.

   The configuration file provides the mechanism for identifying and loading the plug-in. This file conforms to the Java 2 service provider interface specification.
   a. Create a file with the name `com.ibm.task.spi.`*`plug-in_name`*`APIEventHandlerPlugin`, where *plug-in_name* is the name of the plug-in.

      For example, if your plug-in is called `Customer` and it implements the com.ibm.task.spi.APIEventHandlerPlugin3 interface, the name of the configuration file is `com.ibm.task.spi.`**`Customer`**`APIEventHandlerPlugin`.
   b. In the first line of the file that is neither a comment line (a line that starts with a number sign (#)) nor a blank line, specify the fully qualified name of the plug-in class that you created in step 1.

      For example, if your plug-in class is called `MyAPIEventHandler` and it is in the `com.customer.plugins` package, then the first line of the configuration file must contain the following entry: `com.customer.plugins.MyAPIEventHandler`.

Results

You have an installable JAR file that contains a plug-in that handles API events and a service provider configuration file that can be used to load the plug-in.

Notes: You only have one eventHandlerName property available to register both API event handlers and notification event handlers. If you want to use both an API

event handler and a notification event handler, the plug-in implementations must have the same name, for example, `Customer` as the event handler name for the SPI implementation.

You can implement both plug-ins using a single class, or two separate classes. In both cases, you need to create two files in the META-INF/services/ directory of your JAR file, for example, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin and com.ibm.task.spi.CustomerAPIEventHandlerPlugin.

Package the plug-in implementation and the helper classes in a single JAR file.

To make a change to an implementation effective, replace the JAR file in the shared library, deploy the associated EAR file again, and restart the server.

What to do next

You now need to install and register the plug-in so that it is available to the human task container at runtime. You can register API event handlers with a task instance, a task template, or an application component.

API event handlers

API events occur when a human task is modified or it changes state. To handle these API events, the event handler is invoked directly before the task is modified (pre-event method) and just before the API call returns (post-event method).

If the pre-event method throws an ApplicationVetoException exception, the API action is not performed, the exception is returned to the API caller, and the transaction associated with the event is rolled back. If the pre-event method was triggered by an internal event and an ApplicationVetoException exception is thrown, the internal event, such as an automatic claim, is not performed but an exception is not returned to the client application. In this case, an information message is written to the SystemOut.log file. If the API method throws an exception during processing, the exception is caught and passed to the post-event method. The exception is passed again to the caller after the post-event method returns.

The following rules apply to pre-event methods:

- Pre-event methods receive the parameters of the associated API method or internal event.
- Pre-event methods can throw an ApplicationVetoException exception to prevent processing from continuing.

The following rules apply to post-event methods:

- Post-event methods receive the parameters that were supplied to the API call, and the return value. If an exception is thrown by the API method implementation, the post-event method also receives the exception.
- Post-event methods cannot modify return values.
- Post-event methods cannot throw exceptions; runtime exceptions are logged but they are ignored.

To implement API event handlers, you can implement either the APIEventHandlerPlugin3 interface, which extends the APIEventHandlerPlugin interface, or extend the default com.ibm.task.spi.APIEventHandler SPI implementation class. If your event handler inherits from the default

implementation class, it always implements the most recent version of the SPI. If you upgrade to a newer version of Business Process Choreographer, fewer changes are necessary if you want to exploit new SPI methods.

If you have both a notification event handler and an API event handler, both of these handlers must have the same name because you can register only one event handler name.

Creating notification event handlers

Notification events are produced when human tasks are escalated. Business Process Choreographer provides functionality for handling escalations, such as creating escalation work items or sending e-mails. You can create notification event handlers to customize the way in which escalations are handled.

About this task

To implement notification event handlers, you can implement the NotificationEventHandlerPlugin interface, or you can extend the default com.ibm.task.spi.NotificationEventHandler service provider interface (SPI) implementation class.

Complete the following steps to create a notification event handler.

Procedure

1. Write a class that implements the NotificationEventHandlerPlugin interface or extends the NotificationEventHandler implementation class. This class can invoke the methods of other classes.

   If you use the NotificationEventHandlerPlugin interface, you must implement all of the interface methods. If you extend the SPI implementation class, overwrite the methods that you need.

   This class runs in the context of a Java 2 Enterprise Edition (J2EE) Enterprise application. Ensure that this class and its helper classes follow the EJB specification.

   The plug-in is invoked with the authority of the EscalationUser role. This role is defined when the human task container is configured.

   Note: If you want to call the HumanTaskManagerService interface from this class, do not call a method that updates the task that produced the event. This action might result in inconsistent task data in the database.
2. Assemble the plug-in class and its helper classes into a JAR file.

   You can make the JAR file available in one of the following ways:
   - As a utility JAR file in the application EAR file.
   - As a shared library that is installed with the application EAR file.
   - As a shared library that is installed with the TaskContainer application. In this case, the plug-in is available for all tasks.
3. Assemble the plug-in class and its helper classes into a JAR file.

   If the helper classes are used by several J2EE applications, you can package these classes in a separate JAR file that you register as a shared library.
4. Create a service provider configuration file for the plug-in in the META-INF/services/ directory of your JAR file.

   The configuration file provides the mechanism for identifying and loading the plug-in. This file conforms to the Java 2 service provider interface specification.

a. Create a file with the name `com.ibm.task.spi.`*`plug-in_name`*`NotificationEventHandlerPlugin`, where *plug-in_name* is the name of the plug-in.

   For example, if your plug-in is called `HelpDeskRequest` (event handler name) and it implements the com.ibm.task.spi.NotificationEventHandlerPlugin interface, the name of the configuration file is `com.ibm.task.spi.`**`HelpDeskRequest`**`NotificationEventHandlerPlugin`.

b. In the first line of the file that is neither a comment line (a line that starts with a number sign (#)) nor a blank line, specify the fully qualified name of the plug-in class that you created in step 1.

   For example, if your plug-in class is called `MyEventHandler` and it is in the `com.customer.plugins` package, then the first line of the configuration file must contain the following entry: `com.customer.plugins.MyEventHandler`.

