



## **Technical product overview**





## Technical product overview

**Note**

Before using this information and the product it supports, read the information in Notices at the end of this book.

**Second Edition (December 2005)**

**© Copyright International Business Machines Corporation 2005. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## Chapter 1. Business integration . . . . 1

Integration between business units . . . . .	2
Integration between enterprises . . . . .	2
WebSphere Integration Developer . . . . .	3
Standards . . . . .	4
Role of the integration developer . . . . .	4

## Chapter 2. Service Component

### Architecture . . . . . 5

Service components . . . . .	6
Service data objects . . . . .	8
Service qualifiers . . . . .	9
Modules . . . . .	9
Imports and exports . . . . .	11
Accessing EIS systems . . . . .	11
Accessing other modules . . . . .	12
Service import and export binding types . . . . .	12
Selecting appropriate bindings . . . . .	13
Service implementation types . . . . .	14

Java objects . . . . .	14
BPEL process . . . . .	15
State machines . . . . .	16
Business rules . . . . .	16
Selectors . . . . .	17
Human task . . . . .	17
Interface map . . . . .	18
Mediation flow . . . . .	19
Stand-alone references . . . . .	20
Related information . . . . .	20

## Chapter 3. Learning about the tools . . 21

Welcome view overview . . . . .	21
Tutorials in the Welcome view . . . . .	21
Samples in the Welcome view . . . . .	22
Information center . . . . .	23

## Notices . . . . . 25



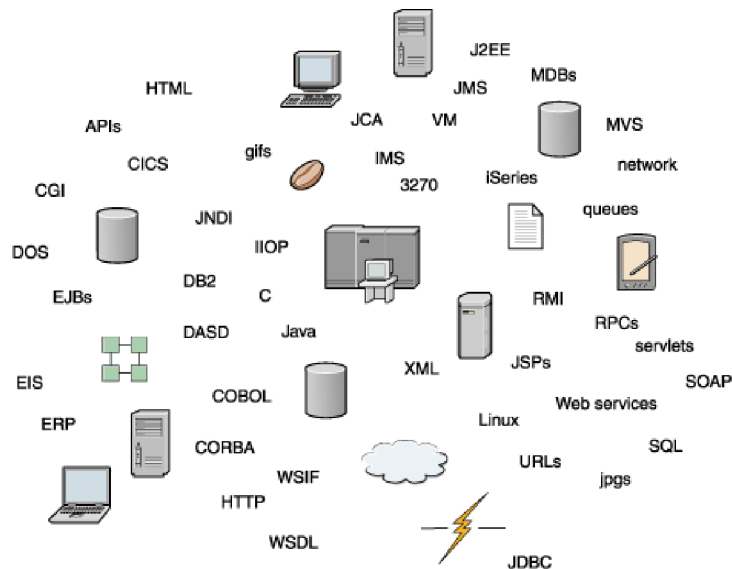
---

## Chapter 1. Business integration

Business integration means integrating applications, data, and processes within an enterprise or amongst a set of enterprises. The challenge of this task and how it is met by WebSphere Integration Developer is examined.

You have been told to build a portal for your customers to your business applications. It must provide access to dozens of major business applications and related data spread throughout all of your business units. You have also been asked to add your business partners' applications to the portal. Integration also means developing processes, since there will be some logic to the sequence of these assembled applications. You have 20 business units and a dozen business partners. Your portal must be available on the Web 24 hours a day. You have been given a staff of six developers, including yourself, and you have four months to get it up and running.

Most people given that challenge would look at the technology they had accumulated over the past few decades and see a collage like the following picture:



It is an overwhelming, but not impossible situation. The most difficult problem you face in addition to the disjointed collection of hardware and software is your time and resource constraints. You must find powerful tools to quickly bring together the applications and data scattered across the enterprises involved. Hand-coding is not an option.

Are you the only one facing this situation? No. This is a widespread problem that has been many years in the making. According to *CIO Survey*, December 2001, application integration is consistently one of the top three technology priorities. According to *The Business Integrator Journal*, Winter 2001, in a recent survey, two out of three developers use integration software to develop Web-based solutions. On average, each developer was integrating three different systems.

Let us look at two forces in business that are driving the problem to the crisis point: integration between business units within an enterprise and integration between enterprises. Then we will look at how WebSphere Integration Developer addresses these issues and, in particular, the significance of its commitment to industry standards. Finally, we will look at the integration specialist, who is the type of person who will use WebSphere Integration Developer's tools to solve the problems's posed earlier.

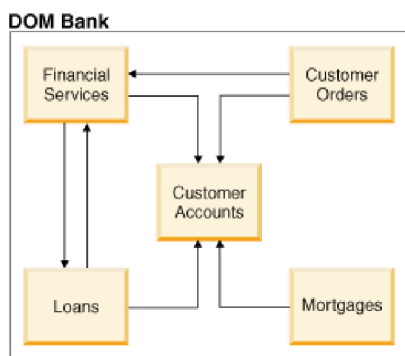
---

## Integration between business units

Business units frequently find themselves collaborating today, creating a need for close integration of their applications.

Formerly autonomous business units are being integrated because technology allows them to be connected and because efficiency says they must operate in a more cooperative way to minimize overhead and maximize output. A common corporate goal also drives business units together. A marketing unit and a research-and-development unit both want to produce a profitable product. By integrating the knowledge of the market with the product development information, the odds of producing that successful product increase. Collaboration between business units also lets corporations leverage their many existing business applications by permitting their reuse in different business contexts.

Integration between business units is easier than integration between enterprises because there is less security risk, and managing the interactions between the units should not be as difficult. The business units are probably using the same protocols, operating systems, and computer languages. In other words, it is a relatively homogenous environment. The key, however, is to have the right tools to quickly integrate the applications. In the following diagram, the DOM bank has several business units that need to share each other's information. Several years ago, the DOM bank could manage by printing copies of information from one business unit and walking them over to another, which had its own systems and applications. Today, the DOM bank must create integrated applications encompassing its business units if it wishes to keep up with its competitors.



---

## Integration between enterprises

The forces driving the integration of applications between business units also applies between enterprises, as partnerships or takeovers require shared data and processes.

Technology enables enterprises to be linked in mutually beneficial areas. For example, an automobile manufacturer can set up an integrated process with a tire supplier so that when the stock of tires is low the supplier is notified automatically. Integration between enterprises is being driven by economic necessity. Having closer ties amongst corporations means less time delays and less overhead to get things done. These automated processes mean that people spend less time to process transactions between enterprises and travel costs and face-to-face meeting time can be reduced significantly. Administration costs are similarly reduced and turnaround time between notification, delivery and invoices is improved.

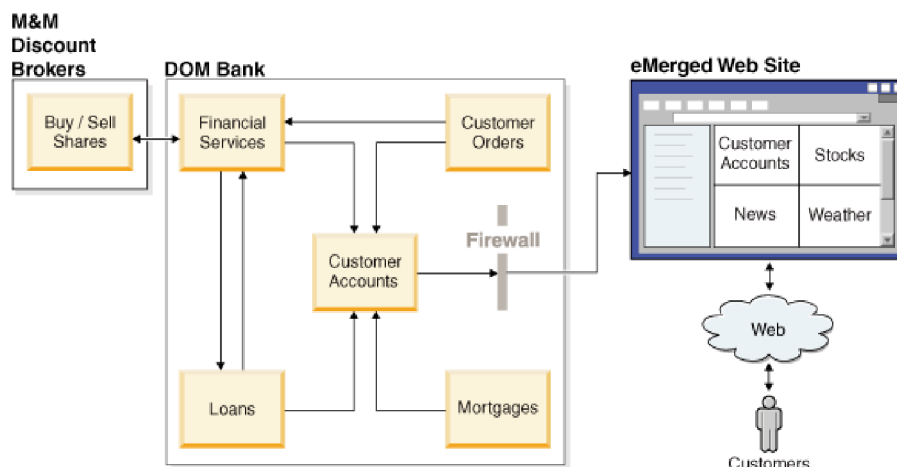
But different enterprises have different histories. Their applications are coded in different languages on different platforms using different communication protocols. There are also greater security risks when working with different organizations. Whatever the benefits and even the necessity of integration between enterprises, the costs in development time can be significant without the right tools.



