IBM Business Process Manager
Version 8 Release 0

Overview of IBM Business Process Manager

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. Some links within the PDF books have been tailored for use in the information centers and may not work properly.

The PDF documentation is available within a quarter after a major release of the information center, such as Version 7.0 or Version 7.5.

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.

Contents

Chapter 2. Learn more about key concepts 177

Chapter 1. Getting started with IBM Business Process Manager

Understand what capabilities IBM® Business Process Manager provides for business process management, and how the various phases of business process management, such as creating and deploying process applications, relate to one another.

The process application is the fundamental container for processes and their components in IBM Business Process Manager. Process designers create process applications in the authoring environments, and might include services, tasks, and artifacts needed to support execution.

Advanced Integration Services are implemented in IBM Integration Designer and associated with process applications. From the Process Center, process applications are deployed to the Process Server, which is the process runtime environment for IBM Business Process Manager.

Similarly, automated processes created in Integration Designer can use human activity flows that have been developed in IBM Process Designer.

Release summary

Become familiar with what's new in IBM Business Process Manager V8.0 and access valuable resources that help you get started with different areas of the product.

What's new

IBM BPM V8.0 provides numerous new product features and various enhancements and improvements to existing product capabilities. See the complete list of new product features and enhanced capabilities for IBM BPM V8.0.

"What's new in IBM Business Process Manager V8.0" on page 14

Improvements

Review the most important improvements and fixes that have been made to existing features in IBM BPM V8.0.

High performance and scalability on z/OS

The zEnterprise environment delivers high performance to its applications by collocating WebSphere applications and transactions managers close to the data (DB2 z/OS) and transactions (CICS, IMS, or WebSphere MQ). The benefits of having applications under z/OS control provide significant value to client applications and businesses. Improvements in the following areas support high performance on z/OS:

- Database consistency in naming conventions
- Thread identity support for configuration-generated data sources
- Removal of Common Event Infrastructure (CEI) database configuration in stand-alone profiles and network deployment scenarios
- Support of the Business Process Choreographer database configuration for stand-alone profiles on databases other than DB2
- Generation of CEI Data Definition Language (DDL) files by the database design tool
- Ability to process SQL and database grouping of SQL files

Considerations for a network deployment configuration

Enhanced WebSphere Customization Toolbox (WCT) support for network deployment installation scenarios

Preceding customizing the environment, z/OS administrators and system programmers can follow a simplified initial installation and configuration process. The simplified configuration is achieved through the following improvements:

- z/OS support for applications and data
- Highlighted and well-documented roles
- Documentation and scripts that install a clustered BPM environment can be used at customer sites
- Fewer tasks that are outlined in installation spreadsheets

Network deployment configuration for z/OS

Improved administrative support to start and stop deployment environments

Using administrative console actions and wsadmin commands, you can now start and stop a clustered topology in the correct order, without errors. The status is communicated while the start or stop operation progresses. When errors are encountered, an improved error-reporting experience offers access to the log files that indicate which cluster or node are affected and the specific error.

Starting and stopping deployment environments

Improved system retry control

System-based automated retries for Service Component Architecture (SCA) asynchronous invocations and long-running BPEL processes are provided to eliminate uncontrollable retries that lead to data-integrity issues. The following capabilities have been added:

- The ability to disable retries for the entire BPM system (SCA, BPC, bindings)
- Declarative ability, at design time, to indicate at operation level where a retry should occur by specifying a retry count and interval
- Administrative capability to adjust declared retries (count and interval) at run time without stop, start, uninstallation, or installation

Controlling system retries

Prerequisite checking during the DB2 Express installation

An improved installation experience helps you find and rectify issues while the embedded DB2 Express installation progresses.

- The password selection matches the operating system-level policy.
- For a typical installation, the default user ID and password that are used for the DB2 Express installation are validated.
- For a custom installation, no default password is specified for the DB2 database user ID. You are required to enter a password and a matching confirmation password for the user ID.
- The installation program verifies whether a BPMINST database instance already exists. If it exists, you are prompted to delete the existing database instance before you continue. An automated check also verifies whether port 50000 is free and prompts you to rectify it if the port is not free.

Preparing to install and configure the software

Interactive uninstallation of DB2 Express

You can now cancel the removal of the embedded DB2 Express database if there are other installations that depend on it. Alternatively, you can choose to completely uninstall the BPMINST database installation as part of the DB2 Express uninstallation.

Uninstalling IBM Business Process Manager interactively

Improved service creation and error handling

An additional level of granularity enables you to differentiate different types and severity of errors and manage exception paths more effectively. Now you can identify, differentiate, and raise declared exceptions as faults when they are defined on an Advanced Integration service (AIS). A catch-all error event identifies errors that are not declared. Improved consistency in error management provides the following benefits:

- Promotes operational responsiveness
- Facilitates problem determination
- Reduces time to recovery
- Increases development productivity

Handling errors in BPDs

Deprecated features

IBM BPM extends capabilities found in previous versions of WebSphere® Integration Developer, WebSphere Lombardi Edition, IBM Business Process Manager, WebSphere Process Server, WebSphere Enterprise Service Bus, and other IBM business process management products.

See a summary of product features that are deprecated and removed in IBM BPM V8.0:

Deprecated and removed features of IBM Business Process Manager

System requirements

Review the system requirements and ensure that they are met before installing each product in the IBM BPM V8.0 suite.

IBM Business Process Manager Advanced
IBM Business Process Manager Advanced detailed system requirements

IBM Business Process Manager Standard
IBM Business Process Manager Standard detailed system requirements

IBM Business Process Manager Express
IBM Business Process Manager Express detailed system requirements

IBM Business Process Manager Tools & Add-Ons
IBM Business Process Manager Tools & Add-Ons detailed system requirements

IBM Integration Designer
IBM Integration Designer detailed system requirements

IBM Business Monitor
IBM Business Monitor and WebSphere Business Monitor detailed system requirements

Release notes

Check the release notes on the Support website for limitations and workarounds.

IBM Business Process Manager Advanced
http://www.ibm.com/support/search.wss?q=ibpma80relnotes

IBM Business Process Manager Standard
http://www.ibm.com/support/search.wss?q=ibpms80relnotes

IBM Business Process Manager Express
http://www.ibm.com/support/search.wss?q=ibpme80relnotes

IBM Integration Designer
http://www.ibm.com/support/search.wss?q=iid80relnotes

IBM Business Monitor
http://www.ibm.com/support/search.wss?q=mon80relnotes

Process Designer
http://www.ibm.com/support/search.wss?q=pd80relnotes

Business Space
http://www.ibm.com/support/search.wss?q=bsp80relnotes

Additional resources

Use the following resources to access the IBM BPM community links and share knowledge and resources.

IBM Business Process Manager Community Wiki
Find and share knowledge about IBM Business Process and Decision Management with other practitioners and users. The IBM Business Process Manager Community wiki is also where you find the WebSphere Lombardi Edition and Lombardi Teamworks community content.

IBM BPM Community

Sample Exchange
Find and share sample applications, toolkits, and other code you can use in your IBM Business Process and Decision Management solutions.

Sample Exchange Home

Service releases

The following interim fixes, including WebSphere Application Server fixes, are available for several products in the Business Process Manager V8.0 suite.

Review the information for your product, and download and install the applicable packages from the following Support page: Required interim fixes for IBM Business Process Management products

Table 1. WebSphere Application Server interim fixes

Product name	Interim fix type	Interim fix name	Release date	Description
IBM WebSphere Application Server 8.0.0 Fix Pack 3 (8.0.0.3)	Required	WAS 8.0.0.3 – Required Interim Fixes	May 18, 2012	Provides required fixes for WebSphere Application Server V8.0.0 Fix Pack 3.
	Optional	PM58769 PM58769 Z-Series	May 18, 2012	Corrects a possible performance downgrade when the *java.util.logging.Logger.isLoggable* method is called.
	Optional	PM62434	May 18, 2012	Fixes the CeaNotificationConsumer.wsdl issue without Internet access.
	Optional	PM63049	May 18, 2012	Corrects an out-of-memory error while creating profiles on Solaris non-English operating systems.
	Optional	PM63098 Z-Series	May 18, 2012	Fixes an *UnsupportedOperationException* during the HTTP servlet filter on the z/OS operating system.
	Optional	PM63369 PM63369 Z-Series	May 18, 2012	Corrects the following issues: • J2CCommandHelper cannot find nested Java™ archive (JAR) files in .RAR format. • Normal polling does not start after recovery for a FlatFile resource.
	Optional	PM63963	May 18, 2012	Increases the maximum heap size in *manageprofiles* to prevent an out-of-memory error.

Table 2. Interim fixes per business process management product

Product name	Interim fix type	Interim fix name	Release date	Description
IBM Business Process Manager Standard V8.0	Recommended	Standard – JR42732	May 18, 2012	Corrects an SQL exception during a server startup for the stand-alone profile, when DB2 version 9.5 is used.
IBM Business Process Manager Express V8.0	Recommended	Express – JR42732	May 18, 2012	Corrects an SQL exception during a server startup for the stand-alone profile, when DB2 version 9.5 is used.

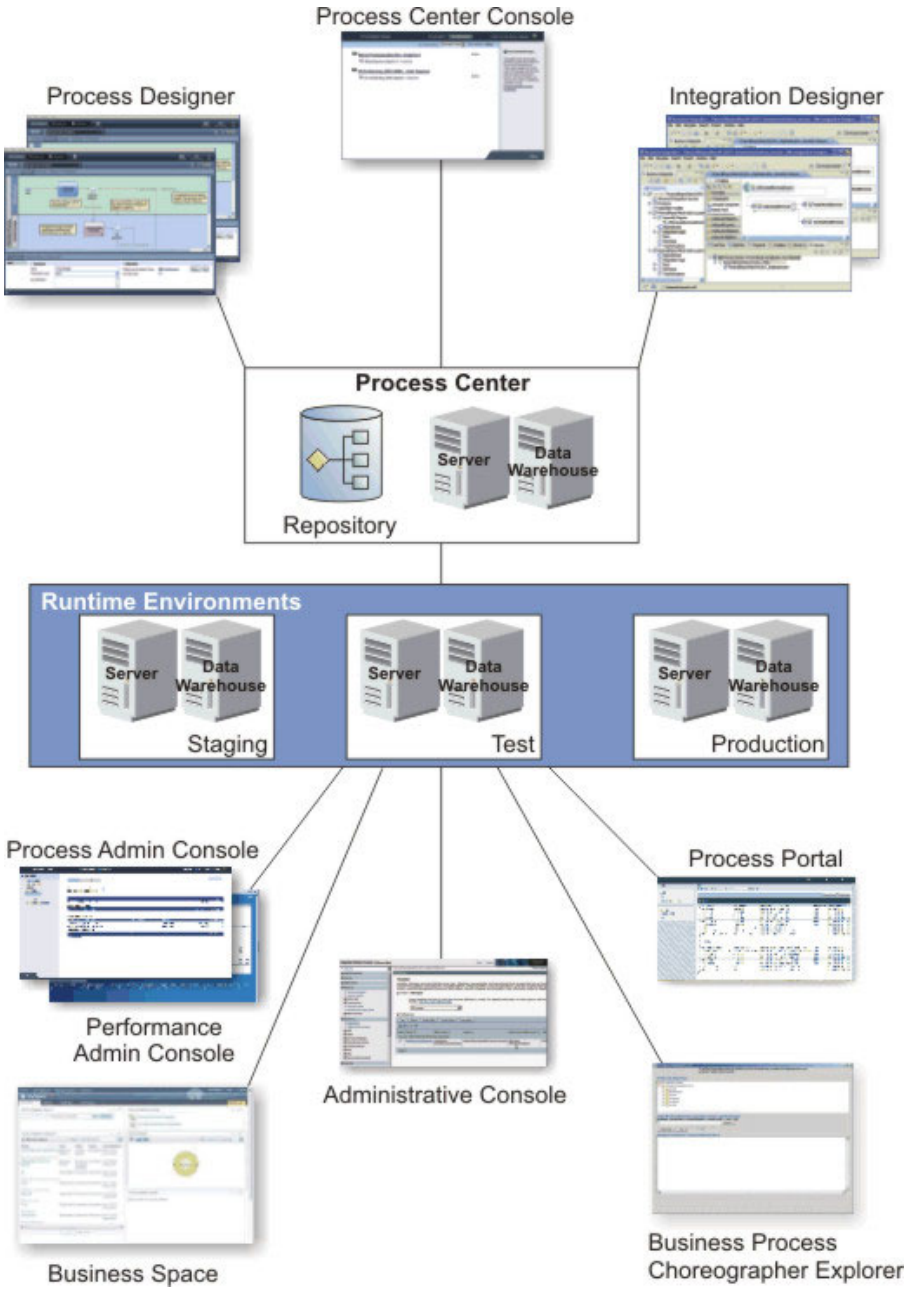
Table 2. Interim fixes per business process management product (continued)

Product name	Interim fix type	Interim fix name	Release date	Description
IBM Business Process Manager Advanced V8.0	Recommended	Advanced – JR42732	May 18, 2012	Corrects an SQL exception during a server startup for the stand-alone profile, when DB2 version 9.5 is used.
	Recommended	Advanced – JR42736	May 18, 2012	Fixes the authentication data aliases for the SCA and CEI Messaging engines that are missing for the MSSQL Server. Corequisite with JR42749.
	Recommended	Advanced – JR42749	May 18, 2012	Fixes the authentication data aliases for the SCA and CEI Messaging engines that are missing for the MSSQL Server. Corequisite with JR42736.
IBM Business Process Manager Advanced – Process Server V8.0	Recommended	Advanced – JR42732	May 18, 2012	Corrects an SQL exception during a server startup for the stand-alone profile, when DB2 version 9.5 is used.
	Recommended	Advanced – JR42736	May 18, 2012	Fixes the authentication data aliases for the SCA and CEI Messaging engines that are missing for the MSSQL Server. Corequisite with JR42749.
	Recommended	Advanced – JR42749	May 18, 2012	Fixes the authentication data aliases for the SCA and CEI Messaging engines that are missing for the MSSQL Server. Corequisite with JR42736.

Product overview

The components of IBM Business Process Manager provide a unified BPM repository; tooling for authors, administrators, and users; and a runtime platform. The product can be configured to support various levels of complexity and involvement with business process management.

The following diagram illustrates a typical IBM Business Process Manager configuration:



- From the IBM Process Designer and IBM Integration Designer authoring environments, developers connect to Process Center. From either of these GUI-based application development tools, developers can create, test, debug, and deploy business applications. Choose one tool or the other, depending on the type of application you are developing. There might also be cases where using both tools brings significant advantages.
- In the Process Designer and Integration Designer authoring environments, process and service designers create deployable process applications and reusable toolkits. Process applications contain process models and service implementations, including any required supporting files. They are stored in the Process Center repository, where they can be shared.
- The Process Center includes two servers: The Process Center server and the Performance Data Warehouse. These servers allow developers working in IBM Process Designer to run their process applications and store performance data for testing and playback purposes during development efforts. Performance Data Warehouse retrieves tracked data from the Process Server or Process Center server at regular intervals.

- Process Center also supports a number of administrative functions. From the Process Center Console, administrators install process applications that are ready for staging, testing, or production on the Process Servers. They also manage running instances of process applications in configured environments.
- Install process applications on a Process Server for staging, testing, and production. The runtime environments support Business Process Model and Notation (BPMN) 2.0 processes. IBM Business Process Manager Advanced also supports Business Process Execution Language (BPEL) processes.
- From the Process Admin Console and Performance Admin Console, administrators can manage and maintain all runtime servers. Use the Process Admin Console to manage the Process Center server as well as process servers in your runtime environments. Use the Performance Admin Console to help identify performance bottlenecks and to capture instrumentation data for further analysis.
- Use the administrative console to create and manage objects such as resources, applications, and servers. In addition, use the administrative console to view product messages.
- Use Business Space to create custom business spaces with widgets for monitoring or administering different aspects of your system (for example, monitoring business activities, services, and system health, or administering mediation policies and business calendars). You can also create a business space with the Human Task Management widgets and use it to participate in business processes.
- Using the Process Portal, process participants can connect to the Process Center server or a Process Server in any configured runtime environment, depending on whether a process is being developed, tested, or has been released to a production environment.
- Business Process Execution Language (BPEL) process instances are typically managed in the Business Process Choreographer Explorer or in Business Space.

IBM Business Process Manager V8.0 configurations

Different configurations of IBM Business Process Manager correlate with typical entry points or stages in a company's business process management program.

Table 3. IBM Business Process Manager configurations

Configuration	Phase
Advanced	**Transformation** Complete set of business process management capabilities • Extended support for high-volume process automation • Built-in SOA components for extensive enterprise-wide service integration, orchestration
Standard	**Program** Configured for typical business process management projects • For multi-project improvement programs, with high business involvement • Basic system integration support • Rapid time-to-value and improved user productivity
Express	**Project** Configured for first business process management project • Rapid time-to-value: improved user productivity • Low entry price • Easy installation and configuration

IBM Business Process Manager V8.0 configuration capabilities

Understand what products and capabilities IBM offers for business process management, and choose the right one for your enterprise.

IBM Business Process Manager is a single BPM platform that combines human-centric and integration-centric capabilities into a unified product. Different configurations of the product are available for different users, and satisfy different needs in the enterprise. Product configurations can be combined for collaborative authoring and network-deployed runtime environments.

Table 4. IBM Business Process Manager configuration capabilities

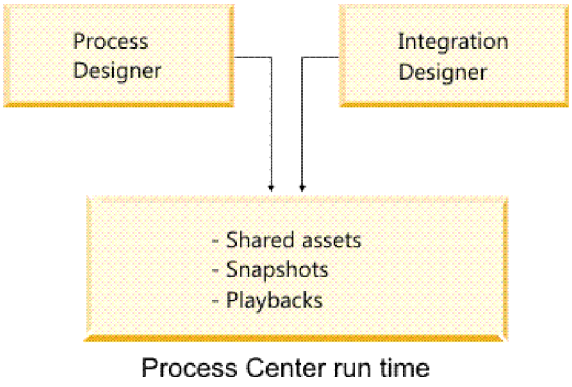
Capability	**Adv.**	**Std.**	**Express®**
WebSphere® Lombardi Edition compatible execution	X	X	X
Process Designer (BPMN)	X	X	X
Collaborative editing / immediate playback	X	X	X
Interactive "process coach" user interfaces	X	X	X
ILOG®-based process rules	X	X	X
Process Portal	X	X	X
Real-time monitoring and reporting	X	X	X
Performance analytics & optimizer	X	X	X
Performance Data Warehouse	X	X	X
Process Center / shared asset repository	X	X	X
Unlimited process authors and end-users	X	X	200 users / 3 authors
High availability: clustering and unlimited cores	X	X	• 4 cores production • 2 cores development • No cluster
WebSphere Process Server compatible execution	X		
Integration Designer (BPEL / SOA)	X		
Built-in enterprise service bus (ESB)	X		
Transaction support	X		
Integration adapters	X		
Flexible Business Space user interface	X		
Advanced platform support (Linux on System z®, IBM AIX®, Solaris)	X	X	

The Process Center repository

The Process Center includes a repository for all processes, services, and other assets created in the IBM Business Process Manager authoring environments, Process Designer and Integration Designer.

Process Center is a software component that runs as a server where Process Designer and Integration Designer share assets, in effect letting them develop business processes cooperatively in a highly interactive manner. These business processes can use monitoring points created with the Business Monitor development toolkit. The result is a business process that can be examined at run time for effectiveness under real working conditions.

In the diagram that follows, you see several related components that together let you build complex business processes.



The Process Center console provides the tools that you need to maintain the repository.

- From the Process Center console, you can create process applications and toolkits and grant other users access to those process applications and toolkits.
- In the authoring environments, you can create process models, services, and other assets within process applications.
- The Process Center includes a Process Center server and performance data warehouse, allowing users working in the authoring environments to run processes and store performance data for testing and playback purposes.
- From the Process Center console, administrators install process applications that are ready for testing or production on the process servers in those environments.
- From the Process Center console, administrators manage running instances of process applications in configured environments.

The Process Center console provides a convenient location in which to create and maintain high-level containers such as process applications and toolkits. Administrators who do not actively work in the Designer view can use the Process Center console to provide a framework in which BPM analysts and developers can build their processes and underlying implementations. Another primary task for administrators is managing access to the Process Center repository by setting up the appropriate authorization for users and groups.

Users with appropriate authorization can perform some administrative tasks directly in Process Designer and Integration Designer. For example, a developer with write access to the process application who wants to capture the state of all project assets at a significant stage of development can create a snapshot while working in the Designer view.

Process Server and runtime environments

Process Server provides a single BPM runtime environment that can support a range of business processes, service orchestration, and integration capabilities.

In your authoring environments, the integrated process server within Process Center enables you to run processes as you build them. When you are ready, you can install and run those same processes on the process servers in your runtime environments. The Business Performance Data Warehouse component collects and aggregates process data from processes running on the process servers. You can use this data to improve your business processes.

The Process Admin Console enables you to manage the process servers in your runtime environments, for example, staging, test, production, as well as the process server that is part of the Process Center.

Authoring environments

IBM Business Process Manager Advanced offers two authoring environments. Use IBM Process Designer to efficiently model business processes that involve human tasks. Use IBM Integration Designer to build services that are self-contained or that invoke other existing services such as web services, enterprise resource applications, or applications running in CICS and IMS.

- "Process Designer"
- "Integration Designer"

Process Designer

Process Designer is available in all editions of the product. IBM Business Process Manager Advanced also offers Integration Designer with its associated editors and adapters.

A process is the major unit of logic in IBM Business Process Manager. It is the container for all components of a process definition, including services, activities and gateways; timer, message, and exception events; sequence lines, rules, and variables. When you model a process, you are creating a reusable business process definition (BPD). Both Process Designer and Integration Designer create process models that can contain human tasks.

Process Designer helps you develop business processes. With an easy-to-use graphics-oriented tool, you can create a sequence of actions that compose a business process, and you can redraw that process over time as circumstances change. If one or more activities require access to large backend systems or services that provide data for the business process, for example to get information on customers, you can meet that need using Integration Designer. Using a simple interface, an activity in Process Designer can call a service created in Integration Designer. That service can use mediation flows to transform, route, and enhance data and adapters to get to many backend systems in standard way. In short, Process Designer focuses on the business process and Integration Designer focuses on automated services to complement the business process. See Getting started with IBM Process Designer.

All Process Designer projects are contained in process applications. You store those process applications and associated artifacts in the Process Center repository. Process applications can share assets that have been placed in toolkits.

IBM Business Process Manager provides several user interfaces to enable you to model, implement, simulate, and inspect business processes. You create and manage process applications, toolkits, tracks, and snapshots in the Process Center Console. You can create process models, reports, and simple services in Process Designer. You can run and debug processes in the Inspector. And you can run simulations in the Optimizer.

Process applications developed in Process Designer can at any time be run on the Process Center server or saved to a snapshot and deployed on the Process Server. The same is true of services developed in Integration Designer and associated with process applications.

Integration Designer

Process Designer is available in all editions of the product. IBM Business Process Manager Advanced also offers Integration Designer with its associated editors and adapters.

Integration Designer provides editors and aids to help developers create complex automated processes and services. It is available as a component in IBM Business Process Manager Advanced or as a stand-alone toolset for other uses.

IBM Integration Designer has been designed as a complete integration development environment for those building integrated applications. 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. 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 assemble an integrated application without detailed knowledge of the underlying implementation of each component.

Integration Designer tools are based on a service-oriented architecture. Components are services and an integrated application involving many components is also a service. The services created comply to the leading, industry-wide standards. BPEL processes, which also become components, are similarly created with easy-to-use visual tools that comply to the industry-standard Business Process Execution Language.

In the Integration Designer paradigm, components are assembled in modules. Imports and exports are used to share data between modules. Artifacts placed in a library can be shared among modules.

Modules and libraries can be associated with a process application for use with the Process Center and can be used as services by processes created in Process Designer. In such cases, they can also be deployed with the process application.

Alternatively, modules and libraries can be deployed directly to the test environment or to the Process Server. You can use mediation modules to create mediation flows, which you can deploy to WebSphere Enterprise Service Bus or to the Process Server.

IBM Integration Designer also provides the capability for creating data types and xml maps that can be deployed on the WebSphere DataPower appliance. You can also transfer files to and from WebSphere DataPower.

Administration tools

IBM Business Process Manager includes a set of administration tools to help you accomplish tasks ranging from installing and managing snapshots to administering processes and working with the resources in your IT environment.

Command-line tools

IBM Business Process Manager provides command-line tools, scripting interfaces, and programming interfaces to administer the runtime environment.

- Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks. Using these tools, you can start and stop application servers, check server status, add or remove nodes, and other tasks.
- The WebSphere administrative (wsadmin) scripting program is a non-graphical command interpreter environment that enables you to run administrative options in a scripting language and to submit scripting language programs for execution. It supports the same tasks as the administrative console, as well as many of the Process Center console tasks. The wsadmin tool is intended for production environments and unattended operations.
- Administrative programming interfaces are a set of Java classes and methods under the Java Management Extensions (JMX) specification that provide support for administering Service Component Architecture (SCA) and business objects. Each programming interface includes a description of its purpose, an example that demonstrates how to use the interface or class, and references to the individual method descriptions.

Process Center console

The Process Center console provides a convenient location for users to create and maintain high-level library items such as process applications and toolkits. It helps provide a framework in which BPM analysts and developers can build their processes and underlying implementations. In addition, the Process Center console provides tools for maintaining the repository, including setting up the appropriate authorization for users and groups.

Access the Process Center console through a web browser (for example, http://host:9080/ProcessCenter).

Process Admin Console

The Process Admin Console is used to administer the process servers in your environment, including the users and installed snapshots for each server. In addition, it provides tools to help you manage queues and caches.

The Process Admin Console includes the Process Inspector, a tool to view and manage process instances for process applications that are running on a specific process server.

Access the Process Admin Console through a web browser (for example, http://host:9080/ProcessAdmin).

Business Performance Admin Console

The Business Performance Admin Console includes tools for managing the Performance Data Warehouses in your environment. You can use this tool to manage server queues and monitor server performance.

Access the Business Performance Admin Console through a web browser (for example, http://host:9080/PerformanceAdmin).

WebSphere Application Server administrative console

The administrative console is used to administer applications, services, and other resources at a cell, node, server, or cluster scope. You can use the console with stand-alone servers and with deployment managers that manage all servers in a cell in a networked environment.

If you have installed a stand-alone profile, you have a single node in its own administrative domain, known as a cell. Use the administrative console to manage applications, buses, servers, and resources within that administrative domain. Similarly, if you have installed and configured a network deployment cell, you have a deployment manager node and one or more managed nodes in the same cell. Use the administrative console to manage applications, set up managed nodes in the cell, and monitor and control those nodes and their resources.

Access this console through a web browser (for example, http://host:9043/ibm/console).

Business Process Choreographer Explorer and Business Process Archive Explorer

Depending on your user role, you can use these client interfaces to manage BPEL processes and human tasks created in IBM Integration Designer, work with your assigned tasks, view completed BPEL processes and human tasks that are in an archive database, or delete processes and tasks from the archive.

Business Space powered by WebSphere

Business Space powered by WebSphere is an integrated user experience for business users across the IBM business process management portfolio. Business Space is a browser-based graphical user interface that provides a customizable and collaborative environment for you to monitor and administer common business processes (such as human task flows) and performance indicators. Use it to view and interact with content from various products in the business process management portfolio. For example, a business space might contain widgets from IBM Business Monitor that monitor key performance indicators (KPIs) in your business; it might also contain widgets from Business Process Manager that take action (assigning people to tasks, or adjusting business rules) based on the information in the KPI monitoring widgets.