Results

You have an installable JAR file that contains a plug-in that handles notification events and a service provider configuration file that can be used to load the plug-in. You can register API event handlers with a task instance, a task template, or an application component.

Notes: You only have one eventHandlerName property available to register both API event handlers and notification event handlers. If you want to use both an API event handler and a notification event handler, the plug-in implementations must have the same name, for example, `Customer` as the event handler name for the SPI implementation.

You can implement both plug-ins using a single class, or two separate classes. In both cases, you need to create two files in the META-INF/services/ directory of your JAR file, for example, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin and com.ibm.task.spi.CustomerAPIEventHandlerPlugin.

Package the plug-in implementation and the helper classes in a single JAR file.

To make a change to an implementation effective, replace the JAR file in the shared library, deploy the associated EAR file again, and restart the server.

What to do next

You now need to install and register the plug-in so that it is available to the human task container at runtime. You can register notification event handlers with a task instance, a task template, or an application component.

Installing API event handler and notification event handler plug-ins

To use API event handler or notification event handler plug-ins, you must install the plug-in so that it can be accessed by the task container.

About this task

The way in which you install the plug-in depends on whether the plug-in is to be used by only one Java 2 Enterprise Edition (J2EE) application, or several applications.

Complete one of the following steps to install a plug-in.

- Install a plug-in for use by a single J2EE application.

  Add your plug-in JAR file to the application EAR file. In the deployment descriptor editor in WebSphere Integration Developer, install the JAR file for your plug-in as a project utility JAR file for the J2EE application of the main enterprise JavaBeans (EJB) module.
- Install a plug-in for use by several J2EE applications.

  Put the JAR file in a WebSphere Application Server shared library and associate the library with the applications that need access to the plug-in. To make the JAR file available in a network deployment environment, manually distribute the JAR file on each node that hosts a server or cluster member on which any of your applications is deployed. You can use the deployment target scope of your applications, that is the server or cluster on which the applications are deployed, or the cell scope. Be aware that the plug-in classes are then visible throughout the selected deployment scope.

What to do next

You can now register the plug-in.

Registering API event handler and notification event handler plug-ins with task templates, task models, and tasks

You can register plug-ins for API event handlers and notification event handlers with tasks, task templates, and task models at various times: when you create an ad-hoc task, update an existing task, create an ad-hoc task model, or define a task template.

About this task

You can register plug-ins for API event handlers and notification event handlers with tasks on the following levels:

Task template
All of the tasks that are created using the template use the same handlers

Ad-hoc task model
The tasks that are created using the model use the same handlers

Ad-hoc task
The task that is created uses the specified handlers

Existing task
The task uses the specified handlers

You can register a plug-in in one of the following ways.

- For task templates modeled in WebSphere Integration Developer, specify the plug-in in the task model.
- For ad-hoc tasks or ad-hoc task models, specify the plug-in when you create the task or task model.

  Use the setEventHandlerName method of the TTask class to register the name of the event handler.
- Change the event handler for a task instance at runtime.

  Use the update(Task task) method to use a different event handler for a task instance at runtime. The caller must have task administrator authority to update this property.

Creating, installing, and running plug-ins to post-process people query results

People resolution returns a list of the users that are assigned to a specific role, for example, potential owner of a task. You can create a plug-in to change the results of people queries returned by people resolution. For example, to improve workload balancing, you might have a plug-in that removes users from the query result who already have a high workload.

About this task

You can have only one post-processing plug-in; this means that the plug-in must handle the people query results from all tasks. Your plug-in can add or remove users, or change user or group information. It can also change the result type, for example, from a list of users to a group, or to everybody.

Because the plug-in runs after people resolution completes, any rules that you have to preserve confidentiality or security have already been applied. The plug-in receives information about users that have been removed during people resolution (in the HTM_REMOVED_USERS map key). You must ensure that your plug-in uses this context information to preserve any confidentiality or security rules you might have.

To implement post-processing of people query results, you use the StaffQueryResultPostProcessorPlugin interface. The interface has methods for modifying the query results for tasks, escalations, task templates, and application components.

Complete the following steps to create a plug-in to post-process people query results.

Procedure

1. Write a class that implements the StaffQueryResultPostProcessorPlugin interface.

   This class runs in the context of a Java 2 Enterprise Edition (J2EE) Enterprise application. This class can invoke methods of other classes. Ensure that this class and its helper classes follow the EJB specification.

   Note: If you want to call the HumanTaskManagerService interface from this class, do not call a method that updates the task that produced the event. This action might result in inconsistent task data in the database.

   You must implement all of the methods in the interface. These methods include information relating to the people assignment criteria for the specific task template, task or escalation role.

   - The people assignment criteria definition is specified as an entry in the **context** parameter of type Map. To access this information proceed as follows:

     ```
     Map pacAsMap = (Map) context.get("HTM_VERB");

     // to retrieve the name of the PAC
     String pacName = (String) pacAsMap.get("HTM_VERB_NAME");

     // to retrieve the PAC parameter names
     Set paramNames = pacAsMap.keySet();

     // to retrieve the value of a specific parameter
     String paramValue = (String) pacAsMap.get(paramName);
     ```

- The replacement variables specified as people assignment criteria parameter values are entries of the **context** parameter of type `Map`. To access this information proceed as follows:

```
Object replVarObj = pacAsMap.get(replVarName);
if (replVarObj instanceof String)
     String replVarValue = (String) replVarObj;
if (replVarObj instanceof String[])
     String[] replVarValues = (String[]) replVarObj;
```

- The StaffQueryResult object that is created by accessing a people directory during people resolution, for example, by accessing the virtual member manager people directory.