## WebSphere Integration Developer

WebSphere Integration Developer is the answer to the integration challenges organization's face on a daily basis. It has been designed as a complete integration development environment for those building integrated applications. To simplify and accelerate the development of integrated applications, this environment provides a layer of abstraction that separates the visually-presented components you work with from the underlying implementation.

Integrated applications are not simple. They can call applications on Enterprise Information Systems (EIS), involve business processes across departments or enterprises, and invoke applications locally or remotely written in a variety of languages and running on a variety of operating systems. For example, eMerged Corporation was created by merging DOM bank and M&M Discount brokers. The merger meant all of the above: applications on EIS systems, business processes, and applications within each former corporation had to be shared between the corporations and presented in a seamless way to the new set of customers. However, eMerged accomplished the task and, as shown in the following diagram, customers from both of the former separate businesses can access all their financial information online.



eMerged used WebSphere Integration Developer's tools to build the integrated applications for themselves and their customers. These tools present applications, including applications that exist remotely on EIS systems, and business processes as components. The components are created and assembled into other integrated applications (that is, applications created from a set of components) through visual editors. The visual editors present a layer of abstraction between the components and their implementations. A developer using the tools can create an integrated application without detailed knowledge of the underlying implementation of each component.

The tools allow both a top-down design approach to building an integrated application, where the implementation for one or more components does not exist and is added later; or a bottom-up approach, where the components are already implemented and the developer assembles them by dragging and dropping them in a visual editor and then creates a logical flow amongst them by joining them with lines. A debugging and test environment means full testing before your applications are deployed to a production server. Setting monitoring points lets you see how an application is used in real time in order to fine tune it for optimal performance.

WebSphere Integration Developer's tools are based on a service-oriented architecture. Components are services and an integrated application involving many components is a service. The services created comply to the leading, industry-wide standards. Business processes, which also become components, are similarly created with easy-to-use visual tools that comply to the industry-standard Business Process Execution Language (BPEL). WebSphere Integration Developer is available on both Windows and Linux platforms.

Here are some benefits of WebSphere Integration Developer's tools:

- They are simple to learn
- They can be applied to complex integration situations
- You can produce applications quickly that conform to industry-wide standards

## Standards

Applications created by WebSphere Integration Developer conform to industry-wide standards associated with service-oriented architecture.

No one wants to create applications that are tied to proprietary code that may be unsupported in a few years or involve costly licensing fees. Standards-based integration is therefore a fundamental aspect of WebSphere Integration Developer. For connectivity, J2EE Connector Architecture standards are used. For asynchronous messaging, often used in large applications requiring guaranteed delivery of data, the Java Message Service (JMS) standard is used. WebSphere Integration Developer can easily integrate Web services based on Simple Object Access Protocol (SOAP). To describe a service, the well-established Web Services Description Language (WSDL) standard is used. To define a business process, the Business Process Execution Language (BPEL) standard is employed.

These standards-based interfaces and components comprise an open-ended and pluggable architecture. Proprietary elements, however, are not excluded; they are accessed through the use of standardized interfaces. This means applications created in WebSphere Integration Developer may interact with .NET applications, for example. In the architecture section, a link is provided to the full Service Component Architecture, which provides an extensive list of the many standards supported.

## Role of the integration developer

The integration developer is the primary user of WebSphere Integration Developer. This person, through the use of visual tools, can build a complex integrated application without requiring extensive knowledge of the underlying implementation.

WebSphere Integration Developer presents applications and business processes as *components*. The implementation of the components remains hidden and the components interoperate through interfaces. As a result, integration developers do not need to have extensive knowledge of the underlying implementation of the components to create an integrated application that uses them. Integration developers, however, will likely have a broad technical knowledge in the integration field since they need some understanding of EIS systems, business processes, and applications coded in Java or other languages. For example, an architect has a broad understanding of how a system works without knowing what each component does in detail. Like an architect, an integration developer might be the person in an organization who designs the overall application, and then has others who code the implementation of the specific components.

---

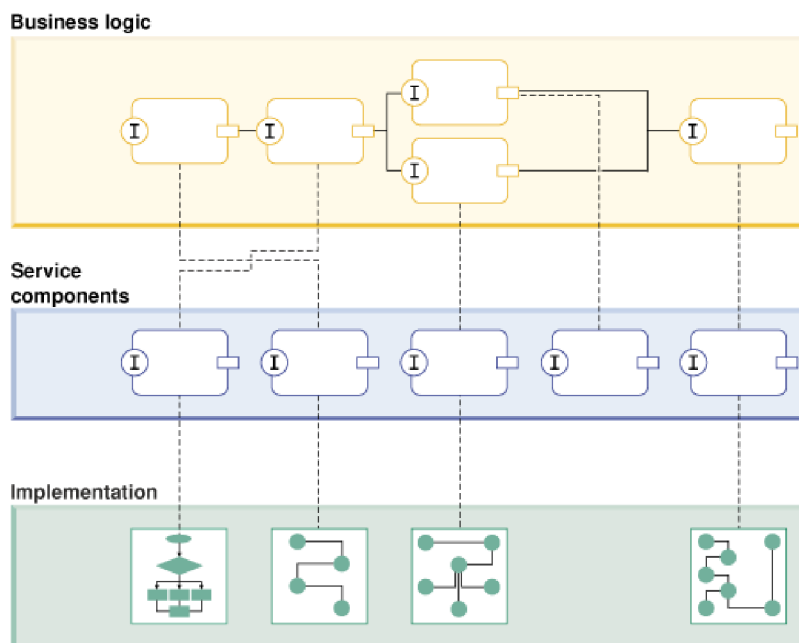
## Chapter 2. Service Component Architecture

Service Component Architecture, based on the industry-standard service-oriented architecture, presents all business processes - Web services, Enterprise Information System (EIS) service assets, workflows, databases, and so on - in a service-oriented way. In this section, we examine at a high level the services and service data objects created by this architecture, which together express business logic and refer to business data.

The goal of Service Component Architecture is to separate business integration logic from implementation so that an integration developer can focus on assembling an integrated application rather than on the implementation details. To achieve that end, service components that contain the implementation of individual services required by business processes are created. The result is an architecture of three layers - business integration logic, service components, and implementation - as shown in the following diagram.



Since the service components contain the implementation, they can be assembled graphically by the integration developer without the knowledge of low-level implementation details. Service components also provide the option of letting the integration developer, or someone who works for the integration developer, add the implementation later. As you will see in the product, components are assembled together visually. In other words, you are not exposed to the code within the components. In the business logic level shown in the diagram that follows, the components are assembled independently of their implementation. The service-oriented architecture, then, lets you focus on solving your business problems by using and reusing components rather than diverting your attention to the technology that is implementing the services you are using.