Business process rules manager

The business process rules manager is a web-based tool that assists the business analyst in browsing and modifying business rule values. The tool is an option of the IBM Process Server that you can select to install at profile creation time or after installing the server.

What's new in IBM Business Process Manager V8.0

IBM Business Process Manager V8.0 brings a new redesigned Process Portal, integration with Enterprise Content Management systems, searching and sharing of content between process centers, enhanced governance capabilities, and various other new features to the IBM Business Process Manager V8.0 product.

Process Portal

The redesigned Process Portal gives process participants a highly collaborative work experience with increased social capabilities. Process Portal now includes the following new features:

- The ability to request help from experts and collaborate with experts and other users in real time to complete work on a task
- The ability to add comments and attach documents to a specific process or task
- One-click subscription to process instances that a user is interested in, providing process-related on-screen notifications and activity updates in the subscribing user's activity stream
- Activity streams that display activity updates, such as task creation and completion, user comments and actions, and notifications that are related to tasks that are owned by a user or related to particular process instances that a user is following
- Enhanced user profile information, including avatars and configuration of notifications

Process Designer

Process design enhancements

The following new features improve the functionality offered to process participants in Process Portal:

- Automatic starting of the next task - You can configure individual activities to start automatically if they are assigned to the same person who is assigned to the previous task. In Process Portal, if the owner of the current task is the same as the owner of the next task, the next task starts automatically when the current task is complete.
- Restricting ad hoc actions by milestone or participant group - You can configure ad hoc actions, also called *user-initiated actions*, to be available for only a particular phase of a process or for a particular user group by restricting the visibility of the associated ad hoc event to a particular swimlane or milestone in the business process definition (BPD).
- Configuring activities for inline completion - You can configure user tasks that involve a simple decision, such as to approve or reject a request or to choose between a set of options, so that the business user can complete it in Process Portal without having to open the Coach for the task. Instead, the user clicks a button or chooses an option with one click.

Create reusable user interfaces and behavior for Coaches

In IBM BPM V8.0, the Coaches are completely redesigned to contain Coach Views. Coach Views are reusable user interfaces that you can create and customize. Coach Views consist of one or more other Coach Views, data bindings, layout information, and behavior. Because Coach Views are reusable, you can create a library of common user interfaces and behavior that you can use to rapidly develop new Coaches.

For greater flexibility in creating a service flow, Coach Views can broadcast boundary events that you use to connect nodes in the service.

To maintain backwards compatibility, Coaches from previous releases are now called *Heritage Coaches*. You can continue to use and maintain existing Heritage Coaches, but use Coach Views when creating user interfaces for services.

BPMN 2.0: Enhanced support for error handling and termination handling

In V8.0, you now have more options when throwing and catching errors using error events in BPDs, subprocesses, and services (including Advanced Integration Services). You can throw a specific error object by selecting a variable, and you can catch specific errors and map the caught error data to a variable. Improved error-handling capabilities include the option to specify an error code and map to an error type on errors thrown from the flow of a BPD or a service using an error end event. When catching errors, you have the option to filter the errors that are caught by selecting an error from a list of all thrown errors for the linked process, subprocess, or service, using error intermediate events. Also, you can map the error data into a variable by selecting a variable that was previously defined. If you are catching specific errors, you can select the error code, map the error data, or both. Models created in earlier versions follow the behavior of the previous version.

For process instances, you have more flexibility in defining the scope of a terminate end event. You can designate whether all activities in the process instance are ended, even the parent processes. In earlier versions, terminating the entire process instance was the only option. The behavior was not visible when you designed models with terminate end events, and the behavior could not be changed. A new check box that terminates the entire process instance is cleared by default in V8.0 for new models. Therefore, the terminate end event ends activities at the level of the process where you add it, including at the level of subprocesses with lower-level activities. For process models that were created in earlier versions and migrated to V8.0, the previous behavior of terminating all activities in the process instance is preserved, unless you clear the check box. Depending on your needs, you can clear or select the check box.

Integration with Enterprise Content Management systems

Enterprise Content Management systems help you manage documents of all types, such as records, images, and web pages. By incorporating the new Enterprise Content Management service into your business processes in IBM Business Process Manager, you can search, view, and store documents on Enterprise Content Management systems.

- You can use Coach controls to quickly build a user interface for listing, viewing, and storing documents.
- Using a graphical user interface, you can create queries to the Enterprise Content Management system without having to know the Content Management Interoperability Service (CMIS) query language syntax.
- Because the Enterprise Content Management integration is based on the industry-standard CMIS interface, IBM Business Process Manager can connect to any Enterprise Content Management product that supports CMIS.

Data visibility

A business object can be identified as a shared business object, making the business object and its values accessible to other instances at run time.

Process Center

Control the installation of process application snapshots with governance processes

You can apply a governance process that provides control over the installation of process application snapshots. When this governance is in place on a process application, all requests made from IBM Process Center to install a snapshot of that process application pass through the

governance process. The process application snapshot is installed on a process server only after the approvals that are defined in that governance process are completed.

You also can create a governance process that reacts to the status change of a snapshot.

Reference links

Process documentation now includes rich text content and reference links so that you can attach links to content sources or other sources. The following examples are possible reference links:

- A website or a wiki page
- A change request that is stored in a change management system
- A test case that is stored in a quality management system
- Artifacts that are managed by Open Services for Lifecycle Collaboration (OSLC)-enabled content providers

This linking capability helps you achieve traceability or provide details about the changes to a business object or service interface.

Compare and copy

Using the new compare and copy feature, you can learn more about your changing business process applications:

- Understand which components are new, updated, or conflicting when comparing a snapshot to the tip of a track.
- Compare library items in a process application snapshot to the library items on the tip of a track.
- Select changed components from one snapshot and copy them to the tip of a track that has associated dependencies.
- Include an option to create a track when a process application is being imported.

Searching and sharing of content between process centers

You can now find assets, such as toolkits, process applications, services, or business objects faster using specific syntax or tags using the Search field.

- Search for process applications, toolkits, and library items based upon specified keywords.
- Filter results by type using the directed search capability.
- Administer the Process Center index, which is used to conduct searches on the Process Center repository. The index is automatically created and maintained. You now can manually re-create or update the index. You also can configure the index to be processed automatically.
- Preview individual results and view associated process documentation.
- Register to share process content with another process center. When you register two Process Centers with each other, you can share toolkits with other users or subscribe to toolkits that other users share.
- Share toolkits that provide common or exemplary content.
- Tag key library items as they are released so that subscribers know what to use.
- Publish individual snapshots and notify subscribers that a new version is available.
- Subscribe to shared process content (toolkits) from another Process Center.
- Receive notifications when new versions (snapshots) become available.

Compare snapshots before instances are migrated to identify the potential locations of orphaned tokens

You can now use a policy file to compare snapshots before instances are migrated. Use the file to identify the potential locations of orphaned tokens, tokens that are associated with activities that were removed from a BPD, and specify whether each orphaned token is deleted or moved during instance migration.

Process Server

Enterprise service bus capabilities in IBM Process Server

IBM Business Process Manager Advanced provides the same enterprise service bus capabilities that are available in WebSphere Enterprise Service Bus. Several new features are added to the mediation flow component, and these features are available only when you deploy to IBM Process Server V8.0.

The mediation flow component is updated to include the following new features:

- WebSphere eXtreme Scale primitives

  These mediation flow primitives provide elastic scalability with WebSphere eXtreme Scale, giving you cache content-enhancing connectivity for throttled back ends and large binary data. Key uses include response caching, policy caching, and request persistence. The eXtreme Scale mediation primitives can be used only if WebSphere eXtreme Scale is installed.
- Service invocation-style simplification

  Additional invocation-style options are available to control the invocation style for a service without the need to specify additional parameters and, in general, without the need to consider the invocation style that invoked the mediation flow. The new invocation styles are Async with deferred response, Async with callback, and As target.
- Optimized XSLT transforms

  The XSL Transformation mediation primitive is renamed as the Mapping mediation primitive. You can switch easily between XSLT and Business Object Mapper transformation engines for improved functionality or performance.
- Support for WebSphere Service Registry and Repository V8.0

Installation and configuration

Improved installation

- For a typical or custom installation using embedded IBM DB2® Express, and for a custom installation using a local DB2 database server, the installation catches more problems earlier and provides information about how to fix problems before starting the installation process.
- The default passwords are removed from the custom installation. For a typical installation, the passwords are changed to comply with the password policy on all operating systems.
- When you install Process Server, select **Production** for production use, or **Non-production** to use Process Server only for test, staging, or development. Separate licensing is now available for non-production Process Server use.
- The IBM Support Assistant Data Collector is installed with IBM Business Process Manager so that you can search for information, investigate problems, or submit a problem report to IBM.
- The Interactive Installation and Configuration Guide is a new form used to generate a set of installation and configuration topics that are customized to your precise installation needs. In the Interactive Installation and Configuration Guide form, select the options you need for your installation scenario. As you select each option, the tool automatically removes options that are ruled out by your previous selections. For example, if you indicate that you plan to install the Express configuration, it removes Network Deployment as a potential topology. This form is available in the information center.

Improved database functionality and reliability

- IBM Business Process Manager now supports Oracle Data Guard, a high-availability, disaster-recovery, and data-protection mechanism that is used to create, manage, and monitor one or more standby databases, ensuring that Oracle databases for production can survive disasters and data corruptions.
- You can now scale your database solution using the IBM DB2 pureScale® feature. Multiple database servers, known as *members*, process incoming database requests; these members operate in a clustered system and share data.

Improved flexibility for database password changes

You can now reconfigure the database password, as needed, after your database configuration is completed. Reconfiguring the database password provides flexibility if new users take on the database administrator role, or if your company has a policy of changing passwords regularly.

Integration Designer

Increased visibility and control over system retries for SCA components in your application

You now have more control to design the system retries that you plan for your runtime environment. You can set the retry count on the properties page of your module, or you can change the retry count for more than one module using the Configure Asynchronous Retry Count wizard. When a system error occurs, asynchronous invocations retry until the specified retry count is reached. In earlier versions, modules were created with a retry count of 4. Now new modules are created with a retry count of zero. Modules from earlier versions keep existing retry settings during migration.

Unlike in previous versions, the retry behavior of mediation primitives overrides the asynchronous retry count, even if you do not specify retries. Before this release, the retry logic of these primitives was not integrated with the underlying asynchronous retry logic, so that retries could have happened when you defined no retries, or retries could have happened from both the mediation primitive and the service integration bus destination at the same time. Now the behavior defined in these mediation primitives is honored and overrides the service integration bus destination retry logic.

Because the mediation primitive overrides the asynchronous retry count, failover situations, such as when you have an issue with an application server or a messaging engine, might cause failed event manager messages. In previous releases, it was possible that these messages were handled by the service integration bus destination.

Generate a business object map for increased speed and flexibility

You can now set the map generation type to generate a business object map in addition to an XSLT map.

Improved error handling with support for faults in Advanced Integration Services

An interface operation with faults is supported in an Advanced Integration Service.

Accessibility in IBM Business Process Manager

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use information technology products successfully.

IBM strives to provide products that are accessible to most people, regardless of age or ability. Use assistive technologies, such as screen-reader software and digital speech synthesizer, to use what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.

You can operate features using the keyboard instead of the mouse.

You can customize display attributes such as color, contrast, and font size.

You can magnify information presented in the graphical views for more detail.

A U.S. Section 508 Voluntary Product Accessibility Template (VPAT) can be requested on the IBM web site at http://www.ibm.com/able/product_accessibility/index.html.

The information center documentation includes the following additional features to aid accessibility:

- The documentation is available in HTML formats to help users apply screen-reader software technology.
- Images in the documentation are provided with alternative text so that users with vision impairments can use the contents of the images.

Availability of national languages in IBM Business Process Manager

IBM Business Process Manager supports the following languages. Documentation may not be fully translated.

- Chinese-Simplified
- Chinese-Traditional
- Czech
- English U.S.
- French
- German
- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Portuguese-Brazilian
- Russian
- Spanish

IBM Business Process Manager provides partial support for the following languages. Documentation may not be fully translated.

- Arabic (translated for BPEL human task widgets, Business Process Choreographer Explorer widgets, monitor widgets, and Business Space framework)
- Danish (translated for Business Space monitor widgets)
- Dutch (translated for Process Designer, Process Center, and Business Space framework)
- Finnish (translated for Business Space monitor widgets)
- Greek (translated for Process Designer, Process Center, and Business Space)
- Hebrew (translated for BPEL human tasks, Business Process Choreographer Explorer, and Business Space framework)
- Norwegian (translated for Business Space monitor widgets)
- Portuguese-Portugal (Process Designer, Process Center)
- Romanian (translated for runtime operations)
- Slovak (translated for Business Space)
- Swedish (translated for Business Space monitor widgets)
- Turkish (translated for Business Space)

  Note: For the Turkish locale, you must set the **case-insensitive-security-cache** entry in the `60Database.xml` file to **false** to allow usernames and passwords to contain the letter "**i**" (for example, **tw_admin**). The `60Database.xml` file is located in the *install_root*`\profiles\`*profile_name*`\config\cells\`*cell_name*`\nodes\`*node_name*`\servers\`*server_name*`\process-center\config\system\` directory.

  Important: For the Turkish locale, you must invoke the standalone Profile Management Tool to avoid errors. Do not invoke the Profile Management Tool from the Installation Manager.

IBM Business Process Manager provides support for users to enter bidirectional strings in the Process Designer environment, in coaches, and in Process Portal. It provides JavaScript APIs for bidirectional language test manipulation.

Coaches and Process Portal support the use of Hebrew and Arabic calendars.

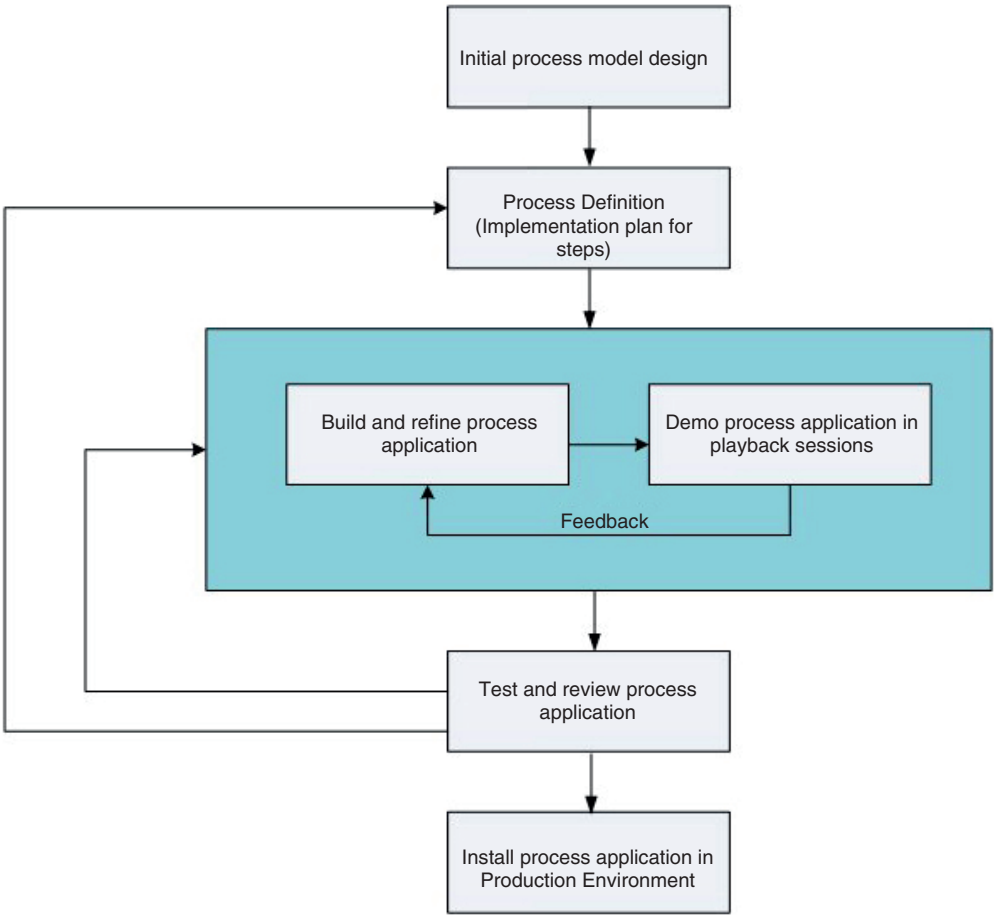
Business process management overview

When developing processes in Process Designer, you need to plan for the eventual installation of your process applications on servers in your test and production environments.

Process Designer can be found in IBM Business Process Manager Express, IBM Business Process Manager Standard and IBM Business Process Manager Advanced. In this section we will focus on the Advanced version, which is designed for high-volume automation and use of complex services developed in Integration Designer. It has built-in SOA components that can be used for extensive enterprise-wide service integration. The Standard version can be used by many business professionals cooperatively to develop multiple sophisticated processes. It has basic system integration. The Express version is for a small number of users on a single server who are getting started in business processes or do not need access to many external systems.

The following diagram shows the lifecycle of a typical process development effort. It includes steps for building and refining an installation service so that you can install your process applications in the production environment.

As this diagram shows, you can work exclusively in your development environment. But you need to configure Process Servers for both your test and production environments.



Overview of process modeling

A process is the major unit of logic in IBM Business Process Manager. It is the container for all components of a process definition, including services, activities and gateways; timer, message, and exception events; sequence lines, rules, and variables. When you model a process, you are creating a reusable Business Process Definition (BPD).

Process components enable you to define the process workflow for end users, creating logic inside a process and integrating with other applications and data sources. To understand what occurs inside a process at run time, it is important to understand the components that make up a process at design time.

Building processes in IBM BPM

Many different individuals from various organizations are usually involved in developing processes using IBM BPM. The overriding concern is to ensure that you are building the best possible solution for meeting the stated goals of your project. To ensure successful results, team members should work together to capture process requirements and iteratively develop the model and its implementations.

Reusing items in Process Designer

Process Designer enables process developers to reuse existing items both within and across process applications. For example, if you know several services already exist that include Coaches and other shared items that you and other developers need, you can access and reuse those items by including them in a toolkit. Then, from your process application, you can add a dependency to the toolkit in which

the shared items reside. This enables you to pick one of the existing services when choosing the implementation for an activity. The items in the toolkit can also be used by other developers working in different process applications.

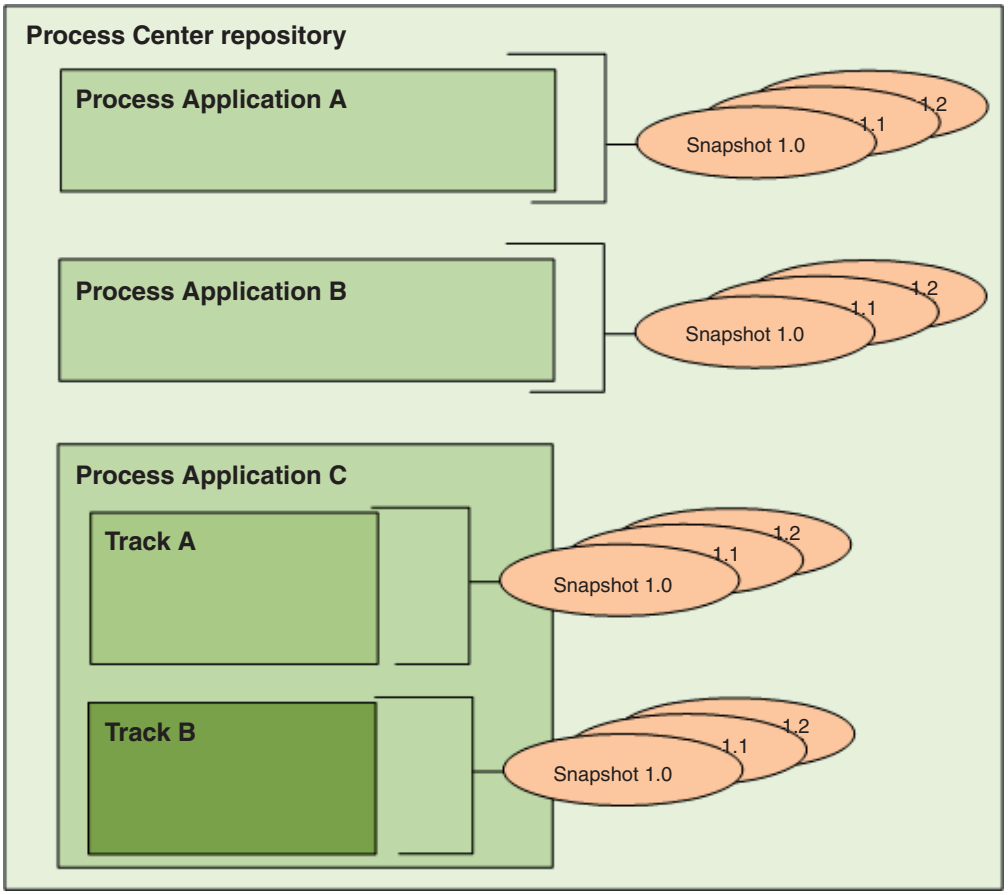
Using the Designer in IBM Process Designer

The Designer interface provides the tools that you need to model your processes in IBM BPM.

Process development with the Process Center

IBM Process Center serves as a central repository for all project assets created in Process Designer. When multiple Process Designer clients connect to the Process Center, users can share items, such as processes and services, and can also see changes made by other users as they happen. The Process Center can also be used as a repository for assets created in IBM Integration Designer.

When you are developing processes in Process Designer, there is a hierarchy available in the Process Center repository which is designed to help you manage your projects. The following figure provides a conceptual overview of the repository hierarchy:



As you can see from the preceding diagram, the Process Center repository includes the following artifacts:

Content Type	Description
Process Applications	Containers for the process models and supporting implementations that BPM analysts and developers create in the Designer in IBM Process Designer.

Content Type	Description
Tracks	Optional subdivisions in a process application based on team tasks or process application versions. When enabled, tracks allow parallel development to occur. Administrators determine if additional tracks are necessary and, thus, enabled for each process application.
Snapshots	Record the state of the items within a process application or track at a specific point in time. Usually snapshots represent a milestone or are used for playbacks or installations. You can compare snapshots and revert to previous snapshots. If an administrator enables tracks for a process application, a snapshot is used as the basis for a new track.

Process applications: Overview

A process application is a container for process models and their supporting implementations; it is stored in the repository. After the artifacts have been authored or otherwise created, they are assembled into a process application.

Process applications contain some or all of the following artifacts:

- One or more process models, also called Business Process Definitions (BPDs)
- References to toolkits
- The services required to implement activities or integrate with other systems, including Advanced Integration Services
- One or more tracks
- Service Component Architecture (SCA) modules and libraries (authored in IBM Integration Designer)
- An IBM Business Monitor model for monitoring business performance
- Any other items required to run the process

Process application tip, snapshots, and tracks

Any changes you make to a process application are dynamically saved to the Process Center repository at the tip, which is the current working version of the process application. You can use playback sessions on the tip to instantly test and manage the current working version of the process application.

The process application remains at that tip level until you decide to create a snapshot, which records the state of library items within a process application or track at a specific point in time. Typically, you take a snapshot every time that you are ready to test the integration or want to install the process application on a process center server or a process server for testing, staging, or production.

Note: The tip is a special snapshot; it is the only type of snapshot in which you can change contents, but you can run it only on the Process Center server. You cannot install a tip on a process server.

By default, each process application has a single track, called `Main`. If you want to allow parallel development on a process application, you can create additional tracks. These optional subdivisions in the process application keep changes isolated. For example, imagine your company is in the process of rebranding; during this transition, the current process applications must be maintained while new versions are being developed based on the updated corporate identity. In this situation, one team might be making minor fixes on the current version of a process application (in the Main track) while another team is building a new version of the process application in a separate track.

Toolkits for process applications

Toolkits are containers that store library items (for example, BPDs) for reuse by process applications or other toolkits. Process applications can share library items from one or more toolkits, and toolkits can share library items from other toolkits. If you have access to a toolkit, you can create a dependency on it and use that toolkit's library items in your process application.

Process applications and business level applications

A process application has a business level application (BLA), which acts as a container for the process application and its assets (assets include things like monitor models, SCA modules, toolkits, and libraries). Each process application snapshot has its own BLA. Many of the administration tasks for a snapshot (for example, stopping or starting it on a production server) are done at the level of the BLA, allowing for quicker and simpler administration of the snapshot and all of its assets.

Running and debugging processes

Using the Inspector, individual developers can run processes and services on the Process Center Server or remote run time Process Server.

The Inspector in IBM Process Designer is key to an iterative approach to process development. An entire development team can use the Inspector to demonstrate current process design and implementation in playback sessions. Playback sessions help capture important information from different stakeholders in a process, such as management, end users, and business analysts. Taking an iterative approach to process development ensures that your process applications meet the goals and needs of everyone involved.

The Inspector in IBM Process Designer includes several tools that enable you to complete tasks like the following in each of your configured environments:

Task	Description
Manage instances of processes	When you run a process, you can view all previously run and currently running instances on the IBM Business Process Manager servers in your environment. You can manage running instances by halting and then resuming them, for example. You can also manage previously run instances by filtering or deleting specific records.
Step through and debug a process	For a selected instance, see the currently executing step and then move forward through the process, evaluating process execution step by step. A tree display of the process combined with indicators called tokens in the process diagram make it easy to understand where you are in the process. You also have the advantage of seeing the variables used in each step and their corresponding values (where applicable).

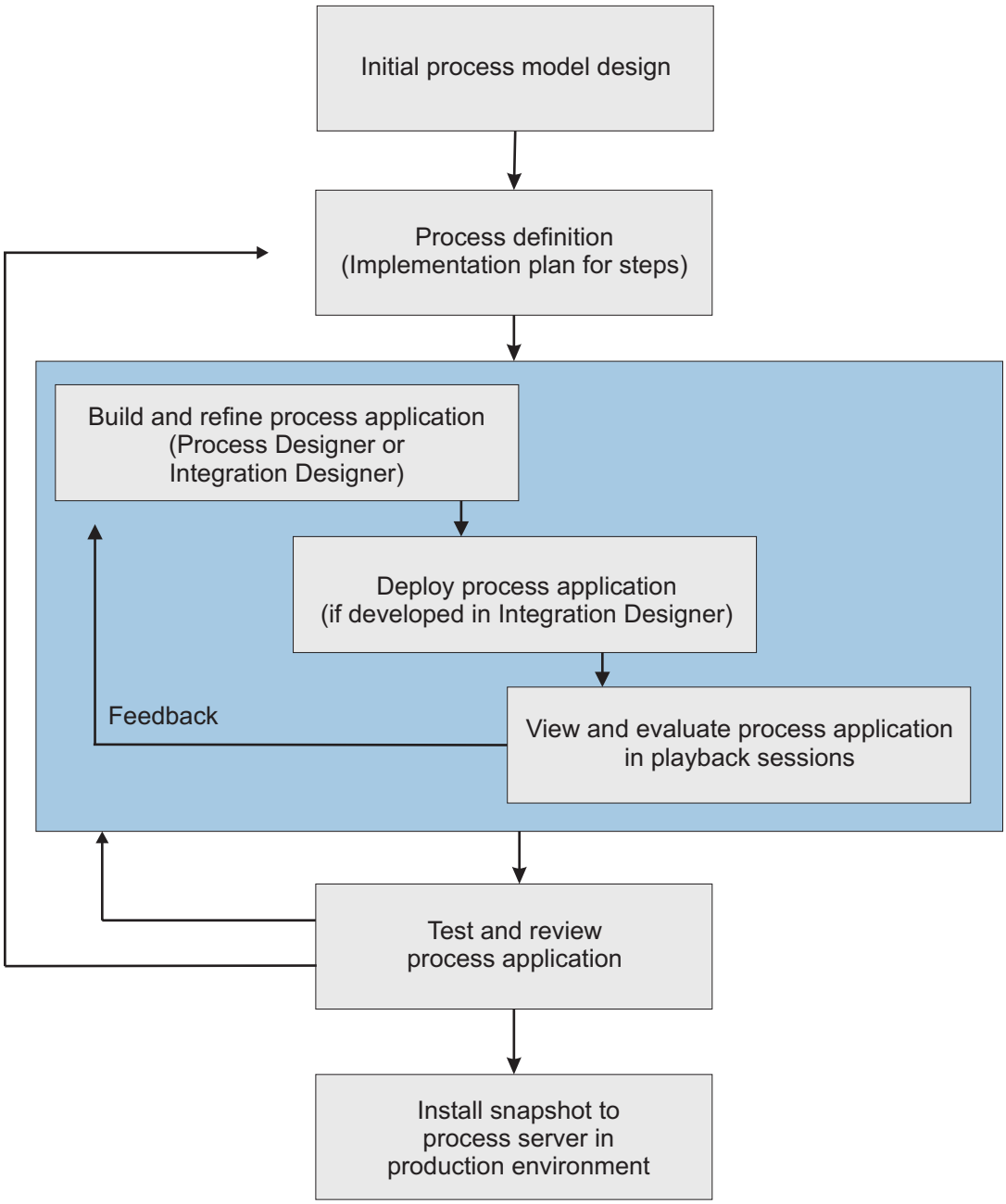
If you are working in IBM Integration Designer, you can use the Inspector if your project is associated with a process application. You also have other debug and test tools available to you. For more information about these Integration Designer tools, see "Testing modules" and "Using the integration debugger for problem determination" in the related links.