  The StaffQueryResult object contains the information about the user entries that are retrieved during people resolution. For more information, see the Javadoc reference information for the StaffQueryResultPostProcessorPlugin interface.
- The list of users that have been explicitly excluded by people resolution is contained as an entry of the **context** parameter of type `Map`. To access this information proceed as follows:

```
String[] removedUserIDs = (String[]) context.get("HTM_REMOVED_USERS");
```

The following example shows how you might change the editor role of a task called SpecialTask.

```
public StaffQueryResult processStaffQueryResult
                         (StaffQueryResult originalStaffQueryResult,
                                           Task task,
                                           int  role,
                                           Map  context)
{
  StaffQueryResult newStaffQueryResult = originalStaffQueryResult;
  StaffQueryResultFactory staffResultFactory =
                          StaffQueryResultFactory.newInstance();
  if (role == com.ibm.task.api.WorkItem.REASON_EDITOR &&
      task.getName() != null &&
      task.getName().equals("SpecialTask"))
  {
      UserData user = staffResultFactory.newUserData
                           ("SuperEditor",
                             new Locale("en-US"),
                            "SuperEditor@company.com");
      ArrayList userList = new ArrayList();
      userList.add(user);

      newStaffQueryResult = staffResultFactory.newStaffQueryResult(userList);
   }
   return(newStaffQueryResult);
}
```

2. Assemble the plug-in class and its helper classes into a JAR file.

   You can make the JAR available as a shared library, and associate it with the task container. In this way, your plug-in is available for all tasks.
3. Create a service provider configuration file for the plug-in in the META-INF/services/ directory of your JAR file.

   The configuration file provides the mechanism for identifying and loading the plug-in. This file conforms to the Java 2 service provider interface specification.
   a. Create a file with the name `com.ibm.task.spi.`*`plug-in_name`*`StaffQueryResultPostProcessorPlugin`, where *plug-in_name* is the name of the plug-in.

      For example, if your plug-in is called `MyHandler` and it implements the com.ibm.task.spi.StaffQueryResultPostProcessorPlugin interface, the name of

the configuration file is
`com.ibm.task.spi.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`.

b. In the first line of the file that is neither a comment line (a line that starts with a number sign (#)) nor a blank line, specify the fully qualified name of the plug-in class that you created in step 1.

   For example, if your plug-in class is called `StaffPostProcessor` and it is in the `com.customer.plugins` package, then the first line of the configuration file must contain the following entry: `com.customer.plugins.StaffPostProcessor`. You have an installable JAR file that contains a plug-in that post processes people query results and a service provider configuration file that can be used to load the plug-in.

4. Install the plug-in.

   You can have only one post-processing plug-in for people query results. You must install the plug-in as a shared library.

   a. Define a WebSphere Application Server shared library for the plug-in. Define the shared library on the scope of the server or cluster where Business Process Choreographer is configured. Then associate this shared library with the TaskContainer application. This step needs to be done only once.
   b. Make the plug-in JAR file available to each affected WebSphere Process Server installation that hosts a server or a cluster member.

5. Register the plug-in.

   a. In the administrative console, go to the Custom Properties page of the Human Task Manager

      Click **Servers** → **Application servers** → *server_name* in a stand-alone environment, or **Servers** → **Clusters** → *cluster_name* if Business Process Choreographer is configured in a cluster. Under **Business Integration**, select **Human Task Manager**. Under **Additional Properties**, select **Custom Properties**.
   b. Add a custom property with the name **Staff.PostProcessorPlugin**, and a value of the name that you gave to your plug-in, `MyHandler` in this example.

   The plug-in is now available for post processing people query results. If you change the JAR file, replace the file in the shared library, and restart the server.

6. Run the plug-in. The post processing plug-in is invoked after both the people assignment and people substitution have run. The plug-in is invoked with the information that is specified by the StaffQueryResultPostProcessorPlugin interface.

Part 2. Deploying applications

© Copyright IBM Corp. 2005, 2009

Chapter 5. Overview of preparing and installing modules

Installing modules (also known as deploying) activates the modules in either a test environment or a production environment. This overview briefly describes the test and production environments and some of the steps involved in installing modules.

Note: The process for installing applications in a production environment is similar to the process described in "Developing and deploying applications" in the WebSphere Application Server Network Deployment, version 6 information center. If you are unfamiliar with those topics, review those first.

Before installing a module to a production environment, always verify changes in a test environment. To install modules to a test environment, use WebSphere Integration Developer (see the WebSphere Integration Developer information center for more information). To install modules to a production environment, use WebSphere Process Server.

This topic describes the concepts and tasks needed to prepare and install modules to a production environment. Other topics describe the files that house the objects that your module uses and help you move your module from your test environment into your production environment. It is important to understand these files and what they contain so you can be sure that you have correctly installed your modules.

Libraries and JAR files overview

Modules often use artifacts that are located in libraries. Artifacts and libraries are contained in Java archive (JAR) files that you identify when you deploy a module.

While developing a module, you might identify certain resources or components that could be used by various pieces of the module. These resources or components could be objects that you created while developing the module or already existing objects that reside in a library that is already deployed on the server. This topic describes the libraries and files that you will need when you install an application.

What is a library?

A library contains objects or resources used by multiple modules within WebSphere Integration Developer. The artifacts can be in JAR, resource archive (RAR), or Web service archive (WAR) files. Some of these artifacts include:

- Interfaces or Web services descriptors (files with a .wsdl extension)
- Business object XML schema definitions (files with an .xsd extension)
- Business object maps (files with a .map extension)
- Relationship and role definitions (files with a .rel and .rol extension)

When a module needs an artifact, the server locates the artifact from the EAR class path and loads the artifact, if it is not already loaded, into memory. From that point on, any request for the artifact uses that copy until it is replaced. Figure 29 on page 354 shows how an application contains components and related libraries.