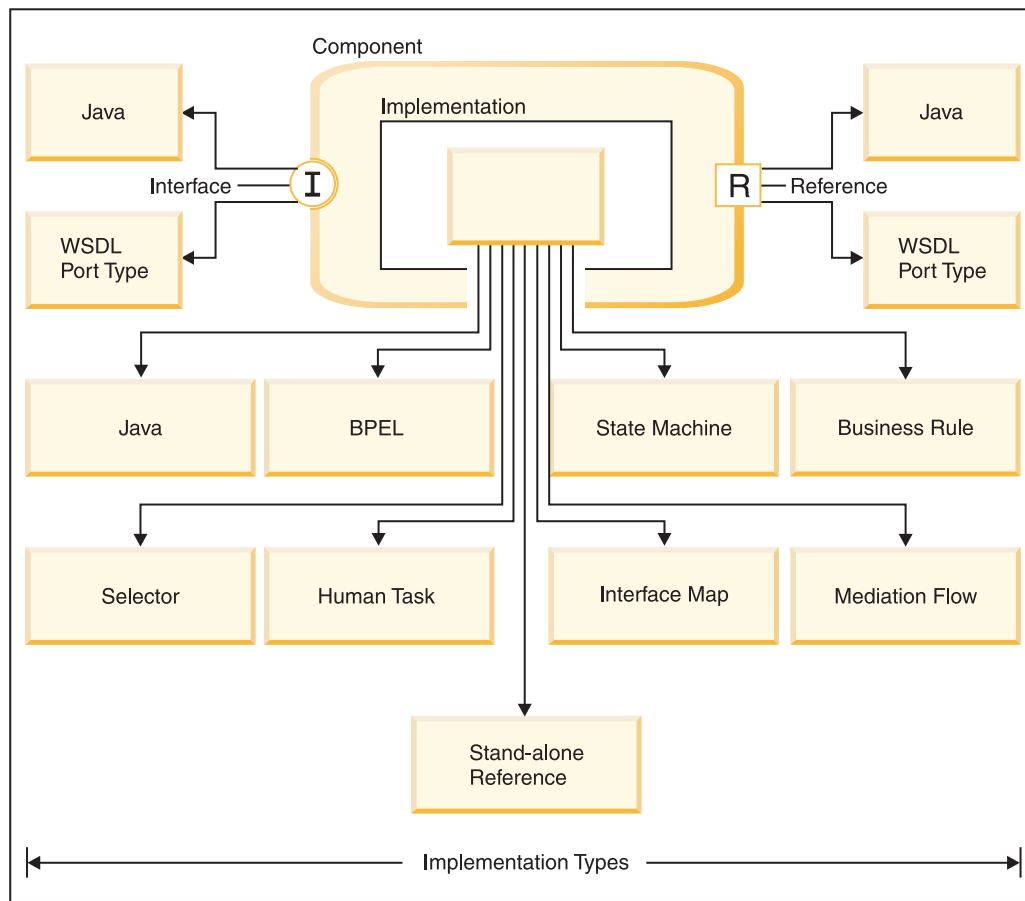


## Service components

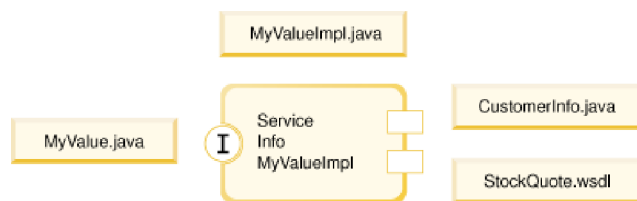
A service component configures a service implementation. A service component is presented in a standard block diagram.

A component consists of an implementation, which is hidden when using WebSphere® Integration Developer's tools, one or more interfaces, which defines its inputs, outputs and faults, and zero or more references. A reference identifies the interface of another service or component that this component requires or consumes. An interface may be defined in one of two languages: a WSDL port type or Java™. An interface supports synchronous and asynchronous interaction styles. A component's implementation can be in various languages.

An interface type can be WSDL or Java, but if there are multiple interfaces you cannot mix WSDL with Java.

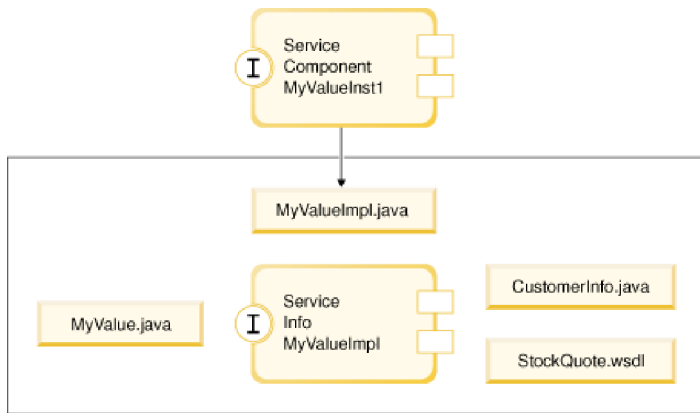


In the picture below, we have a component in the center. It's implementation, MyValueImpl, is in Java as is its interface. It has two references: another Java interface and a WSDL interface.



When working with this component, as shown below, you effectively only see the component itself. A reference to this component from another component would be revealed visually by a line to its interface. A reference from this component would be revealed by a line from its reference point to the interface of other component. A reference represents a service that this component consumes. By naming a reference and only specifying its interface, it allows the component implementation author to defer binding that reference to an actual service until later. At that later time, the integration specialist will do so by wiring from the reference to the interface of another component or import. This loose coupling, which allows for deferred binding and the re-use of implementations, is one of the key reasons for using WebSphere Integration Developer's Service Component Architecture.

A component may also have properties and qualifiers. A qualifier is a quality of service (QoS) directive on interfaces and references for the run time.

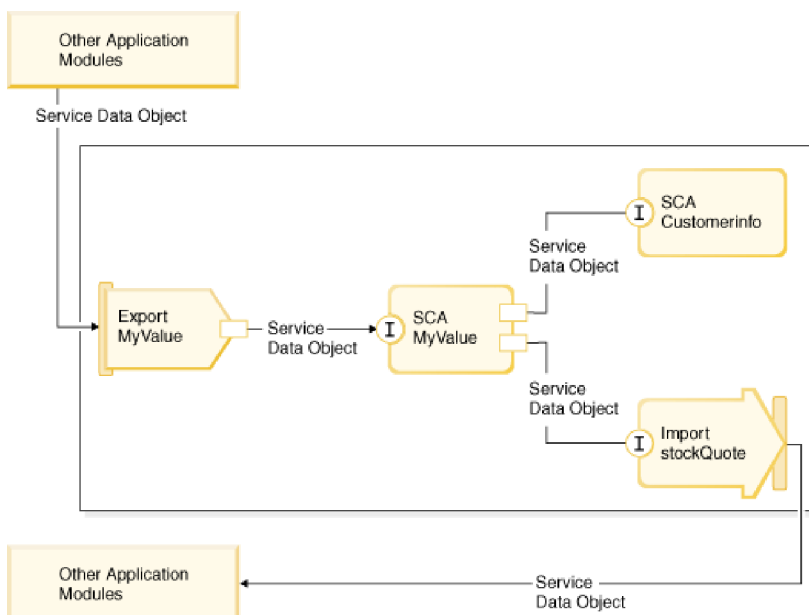


## Service data objects

Service data objects complement Service Component Architecture. Service Component Architecture defines the services as components and the connectivity between them. *Service data objects* define the data flowing between components.

Each component passes information as input and output. When a service is invoked, data objects are passed as an XML document with document literal encoding when using a WSDL port type or as a Java object when using a Java interface. Data objects are the preferred form for data and metadata in Service Component Architecture services. Similar to components, service data objects separate the data object from its implementation. For example, a component interacts with purchase orders while the purchase order itself might use JDBC, EJB, and so on, to perform the updates to the data. Service data objects let the integration developer focus on working with business artifacts. In fact, service data objects are transparent to the integration developer. They are defined by a service data objects Java Specification Request (JSR).

In the diagram that follows, service data objects are passed from an external service to an export, from an export to a component, from a component to a component, from a component to an import, and from an import to a service. Imports and exports are discussed in a subsequent imports and exports section.



---

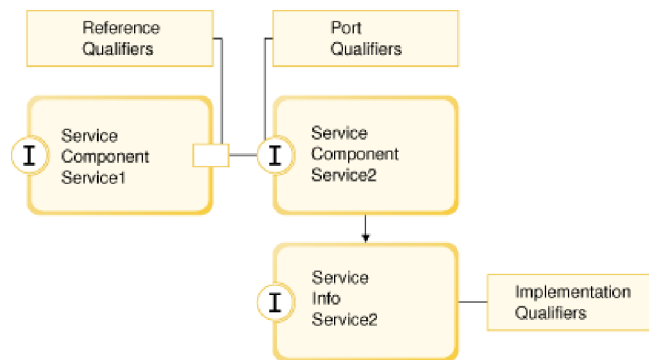
## Service qualifiers

An application communicates its quality of service (QoS) needs to the runtime environment by specifying *service qualifiers*. They govern the interaction between a service client and a target service.