Installing and managing process applications

The process application lifecycle includes installing, administering, and undeploying snapshots. Versioning considerations are also part of the lifecycle.

When developing processes, you can take full advantage of the iterative approach supported by the tools within the Process Designer. Processes evolve over time, initially from a development state, and then to testing, and then on to production. Even in production, your processes might continue to evolve because of changing needs. Being prepared for the ongoing lifecycle of your processes is important and will help you design effectively from the outset.

The following figure illustrates an iterative approach to process development.



A typical Business Process Manager configuration includes three environments to support the development and eventual installation of your processes.

Environment	Description
Development	Build and refine your process applications in IBM Process Designer. Create your process models and implement the steps in those models using the Designer. Using the Inspector, demonstrate your development progress in playback sessions so that you can quickly evaluate and refine your prototype. Using the Process Center console, install your process applications on test or production process servers.
Test	Using the Process Center console, install your process applications on the Process Server in your test environment to implement formal quality assurance tests. You can use the Inspector to help verify and resolve issues.

Environment	Description
Production	When all issues reported from formal testing are resolved, use the Process Center console to install your process applications on the Process Server in your production environment. You can use the Inspector to investigate and resolve any issues reported in your production environment.

If you want to test, install, or administer a process application snapshot that contains IBM Business Process Manager Advanced content or an IBM Business Monitor model, the user or group to which you belong must be assigned to the Configurator, Operator *and* Deployer administrative security role. If you are not currently assigned to all of these roles, click **Users and Groups** in the WebSphere administrative console to modify the user or group roles. See "Administrative security roles" in the related links.

Release and installment strategies

To ensure the process applications that you implement and install meet the quality standards of your organization, consider defining a release and installment strategy. When you have identified the goals and requirements for release and installment of new and updated process applications, you can automate the processes required to approve and launch the programs.

For example, you might want to route a process to several different managers across different reporting structures in your organization. Only after each manager signs off on the new or updated process can it be installed in your production environment and rolled out to end users. You can create and implement the steps involved in such a review in IBM Business Process Manager Advanced to ensure that all corporate guidelines have been satisfied and that you have the required signatures. The final step in the review might be notification to the IT team that the approved process application is ready for installation.

Creating, accessing, and incorporating services

Business processes often use services, which provide necessary functions for the business process. These services appear in a business process diagram as an activity or step. For example, a service in Process Designer can invoke an external web service or it can invoke a complex and automated service designed in Integration Designer.

Accessing services external to an application

This scenario discusses different ways to access services that are external to the application, and provides high level tasks for accessing these external services.

Note: This scenario is applicable for WebSphere Enterprise Service Bus and IBM Business Process Manager. Mediation modules can be deployed to WebSphere Enterprise Service Bus and IBM Business Process Manager. Modules can be deployed to IBM Business Process Manager.

In an integrated business application, *business services* interact with each other to provide a required function. A business service performs a repeatable function or task that contributes towards achieving a business goal. But the work of locating a service and connecting to it is not related to the business function. Separating the business function from the task of managing service connections provides flexibility to a solution.

Service interaction begins when a *service requester* sends a request to a *service provider* to perform a business function. This request is sent in the form of a *message*, which defines the function to be performed. The service provider performs the requested function and sends the result in a message to the service requester. Typically, messages need to be processed in order to allow the services to exchange data, and to implement other low level IT functions that are independent of the business functions and data. For example, routing, protocol conversion, transformation, retry of a failed invocation, and dynamic service invocation. This processing is known as *mediation*.

There are two types of modules in IBM Integration Designer; modules (or business integration modules), which are primarily designed to contain business logic (such as business processes, business rules and business state machines), and mediation modules, which implement mediation flows. Although there is some overlap of function between the two types of modules, in general, we recommend that business logic be isolated in business modules and mediation logic be performed by mediation modules.

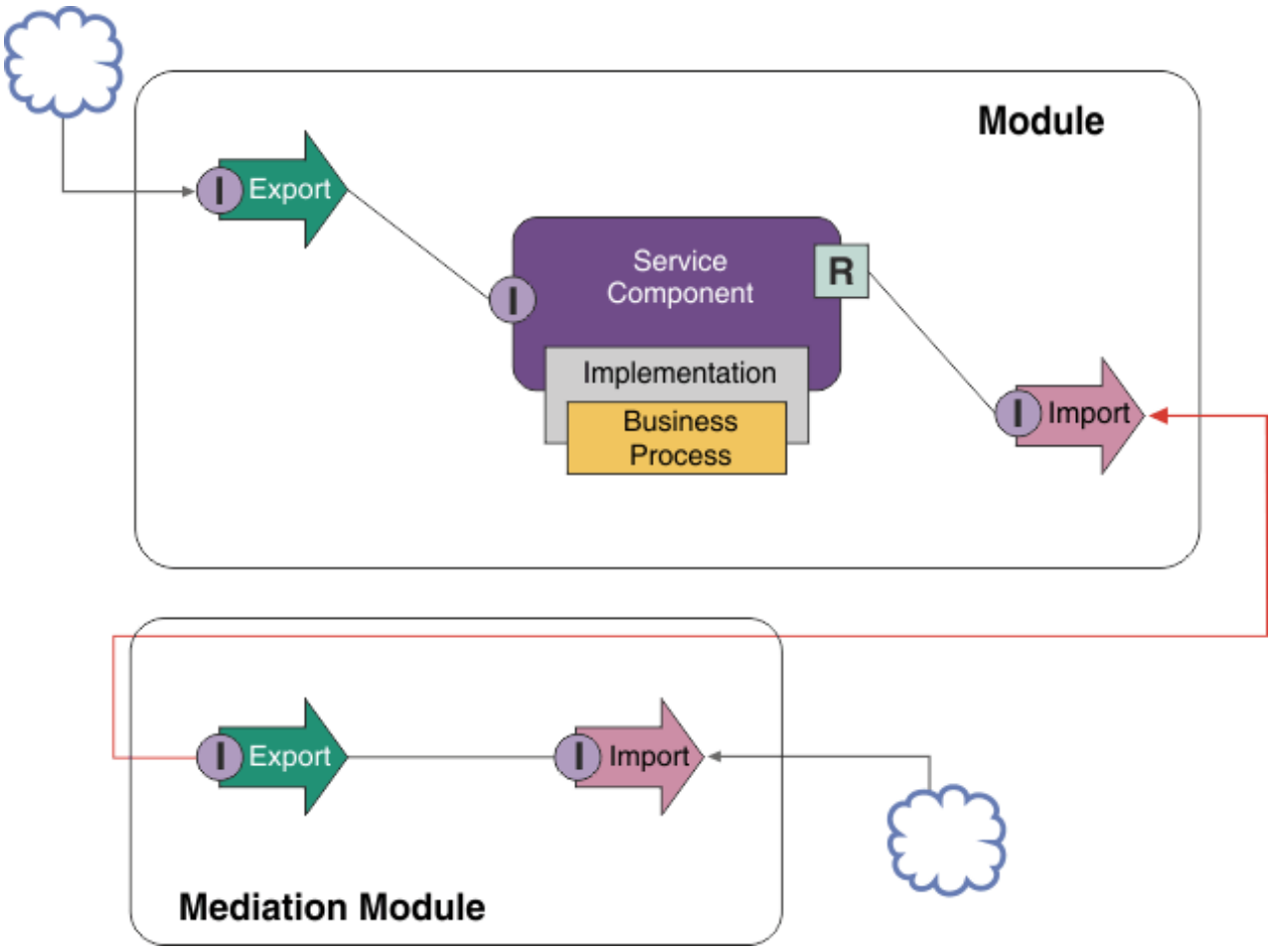
But there is not always a clear separation between business and mediation logic. In these cases, consider the amount of *state*, or data in variables that will need to be processed between service invocations. In general, if there is little or no state processing required, consider using a mediation flow component. If you need to store state between service invocations, or if you have data that will need to be stored in variables and processed, consider using a business process component. For example, if you are calling multiple services and recording the information that is returned from each, so that after invoking all the services, you want to do some further processing with the returned data, use a business process where you can easily assign the returned information to variables. In other words, when you have too much state you have crossed the line into business logic. The following sections help clarify this guidance.

There is no one integration scenario, and there is no technically wrong answer. The guidelines that we discuss here are good practice to enable flexibility and reuse, and are presented for your consideration. As usual, you should carefully consider the benefits and drawbacks of implementing these patterns for your business integration application. Let's consider some situations.

Accessing an SCA component

A basic example of accessing a service is when an import calls another SCA component, without requiring any data transformation. Even in this situation, you could access the external service from a mediation module, rather than accessing it directly from a business module. This would allow flexibility in the future, for changing the service endpoint or qualities of service or governance (for example, adding logging) without impacting the business components that consume the service. This architectural pattern is known as "separation of concerns".

Before you decide to implement this pattern, weigh the benefits of the pattern against the potential effects of overhead introduced by another module. If your main requirement is flexibility, and you are going to make frequent changes to the services accessed, consider using a separate module as shown here. If performance is most important, and you are willing to update and redeploy the business logic, consider using a single module.



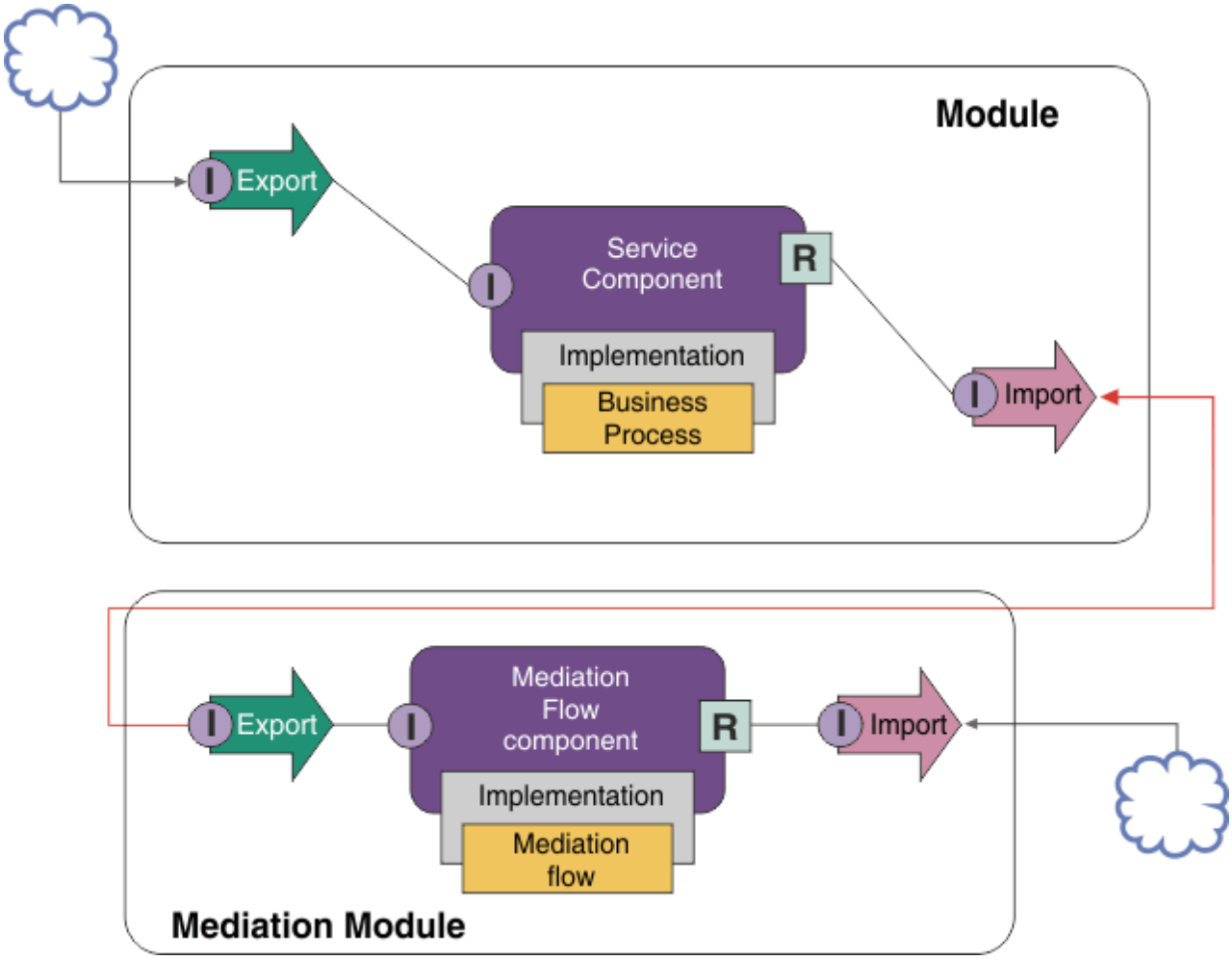
Here are the high level tasks to achieve this example.

1. Create a mediation module. For step-by-step instructions, see Creating mediation modules .
2. In the mediation module, create an import with the appropriate binding for the external service that you want to access. For step-by-step instructions, see Creating imports. For more information on bindings, see Bindings
3. Create an export, and give it the same interface as the import. For step-by-step instructions, see Creating exports.
4. Generate an SCA binding for the export. For step-by-step instructions, see Generating SCA bindings
5. In the mediation module's assembly, wire the export to the import. Save the mediation module.
6. Create a module. For step-by-step instructions, see Creating a module for business services
7. Add an export, and a component.
8. In the Business integration view, drag the export that you created in the mediation module (in step 4) into the module assembly. An import with the same binding as the export will be created.
9. Wire the export to the component, and the component to the import.
10. Add the component's implementation. For information on implementation types, see Implementations

Later, you can add mediation logic such as logging or routing to the mediation module without affecting the business module.

Adding mediation

Sometimes it is not sufficient to simply invoke an external service. Sometimes you need to do processing first, by adding a mediation module as an intermediary between the service requester and provider.



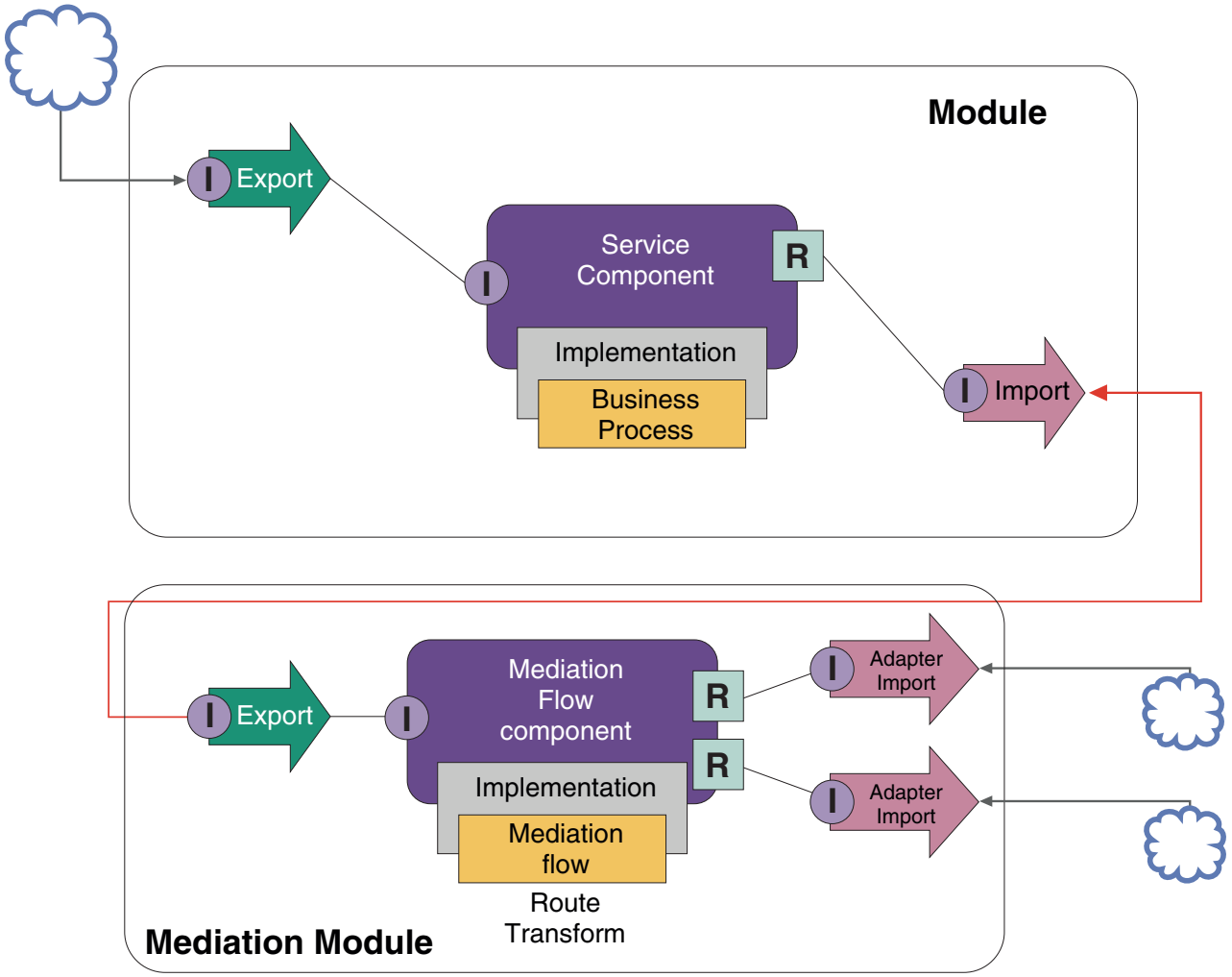
These are some of the functions that the intermediary mediation flow would perform:

- Setting protocol headers. For more information, see Protocol conversion in the WebSphere Enterprise Service Bus information center.
- Interface or parameter transformation by using a Business Object Map or Mapping primitive.Transforming messages
- Selecting a particular service from a static list, by using a Message Filter primitive.Message filter
- Invoking multiple services to aggregate the results, by using the Fan out and Fan In primitives.Aggregating and broadcasting messages
- Dealing with service invocation failures by retrying the same service or invoking a different service, by using a Service Invoke primitive. Retrying a failed service invocation
- Dynamic routing by choosing which service to use at the run time, rather than integration time, which enables services to be more loosely coupled and businesses to react faster to change. New services can be added without touching the modules that have been deployed to the runtime environment. Dynamic routing is most powerful when used with a registry, which requires the Endpoint Lookup mediation primitive to be used.Selecting endpoints dynamically

Accessing Enterprise Information Systems

Services and artifacts on external systems can be imported into Integration Designer. A wizard discovers applications and data on an Enterprise Information System (EIS) and lets you generate services from the discovered applications and data. The generated artifacts are interfaces and business objects, which can be used by components in a module.

Using an intermediary mediation module between a module and a host system makes it more reusable. In the example below, a mediation flow is used to route to the correct host system, and to transform the data to the format required by the host system.



Here are the high level tasks for this example:

1. Use the external service wizard to connect to the host system. Using the external service wizard to access external services follows a similar pattern regardless of the adapter you are using. For information on how to use the external service wizard, see Pattern of accessing external services with adapters
2. Create a module. For step-by-step instructions, see Creating a module for business services
3. Add an export, a component, and an import with SCA binding. For more information, see Calling services
4. Add an interface to the export, and wire the export to the component.
5. Add the component's implementation. In the implementation, set a property that indicates which host service will be accessed. For information on implementation types, see Implementations

6. Create a mediation module with an export that has an SCA binding, and the same interface as the import of the module that you created in step 2.
7. Wire the export to a mediation flow component.
8. Create an import for each host system that you want to access, using the appropriate outbound adapter from the assembly editor palette.
9. Wire the mediation flow component to the imports.
10. Implement the mediation flow component. Use a Message Filter primitive to choose the import based on a property set in the business logic, and use a Mapping primitive for each adapter import. Message filter.
11. In the module, select the mediation module's export as the service to be imported into the module. For step-by-step information, see Invoking a service from another module.

Later, you can make a change such as adding an adapter, or changing an adapter to point to a different host system, with minimal impact to the business logic.

Accessing messaging systems

In order for your service component architecture (SCA) module to communicate to an existing JMS, MQ or MQ JMS messaging client, you need to create interfaces, business objects and bindings for imports and exports. See Mapping a message to an SCA interface.

Mediation flows use messages, which provide access to context and header information in addition to business objects. If you want access to JMS header information or a custom JMS property, use a mediation flow. If you are integrating with an MQ system, and you want to access MQ header information, use a mediation flow.

Creating or calling a web service

Web services are self-contained applications that perform business functions, ranging from a simple query to complex business process interactions. You can call an existing web service or you can develop a new web service to fit your needs. This scenario will describe the steps and point you to additional information.

Although you may not create all of your services from the ground up using IBM Integration Designer, some of your services will indeed be created this way. When working with the assembly editor and the business process editor to assemble services into a business process, you are likely to find that some services are missing. It may therefore be helpful to create those missing services using IBM Integration Designer tools. The reverse is also true - after you have created a new process, you may decide that it would be useful to expose all, or some subset of the process operations as a service for others to consume.

Note: This scenario applies to users of IBM Integration Designer for IBM Process Server and WebSphere Enterprise Service Bus.

There are several reasons for developing web services using IBM Integration Designer:

- Creating services in IBM Integration Designer allows you to implement the service using business rules.
- Developing in IBM Integration Designer allows you to develop a Java™ service and expose it as both a web service and through SCA.
- Interface mapping without having to code is an advantage. You can take all of the data mappings out of the Java code leaving a simple black box Java program for the Java developer.
- IBM Integration Designer shows all of the services and relationships in one place.
- The ability to refactor will also aid in the development of web services using IBM Integration Designer.

Keep in mind that web services should not be regarded as the solution to all of your integration problems. However, just as with any other technology or architectural approach, there are inherent advantages of using web services in the right place and at the right time.

Exports, imports and bindings

IBM Integration Designer allows you to import standard web services and make use of these services in your composite applications.

In IBM Integration Designer, use the assembly editor to develop services. Follow the standard process to create modules, mediation modules, libraries, and components. Then, you can use exports, imports, and bindings to share and access those services. The steps for those basic tasks are listed below and the links lead to more detailed information for each task.

You can use either of two bindings for web services - a web service binding or an HTTP binding. A web service binding provides a specification for transmitting messages to and from a web service. The tools help you to generate a web service binding automatically. An HTTP binding is a standard request-and-response protocol between clients and server as defined by the HTTP protocol published by the World Wide Web consortium (W3C). You will need to provide some initial binding configuration information if you use an HTTP binding.

1. Create an export to publish the module's service for use by other modules.
2. Generate a binding for the export.
   - Generate a web service binding for the export.
   - Generate an HTTP export binding.
3. Create an import to call an existing service that is not part of the module that you are assembling.
   - Generate a web service binding for the import.
   - Generating an HTTP import binding.

Read the linked topic if you want to Invoke a web service from JavaServer Pages.

Web services development capabilities

When opening an editor associated with the web services creation process, you might see the Confirm Enablement window, which displays the following information:

```
This action requires the enablement of "Web Services Deployment".
Enable the required capability?
```

IBM Integration Designer provides a filtering function known as *capabilities*. In the Preferences settings, functions and tools are categorized into capabilities and you can enable and disable categories of capabilities or the subset functions of any category. See Capabilities for more information.

Learn more about key concepts

Use this section as a starting point to investigate the technologies used in and by IBM Business Process Manager.

Authoring scenarios

Use scenarios to understand and work with components and products from the business process management family.

Versioning

The lifecycle of a process application begins with the creation of the process application and continues through a cycle of updating, deploying, co-deploying, undeploying, and archiving the process

application. *Versioning* is a mechanism used to manage the lifecycle of the process application by uniquely identifying the individual versions of the process application.

The way that versioning works in IBM Business Process Manager depends on what you are deploying–a process application, deployed from the repository in IBM Process Center, or an enterprise application deployed directly from IBM Integration Designer.

The process applications and toolkits that you deploy to a runtime environment from the Process Center are, by default, versioned. For enterprise applications, you can choose to version modules and libraries in IBM Integration Designer.

In addition, you can create versions of a human task or state machine, so that multiple versions of the task or state machine can coexist in the runtime environment.

Versioning process applications

Versioning provides the ability for the runtime environment to identify snapshots in the lifecycle of a process application, and to be able to concurrently run multiple snapshots on a process server.

To understand how process applications are versioned, it is important to remember that a process application is a container that holds various artifacts used in or by the process application (for example, process models or BPDs, toolkit references, services, tracks, or monitor models). Any versioning is done at this container level, not at the level of the individual artifacts. For process applications, that means that versioning happens when you take a snapshot.

You can compare snapshots to determine differences between the versions. For example, if a developer fixed a problem with a service and took a snapshot of its containing process application or toolkit at that point, and then a different developer made several additional changes to the same service and took a new snapshot, the project manager can compare the two snapshots to determine which changes were made when and by whom. If the project manager decided that the additional changes to the service were not worthwhile, the project manager can revert to the snapshot of the original fix.

You can run different versions (snapshots) of a process application concurrently on a server; when you install a new snapshot, either remove the original or leave it running.

Version context

Each snapshot has unique metadata to identify the version (referred to as version context). You assign that identifier, but IBM recommends using a three-digit numeric version system in the format <major>.<minor>.<service>. See the topics about naming conventions for a more detailed description of this versioning scheme.

IBM Business Process Manager assigns a global namespace for each process application. The global namespace is specifically either the process application's tip or a particular process application snapshot. The version name used by the server cannot be longer than seven characters, so the assigned name is an acronym that uses characters from the snapshot name that you assigned. Snapshot acronyms areidentical to their snapshot names if the snapshot names conform to the recommended IBM VRM style and are not more that seven characters. For example, a snapshot name of 1.0.0 will have an acronym of 1.0.0, and a snapshot name of 10.3.0 will have the acronym of 10.3.0. The snapshot acronym will be guaranteed to be unique within the context of the process application within the scope of the Process Center server. For that reason, you cannot edit the snapshot acronym.