© Copyright IBM Corp. 2005, 2009

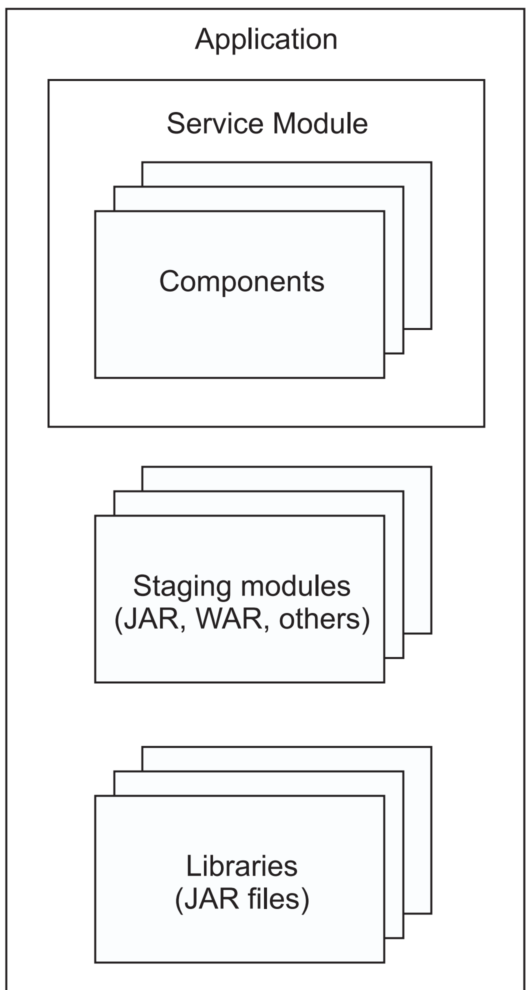


Figure 29. Relationship amongst module, component and library

What are JAR, RAR, and WAR files?

There are a number of files that can contain components of a module. These files are fully described in the Java Platform, Enterprise Edition specification. Details about JAR files can be found in the JAR specification.

In WebSphere Process Server, a JAR file also contains an application, which is the assembled version of the module with all the supporting references and interfaces to any other service components used by the module. To completely install the application, you need this JAR file, any other libraries such as JAR files, Web services archive (WAR) files, resource archive (RAR) files, staging libraries (Enterprise Java Beans - EJB) JAR files, or any other archives, and create an installable EAR file using the serviceDeploy command.

Naming conventions for staging modules

Within the library, there are requirements for the names of the staging modules. These names are unique for a specific module. Name any other modules required to deploy the application so that conflicts with the staging module names do not occur. For a module named *myService*, the staging module names are:

- *myService*App
- *myService*EJB

- *myService*EJBClient
- *myService*Web

Note: The serviceDeploy command only creates the *myService* Web staging module if the service includes a WSDL port type service.

Considerations when using libraries

Using libraries provides consistency of business objects and consistency of processing amongst modules because each calling module has its own copy of a specific component. To prevent inconsistencies and failures it is important to make sure that changes to components and business objects used by calling modules are coordinated with all of the calling modules. Update the calling modules by:

1. Copying the module and the latest copy of the libraries to the production server
2. Rebuilding the installable EAR file using the serviceDeploy command
3. Stopping the running application containing the calling module and reinstall it
4. Restarting the application containing the calling module

EAR file overview

An EAR file is a critical piece in deploying a service application to a production server.

An enterprise archive (EAR) file is a compressed file that contains the libraries, enterprise beans, and JAR files that the application requires for deployment.

You create a JAR file when you export your application modules from WebSphere Integration Developer. Use this JAR file and any other artifact libraries or objects as input to the installation process. The serviceDeploy command creates an EAR file from the input files that contain the component descriptions and Java code that comprise the application.

Preparing to deploy to a server

After developing and testing a module, you must export the module from a test system and bring it into a production environment for deployment. To install an application you also should be aware of the paths needed when exporting the module and any libraries the module requires.

Before you begin

Before beginning this task, you should have developed and tested your modules on a test server and resolved problems and performance issues.

Important: To prevent replacing an application or module already running in a deployment environment make sure the name of the module or application is unique from any already installed.

About this task

This task verifies that all of the necessary pieces of an application are available and packaged into the correct files to bring to the production server.

Note: You can also export an enterprise archive (EAR) file from WebSphere Integration Developer and install that file directly into WebSphere Process Server.

Important: If the services within a component use a database, install the application on a server directly connected to the database.

Procedure

1. Locate the folder that contains the components for the module you are to deploy.

   The component folder should be named *module-name* with a file in it named *module*.module, the base module.
2. Verify that all components contained in the module are in component subfolders beneath the module folder.

   For ease of use, name the subfolder similar to *module/component*.
3. Verify that all files that comprise each component are contained in the appropriate component subfolder and have a name similar to *component-file-name*.component.

   The component files contain the definitions for each individual component within the module.
4. Verify that all other components and artifacts are in the subfolders of the component that requires them.

   In this step you ensure that any references to artifacts required by a component are available. Names for components should not conflict with the names the serviceDeploy command uses for staging modules. See Naming conventions for staging modules.
5. Verify that a references file, *module*.references, exists in the module folder of step 1.

   The references file defines the references and the interfaces within the module.
6. Verify that a wires file, *module*.wires, exists in the component folder.

   The wires file completes the connections between the references and the interfaces within the module.
7. Verify that a manifest file, *module*.manifest, exists in the component folder.

   The manifest lists the module and all the components that comprise the module. It also contains a class path statement so that the serviceDeploy command can locate any other modules needed by the module.
8. Create a compressed file or a JAR file of the module as input to the serviceDeploy command that you will use to prepare the module for installation to the production server.

Example folder structure for MyValue module prior to deployment

The following example illustrates the directory structure for the module MyValueModule, which is made up of the components MyValue, CustomerInfo, and StockQuote.