Qualifiers can be specified on service component references, interfaces, and implementations. Since declaration of the QoS values are external to an implementation, you can change these values without changing the implementation, or set them differently when several instances of the same implementation are used in different contexts.

These are the categories of qualifiers:

- Transaction - rules for the type of transaction
- Activity session - rules for joining the active session
- Security - rules for permission
- Asynchronous reliability - rules for asynchronous message delivery



---

## Modules

A *module* is a unit of deployment that determines which artifacts are packaged together in an Enterprise Archive (EAR) file. Components within a module are collocated for performance, and can pass their data by reference. A module can be seen as a scoping mechanism; that is, it sets an organizational boundary for artifacts.

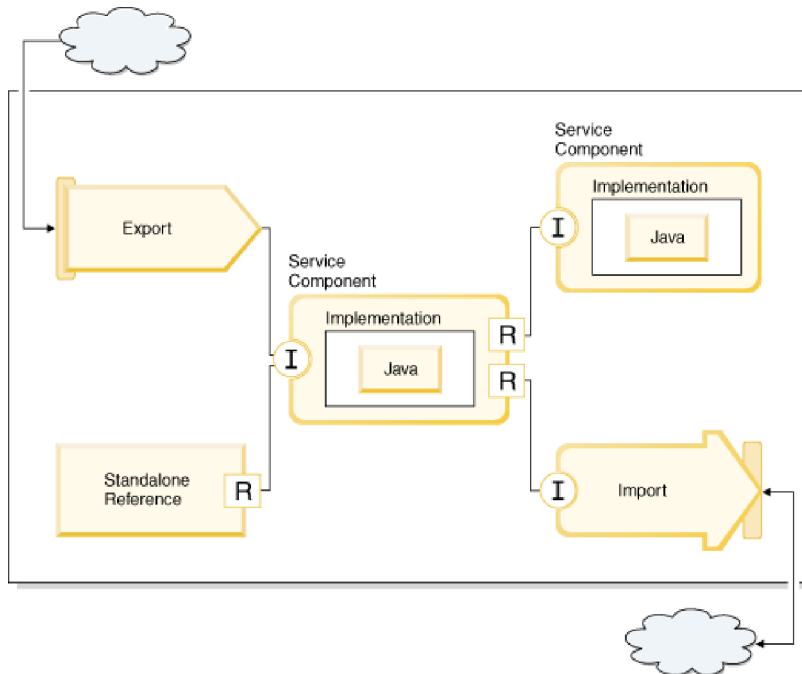
A module is a composite of service components, imports and exports. The service components, imports and exports reside in the same project and root folder, which also contain the wiring that links the components and the bindings needed for the imports and exports. A module may also contain the implementations and interfaces referenced by its components, imports and exports, or these may be placed in other projects, such as a library project.

There are two types of modules. First, a module called *module* (sometimes referred to as a business integration module) that contains a choice of many component types, often used to support a business process. Second, a module called a *mediation module*, which contains up to one component, a mediation flow component, plus zero or more Java components that augment the mediation flow component.

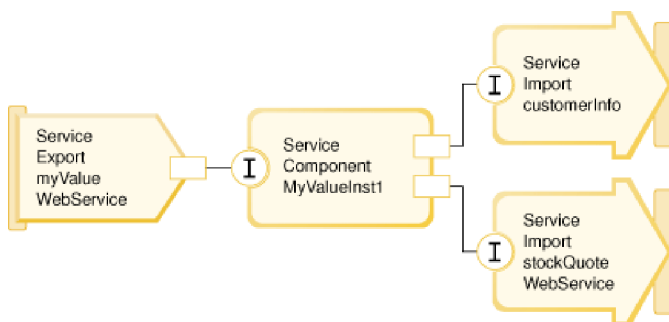
Why are there two module types? The first type of module is primarily designed for business processes. A mediation module is like a gateway to existing external services, which is common in enterprise service bus architectures. These external services or exports are accessed in a mediation module by imports or service providers. By decoupling client service requesters from service providers by a mediation flow, your applications gain flexibility and resilience, a goal of service-oriented architecture. For example, your mediation flow can log incoming messages, route messages to a specific service determined at run time or transform data to make it suitable to pass to another service. These functions can be added and changed over time without modifying the requester or provider services.

A module results in a service application tested and deployed to the WebSphere Process Server. A mediation module results in a service application tested and deployed to either the WebSphere Process Server or the WebSphere Enterprise Service Bus server. Both types of modules support imports and exports.

Implementations, interfaces, business objects, business object maps, roles, relationships and other artifacts often need to be shared among modules. A *library* is a project used to store these shared resources.



In the diagram that follows, the module contains an export, two imports and a service component that uses them. Wiring is shown linking the interfaces and references.



Module and mediation module artifacts include:

- Module definition - defines the module.
- Service components - definitions of the services in the module. A service component name inside a module is unique. However, a service component can have an arbitrary display name, which is typically a name more useful to a user.
- Imports - definitions of imports, which are calls to services external to this module. Imports have bindings, which are discussed in the Imports and exports section.
- Exports - definitions of exports, which are used to expose components to callers that are external to this module. Exports have bindings, which are discussed in the Imports and exports section.



- References - references from one component to another in the module.
- Stand-alone references - references applications that are not defined as Service Component Architecture components (for example, JavaServer Pages), which enable these applications to interact with Service Component Architecture components. There can be only one stand-alone references artifact per module.
- Other artifacts - these artifacts include WSDL files, Java classes, XSD files, BPEL processes, and so on.

---

## Imports and exports

Imports and exports define a module's external interfaces or access points. *Imports* identify services outside of a module, so they can be called from within the module. *Exports* allow components to provide their services to external clients.

There are two levels of imports and exports. An application on an Enterprise Information System (EIS) can be brought into a module as an import or an export component; this level is a system level import. A system level import lets your application access applications on EIS systems as if they were local components.

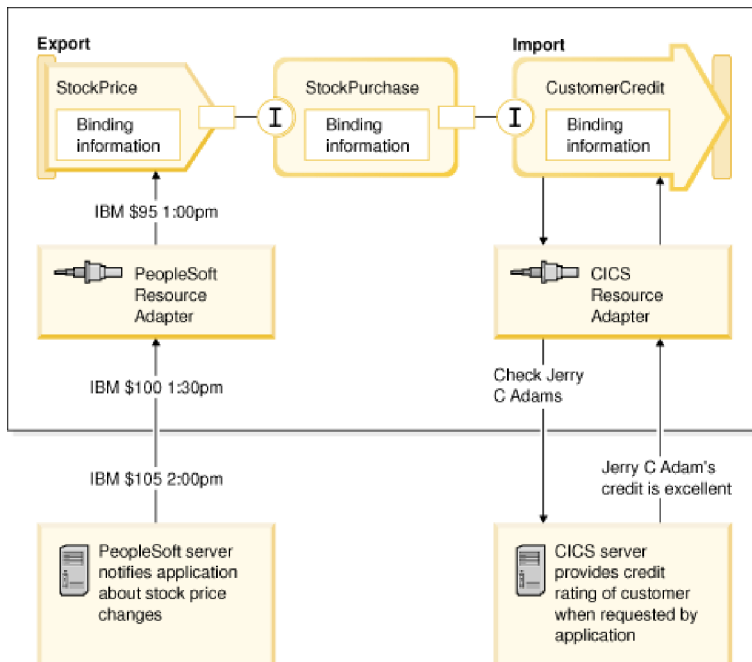
There is also a module level import and export, which lets modules access other modules.