Versioning considerations for process applications in multiple clusters

You can install the same version of a process application to multiple clusters within the same cell. To differentiate among these multiple installations of the same version of the process application, create a snapshot for each installation and include a cell-unique ID in the snapshot name (for example,

v1.0_cell1_1 and v1.0_cell1_2). Each snapshot is a new version of the process application (from a pure lifecycle management perspective), but the content and function are the same.

When you install a process application in a cluster, an automatic synchronization of the nodes is performed.

Versioning considerations for Process Designer toolkits

Remember that process application snapshots are typically taken when you are ready to test or install. Toolkit snapshots, however, are typically taken when you are ready for that toolkit to be used by process applications. Afterward, if you want to update the toolkit, you must take another snapshot of "tip" when you are ready, and then the owners of process applications and toolkits can decide whether they want to move up to the new snapshot.

Versioning modules and libraries

If a module or library is in a process application or toolkit, it takes on the lifecycle of the process application or toolkit (versions, snapshots, tracks, and so on). Module and library names must be unique within the scope of a process application or toolkit.

This topic describes the versioning of modules and libraries that are used with process applications. Note, however, that if you deploy modules directly from IBM Integration Designer to Process Server, you can continue to follow the procedure of assigning version numbers to modules during deployment, as described in “Creating versioned modules and libraries”.

A module or library that is associated with the IBM Process Center must have its dependent libraries in the same process application or in a dependent toolkit.

The following table lists the selections you can make in the dependency editor in IBM Integration Designer when a library is associated with a process application or toolkit:

Table 5. Dependencies for Module, Process App or Toolkit, and Global libraries

Library scope	Description	Can depend on . . .
Module	A copy of this library exists on the server for each module that uses it.	A module-scoped library can depend on all types of libraries.
Process App or Toolkit	The library is shared among all modules in the scope of the process application or toolkit. This setting takes effect if deployment is done through the IBM Process Center. If deployment occurs outside of the IBM Process Center, the library is copied into each module. **Note:** Libraries created in IBM Integration Designer version 8 have a sharing level of **Process App or Toolkit** by default.	A library of this type can depend only on global libraries.
Global	The library is shared among all modules that are running.	A global library can depend only on other global libraries. **Note:** You must configure a WebSphere shared library in order to deploy the global library. See “Module and library dependencies” for more information.

Modules and libraries associated with process applications or toolkits

You do not need to version modules and libraries associated with process applications or toolkits.

Modules and libraries that are associated with a process application or toolkit do not need to be versioned. In fact, you cannot create a version of a module or library associated with a process

application or toolkit in the dependency editor. Modules and libraries associated with a process application or toolkit use snapshots, a function in the Process Center, to achieve the same result as a version.

Libraries associated with a process application or toolkit will not have a required version number in the Libraries section of the dependency editor because no version is needed.

Naming conventions

A naming convention is used to differentiate the various versions of a process application as it moves through the lifecycle of updating, deploying, co-deploying, undeploying, and archiving.

This section provides you with the conventions that are used to uniquely identify versions of a process application.

A *version context* is a combination of acronyms that uniquely describes a process application or toolkit. Each type of acronym has a naming convention. The acronym is limited to a maximum length of seven characters from the [A-Z0-9_] character set, except for the snapshot acronym, which can also include a period.

- The process application acronym is created when the process application is created. It can be a maximum of seven characters in length.
- The snapshot acronym is created automatically when the snapshot is created. It can be a maximum of seven characters in length.

  If the snapshot name meets the criteria for a valid snapshot acronym, the snapshot name and acronym will be the same.

  Note: When using the mediation flow component version-aware routing function, name your snapshot so that it conforms to the <*version*>.<*release*>.<*modification*> scheme (for example, **1.0.0**). Because the snapshot acronym is limited to seven characters, the digit values are limited to a maximum of five total digits (five digits plus two periods). Therefore, care should be taken when the digit fields are incremented, because anything beyond the first seven characters is truncated.

  For example, a snapshot name **11.22.33** results in a **11.22.3** snapshot acronym.
- The track acronym is automatically generated from the first character of each word of the track name. For example, a new track created with the name **My New Track** would result in an acronym value of **MNT**.

  The default track name and acronym are **Main**. Deployment to a IBM Process Center server includes the track acronym in the versioning context if the track acronym is not **Main**.

A business process definition in a process application is typically identified by the process application name acronym, the snapshot acronym, and the name of the business process definition. Choose unique names for your business process definitions whenever possible. When duplicate names exist, you might encounter the following problems:

- You might be unable to expose the business process definitions as web services without some form of mediation.
- You might be unable to invoke a business process definition created in IBM Process Designer from a BPEL process created in IBM Integration Designer.

The version context varies, depending on how the process application is deployed.

Naming conventions for Process Center server deployments:

On the IBM Process Center server, you can deploy a snapshot of a process application as well as a snapshot of a toolkit. In addition, you can deploy the tip of a process application or the tip of a toolkit. (A *tip* is the current working version of your process application or toolkit.) The version context varies, depending on the type of deployment.

For process applications, the process application tip or the specific process application snapshot is used to uniquely identify the version.

Toolkits can be deployed with one or more process applications, but the lifecycle of each toolkit is bound to the lifecycle of the process application. Each process application has its own copy of the dependent toolkit or toolkits deployed to the server. A deployed toolkit is not shared between process applications.

If the track associated with the process application is named something other than the default of **Main**, the track acronym is also part of the version context.

For more information, see the "Examples" on page 37 section, later in this topic.

Process application snapshots

For process application snapshot deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application track acronym (if a track other than **Main** is used)
- Process application snapshot acronym

Stand-alone toolkits

For toolkit snapshot deployments, the version context is a combination of the following items:

- Toolkit name acronym
- Toolkit track acronym (if a track other than **Main** is used)
- Toolkit snapshot acronym

Tips

Process application tips are used during iterative testing in Process Designer. They can be deployed to Process Center servers only.

For process application tip deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application track acronym (if a track other than **Main** is used)
- "Tip"

Toolkit tips are also used during iterative testing in Process Designer. They are not deployed to a production server.

For toolkit tip deployments, the version context is a combination of the following items:

- Toolkit name acronym
- Toolkit track acronym (if a track other than **Main** is used)
- "Tip"

Examples

Resources should be uniquely named and identified externally using the version context.

- The following table shows an example of names that are uniquely identified. In this example, a process application tip uses the default track name (**Main**):

Table 6. Process application tip with default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Main**
Process application track acronym	"" (when the track is **Main**)
Process application snapshot	
Process application snapshot acronym	**Tip**

Any SCA modules associated with this process application tip include the version context, as shown in the following table:

Table 7. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-Tip-M1	PA1-Tip-M1.ear
M2	PA1-Tip-M2	PA1-Tip-M2.ear

- The following table shows an example of a process application tip that uses a non-default track name:

Table 8. Process application tip with non-default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Track1**
Process application track acronym	**T1**
Process application snapshot	
Process application snapshot acronym	**Tip**

Any SCA modules associated with this process application tip include the version context, as shown in the following table:

Table 9. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-T1-Tip-M1	PA1-T1-Tip-M1.ear
M2	PA1-T1-Tip-M2	PA1-T1-Tip-M2.ear

Similar naming conventions apply to advanced Toolkit tip and snapshot deployments. They also apply to advanced snapshots installed to Process Server.

- The following table shows an example of names that are uniquely identified. In this example, a process application snapshot uses the default track name (**Main**):

Table 10. Process application snapshot with default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Main**
Process application track acronym	"" (when the track is **Main**)
Process application snapshot	**Process Shapshot V1**
Process application snapshot acronym	**PSV1**

Any SCA modules associated with this process application snapshot include the version context, as shown in the following table:

Table 11. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-PSV1-M1	PA1-PSV1-M1.ear
M2	PA1-PSV1-M2	PA1-PSV1-M2.ear

- The following table shows an example of a process application snapshot that uses a non-default track name:

Table 12. Process application snapshot with non-default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Track1**
Process application track acronym	**T1**
Process application snapshot	**Process Snapshot V1**
Process application snapshot acronym	**PSV1**

Any SCA modules associated with this process application snapshot include the version context, as shown in the following table:

Table 13. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-T1-PSV1-M1	PA1-T1-PSV1-M1.ear
M2	PA1-T1-PSV1-M2	PA1-T1-PSV1-M2.ear

Naming conventions for Process Server deployments:

On the Process Server, you can deploy the snapshot of a process application. The process application snapshot acronym is used to uniquely identify the version.

For process application snapshot deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application snapshot acronym

Resources should be uniquely named and identified externally using the version context. The following table shows an example of names that are uniquely identified:

Table 14. Example of names and acronyms

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application snapshot	**1.0.0**
Process application snapshot acronym	**1.0.0**

A resource, such as a module or library, has the version context as part of its identify.

The following table shows an example of two modules and how the associated EAR files include the version context:

Table 15. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-1.0.0-M1	PA1-1.0.0-M1.ear
M2	PA1-1.0.0-M2	PA1-1.0.0-M2.ear

The following table shows an example of two process-application-scoped libraries and how the associated JAR files include the version context:

Table 16. Process-application-scoped libraries and version-aware JAR files

SCA process-application-scoped library name	Version-aware name	Version-aware JAR name
Lib1	PA1-1.0.0-Lib1	PA1-1.0.0-Lib1.jar
Lib2	PA1-1.0.0-Lib2	PA1-1.0.0-Lib2.jar

Version-aware bindings

Process applications can contain SCA modules that include import and export bindings. When you co-deploy applications, the binding for each version of the application must be unique. Some bindings are automatically updated during deployment to ensure the uniqueness between versions. In other cases, you have to update the binding after deployment to ensure its uniqueness.

A *version-aware* binding is scoped to a particular version of a process application, which guarantees its uniqueness between process applications. The following sections describe the bindings that are automatically updated to be version-aware as well as any actions you need to take at run time when a binding is not version-aware. For information about things to consider when you create modules, see "Considerations when using bindings".

SCA

The target of an SCA binding is automatically renamed to be version-aware during deployment if the import and export bindings of the module are defined in the same process application scope.

If the bindings are not defined in the same process application scope, an information message is logged. You must modify the import binding after deployment to change the endpoint target address. You can use the administrative console to change the endpoint target address.

Web service (JAX-WS or JAX-RPC)

The endpoint target address of a web service binding is automatically renamed to be version-aware during deployment if all the following conditions are true:

- You followed the default naming convention for the address:
  http://*ip*:*port*/*ModuleName*Web/sca/*ExportName*
- The endpoint address is SOAP/HTTP.
- The import and export bindings of the module are defined in the same process application scope.

If these conditions are not true, an information message is logged. The action you then take depends on how you are deploying your process application:

- If you are co-deploying your process application, you must manually rename the SOAP/HTTP endpoint URL or the SOAP/JMS destination queue to be unique between versions of the process application. You can use the administrative console after deployment to change the endpoint target address.
- If you are deploying only a single version of the process application, you can ignore this message

For SOAP/ JMS web service binding snapshot co-deployment, the action you take depends on how you are deploying your process application:

- If the import and target export are in the same process application, perform the following steps before you publish the process application to the Process Center and create the snapshot:
  1. Change the endpoint URL of the export. Make sure the destination and connection factory are unique.
  2. Change the endpoint URL of the import so that it is the same as the one you specified for the export in the previous step.
- If the import and target export are in different process applications, perform the following steps:
  1. Change the endpoint URL of the export. Make sure the destination and connection factory are unique.
  2. Publish the process application to the Process Center.
  3. Create the snapshot.
  4. Deploy the process application to the Process Server.
  5. Use the WebSphere administrative console to change the endpoint URL of the corresponding import so that it is the same as the one you specified for the export.

HTTP

The endpoint URL address of an HTTP binding is automatically renamed to be version-aware during deployment if all the following conditions are true:

- You followed the default naming convention for the address:
  http(s)://*ip*:*port*/*ModuleName*Web/*contextPathinExport*
- The import and export bindings of the module are defined in the same process application scope.

If these conditions are not true, an information message is logged. The action you then take depends on how you are deploying your process application:

- If you are co-deploying your process application, you must manually rename the endpoint URL to be unique between versions of the process application. You can use the administrative console after deployment to change the endpoint target address.
- If you are deploying only a single version of the process application, you can ignore this message

JMS and generic JMS

System-generated JMS and generic JMS bindings are automatically version-aware.

Note: For user-defined JMS and generic JMS bindings, no automatic renaming occurs during deployment to enable the bindings to be version-aware. If the binding is user-defined, you must rename the following attributes to be unique between versions of the process application:

- Endpoint configuration
- Receive destination queue
- Listener port name (if defined)

Set the matching Send destination if you change the target module endpoint.

MQ/JMS and MQ

No automatic renaming occurs during deployment to enable bindings of type MQ/JMS or MQ to be version-aware.

You must rename the following attributes to be unique between versions of the process application:

- Endpoint configuration
- Receive destination queue

Set the matching Send destination if you change the target module endpoint.

EJB

No automatic renaming occurs during deployment to enable bindings of type EJB to be version-aware.

You must rename the JNDI names attribute to be unique between versions of the process application.

Note that client applications also need to be updated to use the new JNDI names.

EIS

A resource adapter is automatically renamed to be version-aware during deployment as long as the default resource name (***ModuleName*App:Adapter Description**) was not modified.

If the default resource name was modified, the resource adapter names must be unique between versions of the process application.

If the resource adapter names are not unique, an informational message is logged during deployment to alert you. You can manually rename the resource adapters after deployment using the administrative console.

Version-aware dynamic invocation

You can configure mediation flow components to route messages to endpoints that are determined dynamically at run time. When you create the mediation module, you configure the endpoint lookup to use version-aware routing.

If you use the IBM_VRM style (*<version>.<release>.<modification>*) for the snapshot, you can export the process application EAR file to WebSphere Service Registry and Repository (WSRR). When you create the mediation module, you then configure the endpoint lookup to use version-aware routing. For example, you select **Return endpoint matching latest compatible version of SCA module-based services** from the **Match Policy field**, and you select **SCA** for **Binding Type**.

Future versions of the process application are deployed to the server and published to WSRR, and the mediation module endpoint lookup dynamically invokes the latest compatible version of the service endpoint.

Note that, as an alternative, you can set the target in the SMOHeader, and the value can be carried by the request message.

Deploying process applications with Java modules and projects

Process applications can contain custom Java EE modules and Java projects. When you co-deploy applications, the custom Java EE module for each version of the application must be unique.

Note that custom Java EE modules and Java projects are deployed to a server if they are deployed with an SCA module that has a dependency declared on them. If you do not select **Deploy with module** (which is the default) when you declare the dependency, you need to deploy the module or project manually.

Deploying process applications with business rules and selectors

If you are deploying multiple versions of a process application that includes a business rule or selector component, be aware of the way that the associated metadata is used by the versions.

The dynamic metadata for a business rule or selector component is defined at run time by the component name, component target namespace, and component type. If two or more versions of a process application containing a business rule or selector are deployed to the same runtime environment, they will share the same rule logic (business rule) or routing (selector) metadata.

To enable each version of the business rule or selector component of the process application to use its own dynamic metadata (rule logic or routing), refactor the target namespace so that it is unique for each version of the process application.

Deployment architecture

The IBM Business Process Manager deployment architecture consists of software processes called servers, topological units referenced as nodes and cells, and the configuration repository used for storing configuration information.

Cells

In IBM Business Process Manager, *cells* are logical groupings of one or more nodes in a distributed network.

A cell is a configuration concept, a way for administrators to logically associate nodes with one another. Administrators define the nodes that make up a cell, according to the specific criteria that make sense in their organizational environments.

Administrative configuration data is stored in XML files. A cell retains master configuration files for each server in every node in the cell. Each node and server also have their own local configuration files. Changes to a local node or to a server configuration file are temporary, if the server belongs to the cell. While in effect, local changes override cell configurations. Changes to the master server and master node configuration files made at the cell level replace any temporary changes made at the node when the cell configuration documents are synchronized to the nodes. Synchronization occurs at designated events, such as when a server starts.

Servers

Servers provide the core functionality of IBM Business Process Manager. Process servers extend, or augment, the ability of an application server to handle Service Component Architecture (SCA) modules. Other servers (deployment managers and node agents) are used for managing process servers.

A process server can be either a *stand-alone server* or a *managed server*. A managed server can optionally be a member of a *cluster*. A collection of managed servers, clusters of servers, and other middleware is called a *deployment environment*. In a deployment environment, each of the managed servers or clusters is

configured for a specific function within the deployment environment (for example, destination host, application module host, or Common Event Infrastructure server). A stand-alone server is configured to provide all of the required functions.

Servers provide the runtime environment for SCA modules, for the resources that are used by those modules (data sources, activation specifications, and JMS destinations), and for IBM-supplied resources (message destinations, Business Process Choreographer containers, and Common Event Infrastructure servers).

A *node agent* is an administrative agent that represents a node to your system and manages the servers on that node. Node agents monitor servers on a host system and route administrative requests to servers. The node agent is created when a node is federated to a deployment manager.

A *deployment manager* is an administrative agent that provides a centralized management view for multiple servers and clusters.

A stand-alone server is defined by a stand-alone profile; a deployment manager is defined by a deployment manager profile; managed servers are created within a *managed node*, which is defined by a custom profile.

Stand-alone servers:

A stand-alone server provides an environment for deploying SCA modules in one server process. This server process includes, but is not limited to, an administrative console, a deployment target, the messaging support, the business process rules manager, and a Common Event Infrastructure server.

A stand-alone server is simple to set up, and has a First steps console from which you can start and stop the server and open the samples gallery and the administrative console. If you install the IBM Business Process Manager samples, and then open the samples gallery, a sample solution is deployed to the stand-alone server. You can explore the resources used for this sample in the administrative console.

You can deploy your own solutions to a stand-alone server, but a stand-alone server cannot provide the capacity, scalability, or robustness that is required of a production environment. For your production environment, it is better to use a network deployment environment.

It is possible to start off with a stand-alone server and later include it in a network deployment environment, by federating it to a deployment manager cell, *provided that no other nodes have been federated to that cell*. It is not possible to federate multiple stand-alone servers into one cell. To federate the stand-alone server, use the administrative console of the deployment manager or the **addNode** command. The stand-alone server must not be running when you federate it using the **addNode** command.

A stand-alone server is defined by a stand-alone server profile.

Clusters:

Clusters are groups of servers that are managed together and participate in workload management.

A cluster can contain nodes or individual application servers. A node is usually a physical computer system with a distinct host IP address that is running one or more application servers. Clusters can be grouped under the configuration of a cell, which logically associates many servers and clusters with different configurations and applications with one another depending on the discretion of the administrator and what makes sense in their organizational environments.

Clusters are responsible for balancing workload among servers. Servers that are a part of a cluster are called cluster members. When you install an application on a cluster, the application is automatically installed on each cluster member.

Because each cluster member contains the same applications, you can distribute client tasks according to the capacities of the different machines by assigning weights to each server.

Assigning weights to the servers in a cluster improves performance and failover. Tasks are assigned to servers that have the capacity to perform the task operations. If one server is unavailable to perform the task, it is assigned to another cluster member. This reassignment capability has obvious advantages over running a single application server that can become overloaded if too many requests are made.

Profiles

A profile defines a unique runtime environment, with separate command files, configuration files, and log files. Profiles define three different types of environments on IBM Business Process Manager systems: stand-alone server, deployment manager, and managed node.

Using profiles, you can have more than one runtime environment on a system, without having to install multiple copies of the IBM Business Process Manager binary files.

Use the Profile Management Tool or the **manageprofiles** command-line utility to create profiles.

Note: On distributed platforms, each profile has a unique name. On the z/OS® platform, all profiles are named "default".

The profile directory

Every profile in the system has its own directory containing all of its files. You specify the location of the profile directory when you create the profile. By default, it is in the `profiles` directory in the directory where IBM Business Process Manager is installed. For example, the Dmgr01 profile is in `C:\Program Files\IBM\WebSphere\ProcServer\profiles\Dmgr01`.

The First steps console

Every profile in the system has a First steps console. You can use this interface to familiarize yourself with the stand-alone server, deployment manager, or managed node.

The default profile

The first profile that you create within one installation of IBM Business Process Manager is the *default profile*. The default profile is the default target for commands issued from the `bin` directory in the directory where IBM Business Process Manager was installed. If only one profile exists on a system, every command operates on that profile. If you create another profile, you can make it the default.

Note: The default profile is not necessarily a profile whose name is "default".

Augmenting profiles

If you already have a deployment manager profile, a custom profile, or a stand-alone server profile created for WebSphere Application Server Network Deployment or WebSphere ESB, you can *augment* it to support IBM Business Process Manager in addition to existing function. To augment a profile, first install IBM Business Process Manager. Then use the Profile Management Tool or the **manageprofiles** command-line utility.

Restriction: You cannot augment a profile if it defines a managed node that is already federated to a deployment manager.

Deployment managers

A deployment manager is a server that manages operations for a logical group, or cell, of other servers. The deployment manager is the central location for administering the servers and clusters.

When creating a deployment environment, the deployment manager profile is the first profile that you create. The deployment manager has a First steps console, from which you can start and stop the deployment manager and start its administrative console. You use the administrative console of the deployment manager to manage the servers and clusters in the cell. This includes configuring servers and clusters, adding servers to clusters, starting and stopping servers and clusters, and deploying SCA modules.

Although the deployment manager is a type of server, you cannot deploy modules to the deployment manager itself.

Nodes

A *node* is a logical grouping of managed servers.

A node usually corresponds to a logical or physical computer system with a distinct IP host address. Nodes cannot span multiple computers. Node names usually are identical to the host name for the computer.

Nodes in the network deployment topology can be managed or unmanaged. A managed node has a node agent process that manages its configuration and servers. Unmanaged nodes do not have a node agent.

Managed nodes:

A *managed node* is a node that is federated to a deployment manager and contains a node agent and can contain managed servers. In a managed node, you can configure and run managed servers.

The servers that are configured on a managed node make up the resources of your deployment environment. These servers are created, configured, started, stopped, managed and deleted using the administrative console of the deployment manager.

A managed node has a node agent that manages all servers on a node.

When a node is federated, a node agent process is created automatically. This node agent must be running to be able to manage the configuration of the profile. For example, when you do the following tasks:

- Start and stop server processes.
- Synchronize configuration data on the deployment manager with the copy on the node.

However, the node agent does not need to be running in order for the applications to run or to configure resources in the node.

A managed node can contain one or more servers, which are managed by a deployment manager. You can deploy solutions to the servers in a managed node, but the managed node does not contain a sample applications gallery. The managed node is defined by a custom profile and has a First steps console.

Unmanaged nodes:

An unmanaged node does not have a node agent to manage its servers.

Unmanaged nodes in the Network Deployment topology can have server definitions such as Web servers , but not Application Server definitions. Unmanaged nodes can never be federated. That is, a node agent can never be added to an unmanaged node. Another type of unmanaged node is a stand-alone server. The deployment manager cannot manage this stand-alone server because it is not known to the cell. A stand-alone server can be federated. When it is federated, a node agent is automatically created. The node becomes a managed node in the cell.

Node agents

Node agents are administrative agents that route administrative requests to servers.

A node agent is a server that runs on every host computer system that participates in the Network Deployment configuration. It is purely an administrative agent and is not involved in application-serving functions. A node agent also hosts other important administrative functions such as file transfer services, configuration synchronization, and performance monitoring.

Naming considerations for profiles, nodes, servers, hosts, and cells

This topic discusses reserved terms and issues you must consider when naming your profile, node, server, host, and cell (if applicable). This topic applies to distributed platforms.

Profile naming considerations

The profile name can be any unique name with the following restrictions. Do not use any of the following characters when naming your profile:

- Spaces
- Special characters that are not allowed within the name of a directory on your operating system, such as *, &, or ?.
- Slashes (/) or back slashes (\)

Double-byte characters are allowed.

Windows **Directory path considerations:** The installation directory path must be less than or equal to 60 characters. The number of characters in the *profiles_directory_path**profile_name* directory must be less than or equal to 80 characters.

Node, server, host, and cell naming considerations

Reserved names: Avoid using reserved names as field values. The use of reserved names can cause unpredictable results. The following words are reserved:

- cells
- nodes
- servers
- clusters
- applications
- deployments

Descriptions of fields on the Node and Hosts Names and Node, Host, and Cell Names pages: Table 17 describes the fields found on the Node and Host Names and Node, Host, and Cell Names pages of the Profile Management Tool, including the field names, default values, and constraints. Use this information as a guide when you are creating profiles.

Table 17. Naming guidelines for nodes, servers, hosts, and cells

Field name	Default value	Constraints	Description
Stand-alone server profiles			

Table 17. Naming guidelines for nodes, servers, hosts, and cells (continued)

Field name	Default value	Constraints	Description
Node name	Linux UNIX Windows *shortHostName* Node *NodeNumber* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Avoid using the reserved names.	Select any name you want. To help organize your installation, use a unique name if you plan to create more than one server on the system.
Server name	Linux UNIX Windows server1	Use a unique name for the server.	The logical name for the server.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.
Cell name	Linux UNIX Windows *shortHostName* Node *NodeNumber* Cell where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Use a unique name for the cell. A cell name must be unique in any circumstance in which the product is running on the same physical workstation or cluster of workstations, such as a Sysplex. Additionally, a cell name must be unique in any circumstance in which network connectivity between entities is required either between the cells or from a client that must communicate with each of the cells. Cell names also must be unique if their name spaces are going to be federated. Otherwise, you might encounter symptoms such as a javax.naming.Name NotFoundException exception, in which case, you need to create uniquely named cells.	All federated nodes become members of a deployment manager cell.
Deployment manager profiles			

Table 17. Naming guidelines for nodes, servers, hosts, and cells (continued)

Field name	Default value	Constraints	Description
Node name	Linux UNIX Windows *shortHostName* Cell Manager*Node Number* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Use a unique name for the deployment manager. Avoid using the reserved names.	The name is used for administration within the deployment manager cell.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. Avoid using the reserved names. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.
Cell name	Linux UNIX Windows *shortHostName* Cell *CellNumber* where: • *shortHost Name* is the short host name. • *CellNumber* is a sequential number starting at 01.	Use a unique name for the deployment manager cell. A cell name must be unique in any circumstance in which the product is running on the same physical workstation or cluster of workstations, such as a Sysplex. Additionally, a cell name must be unique in any circumstance in which network connectivity between entities is required either between the cells or from a client that must communicate with each of the cells. Cell names also must be unique if their name spaces are going to be federated. Otherwise, you might encounter symptoms such as a javax.naming.Name NotFoundException exception, in which case, you need to create uniquely named cells.	All federated nodes become members of the deployment manager cell , which you name in the Node, Host, and Cell Names page of the Profile Management Tool.
Custom profiles			

Table 17. Naming guidelines for nodes, servers, hosts, and cells (continued)

Field name	Default value	Constraints	Description
Node name	Linux UNIX Windows *shortHostName* Node *NodeNumber* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Avoid using the reserved names. Use a unique name within the deployment manager cell.	The name is used for administration within the deployment manager cell to which the custom profile is added. Use a unique name within the deployment manager cell.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.

Host name considerations:

The host name is the network name for the physical workstation on which the node is installed. The host name must resolve to a physical network node on the server. When multiple network cards exist in the server, the host name or IP address must resolve to one of the network cards. Remote nodes use the host name to connect to and to communicate with this node.