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
```

```
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
   StockQuote.java
   StockQuoteAsynch.java
   StockQuoteCallback.java
   StockQuoteImpl.java
```

What to do next

Install the module onto the production systems as described in Installing a module on a production serverInstalling a module on a production server.

Considerations for installing service applications on clusters

Installing a service application on a cluster places additional requirements on you. It is important that you keep these considerations in mind as you install any service applications on a cluster.

Clusters can provide many benefits to your processing environment by providing economies of scale to help you balance request workload across servers and provide a level of availability for clients of the applications. Consider the following before installing an application that contains services on a cluster:

- Will users of the application require the processing power and availability provided by clustering?

  If so, clustering is the correct solution. Clustering will increase the availability and capacity of your applications.
- Is the cluster correctly prepared for service applications?

  You must configure the cluster correctly before installing and starting the first application that contains a service. Failure to configure the cluster correctly prevents the applications from processing requests correctly.
- Does the cluster have a backup?

  You must install the application on the backup cluster also.

Chapter 6. Deploying a module

You can deploy a module or a mediation module, as generated by WebSphere® Integration Developer, into a production WebSphere Process Server environment using these steps.

Before you begin

Before deploying a service application to a production server, assemble and test the application on a test server. After testing, export the relevant files as described in *Preparing to deploy to a server* in the Developing and Deploying Modules PDF and bring the files to the production system to deploy. See the information centers for WebSphere Integration Developer and WebSphere Application Server Network Deployment for more information.

Procedure

1. Copy the module and other files onto the production server.

   The modules and resources (EAR, JAR, RAR, and WAR files) needed by the application are moved to your production environment.
2. Run the serviceDeploy command to create an installable EAR file.

   This step defines the module to the server in preparation for installing the application into production.
   a. Locate the JAR file that contains the module to deploy.
   b. Issue the serviceDeploy command using the JAR file from the previous step as input.
3. Install the EAR file from step 2. How you install the applications depends on whether you are installing the application on a stand alone server or a server in a cell.

   Note: You can either use the administrative console or a script to install the application. See the WebSphere Application Server information center for additional information.
4. Save the configuration. The module is now installed as an application.
5. Start the application.

Results

The application is now active and work should flow through the module.

What to do next

Monitor the application to make sure the server is processing requests correctly.

Installing versioned SCA modules in a production environment

You can deploy versioned SCA modules into the run time. Each version of a module exists alongside any other versions currently installed on the server or in the cell.

Before you begin

Make sure you perform the following tasks before installing a versioned SCA module into your production environment:

- In WebSphere Integration Developer, specify that the module is versioned and export it for command-line deployment. See Creating versioned modules and libraries for more information.
- Determine whether you want to co-deploy different versions of the module on a single server or whether you need to co-deploy multiple instances of the same versioned module on more than one cluster in the same cell.

About this task

To install versioned modules, perform the following steps.

Procedure

1. Run serviceDeploy against the versioned module you exported to generate an installable EAR file.

   ```
   serviceDeploy moduleName.zip
   ```

   The serviceDeploy command returns an installable EAR file whose name contains the version and, optionally, cell ID information.
2. Install the module using one of the following methods:
   - From within the administrative console, click SCA Modules and use the **Install** button on the SCA Modules page.
   - From within the administrative console, click Applications > Install New Application.
   - Use the AdminApp.install wsadmin command.
3. If you want to install a versioned module on multiple servers or clusters in a cell, do the following for each module instance you require:
   a. Use the createVersionedSCAModule command to create an instance of the module.

      ```
      createVersionedSCAModule -archiveAbsolutePath input_archive_dir
      -workingDirectory working_dir -uniqueCellID cell_ID
      ```

   b. Install the resulting EAR file as described in Step 2.
4. Optional: Use the validateSCAImportExportInformation command to validate that all SCA bindings and selector export bindings in the specified EAR file exist on the bus.

Results

You now have one or more versioned applications in your production environment. They can all be administered through the administrative console or through corresponding administrative commands.

Note: To preserve versioning information, the installation process automatically modifies the module name to ensure it is unique within the server or cell through the use of either the serviceDeploy or createVersionedSCAModule command. These commands add the version number, a unique cell ID, or both to the original module name.

```
moduleName_vversionValue_uniqueCellID
```

For example, if you followed the steps in this topic, deploying version 1.0.1 of the module billingProcess results in a module called billingProcess_v1_0_1 and an installed service application called billingProcess_v1_0_1App. If you also specify a unique cell ID (for example, Cell5), then the module is called billingProcess_v1_0_1_Cell5 and the installed service application is called billingProcess_v1_0_1_Cell5App.

Installing an SCA module with the console

Before you can start running a module or a mediation module, you must deploy it to a server or cluster. Deployment involves creating an installable EAR file and installing the EAR file onto the server or cluster.

Before you begin

If you have exported either a module or a mediation module to a JAR file, use the serviceDeploy command to create an installable EAR file from the JAR file. For more information, see Chapter 6, “Deploying a module,” on page 359.

About this task

You must install the EAR file onto a server or cluster before you can start running the module or mediation module.

Instead of using the administrative console, you can use other methods to install the EAR file, such as the AdminApp.install or AdminApp.installinteractive command with the wsadmin tool.

Important: If, after you start performing the steps, you decide not to install the application you must click **Cancel**: do not simply move to another administrative console page.

Procedure

1. From the administrative console, click **Applications** → **Install New Application** in the console navigation pane. The first of two **Preparing for application installation** pages is displayed.
2. On the first **Preparing for application installation** page:
   a. Specify the full path name of the EAR file. For more information, see Installing applications with the console.
   b. Select whether to use default values or specify some of the values yourself:
      Prompt me only when additional information is required
         Displays only the module mapping step and other steps where you must specify information.
      Show me all installation options and parameters
         Displays all installation steps. To use **Generate default bindings**, which supplies default values for incomplete bindings, select this option.
   c. Click **Next**.
3. Installing an EAR file containing a mediation flow is similar to installing any other enterprise application EAR file into WebSphere Application Server. For detailed information about filling in the second **Preparing for application installation** page and specifying the options in the remaining wizard steps, see Installing applications with the console.

4. When you are installing a mediation module, or a module containing a mediation flow, there is one additional optional step you can perform. On the **Edit module properties** panel, you can edit the values of the properties in the module. If properties belong to a group they are displayed inside an expandable section; if they do not belong to a group you can view them immediately.

Results

You can now start the module or mediation module.

Creating an installable EAR file using serviceDeploy

To install an application in the production environment, take the files copied to the production server and create an installable EAR file.

Before you begin

Before starting this task, you must have a JAR file that contains the module and services you are deploying to the server. See "Preparing to deploy to a server" for more information.

About this task

The serviceDeploy command takes a JAR file, any other dependent EAR, JAR, RAR, WAR and ZIP files and builds an EAR file that you can install on a server.

Procedure

1. Locate the JAR file that contains the module to deploy.
2. Issue the serviceDeploy command using the JAR file from the previous step as input.

   This step creates an EAR file.

   Note: Perform the following steps at an administrative console.
3. Select the EAR file to install in the administrative console of the server.
4. Click **Save** to install the EAR file.

Deploying applications using Apache Ant tasks

ANT tasks allow you to define the deployment of multiple applications to WebSphere Process Server and have them run unattended on a server.

Before you begin

This task assumes the following:
- The applications being deployed have already been developed and tested.
- The applications are to be installed on the same server or servers.
- You have some knowledge of Apache Ant tasks.
- You understand the deployment process.

Information about developing and testing applications is located in the WebSphere Integration Developer information center.

The Generated API and SPI documentation reference section provides details of application programming interfaces. Apache Ant tasks are described in the package

com.ibm.websphere.ant.tasks. For the purpose of this topic, the tasks of interest are ServiceDeploy and InstallApplication.

About this task

If you need to install multiple applications concurrently, develop an Apache Ant task before deployment. The Apache Ant task can then deploy and install the applications on the servers without your involvement in the process.

Procedure

1. Identify the applications to deploy.
2. Create a JAR file for each application.
3. Copy the JAR files to the target servers.
4. Create an Apache Ant task to run the ServiceDeploy command to create the EAR file for each server.
5. Create an Apache Ant task to run the InstallApplication command for each EAR file from step 4 on the applicable servers.
6. Run the ServiceDeploy Apache Ant task to create the EAR file for the applications.
7. Run the InstallApplication Apache Ant task to install the EAR files from step 6.

Results

The applications are correctly deployed on the target servers.

Example of deploying an application unattended

This example of deploying an application unattended shows an Apache ANT task contained in a file myBuildScript.xml.