## Accessing EIS systems

*Enterprise Information Systems* (EIS) are large systems containing applications and data, and which often reside on mainframe computers. An application on an EIS system can be imported into a module and represented as an import service component. An application can also be represented as an export component if the application on the EIS system is invoking a transaction on a service in the module.

The enterprise service discovery wizard discovers EIS systems and their applications and data, and lets you import these applications into your application as an import component. The wizard generates the mapping and marshalling code required to map SOAP requests and responses to methods and data structures on the EIS systems. From your own integrated application's perspective, the application on the EIS system appears and behaves as a local component. Services may invoke transactions on it as they do other components. The wizard can also create an export component, where the application on the EIS system invokes a transaction on a service in the module. With an export component, the initiation takes place in the external EIS system.

In the diagram that follows, an export component notifies the StockPurchase component of the changing value of the IBM stock price. The export component was created from an application on a PeopleSoft server. An import component is invoked by the StockPurchase component to get the credit rating of customers buying stock. It is a two-way or *request-response* interaction. The import component was created from an application at a CICS server.

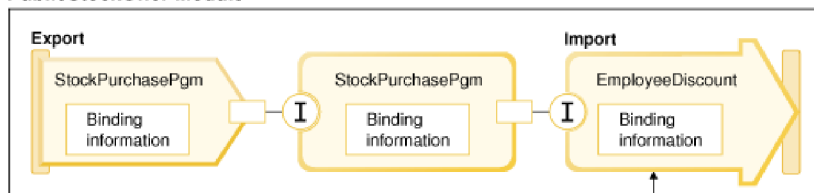


## Accessing other modules

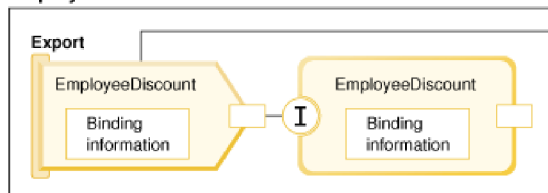
Importing and exporting can take place between modules. If you divide your application amongst a set of modules, each containing a logically-related group of components, you can use importing and exporting to tie the modules to one another for the finished application.

Importing and exporting at the module level is similar to importing and exporting from external applications located on EIS systems. In the diagram that follows, the EmployeeDiscount export component in the EmployeeStockOffer module becomes the EmployeeDiscount import component in the PublicStockOffer module. The StockPurchasePgm export component is the export component representing the StockPurchasePgm component in the PublicStockOffer module.

### PublicStockOffer Module



### EmployeeStockOffer Module



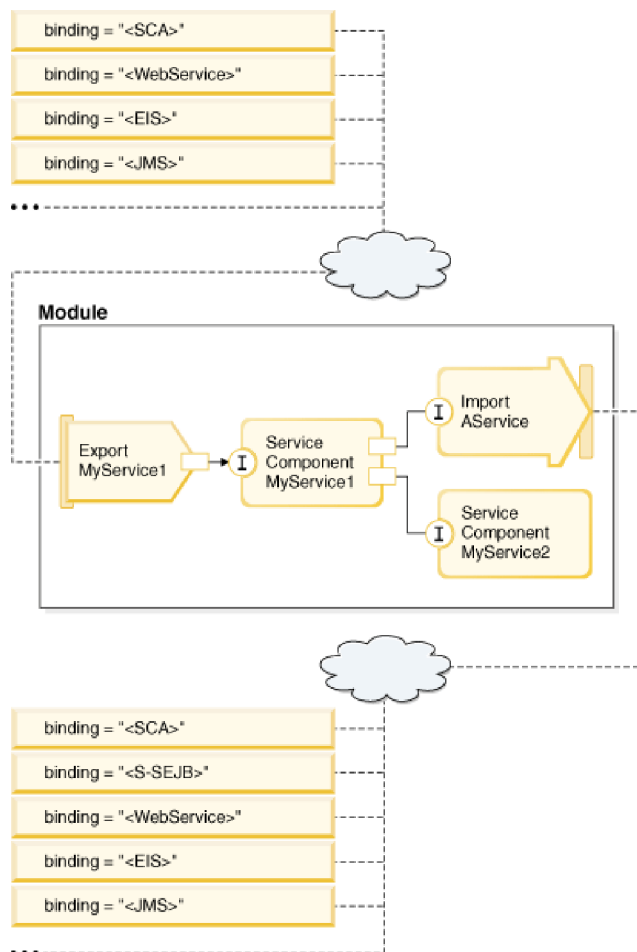
## Service import and export binding types

Imports and exports require binding information, which specifies the means of transporting the data from the modules. An *import binding* describes the specific way an external service is bound to an import component. An *export binding* describes the specific way a module's services are made available to clients.

At the system level, the enterprise service discovery wizard creates import and export components representing application on EIS systems and creates the bindings on generation of the import or export component. The bindings created are of an EIS or Java Message Service (JMS) type. JMS would typically be used in interactions with large EIS systems where asynchronous communication via a message queues is critical for reliability.

At the module level, the assembly editor lists the bindings supported and simplifies the creation of them when you want to create an import component that accesses another module's export component. The assembly editor likewise assists in the creation of bindings for an export component that makes the module's service available externally. A properties view in the assembly editor displays the binding information of any import or export component.

In addition to the EIS and JMS bindings mentioned previously, there are Web service bindings. A Web service import binding allows you to bind an external Web service to an import component. A Web service export binding allows you to provide a service to external clients as a Web service. The SCA or default binding lets you to communicate with other services in other modules. An import component with an SCA binding lets you access a service in another module. An export component with an SCA binding lets you offer a service to other modules. An import component (though not an export component) may also have a stateless session EJB binding.



## Selecting appropriate bindings

When a certain binding might be more appropriate to your application's needs is discussed in this section.

The bindings available in WebSphere Integration Developer provide a range of choices. This list helps you identify when one type of binding could be more suitable for your application's needs than another type of binding.

Consider an *SCA* binding when these factors are applicable:

- All services are contained in WebSphere Integration Developer modules; that is, there are no external services
- Performance is important
- The modules are tightly coupled.

Consider a *Web Services* binding when these factors are applicable:

- You need to access an external service over the Internet or provide a service over the Internet
- The services are loosely coupled
- The protocol of an external service you are accessing or a service you wish to provide is SOAP/HTTP or JMS/HTTP.

Consider an *EIS* binding when these factors are applicable:

- You need to access a service on an EIS system using a resource adapter
- Performance is more important than reliability; that is, synchronous data transmission is preferred over asynchronous.

Consider a *JMS* binding when these factors are applicable:

- You need to access a messaging system
- The services are loosely coupled
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider a *Stateless Session EJB* binding when these factors are applicable:

- The binding is for an imported service that is itself an EJB
- The imported service is loosely coupled
- The state of the EJB is not important.

---

## Service implementation types

Service implementation types are the implementations of the service components.

The standard implementations of the services are described in this section. These implementations will appear in services in the assembly editor and or within BPEL processes.

### Java objects

An implementation of a component in Java is referred to as a Java object.

One common implementation is a component written in Java. This implementation is sometimes nicknamed a "plain old Java object" or POJO. Generally, this implementation will have a WSDL interface type, though this implementation could also have a Java interface. If there are multiple interfaces specified, then you cannot mix WSDL interfaces with Java interfaces. You can, however, "join" an application created with a set of WSDL interfaces to an application with a set of Java interfaces. A sample listed in the samples gallery of the Welcome view shows you how.

When working with a Java object, the code remains hidden from you within the context of the editors.

A Java object can be used in a mediation module. It can be deployed to either a WebSphere Process Server or a WebSphere Enterprise Service Bus server.