IBM Business Process Manager is compliant to both Internet Protocol version 4 (IPv4) and version 6 (IPv6). Wherever you can enter IP addresses in the administrative console, or elsewhere, you can do so in either format. Note that if IPv6 is implemented on your system you must enter the IP address in IPv6 format, and conversely, if IPv6 is not yet available to you, enter IP addresses in IPv4 format. For more information on IPv6 refer to the following description: IPv6.

The following guidelines can help in determining the appropriate host name for your workstation:

- Select a host name that other workstations can reach within your network.
- Do not use the generic identifier, localhost, for this value.
- Do not attempt to install IBM Business Process Manager products on a server with a host name that uses characters from the double-byte character set (DBCS). DBCS characters are not supported when used in the host name.
- Avoid using the underscore (_) character in server names. Internet standards dictate that domain names conform to the host name requirements described in Internet Official Protocol Standards RFC 952 and RFC 1123. Domain names must contain only letters (upper or lower case) and digits. Domain names can also contain dash characters (-) as long as the dashes are not on the ends of the name. Underscore characters (_) are not supported in the host name. If you have installed IBM Business Process Manager on a server with an underscore character in the server name, access the server with its IP address until you rename it.

If you define coexisting nodes on the same computer with unique IP addresses, define each IP address in a domain name server (DNS) look-up table. Configuration files for servers do not provide domain name resolution for multiple IP addresses on a workstation with a single network address.

The value that you specify for the host name is used as the value of the hostName property in configuration documents. Specify the host name value in one of the following formats:

- Fully qualified domain name servers (DNS) host name string, such as xmachine.manhattan.ibm.com
- The default short DNS host name string, such as xmachine
- Numeric IP address, such as 127.1.255.3

The fully qualified DNS host name has the advantages of being totally unambiguous and flexible. You have the flexibility of changing the actual IP address for the host system without having to change the server configuration. This value for host name is particularly useful if you plan to change the IP address frequently when using Dynamic Host Configuration Protocol (DHCP) to assign IP addresses. A disadvantage of this format is being dependent on DNS. If DNS is not available, then connectivity is compromised.

The short host name is also dynamically resolvable. A short name format has the added ability of being redefined in the local hosts file so that the system can run the server even when disconnected from the network. Define the short name to 127.0.0.1 (local loopback) in the hosts file to run disconnected. A disadvantage of the short name format is being dependent on DNS for remote access. If DNS is not available, then connectivity is compromised.

A numeric IP address has the advantage of not requiring name resolution through DNS. A remote node can connect to the node you name with a numeric IP address without DNS being available. A disadvantage of this format is that the numeric IP address is fixed. You must change the setting of the hostName property in configuration documents whenever you change the workstation IP address. Therefore, do not use a numeric IP address if you use DHCP, or if you change IP addresses regularly. Another disadvantage of this format is that you cannot use the node if the host is disconnected from the network.

BPMN 2.0

IBM Business Process Manager business process definitions support the Common Executable subclass of the BPMN 2.0 Process Modeling conformance class, which deals with models that you can run.

BPMN (Business Process Model and Notation) is the foundational standard for the processes in IBM Process Designer and IBM Process Center. Business process definition (BPD) diagrams are based on the BPMN specification. This topic introduces some of the ways BPMN 2.0 is applied in IBM Business Process Manager. For detailed information about BPMN, see the BPMN Specification page at http://www.bpmn.org/.

IBM Business Process Manager supports the following BPMN 2.0 task types:

- None (abstract task in the BPMN 2.0 specification)
- System task (service task in the BPMN 2.0 specification)
- User task
- Script
- Decision task (business rule task in the BPMN 2.0 specification)

The IBM BPM intermediate message events provide similar functions to the BPMN send task and receive task.

BPMN 2.0 notation

Starting in V7.5.1, Process Designer BPMN 2.0 task icons in the BPD diagrams are collected on a simplified palette and displayed in process diagrams. The icons show whether your activity is a system task, user task, decision task, script, or linked process. Activities in models that were created in earlier versions also show appropriate BPMN 2.0 tasks types and task icons when you view them in version 7.5.1 or later.

Activities and tasks

There are some terminology changes from previous versions of Process Designer. A number of those changes involve activity types that have been renamed.

- Service (automated) activities are now system tasks.
- Service (task) activities in a non-system swimlane are now user tasks.
- Service (task) activities in a system swimlane are now decision tasks if they reference a decision service.
- Service (task) activities in a system swimlane are now system tasks if they reference any kind of service other than a decision service.
- Javascript activities are now script tasks.
- Nested process activities are now linked processes.
- External activities from previous versions of Process Designer are available as external implementations for user tasks or system tasks.

Gateways

There are no notation changes to the gateways of previous versions. However, there are three terminology changes. The decision gateway is now the *exclusive gateway*, the simple split or join gateway is now the *parallel gateway*, and the conditional split or join gateway is now the *inclusive gateway*.

There is also a new gateway type, the *event gateway*. An event gateway represents a branching point in a process where the alternative paths that follow the gateway are based on events that occur, rather than the evaluation of expressions using process data (as with an exclusive or inclusive gateway). A specific event, usually the receipt of a message, determines the path that will be taken.

Non-interrupting events

BPMN 2.0 added notation for non-interrupting events. By default, a boundary event interrupts the activity that it is attached to. When the event is triggered, the activity stops and the token continues down the outgoing sequence flow of the event. If the event is set as non-interrupting, when the event is triggered the attached activity continues in parallel, and a new token is generated and is passed down the outgoing sequence flow of the event. The event boundary changes to a dashed line for non-interrupting events.

Intermediate events that are attached to activities are interrupting intermediate events if they close their attached activities or non-interrupting intermediate events if they do not close their attached activities.

Start event

The BPMN specification permits process models to omit start and end event symbols. Process Designer requires that process models use start and stop events.

There are various types of start events available in Process Designer:

processes

- none
- message
- ad hoc

subprocesses

- none

event subprocesses

- error
- message
- timer

You can change the type of a start event by editing the properties of the event. You can have numerous message start events in a process, but you can use only one none start event.

End events

Four types of end events are available: *message*, *terminate*, *error* and *none*. You can change the type of an end event.

When a parent process calls a child process and the child process runs a terminate event action, the BPMN semantics say that the child process immediately stops and the parent process then continues to its next steps. In Process Designer, if a child process runs a terminate event activity, both the child process and its parent process stop.

Subprocesses

The BPMN specification defines two types of subprocesses, embedded and reusable. You can reate both types in Process Designer. Embedded subprocesses are just called *subprocesses* in Process Designer and are new in version 7.5.1. The BPMN reusable subprocess is called a *linked process* in Process Designer.

A subprocess exists within the containing process and is a way of grouping process steps to reduce diagram complexity and clutter. Subprocesses collapse multiple steps into one activity. The subprocess can be seen only by the process in which it is defined. A subprocess exists within the scope of its caller and has access to all the variables within that environment. There is no parameter passing in and out of the embedded subprocess.

Separate from the subprocess and the linked process, Process Designer has an event subprocess, which is a specialized subprocess that is used for event handling. It is not connected to other activities through sequence flow, and it occurs only if its start event is triggered.

Linked processes

A BPMN reusable subprocess is called a *linked process* in Process Designer. It is a process created outside the current process that can be called by the current process. It is reusable because other process definitions can call this process as well. The linked process defines its input and output parameters and has no access to the caller's scope or environment. The linked process is similar to the nested process available in former versions; there is no change to the behavior of the activity. Previous nested processes are migrated to linked processes. The linked process looks like a subprocess with a thick boundary and is highlighted in the Inspector window.

Loops

BPMN provides the concept of an activity that can be repeated. The activity can be atomic, meaning that the activity repeats, or it can be a subprocess, encapsulating a series of steps that are repeated. If you

expand the repeated activity, you see the contained activities that are to be run repeatedly. The condition is always evaluated at the start of each loop iteration. There is no ability to evaluate at the end of each loop iteration.

IBM Business Process Manager has a *multi-instance loop*, which is run some finite number of times with the activities contained within run sequentially or in parallel.

Importing non-BPMN processes

You can import models that were created in IBM WebSphere Business Modeler and use them in Process Designer. For information about the BPMN 2.0 import, see Mapping IBM WebSphere Business Modeler elements to IBM Business Process Manager constructs. You also can import BPMN 2.0 models that were created in IBM WebSphere Business Compass, Rational Software Architect, or other modeling environments.

Business process definitions (BPDs)

To model a process in IBM Process Designer, you must create a business process definition (BPD). The business process definition can be based on an imported BPMN model.

A BPD is a reusable model of a process, defining what is common to all runtime instances of that process model. A BPD must contain a start event, an end event, at least one lane, and one or more activities. See "IBM Process Designer naming conventions" in the related links for details about the character limitations that apply to BPDs.

A Business Process Definition (BPD) needs to include a lane for each system or group of users who participates in a process. A lane can be a participant lane or a system lane. However, you can create a BPD that groups the activities of a group and a system into a single lane if that is your preference. See "Creating a business process definition (BPD)" in the related links for information about how to create a BPD.

You can designate any specific person or group to be responsible for the activities in a participant lane. Each lane that you create is assigned to the All Users participant group by default. You can use this default participant group for running and testing your BPD in the Inspector. The All Users participant group includes all users who are members of the tw_allusers security group, which is a special security group that automatically includes all users in the system.

A system lane contains activities handled by a specific IBM Process Center system. Each activity needs an implementation, which defines the activity and sets the properties for the task. During implementation, a developer creates a service or writes the JavaScript necessary to complete the activities in a system lane. See "Understanding service types" in the related links for information about services.

For each BPD that you create, you need to declare variables to capture the business data that is passed from activity to activity in your process. See "Managing and mapping variables" in the related links to learn about implementing variables.

You can also add events to a BPD. Events in IBM BPM can be triggered by a due date passing, an exception, or a message arriving. The trigger that you want determines the type of event you choose to implement. For detailed information about available event types and their triggers, see "Modeling events".

Bindings

At the core of a service-oriented architecture is the concept of a *service*, a unit of functionality accomplished by an interaction between computing devices. An *export* defines the external interface (or access point) of a module, so that Service Component Architecture (SCA) components within the module can provide their services to external clients. An *import* defines an interface to services outside a module,

so the services can be called from within the module. You use protocol-specific *bindings* with imports and exports to specify the means of transporting the data into or out of the module.

Exports

External clients can invoke SCA components in an integration module over a variety of protocols (such as HTTP, JMS, MQ, and RMI/IIOP) with data in a variety of formats (such as XML, CSV, COBOL, and JavaBeans). Exports are components that receive these requests from external sources and then invoke IBM Business Process Manager components using the SCA programming model.

For example, in the following figure, an export receives a request over the HTTP protocol from a client application. The data is transformed into a business object, the format used by the SCA component. The component is then invoked with that data object.

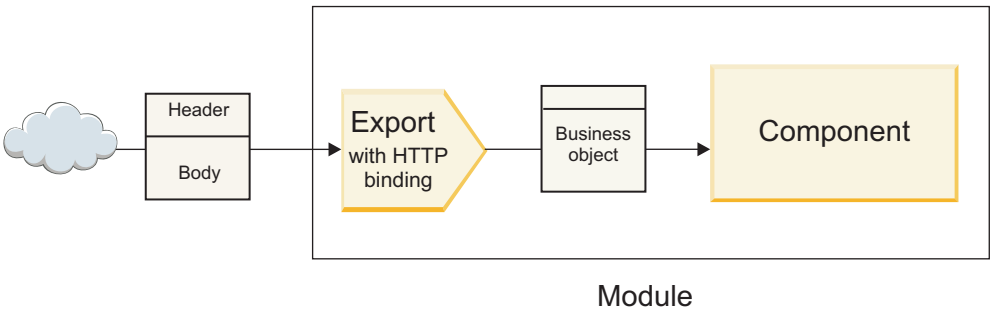


Figure 1. An export with HTTP binding

Imports

An SCA component might want to invoke a non-SCA external service that expects data in a different format. An import is used by the SCA component to invoke the external service using the SCA programming model. The import then invokes the target service in the way that the service expects.

For example, in the following figure, a request from an SCA component is sent, by the import, to an external service. The business object, which is the format used by the SCA component, is transformed to the format expect by the service, and the service is invoked.

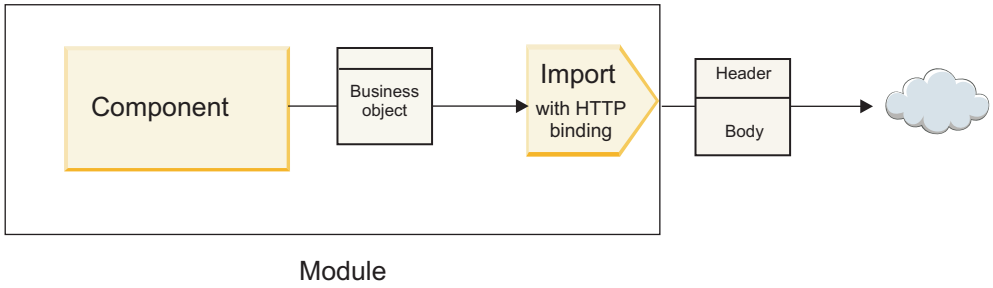


Figure 2. An import with HTTP binding

List of bindings

You use Integration Designer to generate a binding for an import or export and to configure the binding. The types of bindings that are available are described in the following list.

- SCA

The SCA binding, which is the default, lets your service communicate with services in other SCA modules. You use an import with an SCA binding to access a service in another SCA module. You use an export with an SCA binding to offer a service to other SCA modules.

- Web service

  A Web service binding lets you access an external service using interoperable SOAP messages and qualities of service. You can also use Web service bindings to include attachments as part of the SOAP message.

  The Web service binding can use a transport protocol of either SOAP/HTTP (SOAP over HTTP) or SOAP/JMS (SOAP over JMS). Regardless of the transport (HTTP or JMS) used to convey the SOAP messages, Web service bindings always handle request/response interactions synchronously.
- HTTP

  The HTTP binding lets you access an external service using the HTTP protocol, where non-SOAP messages are used, or where direct HTTP access is required. This binding is used when you are working with Web services that are based on the HTTP model (that is, services that use well-known HTTP interface operations such as GET, PUT, DELETE, and so on).
- Enterprise JavaBeans (EJB)

  The EJB bindings let SCA components interact with services provided by Java EE business logic running on a Java EE server.
- EIS

  The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.
- JMS bindings

  Java Message Service (JMS), generic JMS, and WebSphere MQ JMS (MQ JMS) bindings are used for interactions with messaging systems, where asynchronous communication through message queues is critical for reliability.

  An export with one of the JMS bindings watches a queue for the arrival of a message and asynchronously sends the response, if any, to the reply queue. An import with one of the JMS bindings builds and sends a message to a JMS queue and watches a queue for the arrival of the response, if any.

  – JMS

    The JMS binding lets you access the WebSphere-embedded JMS provider.

  – Generic JMS

    The generic JMS binding lets you access a non-IBM vendor messaging system.

  – MQ JMS

    The MQ JMS binding lets you access the JMS subset of a WebSphere MQ messaging system. You would use this binding when the JMS subset of functions is sufficient for your application.
- MQ

  The WebSphere MQ binding lets you communicate with MQ native applications, bringing them into the service oriented architecture framework and providing access to MQ-specific header information. You would use this binding when you need to use MQ native functions.

Export and import binding overview

An export lets you make services in an integration module available to external clients, and an import makes it possible for your SCA components in an integration module to call external services. The binding associated with the export or import specifies the relationship between protocol messages and business objects. It also specifies the way that operations and faults are selected.

Flow of information through an export

An export receives a request, which is intended for the component to which the export is wired, over a specific transport determined by the associated binding (for example, HTTP).

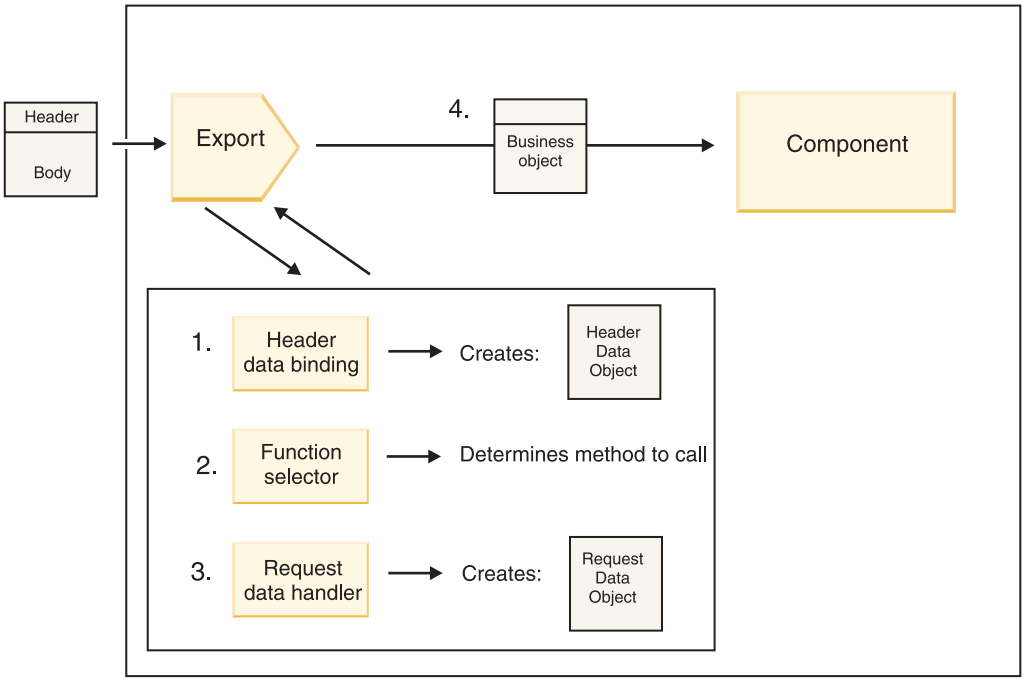


Figure 3. Flow of a request through the export to a component

When the export receives the request, the following sequence of events occurs:

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. The function selector determines the native method name from the protocol message. The native method name is mapped by the export configuration to the name of an operation on the interface of the export.
3. The request data handler or data binding on the method transforms the request to a request business object.
4. The export invokes the component method with the request business object.
   - The HTTP export binding, the Web service export binding, and the EJB export binding invoke the SCA component synchronously.
   - The JMS, Generic JMS, MQ JMS, and WebSphere MQ export bindings invoke the SCA component asynchronously.

Note that an export can propagate the headers and user properties it receives over the protocol, if context propagation is enabled. Components that are wired to the export can then access these headers and user properties. See the “Propagation” topic in the WebSphere Integration Developer information center for more information.

If this is a two-way operation, the component returns a response.

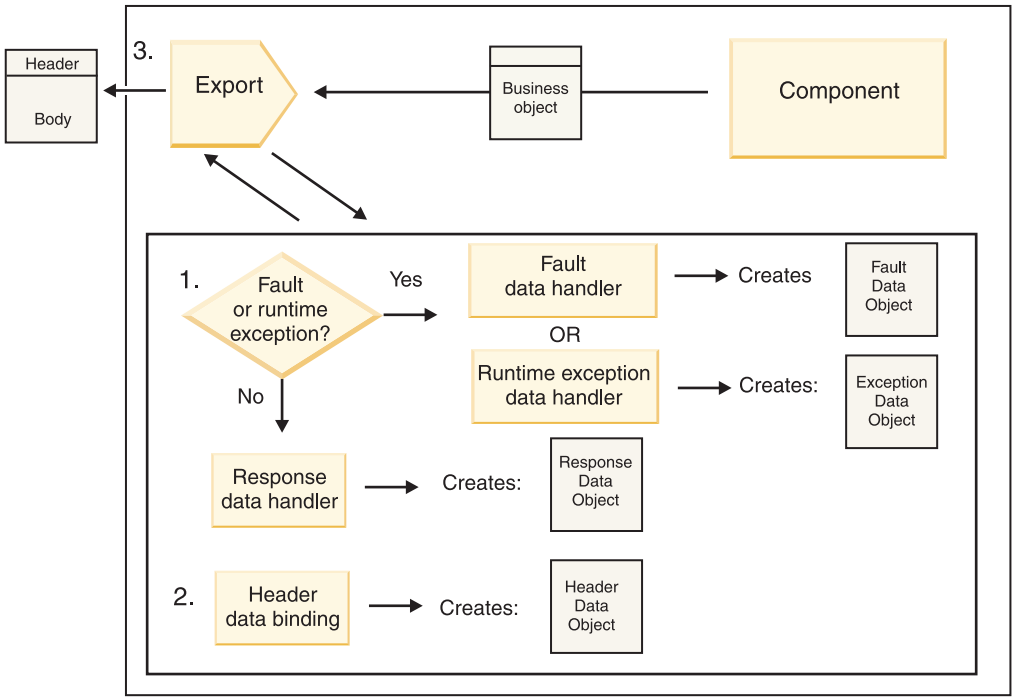


Figure 4. Flow of a response back through the export

The following sequence of steps occurs:

1. If a normal response message is received by the export binding, the response data handler or data binding on the method transforms the business object to a response.

   If the response is a fault, the fault data handler or data binding on the method transforms the fault to a fault response.

   For HTTP export bindings only, if the response is a runtime exception, the runtime exception data handler, if configured, is called.
2. For WebSphere MQ bindings only, the header data binding transforms the header data objects into protocol headers.
3. The export sends the service response over the transport.

Flow of information through an import

Components send requests to services outside the module using an import. The request is sent, over a specific transport determined by the associated binding.

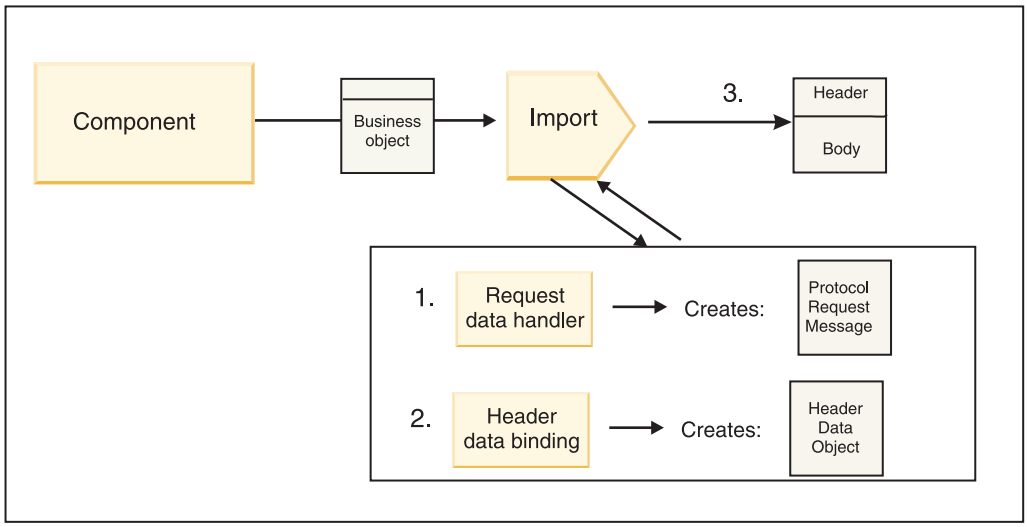


Figure 5. Flow from a component through the import to a service

The component invokes the import with a request business object.

Note:

- The HTTP import binding, the Web service import binding, and the EJB import binding should be invoked synchronously by the calling component.
- The JMS, Generic JMS, MQ JMS, and WebSphere MQ import binding should be invoked asynchronously.

After the component invokes the import, the following sequence of events occurs:

1. The request data handler or data binding on the method transforms the request business object into a protocol request message.
2. For WebSphere MQ bindings only, the header data binding on the method sets the header business object in the protocol header.
3. The import invokes the service with the service request over the transport.

If this is a two-way operation, the service returns a response, and the following sequence of steps occurs:

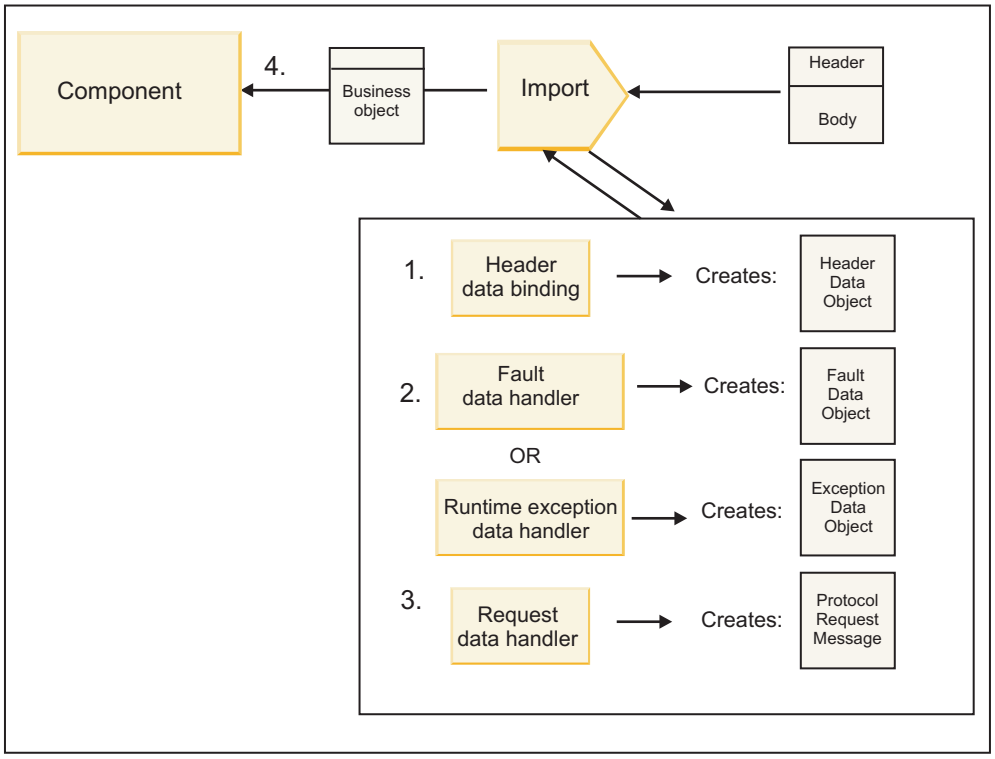


Figure 6. Flow of a response back through the import

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. A determination is made about whether the response is a fault.
   - If the response is a fault, the fault selector inspects the fault to determine which WSDL fault it maps to. The fault data handler on the method then transforms the fault to a fault response.
   - If the response is a runtime exception, the runtime exception data handler, if configured, is called.
3. The response data handler or binding on the method transforms the response to a response business object.
4. The import returns the response business object to the component.

Export and import binding configuration

One of the key aspects of export and import bindings is data format transformation, which indicates how data is mapped (deserialized) from a native wire format to a business object or how it is mapped (serialized) from a business object to a native wire format. For bindings associated with exports, you can also specify a function selector to indicate which operation should be performed on the data. For bindings associated with exports or imports, you can indicate how faults that occur during processing should be handled.

In addition, you specify transport-specific information on bindings. For example, for an HTTP binding, you specify the endpoint URL. For the HTTP binding, the transport-specific information is described in the "Generating an HTTP import binding" and "Generating an HTTP export binding" topics. You can also find information about other bindings in the information center.

Data format transformation in imports and exports:

When an export or import binding is configured in IBM Integration Designer, one of the configuration properties that you specify is the data format used by the binding.

- For export bindings, in which a client application sends requests to and receives responses from an SCA component, you indicate the format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the native data to a business object (which is used by the SCA component) and conversely to transform the business object to native data (which is the response to the client application).
- For import bindings, in which an SCA component sends requests to and receives responses from a service outside the module, you indicate the data format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the business object to native data and vice versa.