```
<?xml version="1.0">

<project name="OwnTaskExample" default="main" basedir=".">
 <taskdef name="servicedeploy"
   classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
 <target name="main" depends="main2">
  <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
   ignoreErrors="true"
   outputApplication="c:/synctest/SyncTargetEAREAR"
   workingDirectory="c:/synctest"
   noJ2eeDeploy="true"
   cleanStagingModules="true"/>
 </target>
</project>
```

This statement shows how to invoke the Apache Ant task.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

Tip: Multiple applications can be deployed unattended by adding additional project statements into the file.

What to do next

Use the administrative console to verify that the newly-installed applications are started and processing the workflow correctly.

Chapter 7. Installing business process and human task applications

You can distribute Service Component Architecture (SCA) modules that contain business processes or human tasks, or both, to deployment targets. A deployment target can be a server or a cluster.

Before you begin

Verify that Business Flow Manager and Human Task Manager are installed and configured for each application server or cluster on which you want to install your application.

About this task

You can install business process and task applications from the administrative console, from the command line, or by running an administrative script.

Results

After a business process or human task application is installed, all of the business process templates and human task templates are put into the start state. You can create process instances and task instances from these templates.

What to do next

Before you can create process instances or task instances, you must start the application.

How business process and human task applications are installed in a network deployment environment

When process templates or human task templates are installed in a network deployment environment, the following actions are performed automatically by the application installation.

The application is installed in stages. Each stage must complete successfully before the following stage can begin.

1. The application installation starts on the deployment manager.

   During this stage, the business process templates and human task templates are configured in the WebSphere configuration repository. The application is also validated. If errors occur, they are reported in the System.out file, in the System.err file, or as FFDC entries on the deployment manager.
2. The application installation continues on the node agent.

   During this stage, the installation of the application on one application server instance is triggered. This application server instance is either part of, or is, the deployment target. If the deployment target is a cluster with multiple cluster members, the server instance is chosen arbitrarily from the cluster members of this cluster. If errors occur during this stage, they are reported in the SystemOut.log file, in the SystemErr.log file, or as FFDC entries on the node agent.

© IBM Corporation 2005, 2008 © IBM 2005, 2009

3. The application runs on the server instance.

   During this stage, the process templates and human templates are deployed to the Business Process Choreographer database on the deployment target. If errors occur, they are reported in the System.out file, in the SystemErr.log file, or as FFDC entries on this server instance.

Deployment of business processes and human tasks

When WebSphere Integration Developer or service deployment generates the deployment code for your process or task, each process component or task component is mapped to one session enterprise bean. All deployment code is packaged in the enterprise application (EAR) file. Additionally, for each process, a Java class which represents Java code in this process is generated and embedded in the EAR file during installation of the enterprise application. Each new version of a model that is to be deployed must be packaged in a new enterprise application.

When you install an enterprise application that contains business processes or human tasks, then these are stored as business process templates or human task templates, as appropriate, in the Business Process Choreographer database. Newly installed templates are, by default, in the started state. However, the newly installed enterprise application is in the stopped state. Each installed enterprise application can be started and stopped individually.

You can deploy many different versions of a process template or task template, each in a different enterprise application. When you install a new enterprise application, the version of the template that is installed is determined as follows:
- If the name of the template and the target namespace do not already exist, a new template is installed
- If the template name and target namespace are the same as those of an existing template, but the valid-from date is different, a new version of an existing template is installed

Note: The template name is derived from the name of the component and not from the business process or human task.

If you do not specify a valid-from date, the date is determined as follows:
- If you use WebSphere Integration Developer, the valid-from date is the date on which the human task or the business process was modeled.
- If you use service deployment, the valid-from date is the date on which the serviceDeploy command was run. Only collaboration tasks get the date on which the application was installed as the valid-from date.

Installing business process and human task applications interactively

You can install an application interactively at runtime using the wsadmin tool and the installInteractive script. You can use this script to change settings that cannot be changed if you use the administrative console to install the application.

About this task

Perform the following steps to install business process applications interactively.

Procedure
1. Start the wsadmin tool.

In the *profile_root*/bin directory, enter `wsadmin`.

2. Install the application.

   At the wsadmin command-line prompt, enter the following command:

   ```
   $AdminApp installInteractive application.ear
   ```

   where *application.ear* is the qualified name of the enterprise archive file that contains your process application. You are prompted through a series of tasks where you can change values for the application.

3. Save the configuration changes.

   At the wsadmin command-line prompt, enter the following command:

   ```
   $AdminConfig save
   ```

   You must save your changes to transfer the updates to the master configuration repository. If a scripting process ends and you have not saved your changes, the changes are discarded.

Configuring process application data source and set reference settings

You might need to configure process applications that run SQL statements for the specific database infrastructure. These SQL statements can come from information service activities or they can be statements that you run during process installation or instance startup.

About this task

When you install the application, you can specify the following types of data sources:

- Data sources to run SQL statements during process installation
- Data sources to run SQL statements during the startup of a process instance
- Data sources to run SQL snippet activities

The data source required to run an SQL snippet activity is defined in a BPEL variable of type tDataSource. The database schema and table names that are required by an SQL snippet activity are defined in BPEL variables of type tSetReference. You can configure the initial values of both of these variables.

You can use the wsadmin tool to specify the data sources.

Procedure

1. Install the process application interactively using the wsadmin tool.
2. Step through the tasks until you come to the tasks for updating data sources and set references.

   Configure these settings for your environment. The following example shows the settings that you can change for each of these tasks.
3. Save your changes.

Example: Updating data sources and set references, using the wsadmin tool

In the **Updating data sources** task, you can change data source values for initial variable values and statements that are used during installation of the process or when the process starts. In the **Updating set references** task, you can configure the settings related to the database schema and the table names.