## BPEL process

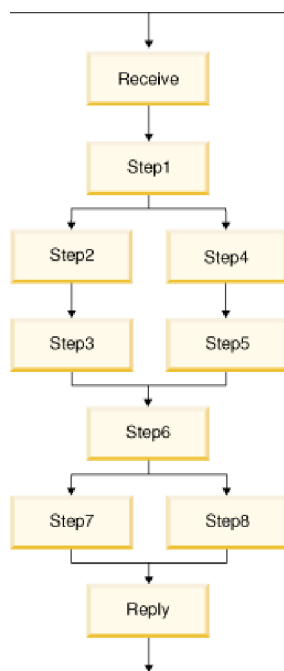
A *BPEL process* component implements a business process.

Its implementation language is the industry standard Business Process Execution Language for Web Services (BPEL4WS) and its IBM extensions. A BPEL process implements a potentially long-running service through the use of more elementary services. A BPEL process created in the process editor can do the following things:

- Describe the orchestration of other services using control flow graphs
- Use variables to keep the process state
- Use sophisticated error handling through fault handling
- Support asynchronous events
- Correlate inbound requests with the right instance of a particular process by using correlation sets to mark that business data within the request that identifies the instance (for example, a customer ID)
- Provide extended transactions through sophisticated compensation support

In addition to these standard BPEL items, WebSphere Integration Developer also extends BPEL to include people into a process with *human task* support. For example, this extension could add to a process the requirement that a person approve a loan.

The process editor uses visual representations of BPEL constructs to build your business process quickly and simply.

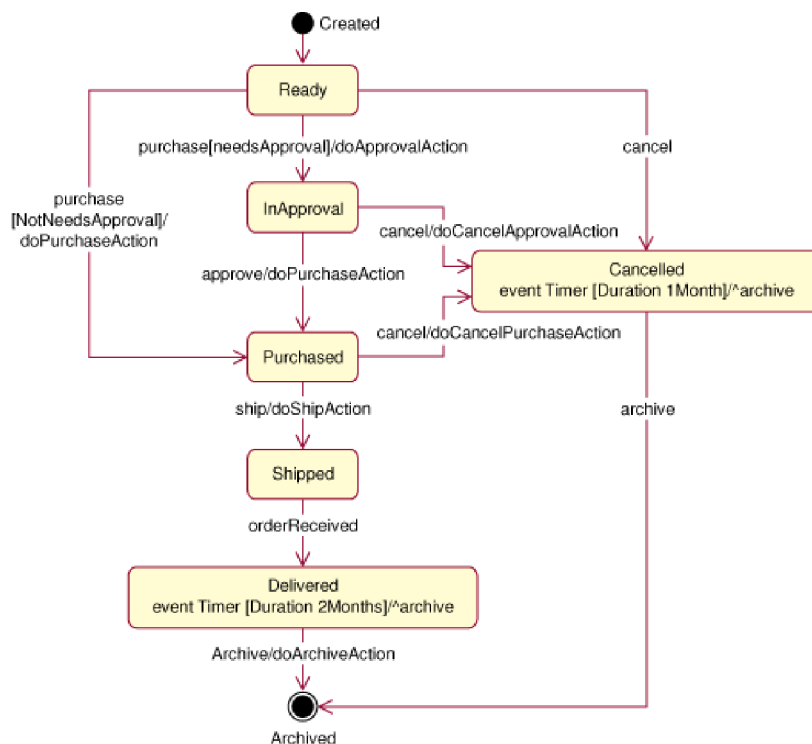


A BPEL process cannot be used in a mediation module. It can only be deployed to a WebSphere Process Server.

## State machines

A state machine is an alternative way of creating a business process. A state machine is suited for processes related to changing states rather than a flow of control. A state defines what an artifact can do at a point in time. A *state machine* is an implementation of this set of states.

State machines are a common way of showing a set of interrelated states in a process. A familiar state machine is a drink dispenser. You put some coins into the machine and along with your drink, which hopefully is dispensed, you get your exact change as the state machine mechanically breaks down the coins that need to be returned to you given the coins you inserted. In the diagram that follows, a typical state machine is shown as created by the state machine editor. In the state machine, an item is purchased and shipped to a customer.



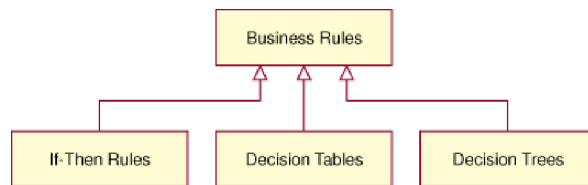
A state machine cannot be used in a mediation module. It can only be deployed to a WebSphere Process Server.

## Business rules

Business rules complement business processes and state machines. If there is condition with a variable, for example, a *business rule* can change the value in that variable at run time. Created by a visual programming language, a business rule makes a decision based on context. The decision can be simple or complex. Business rules are nonprocedural and the rules can be changed independently of an application.

Business rules determine the outcome of a process based on a context. Business rules are used in everyday business situations to make a decision given a specific set of circumstances. This decision may require many rules to cover all the circumstances. Business rules within a business process allow applications to respond quickly to changing business conditions. In an insurance corporation, for example, a business rule for approving car insurance to an applicant could be: *If the applicant is male and over 25 years old, and the car category is sports, and he has been insured with us for the past 5 years, then approve the application for insurance at a fee of \$100 per month.*

WebSphere Integration Developer offers a number of approaches to creating business rules. You can create if-then rules or decision tables, all of which will shape the outcome of your process. Note that these rules are independent of the process itself, meaning that you can change the rules at any time without having to redo your process. For example, based on where your business is located, you might have a rule that says: *If the date is between December 26th and January 1st, then offer a post-holiday sale discount of 20%.* However, should sales continue to be too slow, you could at any time modify the discount to 40%.

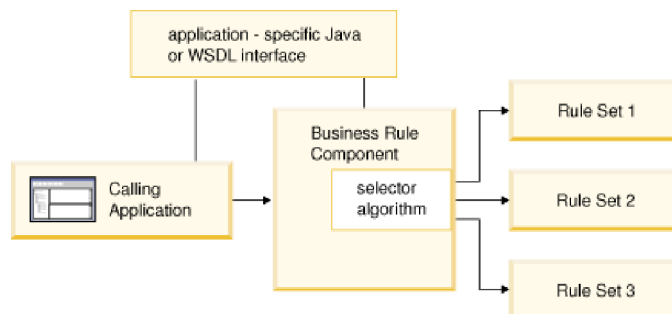


Business rules cannot be used in a mediation module. They can only be deployed to a WebSphere Process Server.

## Selectors

Integrated applications contain many ways to interact. A *selector* is used to route an operation from a client application to one of several possible components for implementation.

Routing to a component is based on dates. For example, here is one route based on a date: *Two weeks before school starts, offer a back-to-school special price on our school-related merchandise.* Businesses may have many such routes based on dates. A selector makes a decision to select one route over another at run time based on a date. For example, if the time is just before school starts, then the previous back-to-school offer would be called. However, if it is the season when school is ending, there could be an offer to prepare children for summer.