IBM Business Process Manager provides a set of predefined data formats and corresponding data handlers or data bindings that support the formats. You can also create your own custom data handlers and register the data format for those data handlers. For more information, see the "Developing data handlers" topic in the IBM Integration Designer information center.

- *Data handlers* are protocol-neutral and transform data from one format to another. In IBM Business Process Manager, data handlers typically transform native data (such as XML, CSV, and COBOL) to a business object and a business object to native data. Because they are protocol-neutral, you can reuse the same data handler with a variety of export and import bindings. For example, you can use the same XML data handler with an HTTP export or import binding or with a JMS export or import binding.
- *Data bindings* also transform native data to a business object (and vice versa), but they are protocol-specific. For example, an HTTP data binding can be used with an HTTP export or import binding only. Unlike data handlers, an HTTP data binding cannot be reused with an MQ export or import binding.

  Note: Three HTTP data bindings (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of IBM Business Process Manager Version 7.0. Use data handlers whenever possible.

As noted earlier, you can create custom data handlers, if necessary. You can also create custom data bindings; however, it is recommended that you create custom data handlers because they can be used across multiple bindings.

Data handlers:

Data handlers are configured against export and import bindings to transform data from one format to another in a protocol-neutral fashion. Several data handlers are provided as part of the product, but you can also create your own data handler, if necessary. You can associate a data handler with an export or import binding at one of two levels: you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

Predefined data handlers

You use IBM Integration Designer to specify the data handler that you want to use.

The data handlers that are predefined for your use are listed in the following table, which also describes how each data handler transforms inbound and outbound data.

Note: Except where noted, these data handlers can be used with JMS, Generic JMS, MQ JMS, WebSphere MQ, and HTTP bindings.
See the "Data handlers" topic in the Integration Designer information center for more detailed information.

Table 18. Predefined data handlers

Data handler	Native data to business object	Business object to native data
ATOM	Parses ATOM feeds into an ATOM feed business object.	Serializes an ATOM feed business object to ATOM feeds.
Delimited	Parses delimited data into a business object.	Serializes a business object to delimited data, including CSV.
Fixed Width	Parses fixed-width data into a business object.	Serializes a business object to fixed-width data.
Handled by WTX	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.
Handled by WTX Invoker	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.
JAXB	Serializes Java beans to a business object using the mapping rules defined by the Java Architecture for XML Binding (JAXB) specification.	Deserializes a business object to Java beans using the mapping rules defined by the JAXB specification.
JAXWS **Note:** The JAXWS data handler can be used only with the EJB binding.	Used by an EJB binding to transform a response Java object or exception Java object to a response business object using the mapping rules defined by the Java API for XML Web Services (JAX-WS) specification.	Used by an EJB binding to transform a business object to the outgoing Java method parameters using the mapping rules defined by the JAX-WS specification.
JSON	Parses JSON data into a business object.	Serializes a business object to JSON data.
Native body	Parses the native bytes, text, map, stream, or object into one of five base business objects (text, bytes, map, stream, or object).	Transforms the five base business objects into byte, text, map, stream, or object.
SOAP	Parses the SOAP message (and the header) into a business object.	Serializes a business object to a SOAP message.
XML	Parses XML data into a business object.	Serializes a business object to XML data.
UTF8XMLDataHandler	Parses UTF-8 encoded XML data into a business object.	Serializes a business object into UTF-8 encoded XML data when sending a message.

Creating a data handler

Detailed information about creating a data handler can be found in the “Developing data handlers” topic in the Integration Designer information center.

Data bindings:

Data bindings are configured against export and import bindings to transform data from one format to another. Data bindings are specific to a protocol. Several data bindings are provided as part of the product, but you can also create your own data binding, if necessary. You can associate a data binding

with an export or import binding at one of two levels–you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

You use IBM Integration Designer to specify which data binding you want to use or to create your own data binding. A discussion of creating data bindings can be found in the "Overview of JMS, MQ JMS and generic JMS bindings" section of the IBM Integration Designer information center.

JMS bindings

The following table lists the data bindings that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

The table also includes a description of the tasks that the data bindings perform.

Table 19. Predefined data bindings for JMS bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object into a business object (which is mapped as the input or output type in the WSDL).	Serializes a business object to the Java serialized object in the JMS object message.
Wrapped bytes	Extracts the bytes from the incoming JMS bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from the JMSBytesBody business object and wraps them into the outgoing JMS bytes message
Wrapped map entry	Extracts the name, value, and type information for every entry in the incoming JMS map message and creates a list of MapEntry business objects. It then wraps the list into the JMSMapBody business object	Extracts the name, value, and type information from the MapEntry list in the JMSMapBody business object and creates the corresponding entries in the outgoing JMS map message.
Wrapped object	Extracts the object from the incoming JMS object message and wraps it into the JMSObjectBody business object.	Extracts the object from the JMSObjectBody business object and wraps it into the outgoing JMS object message.
Wrapped text	Extracts the text from the incoming JMS text message and wraps it into the JMSTextBody business object.	Extracts the text from the JMSTextBody business object and wraps it into the outgoing JMS text message.

WebSphere MQ bindings

The following table lists the data bindings that can be used with WebSphere MQ and describes the tasks that the data bindings perform.

Table 20. Predefined data bindings for WebSphere MQ bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object from the incoming message into a business object (which is mapped as the input or output type in the WSDL).	Transforms a business object to the Java serialized object in the outgoing message

Table 20. Predefined data bindings for WebSphere MQ bindings (continued)

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the unstructured MQ bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from a JMSBytesBody business object and wraps the bytes into the outgoing unstructured MQ bytes message.
Wrapped text	Extracts the text from an unstructured MQ text message and wraps it into a JMSTextBody business object.	Extracts text from a JMSTextBody business object and wraps it in an unstructured MQ text message.
Wrapped stream entry	Extracts the name and type information for every entry in the incoming JMS stream message and creates a list of the StreamEntry business objects. It then wraps the list into the JMSStreamBody business object.	Extracts the name and type information from the StreamEntry list in the JMSStreamBody business object and creates corresponding entries in the outgoing JMSStreamMessage.

In addition to the data bindings listed in Table 20 on page 62, WebSphere MQ also uses header data bindings. See the IBM Integration Designer information center for details.

HTTP bindings

The following table lists the data bindings that can be used with HTTP and describes the tasks that the data bindings perform.

Table 21. Predefined data bindings for HTTP bindings

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the body of the incoming HTTP message and wraps them into the HTTPBytes business object.	Extracts the bytes from the HTTPBytes business object and adds them to the body of the outgoing HTTP message.
Wrapped text	Extracts the text from the body of the incoming HTTP message and wraps it into the HTTPText business object.	Extracts the text from the HTTPText business object and adds it to the body of the outgoing HTTP message.

Function selectors in export bindings:

A function selector is used to indicate which operation should be performed on the data for a request message. Function selectors are configured as part of an export binding.

Consider an SCA export that exposes an interface. The interface contains two operations–Create and Update. The export has a JMS binding that reads from a queue.

When a message arrives on the queue, the export is passed the associated data, but which operation from the export's interface should be invoked on the wired component? The operation is determined by the function selector and the export binding configuration.

The function selector returns the native function name (the function name in the client system that sent the message). The native function name is then mapped to the operation or function name on the interface associated with the export. For example, in the following figure, the function selector returns the native function name (CRT) from the incoming message, the native function name is mapped to the Create operation, and the business object is sent to the SCA component with the Create operation.

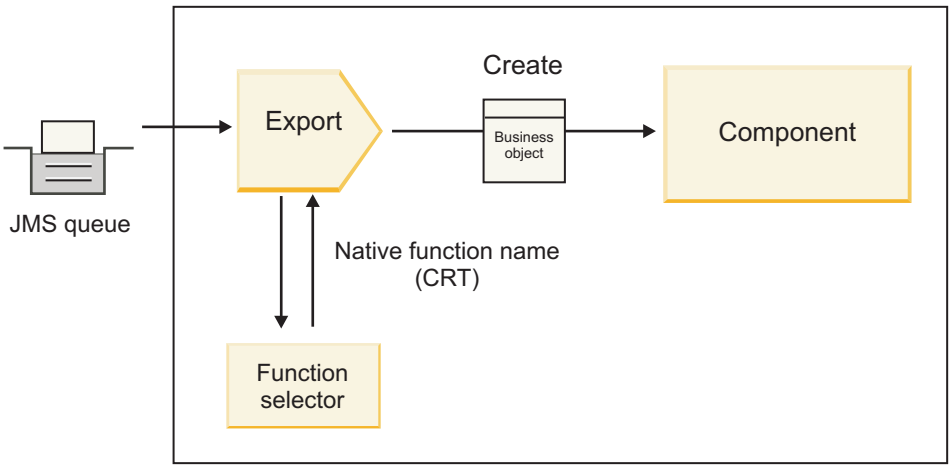


Figure 7. The function selector

If the interface has only one operation, there is no need to specify a function selector.

Several prepackaged function selectors are available and are listed in the sections that follow.

JMS bindings

The following table lists the function selectors that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

Table 22. Predefined function selectors for JMS bindings

Function selector	Description
JMS function selector for simple JMS data bindings	Uses the JMSType property of the message to select the operation name.
JMS header property function selector	Returns the value of the JMS String Property, TargetFunctionName, from the header.
JMS service gateway function selector	Determines if the request is a one-way or two-way operation by examining the JMSReplyTo property set by the client.

WebSphere MQ bindings

The following table lists the function selectors that can be used with WebSphere MQ bindings.

Table 23. Predefined function selectors for WebSphere MQ bindings

Function selector	Description
MQ handleMessage function selector	Returns handleMessage as a value, which is mapped using the export method bindings to the name of an operation on the interface.
MQ uses JMS default function selector	Reads the native operation from the TargetFunctionName property of the folder of an MQRFH2 header.
MQ uses message body format as native function	Finds the Format field of the last header and returns that field as a String.

Table 23. Predefined function selectors for WebSphere MQ bindings (continued)

Function selector	Description
MQ type function selector	Creates a method in your export binding by retrieving a URL containing the Msd, Set, Type and Format properties found in the MQRFH2 header.
MQ service gateway function selector	Uses the MsgType property in the MQMD header to determine the operation name.

HTTP bindings

The following table lists the function selectors that can be used with HTTP bindings.

Table 24. Predefined function selectors for HTTP bindings

Function selector	Description
HTTP function selector based on the TargetFunctionName header	Uses the TargetFunctionName HTTP header property from the client to determine which operation to invoke at runtime from the export.
HTTP function selector based on the URL and HTTP method	Uses the relative path from the URL appended with the HTTP method from the client to determine the native operation defined on the export.
HTTP service gateway function selector based on URL with an operation name	Determines the method to invoke based on the URL if "operationMode = oneway" has been appended to the request URL.

Note: You can also create your own function selector, using IBM Integration Designer. Information about creating a function selector is provided in the IBM Integration Designer information center. For example, a description of creating a function selector for WebSphere MQ bindings can be found in “Overview of the MQ function selectors”.

Fault handling:

You can configure your import and export bindings to handle faults (for example, business exceptions) that occur during processing by specifying fault data handlers. You can set up a fault data handler at three levels–you can associate a fault data handler with a fault, with an operation, or for all operations with a binding.

A fault data handler processes fault data and transforms it into the correct format to be sent by the export or import binding.

- For an export binding, the fault data handler transforms the exception business object sent from the component to a response message that can be used by the client application.
- For an import binding, the fault data handler transforms the fault data or response message sent from a service into an exception business object that can be used by the SCA component.

For import bindings, the binding calls the fault selector, which determines whether the response message is a normal response, a business fault, or a runtime exception.

You can specify a fault data handler for a particular fault, for an operation, and for all operations with a binding.

- If the fault data handler is set at all three levels, the data handler associated with a particular fault is called.
- If fault data handlers are set at the operation and binding levels, the data handler associated with the operation is called.

Two editors are used in IBM Integration Designer to specify fault handling. The interface editor is used to indicate whether there will be a fault on an operation. After a binding is generated with this interface, the editor in the properties view lets you configure how the fault will be handled. For more information, see the “Fault selectors” topic in the IBM Integration Designer information center.

How faults are handled in export bindings:

When a fault occurs during the processing of the request from a client application, the export binding can return the fault information to the client. You configure the export binding to specify how the fault should be processed and returned to the client.

You configure the export binding using IBM Integration Designer.

During request processing, a client invokes an export with a request, and the export invokes the SCA component. During the processing of the request, the SCA component can either return a business response or can throw a service business exception or a service runtime exception. When this occurs, the export binding transforms the exception into a fault message and sends it to the client, as shown in the following figure and described in the sections that follow.

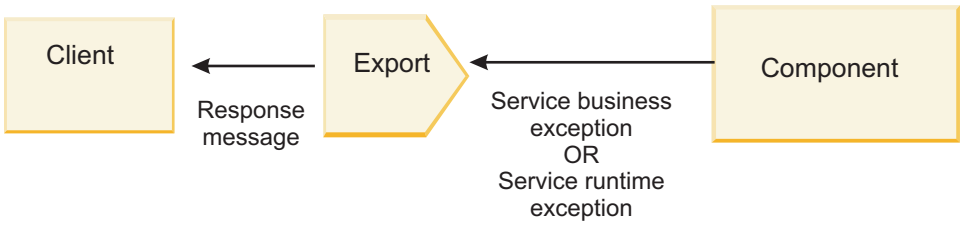


Figure 8. How fault information is sent from the component through the export binding to the client

You can create a custom data handler or data binding to handle faults.

Business faults

Business faults are business errors or exceptions that occur during processing.

Consider the following interface, which has a createCustomer operation on it. This operation has two business faults defined: CustomerAlreadyExists and MissingCustomerId.

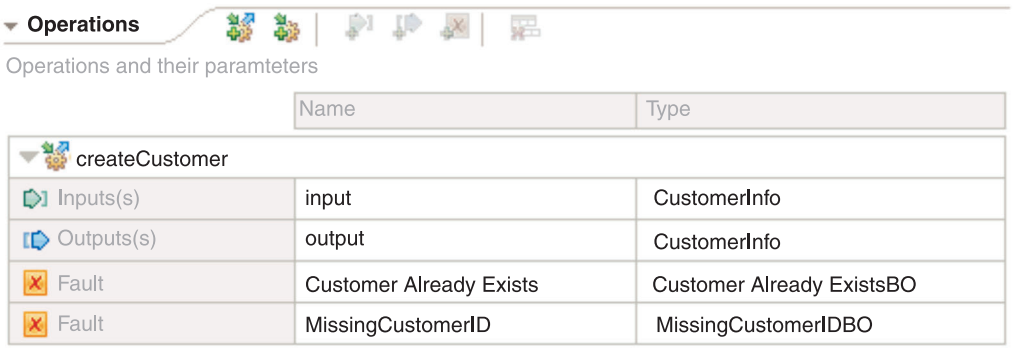


Figure 9. Interface with two faults

In this example, if a client sends a request to create a customer (to this SCA component) and that customer already exists, the component throws a CustomerAlreadyExists fault to the export. The export needs to propagate this business fault back to the calling client. To do so, it uses the fault data handler that is set up on the export binding.

When a business fault is received by the export binding, the following processing occurs:

1. The binding determines which fault data handler to invoke for handling the fault. If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.
2. The binding calls the fault data handler with the data object from the service business exception.
3. The fault data handler transforms the fault data object to a response message and returns it to the export binding.
4. The export returns the response message to the client.

If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.

Runtime exceptions

A runtime exception is an exception that occurs in the SCA application during the processing of a request that does not correspond to a business fault. Unlike business faults, runtime exceptions are not defined on the interface.

In certain scenarios, you might want to propagate these runtime exceptions to the client application so that the client application can take the appropriate action.

For example, if a client sends a request (to the SCA component) to create a customer and an authorization error occurs during processing of the request, the component throws a runtime exception. This runtime exception has to be propagated back to the calling client so it can take the appropriate action regarding the authorization. This is achieved by the runtime exception data handler configured on the export binding.

Note: You can configure a runtime exception data handler only on HTTP bindings.

The processing of a runtime exception is similar to the processing of a business fault. If a runtime exception data handler was set up, the following processing occurs:

1. The export binding calls the appropriate data handler with the service runtime exception.
2. The data handler transforms the fault data object to a response message and returns it to the export binding.
3. The export returns the response message to the client.

Fault handling and runtime exception handling are optional. If you do not want to propagate faults or runtime exceptions to the calling client, do not configure the fault data handler or runtime exception data handler.

How faults are handled in import bindings:

A component uses an import to send a request to a service outside the module. When a fault occurs during the processing of the request, the service returns the fault to the import binding. You can configure the import binding to specify how the fault should be processed and returned to the component.

You configure the import binding using IBM Integration Designer. You can specify a fault data handler (or data binding), and you also specify a fault selector.

Fault data handlers

The service that processes the request sends, to the import binding, fault information in the form of an exception or a response message that contains the fault data.

The import binding transforms the service exception or response message into a service business exception or service runtime exception, as shown in the following figure and described in the sections that follow.

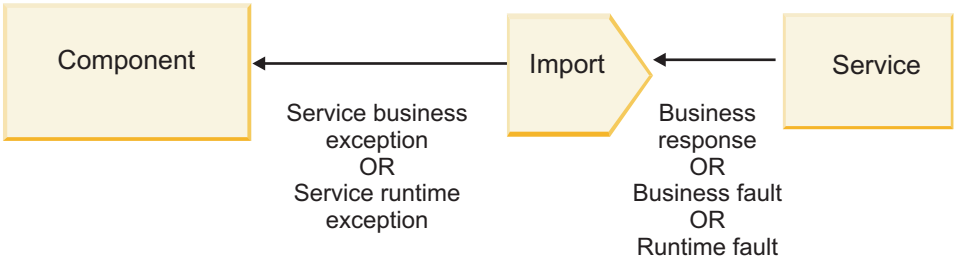


Figure 10. How fault information is sent from the service through the import to the component

You can create a custom data handler or data binding to handle faults.

Fault selectors

When you configure an import binding, you can specify a fault selector. The fault selector determines whether the import response is an actual response, a business exception, or a runtime fault. It also determines, from the response body or header, the native fault name, which is mapped by the binding configuration to the name of a fault in the associated interface.

Two types of prepackaged fault selectors are available for use with JMS, MQ JMS, Generic JMS, WebSphere MQ, and HTTP imports:

Table 25. Prepackaged fault selectors

Fault selector type	Description
Header-based	Determines whether a response message is a business fault, a runtime exception, or a normal message based on the headers in the incoming response message.
SOAP	Determines whether the response SOAP message is a normal response, business fault, or runtime exception.

The following shows examples of header-based fault selectors and the SOAP fault selector.

- Header-based fault selector

  If an application wants to indicate that the incoming message is a business fault, there must be two headers in the incoming message for business faults, which is shown as follows:

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  If an application wants to indicate that the incoming response message is a runtime exception, then there must be one header in the incoming message, which is shown as follows:

  ```
  Header name = FaultType, Header value = Runtime
  ```

- SOAP fault selector

  A business fault can be sent as part of the SOAP message with the following custom SOAP header. "CustomerAlreadyExists" is the name of the fault in this case.

```
<ibmSoap:BusinessFaultName
xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
<ibmSoap:BusinessFaultName>
```

The fault selector is optional. If you do not specify a fault selector, the import binding cannot determine the type of response. The binding therefore treats it as a business response and calls the response data handler or data binding.

You can create a custom fault selector. The steps for creating a custom fault selector are provided in the "Developing a custom fault selector" topic of the IBM Integration Designer information center.

Business faults

A business fault can occur when there is an error in the processing of a request. For example, if you send a request to create a customer and that customer already exists, the service sends a business exception to the import binding.

When a business exception is received by the binding, the processing steps depend on whether a fault selector has been set up for the binding.

- If no fault selector was set up, the binding calls the response data handler or data binding.
- If a fault selector was set up, the following processing occurs:
  1. The import binding calls the fault selector to determine whether the response is business fault, business response, or runtime fault.
  2. If the response is a business fault, the import binding calls the fault selector to provide the native fault name.
  3. The import binding determines the WSDL fault corresponding to the native fault name returned by the fault selector.
  4. The import binding determines the fault data handler that is configured for this WSDL fault.
  5. The import binding calls this fault data handler with the fault data.
  6. The fault data handler transforms the fault data to a data object and returns it to the import binding.
  7. The import binding constructs a service business exception object with the data object and the fault name.
  8. The import returns the service business exception object to the component.

Runtime exceptions

A runtime exception can occur when there is a problem in communicating with the service. The processing of a runtime exception is similar to the processing of a business exception. If a fault selector was set up, the following processing occurs:

1. The import binding calls the appropriate runtime exception data handler with the exception data.
2. The runtime exception data handler transforms the exception data to a service runtime exception object and returns it to the import binding.
3. The import returns the service runtime exception object to the component.

Interoperability between SCA modules and Open SCA services

The IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) provides a simple, yet powerful programming model for constructing applications based on the Open SCA specifications. The SCA modules of IBM Business Process Manager use import and export bindings to interoperate with Open SCA services developed in a Rational® Application Developer environment and hosted by the WebSphere Application Server Feature Pack for Service Component Architecture.

An SCA application invokes an Open SCA application by way of an import binding. An SCA application receives a call from an Open SCA application by way of an export binding. A list of supported bindings is shown in “Invoking services over interoperable bindings” on page 71.

Invoking Open SCA services from SCA modules

SCA applications developed with IBM Integration Designer can invoke Open SCA applications developed in a Rational Application Developer environment. This section provides an example of invoking an Open SCA service from an SCA module using an SCA import binding.

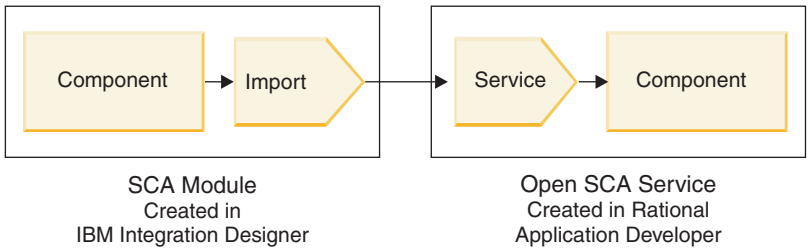


Figure 11. Component in SCA module invoking Open SCA service

No special configuration is required to invoke an Open SCA service.

To connect to an Open SCA service by way of an SCA import binding, you provide the component name and service name of the Open SCA service in the import binding.

1. To obtain the name of the target component and service from the Open SCA composite, perform the following steps:
   a. Ensure that the **Properties** tab is open by clicking **Window** > **Show View** > **Properties**.
   b. Open the composite editor by double-clicking the composite diagram that contains the component and service. For example, for a component named **customer**, the composite diagram is **customer.composite_diagram**.
   c. Click the target component.
   d. In the **Name** field of the **Properties** tab, note the name of the target component.
   e. Click the service icon associated with the component.
   f. In the **Name** field of the **Properties** tab, note the name of the service.
2. To configure the IBM Business Process Manager import to connect it to the Open SCA service, perform the following steps:
   a. In IBM Integration Designer, navigate to the **Properties** tab of the SCA import that you want to connect to the Open SCA service.
   b. In the **Module name** field, enter the component name from step 1d.
   c. In the **Export name** field, enter the service name from step 1f.
   d. Save your work by pressing Ctrl+S.

Invoking SCA modules from Open SCA services

Open SCA applications developed in a Rational Application Developer environment can invoke SCA applications developed with IBM Integration Designer. This section provides an example of invoking an SCA module (by way of an SCA export binding) from an Open SCA service.

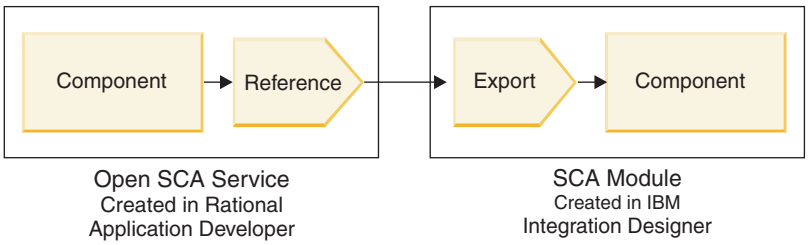


Figure 12. Open SCA service invoking component in SCA module

To connect to an SCA component by way of an Open SCA reference binding, you provide the module name and export name.

1. To obtain the name of the target module and export, perform the following steps:
   a. In IBM Integration Designer, open the module in the assembly editor by double-clicking the module.
   b. Click the export.
   c. In the **Name** field of the **Properties** tab, note the name of the export.
2. Configure the Open SCA reference that you want to connect to the IBM Business Process Manager module and export:
   a. In Rational Application Developer, open the composite editor by double-clicking the composite diagram that contains the component and service.
   b. Click the reference icon of the component reference to display the reference properties in the **Properties** tab.
   c. Click the **Binding** tab on the left side of the page.
   d. Click **Bindings** and then click **Add**.
   e. Select the **SCA** binding.
   f. In the **Uri** field, enter the IBM Business Process Manager module name, followed by a slash ("/"), followed by the export name (which you determined in step 1c).
   g. Click **OK**.
   h. Save your work by pressing Ctrl+S.

Invoking services over interoperable bindings

The following bindings are supported for interoperability with an Open SCA service.

- SCA binding

  In IBM Business Process Manager, when an SCA module invokes an Open SCA service by way of an SCA import binding, the following invocation styles are supported:

  – Asynchronous (one-way)
  – Synchronous (request/response)

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  Note that the SCA binding supports transaction and security context propagation.
- Web service (JAX-WS) binding with either the SOAP1.1/HTTP or SOAP1.2/HTTP protocol

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  In addition, the following qualities of service are supported:

  – Web Services Atomic Transaction
  – Web Services Security

- EJB binding

  A Java interface is used to define the interaction between an SCA module and an Open SCA service when the EJB binding is used.

  Note that the EJB binding supports transaction and security context propagation.
- JMS bindings

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  The following JMS providers are supported:
  - WebSphere Platform Messaging (JMS Binding)
  - WebSphere MQ (MQ JMS Binding)

Note: Business Graphs are not interoperable across any SCA bindings and, therefore, are not supported in interfaces used to interoperate with the WebSphere Application Server Feature Pack for Service Component Architecture.

Binding types

You use protocol-specific *bindings* with imports and exports to specify the means of transporting data into or out of a module.

Selecting appropriate bindings:

When you are creating an application, you need to know how to select the binding that is most appropriate to the needs of your application.

The bindings available in IBM Integration Designer provide a range of choices. In this list, you can determine which type of binding might be most suitable for the needs of your application.

Consider a *Service Component Architecture (SCA)* binding when these factors are applicable:
- All services are contained in modules; that is, there are no external services.
- You want to separate function into different SCA modules that interact directly with each other.
- The modules are tightly coupled.

Consider a *Web Service* 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.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.
- The protocol of the external service you are accessing or the service you want to provide is SOAP/HTTP or SOAP/JMS.

Consider an *HTTP* binding when these factors are applicable:
- You need to access an external service over the Internet or provide a service over the Internet, and you are working with other web services such as GET, PUT, and DELETE.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise JavaBeans (EJB)* binding when these factors are applicable:
- The binding is for an imported service that is itself an EJB or that needs to be accessed by EJB clients.
- The imported service is loosely coupled.
- Stateful EJB interactions are not required.

- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise Information Systems (EIS)* binding when these factors are applicable:
- You need to access a service on an EIS system using a resource adapter.
- Synchronous data transmission is preferred over asynchronous.