```
Task [24]: Updating data sources

//Change data source values for initial variable values at process start

Process name:  Test
// Name of the process template
Process start or installation time:  Process start
// Indicates whether the specified value is evaluated
//at process startup or process installation
Statement or variable:  Variable
// Indicates that a data source variable is to be changed
Data source name:  MyDataSource
// Name of the variable
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name to jdbc/newName
Task [25]: Updating set references

// Change set reference values that are used as initial values for BPEL variables

Process name:  Test
// Name of the process template
Variable:  SetRef
// The BPEL variable name
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name of the data source of the set reference to jdbc/newName
Schema name: [IISAMPLE]
// The name of the database schema
Schema prefix: []:
// The schema name prefix.
// This setting applies only if the schema name is generated.
Table name: [SETREFTAB]: NEWTABLE
// Sets the name of the database table to NEWTABLE
Table prefix: []:
// The table name prefix.
// This setting applies only if the prefix name is generated.
```

Uninstalling business process and human task applications, using the administrative console

You can use the administrative console to uninstall applications that contain business processes or human tasks.

Before you begin

To uninstall an application that contains business processes or human tasks, the following conditions must apply:

- If the application is installed on a stand-alone server, the server must be running and have access to the Business Process Choreographer database.
- If the application is installed on a cluster, the deployment manager and at least one cluster member must be running. The cluster member must have access to the Business Process Choreographer database.
- If the application is installed on a managed server, the deployment manager and the managed server must be running. The server must have access to the Business Process Choreographer database.
- There are no instances of business process or human task templates present in any state.

About this task

To uninstall an enterprise application that contains business processes or human tasks, perform the following actions:

Procedure

1. Click **Applications** → **Enterprise Applications** in the administrative console navigation pane.
2. Select the application that you want to uninstall and click **Stop**.

   This step fails if any process instances or task instances still exist in the application. You can either use the Business Process Choreographer Explorer to delete the instances, or use the **-force** option described in "Uninstalling business process and human task applications, using an administrative command."
3. Select the application that you want to uninstall, and click **Uninstall**.
4. Click **Save** to save your changes.

Results

The application is uninstalled.

Uninstalling business process and human task applications, using an administrative command

Using the bpcTemplates.jacl script provides an alternative to the administrative console for uninstalling applications that contain business processes or human tasks.

Before you begin

To uninstall an application that contains business processes or human tasks, the following conditions must apply:

- If the application is installed on a stand-alone server, the server must be running and have access to the Business Process Choreographer database.
- If the application is installed on a cluster, the deployment manager and at least one cluster member must be running. The cluster member must have access to the Business Process Choreographer database.
- If the application is installed on a managed server, the deployment manager and the managed server must be running. The server must have access to the Business Process Choreographer database.
- There are no instances of business process or human task templates present in any state, unless you intend to use the **-force** option.
- If you want to use the **-force** option, and administrative security is enabled, verify that your user ID has administrator or operator authority.
- Ensure that the server process to which the administrative client connects is running. To ensure that the administrative client automatically connects to the server process, do not use the `-conntype NONE` option as a command option.

About this task

The following steps describe how to use the bpcTemplates.jacl script to uninstall applications that contain business process templates or human task templates.

Procedure

1. If there are still process instances or task instances associated with the templates in the application that you want to uninstall perform one or both of the following:
   - Use the Business Process Choreographer Explorer to delete the instances.
   - Plan to use the **-force** option to delete any instances that are associated with the templates, stop the templates, and uninstall them in one step. Use this option with care because it also deletes all of the data associated with the running instances.
2. Change to the Business Process Choreographer administration scripts directory.

   On Windows platforms, enter:

   ```
   cd install_root\ProcessChoreographer\admin
   ```

   On Linux, UNIX, and i5/OS platforms, enter:

   ```
   cd install_root/ProcessChoreographer/admin
   ```

3. Stop the templates and uninstall the corresponding application.

   On Windows platforms, enter:

   ```
   install_root\bin\wsadmin -f bpcTemplates.jacl
                                 [-user user_name]
                                 [-password user_password]
                                 -uninstall application_name
                                 [-force]
   ```

   On Linux, UNIX, and i5/OS platforms, enter:

   ```
   install_root/bin/wsadmin -f bpcTemplates.jacl
                                 [-user user_name]
                                 [-password user_password]
                                 -uninstall application_name
                                 [-force]
   ```

   Where:

   user_name
      If administrative security is enabled, provide the user ID for authentication.

   user_password
      If administrative security is enabled, provide the user password for authentication.

   application_name
      The name of the application to be uninstalled.

   -force
      Causes any running instances to be stopped and deleted before the application is uninstalled. Use this option with care because it also deletes all of the data associated with the running instances.

Results

The application is uninstalled.

Chapter 8. Adapters and their installation

Adapters allow your application to communicate with other components of your enterprise information system.

The process you use to install adapters is described in Configuring and using adapters in the WebSphere Integration Developer information center.

© Copyright IBM Corp. 2005, 2009

Chapter 9. Troubleshooting a failed deployment

This topic describes the steps to take to determine the cause of a problem when deploying an application. It also presents some possible solutions.

Before you begin

This topic assumes the following things:
- You have a basic understanding of debugging a module.
- Logging and tracing is active while the module is being deployed.

About this task