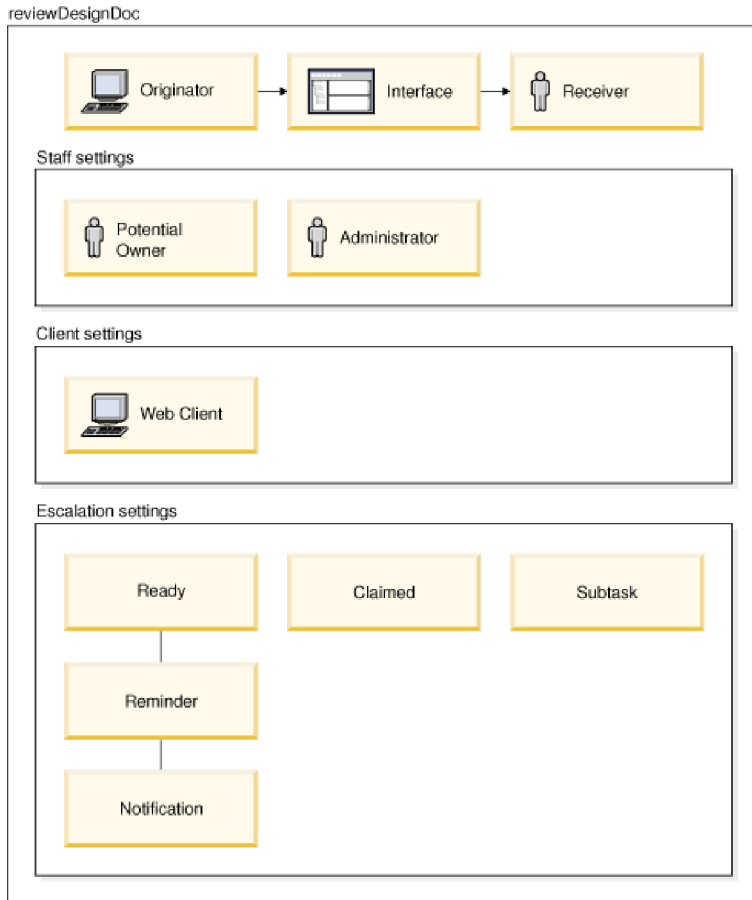


A selector cannot be used in a mediation module. It can only be deployed to a WebSphere Process Server.

## Human task

A *human task* component implements a task done by a person. It represents the involvement of a person in a business process.

Occasionally, people need to intervene in a business process. For example, a customer wants to purchase an item that is above his credit limit. A human task lets you intervene and override a business rule that prevents the customer from making the purchase. A human task has several attributes, such as setting the owner of the task, and providing an escalation process in the case that the human specified is not available. The human task component recognizes the reality that many processes require human intervention for tasks like reviewing, researching and approving.



A human task cannot be used in a mediation module. It can only be deployed to a WebSphere Process Server.

## Interface map

An *interface map* resolves differences between the interfaces of interacting components.

Differences between interfaces in components that need to interact with one another are common. These differences arise because in WebSphere Integration Developer you are often assembling components that were created for different applications. Reusing them to create a new application is one of WebSphere Integration Developer's strengths, since otherwise you would be recoding similar components. But you typically must make some adjustments.

For example, two components can have methods that perform basically the same action but have different names such as `getCredit` and `getCreditRating`. They may also have different operation names and the operations may have different parameter types. An interface map maps the operations and parameters of these methods so that the differences are resolved and the two components may interact. An interface map is like a bridge between the interfaces of two components allowing them to be wired together despite differences.

An interface map exists independent of the components using it, which means the components themselves do not need to be changed.

An interface map cannot be used in a mediation module. It can only be deployed to a WebSphere Process Server.

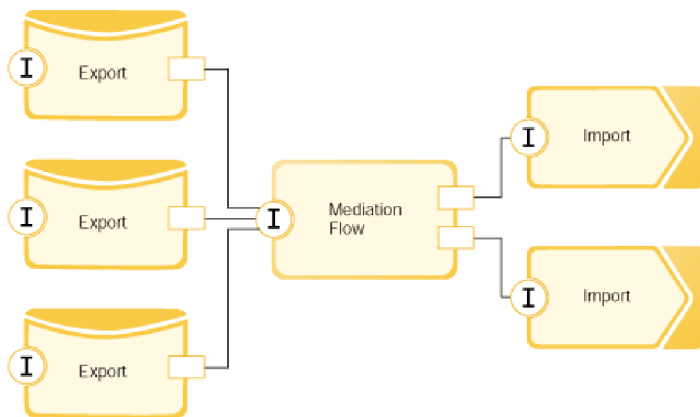


## Mediation flow

*Mediation* is a way of mediating or intervening dynamically between services. A *mediation flow* implements a mediation.

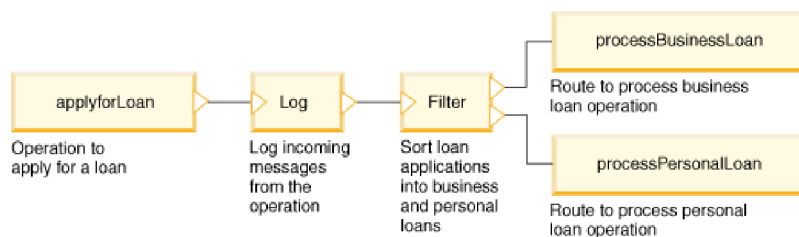
Mediation has several useful functions. For example, you can use mediation when you need to transform data from one service into an acceptable format for a subsequent service. Logging lets you log messages from a service before they are sent to the next service. Routing lets you route data from one service into an appropriate service determined by the mediation flow. A mediation operates independently of the services it connects to. A mediation in the assembly editor appears as a mediation flow component between exports and imports.

In the diagram that follows, three service requesters or exports send their output data to the interface of the mediation flow component. The mediation flow component then routes the appropriate data to two service providers or imports.



A mediation flow is a flow-like construct created with the mediation flow editor. Selecting a mediation flow component in the assembly editor, launches the mediation flow editor. In the mediation flow editor, an operation from one service, the service requester or export, is mapped to the operation of another service, the service provider or import, along with functions provided by the mediation flow editor. These functions are called *mediation primitives* and are wired in a mediation flow as shown in the following diagram. Mediation primitives are IBM-supplied or you can create your own custom primitives. Mediation primitives can act on both message content and message context, where context is binding-specific information such as SOAP or JMS headers, or user-defined properties.

In the diagram that follows an operation, `applyforLoan`, sends a message first to a logging primitive, `Log`, that records the message. `Log` sends the message to the `Filter` primitive, which, depending on the message, routes the message to either a `processBusinessLoan` operation or a `processPersonalLoan` operation.



As discussed in the Modules section, there is a mediation module for a mediation flow component. It can contain up to one mediation flow component plus zero or more Java components that augment the mediation flow component. A mediation module can be deployed to either a WebSphere Process Server or a WebSphere Enterprise Service Bus server.

## Stand-alone references

*Stand-alone references* are references to applications that are not defined as Service Component Architecture components (for example, JavaServer Pages or servlets). Stand-alone references permit these applications to interact Service Component Architecture components.

Stand-alone references have neither an interface nor an implementation (as the implementation is outside the scope of the module). A module can contain no stand-alone references or one stand-alone references artifact. Stand-alone references have the practical value of allowing you to use your existing applications together with Service Component Architecture components created in WebSphere Integration Developer.

Stand-alone references can be used in a mediation module. They can be deployed to either a WebSphere Process Server or a WebSphere Enterprise Service Bus server.

---

## Related information

Several topics related to the architecture are discussed in this section.

These topics provide some additional information associated with the architecture of the product.

- “Developing services to be used with .NET services”
- “Bidirectional support”

## Developing services to be used with .NET services

If you intend to develop services to be used with .NET services, you need to be aware of some special considerations. There are two aspects of these considerations: bringing WSDL files developed in a .NET development environment into the WebSphere Integration Developer development environment and exporting WSDL files from WebSphere Integration Developer so that you can use them with .NET services. The key difference between the two development environments is that the .NET environment uses inline schemas and WebSphere Integration Developer does not. Inline schemas are a way of including the schema within a WSDL file rather than specifying that it be imported as a separate file. Information has been provided in WebSphere Integration Developer to help you with both cases.

Inline schemas in the Creating interfaces section shows how to import WSDL files that contain inline schemas such as .NET WSDL files would have.

Creating proxies to work with .NET services in the Assembling the business service section shows how to publish your services externally in such a way as to work with .NET services. This topic particularly focuses on creating proxies that will work with .NET services.

## Bidirectional support

WebSphere Integration Developer works in a multilingual environment. This means it can display and manipulate data represented in different languages. Several languages that have bidirectional scripts (for example Arabic or Hebrew) are included in this support. Those languages are written from right to left, while numbers and embedded segments of Latin (or Cyrillic or Greek, and so on) text are embedded in this text from left to right.