Consider a *Java Message Service (JMS)* binding when these factors are applicable:

Important: There are several types of JMS bindings. If you expect to exchange SOAP messages using JMS, consider the Web service binding with the SOAP/JMS protocol. See "Web service bindings" on page 74.
- You need to access a messaging system.
- The services are loosely coupled.
- Asynchronous data transmission is preferred over synchronous.

Consider a *Generic Java Message Service (JMS)* binding when these factors are applicable:
- You need to access a non-IBM vendor messaging system.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *Message Queue (MQ)* binding when these factors are applicable:
- You need to access a WebSphere MQ messaging system and need to use the MQ native functions.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *MQ JMS* binding when these factors are applicable:
- You need to access a WebSphere MQ messaging system but can do so within a JMS context; that is, the JMS subset of functions is sufficient for your application.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

SCA bindings:

A Service Component Architecture (SCA) binding lets a service communicate with other services in other modules. An import with an SCA binding lets you access a service in another SCA module. An export with an SCA binding lets you offer a service to other modules.

You use IBM Integration Designer to generate and configure SCA bindings on imports and exports in SCA modules.

If modules are running on the same server or are deployed in the same cluster, an SCA binding is the easiest and fastest binding to use.

After the module that contains the SCA binding is deployed to the server, you can use the administrative console to view information about the binding or, in the case of an import binding, to change selected properties of the binding.

Web service bindings:

A Web service binding is the means of transmitting messages from a Service Component Architecture (SCA) component to a Web service (and vice versa).

Web service bindings overview:

A Web service import binding allows you to call an external Web service from your Service Component Architecture (SCA) components. A Web service export binding allows you to expose your SCA components to clients as Web services.

With a Web service binding, you access external services using interoperable SOAP messages and qualities of service (QoS).

You use Integration Designer to generate and configure Web service bindings on imports and exports in SCA modules. The following types of Web service bindings are available:

- SOAP1.2/HTTP and SOAP1.1/HTTP

  These bindings are based on Java API for XML Web Services (JAX-WS), a Java programming API for creating Web services.
  - Use SOAP1.2/HTTP if your Web service conforms to the SOAP 1.2 specification.
  - Use SOAP1.1/HTTP if your Web service conforms to the SOAP 1.1 specification.

  Important: When you deploy an application with a Web service (JAX-WS) binding, the target server must not have the **Start components as needed** option selected. See "Checking the server configuration" on page 82 for details.

  When you select one of these bindings, you can send attachments with your SOAP messages.

  The Web service bindings work with standard SOAP messages. Using one of the Web service JAX-WS bindings, however, you can customize the way that SOAP messages are parsed or written. For example, you can handle nonstandard elements in SOAP messages or apply additional processing to the SOAP message. When you configure the binding, you specify a custom data handler that performs this processing on the SOAP message.

  You can use policy sets with a Web service (JAX-WS) binding. A policy set is a collection of policy types, each of which provides a quality of service (QoS). For example, the WSAddressing policy set provides a transport-neutral way to uniformly address Web services and messages. You use Integration Designer to select the policy set for the binding.

  Note: If you want to use a Security Assertion Markup Language (SAML) policy set, you must perform some additional configuration, as described in "Importing SAML policy sets" on page 80.
- SOAP1.1/HTTP

  Use this binding if you want to create Web services that use a SOAP-encoded message based on Java API for XML-based RPC (JAX-RPC).
- SOAP1.1/JMS

  Use this binding to send or receive SOAP messages using a Java Message Service (JMS) destination.

Regardless of the transport (HTTP or JMS) that is used to convey the SOAP message, Web service bindings always handle request/response interactions synchronously. The thread making the invocation on the service provider is blocked until a response is received from the provider. See "Synchronous invocation" for more information about this invocation style.

Important: The following combinations of Web service bindings cannot be used on exports in the same module. If you need to expose components using more than one of these export bindings, you need to have each in a separate module and then connect those modules to your components using the SCA binding:

- SOAP 1.1/JMS and SOAP 1.1/HTTP using JAX-RPC
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.1/HTTP using JAX-WS
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.2/HTTP using JAX-WS

After the SCA module that contains the Web service binding is deployed to the server, you can use the administrative console to view information about the binding or to change selected properties of the binding.

Note: Web services allow applications to interoperate by using standard descriptions of services and standard formats for the messages they exchange. For example, the Web service import and export bindings can interoperate with services that are implemented using Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET. When interoperating with such services, you must ensure that:

- The Web Services Description Language (WSDL) file that is used to access a Web service export includes a non-empty SOAP action value for each operation in the interface.
- The Web service client sets either the SOAPAction header or the wsa:Action header when sending messages to a Web service export.

SOAP header propagation:

When handling SOAP messages, you might need to access information from certain SOAP headers in messages that are received, ensure that messages with SOAP headers are sent with specific values, or allow SOAP headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want SOAP headers to be propagated.

- When requests are received at an export or responses received at an import, the SOAP header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When requests are sent from an export or responses sent from an import, SOAP headers can be included in those messages.

The form and presence of the propagated SOAP headers might be affected by policy sets configured on the import or export, as explained in Table 26 on page 76.

To configure the propagation of SOAP headers for an import or export, you select (from the Properties view of Integration Designer) the **Propagate Protocol Header** tab and select the options you require.

WS-Addressing header

The WS-Addressing header can be propagated by the Web service (JAX-WS) binding.

When you propagate the WS-Addressing header, be aware of the following information:

- If you enable propagation for the WS-Addressing header, the header will be propagated into the module in the following circumstances:
  - When requests are received at an export
  - When responses are received at an import
- The WS-Addressing header is not propagated into outbound messages from IBM Business Process Manager (that is, the header is not propagated when requests are sent from an import or when responses are sent from the export).

WS-Security header

The WS-Security header can be propagated by both the Web service (JAX-WS) binding and the Web service (JAX-RPC) binding.

The Web services WS-Security specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

When you propagate the WS-Security header, be aware of the following information:

- If you enable propagation for the WS-Security header, the header will be propagated across the module in the following circumstances:
  - When requests are received at an export
  - When requests are sent from an import
  - When responses are received at an import
- The header will *not*, by default, be propagated when responses are sent from the export. However, if you set the JVM property **WSSECURITY.ECHO.ENABLED** to **true**, the header will be propagated when responses are sent from the export. In this case, if the WS-Security header on the request path is not modified, WS-Security headers might be automatically echoed from requests into responses.
- The exact form of the SOAP message sent from an import for a request or from an export for a response might not exactly match the SOAP message that was originally received. For this reason, any digital signature should be assumed to become invalid. If a digital signature is required in messages that are sent, it must be established using the appropriate security policy set, and WS-Security headers relating to digital signature in received messages should be removed within the module.

To propagate the WS-Security header, you must include the WS-Security schema with the application module. See "Including the WS-Security schema in an application module" on page 77 for the procedure to include the schema.

How headers are propagated

The way that headers are propagated depends on the security policy setting on the import or export binding, as shown in the Table 26:

Table 26. How security headers are passed

	Export binding with no security policy	**Export binding with security policy**
Import binding with no security policy	Security headers are passed as-is through the module. They are not decrypted. The headers are sent outbound in the same form in which they were received. The digital signature might become invalid.	Security headers are decrypted and passed through the module with signature verification and authentication. The decrypted headers are sent outbound. The digital signature might become invalid.

Table 26. How security headers are passed (continued)

	Export binding with no security policy	**Export binding with security policy**
Import binding with security policy	Security headers are passed as-is through the module. They are not decrypted. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.	Security headers are decrypted and passed through the module with signature verification and authentication. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.

Configure appropriate policy sets on the export and import bindings, because this isolates the service requester from changes to the configuration or QoS requirements of the service provider. Having standard SOAP headers visible in a module can then be used to influence the processing (for example, logging and tracing) in the module. Propagating SOAP headers across a module from a received message to a sent message does mean that the isolation benefits of the module are reduced.

Standard headers, such as WS-Security headers, should not be propagated on a request to an import or response to an export when the import or export has an associated policy set that would normally result in the generation of those headers. Otherwise, an error will occur because of a duplication of the headers. Instead, the headers should be explicitly removed or the import or export binding should be configured to prevent propagation of protocol headers.

Accessing SOAP headers

When a message that contains SOAP headers is received from a Web service import or export, the headers are placed in the headers section of the service message object (SMO). You can access the header information, as described in "Accessing SOAP header information in the SMO".

Including the WS-Security schema in an application module

The following procedure outlines the steps for including the schema in the application module:

- If the computer on which Integration Designer is running has access to the Internet, perform the following steps:
  1. In the Business Integration perspective, select **Dependencies** for your project.
  2. Expand **Predefined Resources** and select either **WS-Security 1.0 schema files** or **WS-Security 1.1 schema files** to import the schema into your module.
  3. Clean and rebuild the project.
- If a computer on which Integration Designer is running does not have Internet access, you can download the schema to a second computer that does have Internet access. You can then copy it to the computer on which Integration Designer is running.
  1. From the computer that has Internet access, download the remote schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Remote WSDL** or **XSD file**.
     c. Import the following schemas:

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. Copy the schemas to the computer that does not have Internet access.
  3. From the computer that has no Internet access, import the schema:

a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
b. Select **Local WSDL** or **XSD file**.

4. Change the schema locations for oasis-wss-wssecurity_secext-1.1.xsd:
   a. Open the schema at *workplace_location/module_name*/StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd.
   b. Change:

```
<xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
```

   to:

```
<xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
```

   c. Change:

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
```

   to:

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
```

5. Change the schema location for oasis-200401-wss-wssecurity-secext-1.0.xsd:
   a. Open the schema at *workplace_location/module_name*/StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd.
   b. Change:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
```

   to:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
```

6. Clean and rebuild the project.

Transport header propagation:

When handling SOAP messages, you might need to access information from certain transport headers in messages that are received, ensure that messages with transport headers are sent with specific values, or allow transport headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want transport headers to be propagated.

- When requests are received at an export or responses are received at an import, the transport header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When responses are sent from an export or requests are sent from an import, transport headers can be included in those messages.

Specifying propagation of headers

To configure the propagation of transport headers for an import or export, perform the following steps:

1. From the Properties view of Integration Designer, select **Binding** > **Propagation**.
2. Set the transport header propagation option that you require.

Note: Transport header propagation is disabled by default and can be deployed only to a Version 7.0.0.3 (or later) runtime environment. Also note that, for Version 7.0.0.3, transport header propagation is limited to HTTP transport headers only.

If you enable propagation of transport headers, the headers will be propagated across a module from received messages and, if you do not explicitly remove the headers, the headers will be used in subsequent invocations in the same thread.

Note: Transport headers cannot be propagated when you are using the Web service (JAX-RPC) binding.

Accessing the header information

When transport header propagation is enabled for received messages, all transport headers (including customer-defined headers) are visible in the service message object (SMO). You can set the headers to different values or create new ones. Note, however, that there is no checking or validation of the values you set, and any improper or incorrect headers might cause Web service runtime problems.

Consider the following information about the setting of HTTP headers:

- Any changes to the headers that are reserved for the Web service engine will not be honored in the outbound message. For example, the HTTP version or method, Content-Type, Content-Length and SOAPAction headers are reserved for the Web service engine.
- If the header value is a number, the number (rather than the string) should be set directly. For example, use **Max-Forwards = 5** (rather than **Max-Forwards = Max-Forwards: 5**) and **Age = 300** (rather than **Age = Age: 300**).
- If the request message is less than 32 KB in size, the Web service engine removes the Transfer-Encoding header and instead sets the Content-Length header to the fixed size of the message.
- The Content-language is reset by WAS.channel.http on the response path.
- An invalid setting for Upgrade results in a 500 error.
- The following headers append the value reserved by the Web service engine to the customer settings:
  - User-Agent
  - Cache-Control
  - Pragma
  - Accept
  - Connection

You can access the header information in one of the following ways:

- Using a mediation primitive to access the SMO structures

  See the “Related Information” links to find information about using mediation primitives.
- Using the context service SPI

  The following sample code reads the HTTP transport headers from the context service:

```
HeadersType headerType = ContextService.INSTANCE.getHeaders();
HTTPHeaderType httpHeaderType = headerType.getHTTPHeader();
List HTTPHeader  httpHeaders = httpHeaderType.getHeader();
if(httpHeaders!=null){
 for(HTTPHeader httpHeader: httpHeaders){
   String httpHeadername = httpHeader.getName();
   String httpHeaderValue = httpHeader.getValue();
     }
}
List PropertyType  properties = headerType.getProperties();
if(properties!=null){
  for(PropertyType property: properties){
   String propertyName = property.getName();
   String propertyValue = property.getValue().toString();
     }
}
```

Troubleshooting

If you encounter problems when sending the revised headers, you can intercept the TCP/IP message by using tools such as the TCP/IP Monitor in Integration Designer. You access the TCP/IP Monitor by selecting **Run/Debug** > **TCP/IP Monitor** from the Preferences page.

You can also view the header values using the JAX-WS engine trace: **org.apache.axis2.*=all: com.ibm.ws.websvcs.*=all:**

Working with Web service (JAX-WS) bindings:

When you use Web service (JAX-WS) bindings with your applications, you can add a Security Assertion Markup Language (SAML) quality of service (QOS) to the binding. You must first use the administrative console to import the policy set. You can also use the administrative console to make sure that the server is properly configured for use with the Web service (JAX-WS) binding.

Importing SAML policy sets:

Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. When you configure a web service (JAX-WS) binding in Integration Designer, you can specify an SAML policy set. You first use the administrative console of IBM Business Process Manager to make the SAML policy sets available so that they can be imported into Integration Designer.

The SAML policy sets are typically located in the profile configuration directory:

profile_root/config/templates/PolicySets

Before you begin this procedure, verify that the following directories (which contain the policy sets) are located in the profile configuration directory:

- SAML11 Bearer WSHTTPS default
- SAML20 Bearer WSHTTPS default
- SAML11 Bearer WSSecurity default
- SAML20 Bearer WSSecurity default
- SAML11 HoK Public WSSecurity default
- SAML20 HoK Public WSSecurity default
- SAML11 HoK Symmetric WSSecurity default
- SAML20 HoK Symmetric WSSecurity default
- Username WSHTTPS default

If the directories are not in the profile configuration directory, copy them to that directory from the following location:

app_server_root/profileTemplates/default/documents/config/templates/PolicySets

You import the policy sets into the administrative console, select the ones you want to make available to Integration Designer, and then save a .zip file for each of those policy sets to a location that is accessible by Integration Designer.

1. Import the policy sets by following these steps:
   a. From the administrative console, click **Services** > **Policy Sets** > **Application policy sets**.
   b. Click **Import** > **From Default Repository**.
   c. Select the SAML default policy sets, and click **OK**.
2. Export the policy sets so that they can be used by Integration Designer:

a. From the Application policy sets page, select the SAML policy set you want to export, and click **Export**.

   Note: If the Application policy sets page is not currently displayed, click **Services** > **Policy Sets** > **Application Policy Sets** from the administrative console.

b. On the next page, click the .zip file link for the policy set.

c. In the File Download window, click **Save** and indicate a location that is accessible by Integration Designer.

d. Click **Back**.

e. Complete steps 2a through 2d for each policy set you want to export.

The SAML policy sets are saved in .zip files and are ready to be imported into Integration Designer.

Import the policy sets into Integration Designer, as described in the topic "Policy sets".

Invoking web services that require HTTP basic authentication:

HTTP basic authentication employs a user name and password to authenticate a service client to a secure endpoint. You can set up HTTP basic authentication when sending or receiving web service requests.

You set up HTTP basic authentication for receiving web service requests by configuring the Java API for XML Web Services (JAX-WS) export binding, as described in the Creating and assigning security roles to web service exports.

HTTP basic authentication can be enabled for web service requests that are sent by a JAX-WS import binding in one of two ways:

- When configuring the import binding in an SCA module, you can select the supplied HTTP authentication policy set named BPMHTTPBasicAuthentication (which is provided with the web service (JAX-WS) import binding) or any other policy set that includes the HTTPTransport policy.
- When constructing the SCA module, you can use mediation flow capabilities to dynamically create a new HTTP authentication header and specify the user name and password information in the header.

Note: The policy set has precedence over the value specified in the header. If you want to use the value set in the HTTP authentication header at run time, do not attach a policy set that includes the HTTPTransport policy. Specifically, do not use the default BPMHTTPBasicAuthentication policy set, and, if you have defined a policy set, make sure it excludes the HTTPTransport policy.

For more information about web service policy sets and policy bindings and how they are used, see Web services policy sets of the WebSphere Application Server Information Center.

- To use the supplied policy set, perform the following steps:
  1. Optional: In the administrative console, create a client general policy binding or edit an existing one that includes the HTTPTransport policy with the required user ID and password values.
  2. In IBM Integration Designer, generate a web service (JAX-WS) import binding and attach the BPMHTTPBasicAuthentication policy set.
  3. Perform *one* of the following steps:
     - In IBM Integration Designer, in the web service (JAX-WS) import binding properties, specify the name of an existing client general policy binding that includes the HTTPTransport policy.
     - After deploying the SCA module, use the administrative console to either select an existing client policy binding, or create a new client policy binding and then associate it with the import binding.
  4. Optional: In the administrative console of the process server, edit the selected policy set binding to specify the required ID and password.

- To specify the user name and password in the HTTP authentication header, perform one of the following sets of steps:
  - Use the HTTP Header Setter mediation primitive in IBM Integration Designer to create the HTTP authentication header, and specify the user name and password.
  - If additional logic is required, use Java code in a custom mediation primitive (as shown in the following example) to:
    1. Create an HTTP authentication header.
    2. Specify the user name and password information.
    3. Add the new HTTP authentication header to HTTPControl.
    4. Set the updated HTTPControl back in the Context service.

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
}
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

Checking the server configuration:

When you deploy an application with a web service (JAX-WS) binding, you must make sure that the server on which the application is deployed does not have the **Start components as needed** option selected.

You can check to see whether this option is selected by performing the following steps from the administrative console:

1. Click **Servers** > **Server types** > **WebSphere application servers**.
2. Click the server name.
3. From the Configuration tab, determine whether **Start components as needed** is selected.
4. Perform one of the following steps:
   - If **Start components as needed** is selected, remove the check, and then click **Apply**.
   - If **Start components as needed** is not selected, click **Cancel**.

Attachments in SOAP messages:

You can send and receive SOAP messages that include binary data (such as PDF files or JPEG images) as attachments. Attachments can be *referenced* (that is, represented explicitly as message parts in the service interface) or *unreferenced* (in which arbitrary numbers and types of attachments can be included).

A referenced attachment can be represented in one of the following ways:

- MTOM attachments use the SOAP Message Transmission Optimization Mechanism (http://www.w3.org/TR/soap12-mtom/) specified encoding. MTOM attachments are enabled through a configuration option in the import and export bindings and are the recommended way to encode attachments for new applications.
- As a wsi:swaRef-typed element in the message schema

  Attachments defined using the wsi:swaRef type conform to the Web Services Interoperability Organization (WS-I) *Attachments Profile Version 1.0* (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.
- As a top-level message part, using a binary schema type

  Attachments represented as top-level message parts conform to the *SOAP Messages with Attachments* (http://www.w3.org/TR/SOAP-attachments) specification.

  Attachments represented as top-level message parts can also be configured to ensure that the WSDL document and messages produced by the binding conform to the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html).

An unreferenced attachment is carried in a SOAP message without any representation in the message schema.

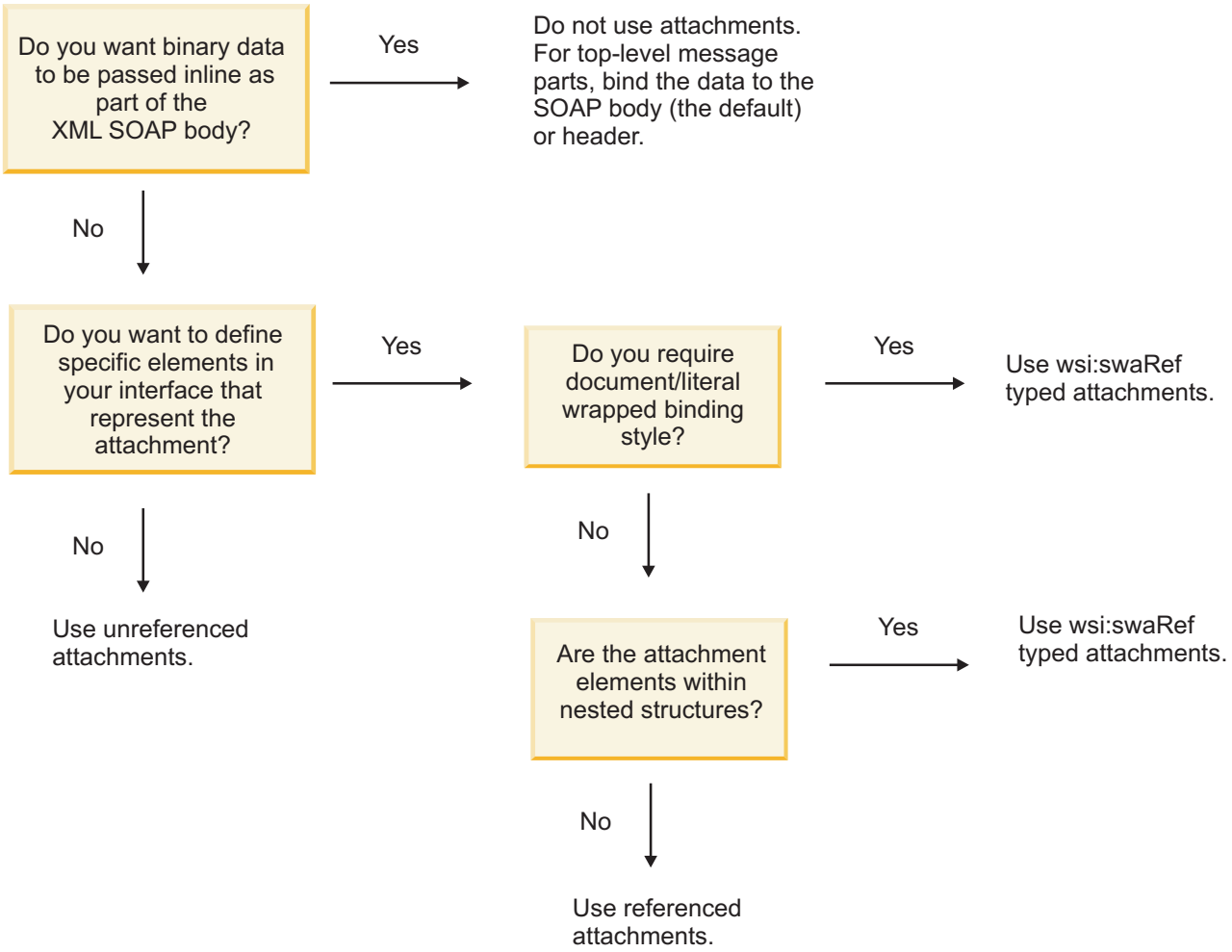
In all cases, except MTOM attachments, the WSDL SOAP binding should include a MIME binding for attachments to be used, and the maximum size of the attachments should not exceed 20 MB.

Note: To send or receive SOAP messages with attachments, you must use one of the Web service bindings based on the Java API for XML Web Services (JAX-WS).

How to choose the appropriate attachment style:

When designing a new service interface that includes binary data, consider how that binary data is carried in the SOAP messages that are sent and received by the service.

Message Transmission Optimization Mechanism (MTOM) should be used for attachments if the connected web service application supports it. If not, the following diagram shows how other attachment styles are chosen. Use the following set of questions to determine the appropriate attachment style:



MTOM attachments: top-level message parts:

You can send and receive web service messages which include SOAP Message Transmission Optimization Mechanism (MTOM) attachments. In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

By sending or receiving a referenced attachment in a SOAP message, the binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

The following is a sample of an MTOM SOAP message:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;
type="application/xop+xml"; start="
<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>"; start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header/>
```

```
      <soapenv:Body>
         <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
            <input>
            <imageData><xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
            href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/></imageData>
            </input>
         </sendImage>
      </soapenv:Body>
   </soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

Note that in the MTOM sample, the content-type for the SOAP envelope is **application/xop+xml** and the binary data is replaced by an **xop:Include** element like below:

```
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
```

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

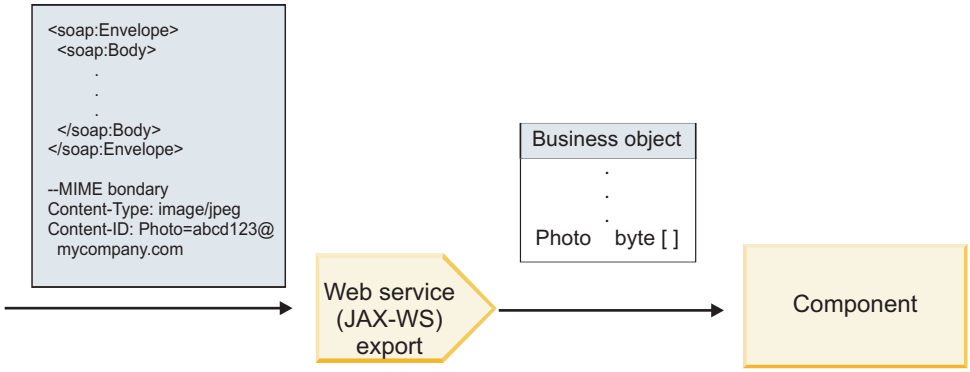


Figure 13. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment

MTOM attachment attributes

- MTOM can support attachment elements within nested structures.
- MTOM is only available for the base64Binary type.
- MTOM can support attachment elements within nested structures meaning that the **bodyPath** for MTOM attachments are the **xpath** location for the element where the MTOM attachment is held. The computing logic for **bodyPath** is strictly following the schema to generate the **xpath** location as shown in the examples below:
  - For a non-array type (**maxOccurs** is 1): /sendImage/input/imageData
  - For a array type (**maxOccurs** > 1): /sendImage/input/imageData[1]
- Mixed attachment types are not supported, meaning that if MTOM is enabled on the import binding, the MTOM attachment will be generated. If MTOM is disabled or if the MTOM configuration value is left as the default value on the export binding, the incoming MTOM message is not supported.

Referenced attachments: swaRef-typed elements:

You can send and receive SOAP messages that include attachments represented in the service interface as swaRef-typed elements.

An swaRef-typed element is defined in the Web Services Interoperability Organization (WS-I) *Attachments Profile* Version 1.0 (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.

In the SOAP message, the SOAP body contains an swaRef-typed element that identifies the content ID of the attachment.

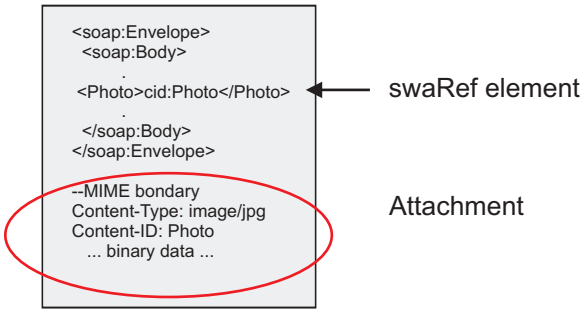


Figure 14. A SOAP message with an swaRef element

The WSDL for this SOAP message contains an swaRef-typed element within a message part that identifies the attachment.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

The WSDL should also contain a MIME binding that indicates MIME multipart messages are to be used.

Note: The WSDL does *not* include a MIME binding for the specific swaRef-typed message element, because MIME bindings apply only to top-level message parts.

Attachments represented as swaRef-typed elements can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Inbound processing of attachments

You use Integration Designer to configure an export binding to receive the attachment. You create a module and its associated interface and operations, including an element of type swaRef. You then create a Web service (JAX-WS) binding.