The task of troubleshooting a deployment begins after you receive notification of an error. There are various symptoms of a failed deployment that you have to inspect before taking action.

Procedure

1. Determine if the application installation failed.

   Examine the SystemOut.log file for messages that specify the cause of failure. Some of the reasons an application might not install include the following:
   - You are attempting to install an application on multiple servers in the same Network Deployment cell.
   - An application has the same name as an existing module on the Network Deployment cell to which you are installing the application.
   - You are attempting to deploy J2EE modules within an EAR file to different target servers.

   Important: If the installation has failed and the application contains services, you must remove any SIBus destinations or J2C activation specifications created prior to the failure before attempting to reinstall the application. The simplest way to remove these artifacts is to click **Save > Discard all** after the failure. If you inadvertently save the changes, you must manually remove the SIBus destinations and J2C activation specifications (see Deleting SIBus destinations and Deleting J2C activation specifications in the Administering section).
2. If the application is installed correctly, examine it to determine if it started successfully.

   If the application did not start successfully, the failure occurred when the server attempted to initiate the resources for the application.
   a. Examine the SystemOut.log file for messages that will direct you on how to proceed.
   b. Determine if resources required by the application are available and/or have started successfully.

      Resources that are not started prevent an application from running. This protects against lost information. The reasons for a resource not starting include:
      - Bindings are specified incorrectly
      - Resources are not configured correctly
      - Resources are not included in the resource archive (RAR) file
      - Web resources not included in the Web services archive (WAR) file

© Copyright IBM Corp. 2005, 2009

c. Determine if any components are missing.

   The reason for missing a component is an incorrectly built enterprise archive (EAR) file. Make sure that the all of the components required by the module are in the correct folders on the test system on which you built the Java archive (JAR) file. "Preparing to deploy to a server" contains additional information.
3. Examine the application to see if there is information flowing through it.

   Even a running application can fail to process information. Reasons for this are similar to those mentioned in step 2b on page 373.
   a. Determine if the application uses any services contained in another application. Make sure that the other application is installed and has started successfully.
   b. Determine if the import and export bindings for devices contained in other applications used by the failing application are configured correctly. Use the administrative console to examine and correct the bindings.
4. Correct the problem and restart the application.

Deleting J2C activation specifications

The system builds J2C application specifications when installing an application that contains services. There are occasions when you must delete these specifications before reinstalling the application.

Before you begin

If you are deleting the specification because of a failed application installation, make sure the module in the Java Naming and Directory Interface (JNDI) name matches the name of the module that failed to install. The second part of the JNDI name is the name of the module that implemented the destination. For example in sca/SimpleBOCrsmA/ActivationSpec, **SimpleBOCrsmA** is the module name.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as administrator or configurator to perform this task.

About this task

Delete J2C activation specifications when you inadvertently saved a configuration after installing an application that contains services and do not require the specifications.

Procedure

1. Locate the activation specification to delete.

   The specifications are contained in the resource adapter panel. Navigate to this panel by clicking **Resources > Resource adapters**.
   a. Locate the **Platform Messaging Component SPI Resource Adapter.**

      To locate this adapter, you must be at the **node** scope for a standalone server or at the **server** scope in a deployment environment.
2. Display the J2C activation specifications associated with the Platform Messaging Component SPI Resource Adapter.

   Click on the resource adapter name and the next panel displays the associated specifications.

3. Delete all of the specifications with a **JNDI Name** that matches the module name that you are deleting.
   a. Click the check box next to the appropriate specifications.
   b. Click **Delete.**

Results

The system removes selected specifications from the display.

What to do next

Save the changes.

Deleting SIBus destinations

Service integration bus (SIBus) destinations are used to hold messages being processed by SCA modules. If a problem occurs, you might have to remove bus destinations to resolve the problem.

Before you begin

If you are deleting the destination because of a failed application installation, make sure the module in the destination name matches the name of the module that failed to install. The second part of the destination is the name of the module that implemented the destination. For example in sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer, **SimpleBOCrsmA** is the module name.

Required security role for this task: When security and role-based authorization are enabled, you must be logged in as administrator or configurator to perform this task.

About this task

Delete SIBus destinations when you inadvertently saved a configuration after installing an application that contains services or you no longer need the destinations.

Note: This task deletes the destination from the SCA system bus only. You must remove the entries from the application bus also before reinstalling an application that contains services (see Deleting J2C activation specifications in the Administering section of this information center.

Procedure

1. Log into the administrative console.
2. Display the destinations on the SCA system bus.
   a. In the navigation pane, click **Service integration** → **buses**
   b. In the content pane, click **SCA.SYSTEM.*cell_name*.Bus**
   c. Under Destination resources, click **Destinations**
3. Select the check box next to each destination with a module name that matches the module that you are removing.
4. Click **Delete.**

Results

The panel displays only the remaining destinations.

What to do next

Delete the J2C activation specifications related to the module that created these destinations.

Part 3. Appendixes

© Copyright IBM Corp. 2005, 2009

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

© Copyright IBM Corp. 2005, 2009

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
1001 Hillsdale Blvd., Suite 400
Foster City, CA 94404
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows: (c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (R or TM), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (http://www.eclipse.org).

IBM WebSphere Process Server for Multiplatforms, Version 6.2

IBM®

Printed in USA