In Overview of bidirectional script support in IBM WebSphere Integration Developer the support for bidirectional languages is discussed including the configuration required, some specific technical points when using the support and limitations.

---

## Chapter 3. Learning about the tools

WebSphere Integration Developer uses visual tools to create integrated applications based on service-oriented architecture.

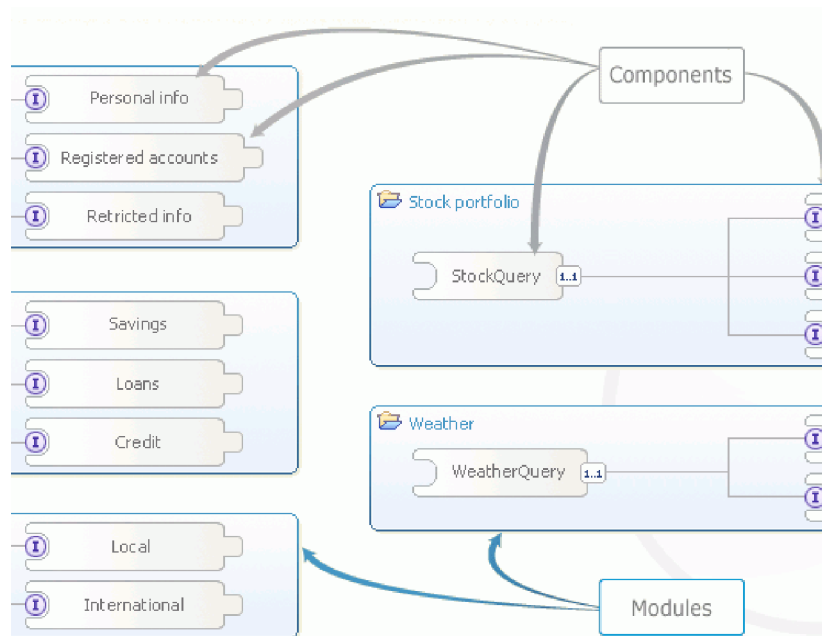
The overview section of the Welcome view provides the highest level of understanding WebSphere Integration Developer's tools. In this case, highest level means the most abstract presentation of the tools. The graphics explain what the tools do. Read the overview section if you are new to the product and want to quickly understand what its tools do. The tutorials section, which can also be found in the Welcome view, lets you watch the tools in action. The samples section lets you start using the tools, but in a fail-safe way. Finally, as you start to develop applications with WebSphere Integration Developer, use the information center to find in-depth information on each tool, including conceptual information, tasks you can perform, and reference information on each tool.

---

### Welcome view overview

The overview section of the Welcome view is the highest level look at the tools and features in the product. Inside the overview, graphics are used to convey what the key tools do in the product.

The overview section of the Welcome view is a way to quickly learn about the product before you start working with it in a hands-on way. Most users that are new to the product should start there. Selecting an icon from the overview launches a graphic on a tool or a topic. For example, the relationship of components and modules is shown in the diagram that follows. You can also launch an audiovisual tour of the product.

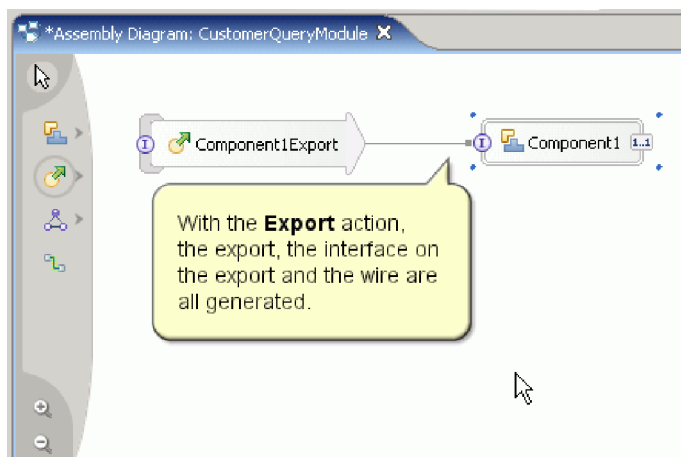


---

### Tutorials in the Welcome view

The tutorials let you see the tools in action in a fail-safe way. You select a tutorial and then watch as it proceeds to perform some tasks as a user would.

Once you understand the product at the conceptual level, you can play with the tools by launching a tutorial. Each tutorial is a movie-like show of a particular tool. In addition to watching the tutorial, bubble text describes what you are seeing. The WebSphere Integration Developer tutorials are found in the Watch and Learn section of the tutorial gallery of the Welcome view.



---

## Samples in the Welcome view

Our samples give you hands-on exposure in the use of WebSphere Integration Developer to develop business service solutions in a fail-safe environment.

The samples can be found in the Samples Gallery of the Welcome view. Selecting a sample gives you several options. You can build it yourself following step-by-step instructions, or you can have it built for you.

The samples vary in scope. A technology sample focuses on a particular tool performing a specific task. An application sample uses several tools to achieve a more complex result. A scenario is a lengthy sample that builds a large application with many tools. Each sample lists the length of time it will take to build.

### Business state machine (simple)

This sample demonstrates how a business state machine can be used to moderate a sales order transaction. Specifically, the state machine emulates an on-line brokerage that manages the selling of a share.

The following tools are used in this application:

- Business state machine editor
- Human task editor

To import the ready-made sample, click the **Import** link below and click **Finish** in the opened wizard. See **Running instructions** to run the imported code.

or

If you want to build the sample for yourself, click **Step-by-step instructions**.

#### Ready-made sample

[Import](#) [Running instructions](#)

#### Build it yourself

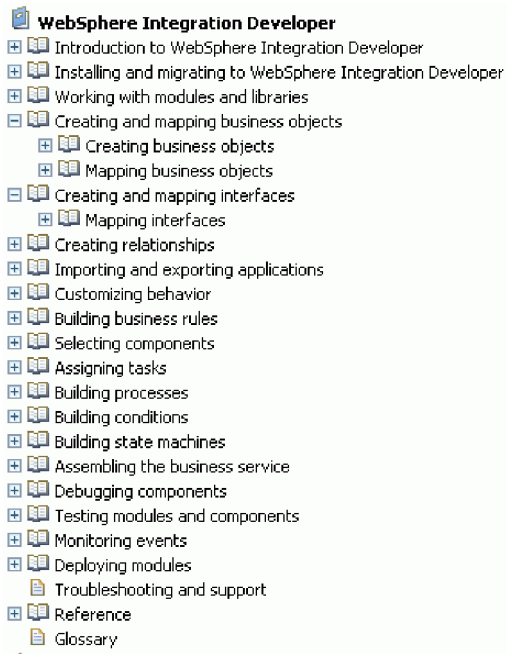
[Step-by-step instructions](#)

---

## Information center

The information center provides complete information for the product. In addition to tutorials and samples, additional conceptual, task, and reference information can be found for each tool for WebSphere Integration Developer.

The navigation of the information center gives you quick access to the information you want. You can also search the entire information center. Select from the search results to see various topics containing in-depth information on the subject, as well as links to related information.





---

## Notices

The XDoclet Documentation included in this IBM® product is used with permission and is covered under the following copyright attribution statement: Copyright (c) 2000-2004, XDoclet Team. All rights reserved.

Portions based on *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Copyright (c) 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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 documentation 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 documentation. The furnishing of this documentation does not give 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 (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, 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 OR CONDITIONS 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.

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:

*Intellectual Property Dept. for Rational Software*  
*IBM Corporation*  
*20 Maguire Road*  
*Lexington, Massachusetts 02421-3112*  
*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 documentation 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 may 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 illustrates 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. 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 IBM's application programming interfaces.

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. 2000, 2005. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## **Programming interface information**

Programming interface information 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**

See <http://www.ibm.com/legal/copytrade.shtml>.







Printed in USA

SC10-4208-00