Note: See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

When a client passes a SOAP message with an swaRef attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the content ID of the

attachment in the business object.

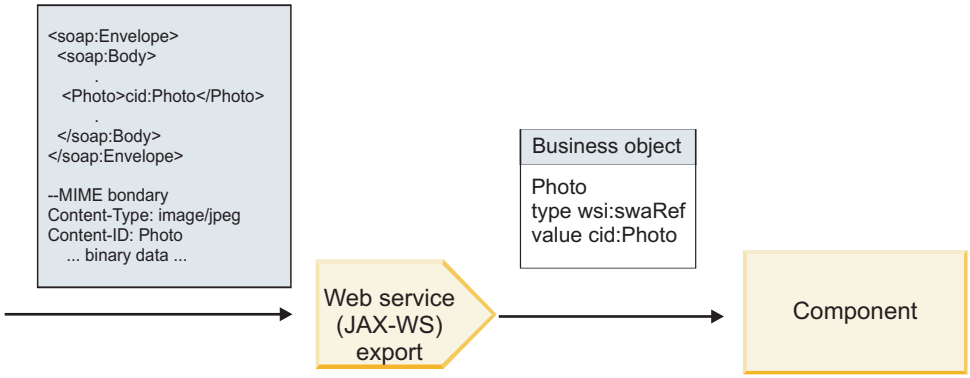


Figure 15. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 16, when swaRef attachments are accessed by components, the attachment content identifier appears as an element of type swaRef.

Each attachment of a SOAP message also has a corresponding **attachments** element in the SMO. When using the WS-I swaRef type, the **attachments** element includes the attachment content type and content ID as well as the actual binary data of the attachment.

To obtain the value of an swaRef attachment, it is therefore necessary to obtain the value of the swaRef-typed element, and then locate the **attachments** element with the corresponding **contentID** value. Note that the **contentID** value typically has the **cid:** prefix removed from the swaRef value.

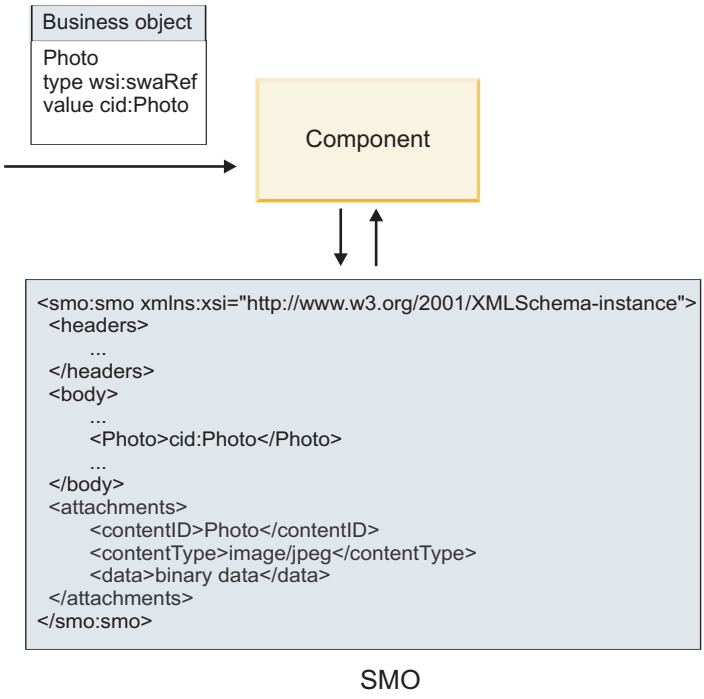


Figure 16. How swaRef attachments appear in the SMO

Outbound processing

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines the attachment that will be passed to the Web service.

When an SCA message is received by a Web service (JAX-WS) import binding, swaRef-typed elements are sent as attachments if the import is wired to a mediation flow component and the swaRef-typed element has a corresponding **attachments** element.

For outbound processing, swaRef-typed elements are always sent with their content ID values; however the mediation module must ensure that there is a corresponding **attachments** element with a matching **contentID** value.

Note: To conform to the WS-I Attachments Profile, the **content ID** value should follow the "content-id part encoding," as described in section 3.8 of the WS-I *Attachments Profile* 1.0.

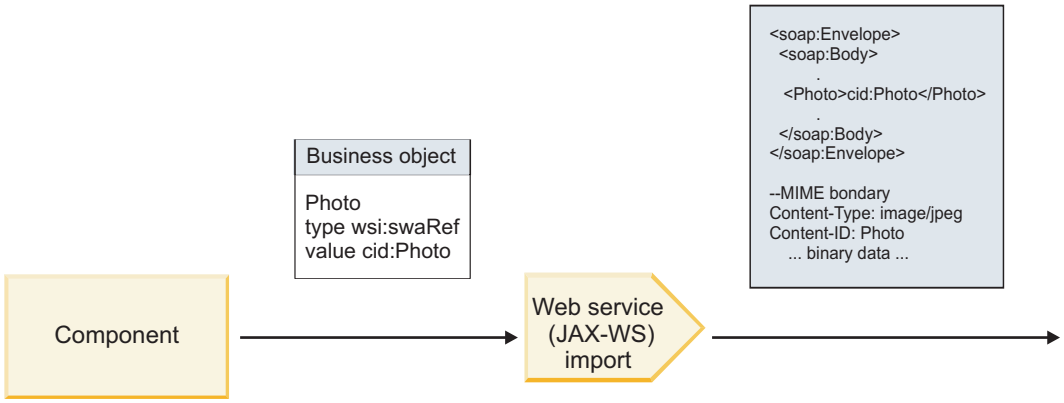


Figure 17. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment

Setting attachment metadata in a mediation flow component

If, in the SMO, there is an swaRef-typed element value and an **attachments** element, the binding prepares the SOAP message (with the attachment) and sends it to a recipient.

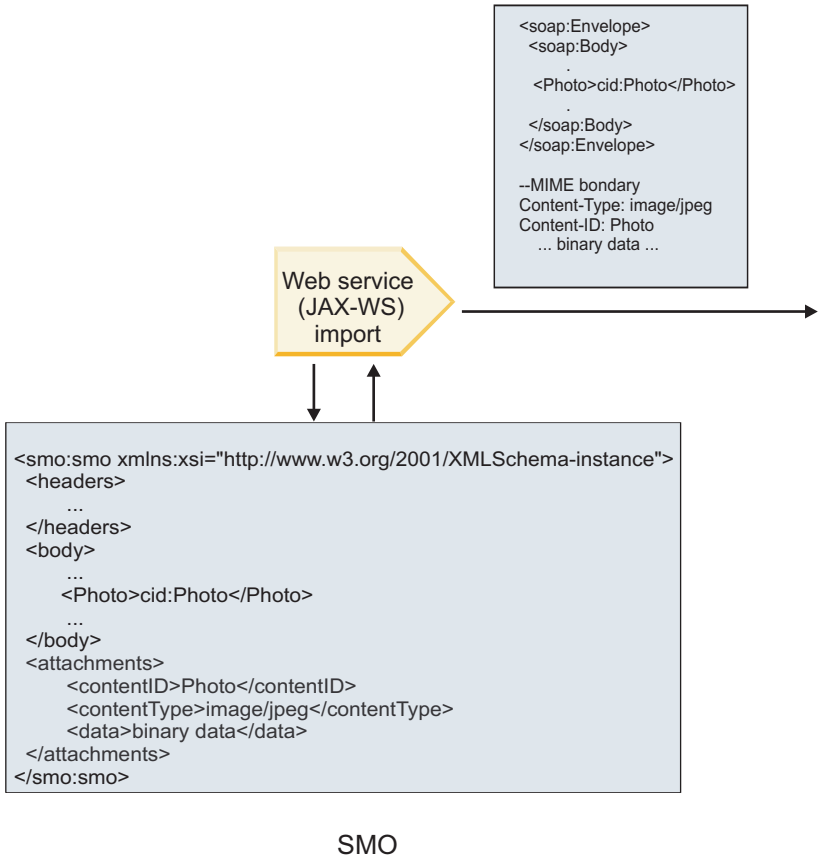


Figure 18. How an swaRef attachment in the SMO is accessed to create the SOAP message

The **attachments** element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Referenced attachments: top-level message parts:

You can send and receive SOAP messages that include binary attachments that are declared as parts in your service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

What is the advantage of sending or receiving a referenced attachment in a SOAP message? The binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

Types of SOAP messages with referenced attachments

Beginning with Version 7.0.0.3 of IBM Business Process Manager, you have a choice about how the SOAP message is generated:

- **WS-I-compliant messages**

  The runtime can generate SOAP messages that comply with the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1*. In a SOAP message that is compliant with these profiles, only one message part is bound to the SOAP body; for those that are bound as attachments, the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is used to relate the attachment to the message part.
- **Non-WS-I-compliant messages**

  The runtime can generate SOAP messages that do not comply with the WS-I profiles but that are compatible with the messages generated in Version 7.0 or 7.0.0.2 of IBM Business Process Manager. The SOAP messages use top-level elements named after the message part with an **href** attribute that holds the attachment **content-id**, but the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is not used.

Selecting WS-I compliance for Web service exports

You use Integration Designer to configure an export binding. You create a module and its associated interface and operations. You then create a Web service (JAX-WS) binding. The Referenced attachments page displays all the binary parts from the created operation, and you select which parts will be attachments. You then specify, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body.

  Note: This option can be used only when the corresponding WSDL file is also WS-I compliant.

  A WSDL file that is generated by Integration Designer Version 7.0.0.3 is compliant with WS-I. However, if you import a WSDL file that is not compliant with WS-I, you cannot select this option.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body.

Note: Only top-level message parts (that is, elements defined in the WSDL portType as parts within the input or output message) that have a binary type (either base64Binary or hexBinary) can be sent or received as referenced attachments.
See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

For WS-I-compliant messages, the content-ID that is generated in the SOAP message is a concatenation of the following elements:

- The value of the **name** attribute of the **wsdl:part** element referenced by the **mime:content**
- The character **=**
- A globally unique value, such as a UUID
- The character **@**
- A valid domain name

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

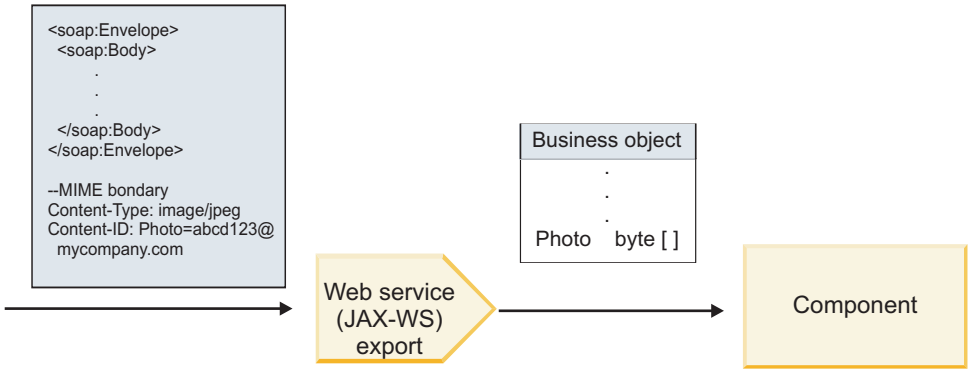


Figure 19. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 19, when referenced attachments are accessed by components, the attachment data appears as a byte array.

Each referenced attachment of a SOAP message also has a corresponding **attachments** element in the SMO. The **attachments** element includes the attachment content type and the path to the message body element where the attachment is held.

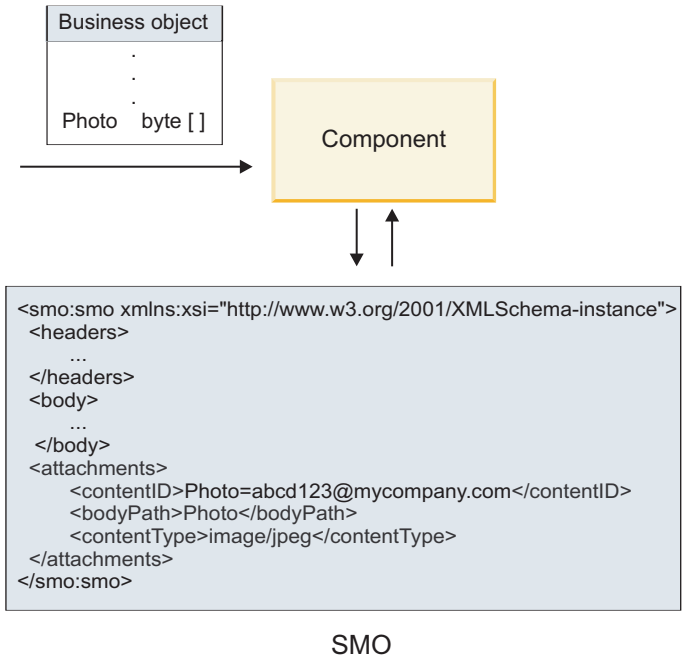


Figure 20. How referenced attachments appear in the SMO

Important: The path to the message body element is not automatically updated if the message is transformed and the attachment moved. You can use a mediation flow to update the **attachments** element with the new path (for example, as part of the transform or using a separate message element setter).

How outbound SOAP messages are constructed

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines which message parts should be passed as attachments. You can also indicate, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding do not include elements in the SOAP body that refer to the attachments; the relationship is expressed by way of the attachment content ID including the message part name.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding include one or more elements in the SOAP body that refer to the attachments by way of an **href** attribute.

Note: The part that represents an attachment, as defined in the WSDL, must be a simple type (either base64Binary or hexBinary). If a part is defined by a complexType, that part cannot be bound as an attachment.

Outbound processing of referenced attachments

The import binding uses information in the SMO to determine how the binary top-level message parts are sent as attachments.

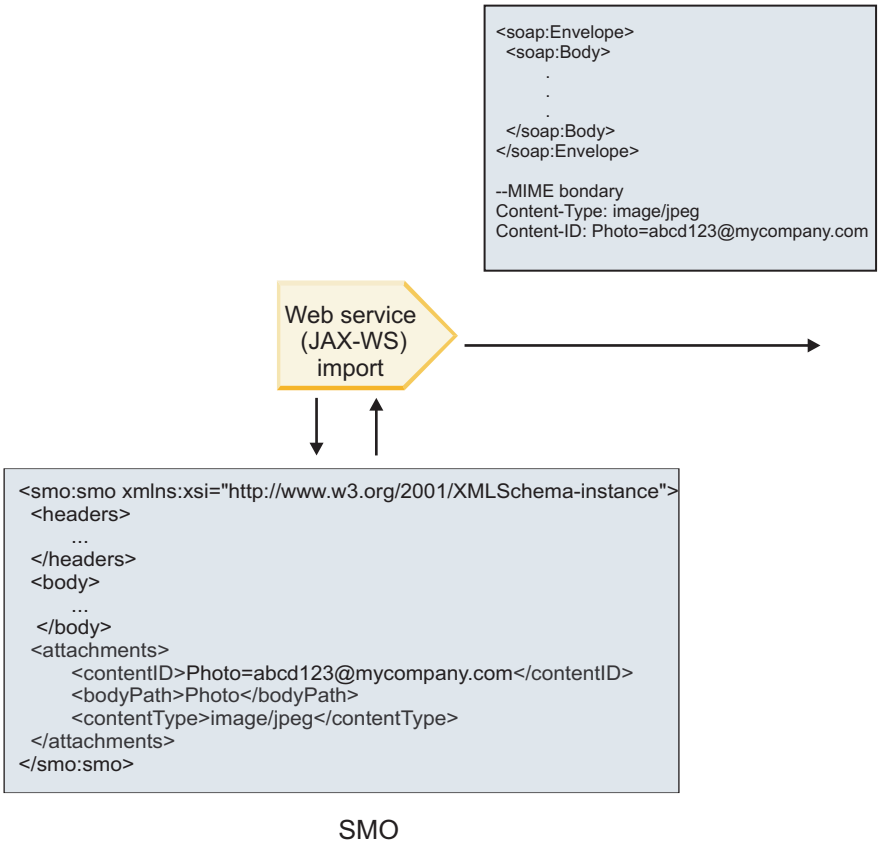


Figure 21. How the referenced attachment in the SMO is accessed to create the SOAP message

The **attachments** element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

The binding uses a combination of the following conditions to determine how (or whether) the message is sent:

- Whether there is a WSDL MIME binding for the top-level binary message part and, if so, how the content type is defined
- Whether there is an **attachments** element in the SMO whose **bodyPath** value references a top-level binary part

How attachments are created when an attachment element exists in the SMO

The following table shows how an attachment is created and sent if the SMO contains an **attachment** element with a **bodyPath** that matches a message name part:

Table 27. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	**How message is created and sent**
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to **application/octet-stream**.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to the type defined in the WSDL MIME content element.
Not present	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to **application/octet-stream**. **Note:** Sending message parts as attachments when not defined as such in the WSDL may break compliance with the WS-I Attachments Profile 1.0 and so should be avoided if possible.

How attachments are created when no attachment element exists in the SMO

The following table shows how an attachment is created and sent if the SMO does not contain an **attachment** element with a **bodyPath** that matches a message name part:

Table 28. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	**How message is created and sent**
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. **Content-Id** is generated. **Content-Type** is set to **application/octet-stream**.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. **Content-Id** is generated. **Content-Type** is set to the type defined in the WSDL MIME content element.
Not present	Message part is not sent as an attachment.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Unreferenced attachments:

You can send and receive *unreferenced* attachments that are not declared in the service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachments are in subsequent parts. No reference to the attachment is included in the SOAP body.

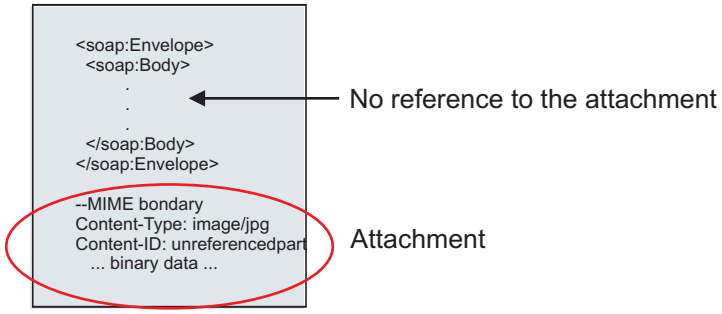


Figure 22. A SOAP message with an unreferenced attachment

You can send a SOAP message with an unreferenced attachment through a Web service export to a Web service import. The output message, which is sent to the target Web service, contains the attachment.

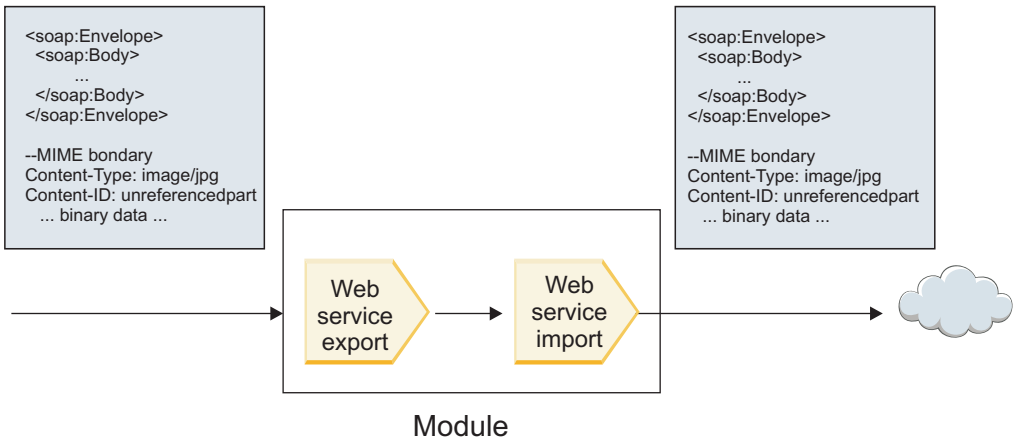


Figure 23. An attachment passing through an SCA module

In Figure 23, the SOAP message, with the attachment, passes through without modification.

You can also modify the SOAP message by using a mediation flow component. For example, you can use the mediation flow component to extract data from the SOAP message (binary data in the body of the message, in this case) and create a SOAP with attachments message. The data is processed as part of the

attachments element of a service message object (SMO).

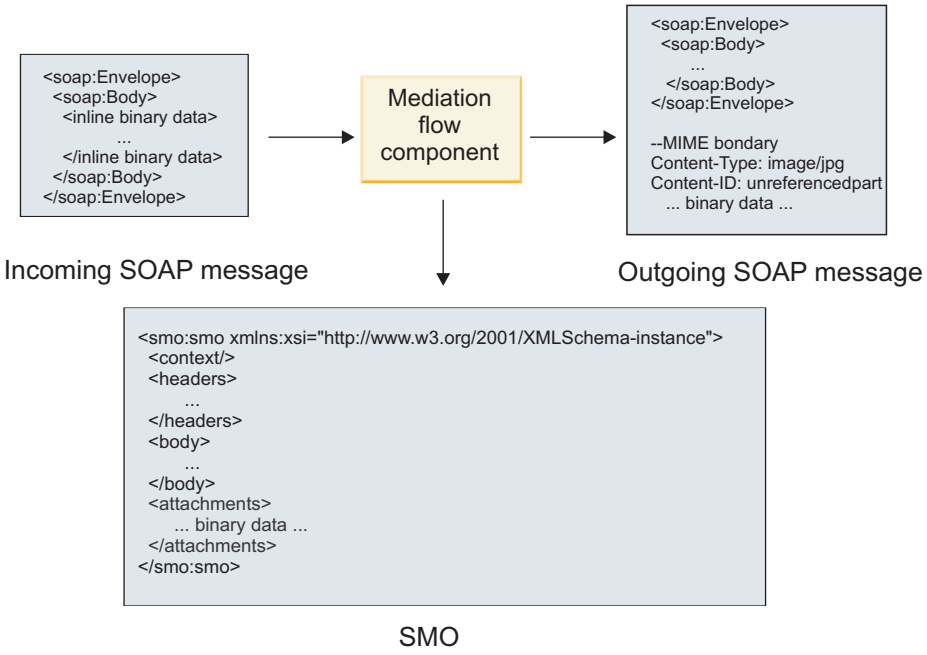


Figure 24. A message processed by a mediation flow component

Conversely, the mediation flow component can transform the incoming message by extracting and encoding the attachment and then transmitting the message with no attachments.

Instead of extracting data from an incoming SOAP message to form a SOAP with attachments message, you can obtain the attachment data from an external source, such as a database.

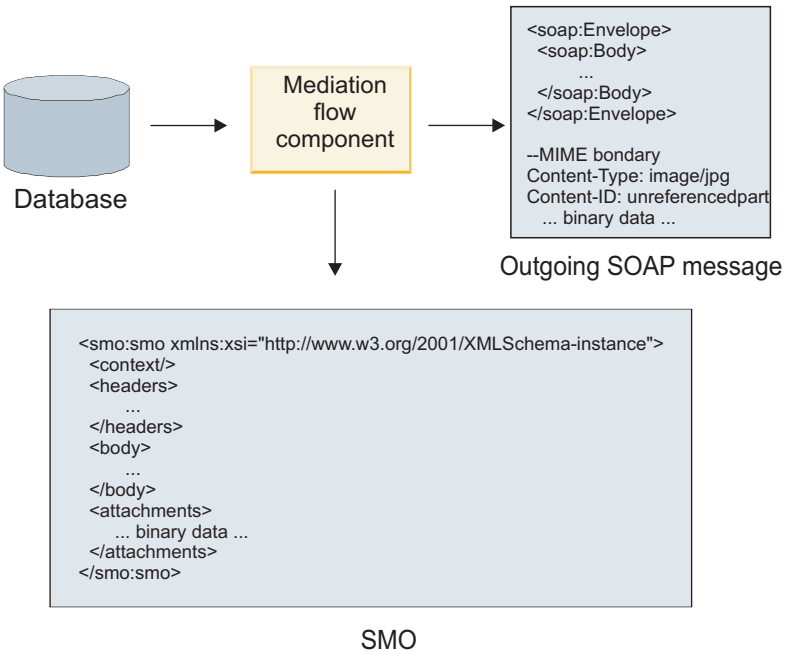


Figure 25. An attachment obtained from a database and added to the SOAP message

Conversely, the mediation flow component can extract the attachment from an incoming SOAP message and process the message (for example, store the attachment in a database).

Unreferenced attachments can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Use of WSDL document style binding with multipart messages:

The Web Services Interoperability Organization (WS-I) organization has defined a set of rules regarding how Web services should be described by way of a WSDL and how the corresponding SOAP messages should be formed, in order to ensure interoperability.

These rules are specified in the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html). In particular, the WS-I Basic Profile 1.1 R2712 states: "A document-literal binding MUST be serialized as an ENVELOPE with a soap:Body whose child element is an instance of the global element declaration referenced by the corresponding wsdl:message part."

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, only one of those parts should be bound to the SOAP body in order to be compliant with the WS-I Basic Profile 1.1.

Further, the WS-I Attachments Profile 1.0 R2941 states: "A wsdl:binding in a DESCRIPTION SHOULD bind every wsdl:part of a wsdl:message in the wsdl:portType to which it refers to one of soapbind:body, soapbind:header, soapbind:fault , soapbind:headerfault, or mime:content.".

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, all parts other than the one selected to be bound to the SOAP body must be bound as attachments or headers.

The following approach is used when WSDL descriptions are generated for exports with Web service (JAX-WS and JAX-RPC) bindings in this case:

- You can choose which message part is bound to the SOAP body if there is more than one non-binary-typed element. If there is a single non-binary typed element, that element is automatically bound to the SOAP body.
- For the JAX-WS binding, all other message parts of type "hexBinary" or "base64Binary" are bound as referenced attachments. See "Referenced attachments: top-level message parts" on page 89.
- All other message parts are bound as SOAP headers.

The JAX-RPC and JAX-WS import bindings honor the SOAP binding in an existing WSDL document with multipart document style messages even if it does bind multiple parts to the SOAP body; however, you are not able to generate Web service clients for such WSDL documents in Rational Application Developer.

Note: The JAX-RPC binding does not support attachments.

The recommended pattern when using multipart messages with an operation that has document style SOAP binding is therefore:

1. Use the document/literal wrapped style. In this case, messages always have a single part; however, attachments have to be unreferenced (as described in “Unreferenced attachments” on page 95) or swaRef-typed (as described in “Referenced attachments: swaRef-typed elements” on page 86) in this case.
2. Use the RPC/literal style. In this case, there are no restrictions on the WSDL binding in terms of number of parts bound to the SOAP body; the SOAP message that results always has a single child that represents the operation being invoked, with the message parts being children of that element.
3. For the JAX-WS binding, have at most one message part that is not of type "hexBinary" or "base64Binary", unless it is acceptable to bind the other non-binary parts to SOAP headers.
4. Any other cases are subject to the behavior described.

Note: Additional restrictions exist when you use are using SOAP messages that do not conform to the WS-I *Basic Profile Version 1.1*.

- The first message part should be non-binary.
- When receiving multipart document-style SOAP messages with referenced attachments, the JAX-WS binding expects each referenced attachment to be represented by a SOAP body child element with an href attribute value that identifies the attachment by its content ID. The JAX-WS binding sends referenced attachments for such messages in the same way. This behavior is not compliant with the WS-I Basic Profile.

  To ensure that your messages comply with the Basic Profile, follow approach 1 or 2 in the previous list or avoid the use of referenced attachments for such messages and use unreferenced or swaRef-typed attachments instead.

HTTP bindings:

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

Hypertext Transfer Protocol (HTTP) is a widely-used protocol for transferring information on the Web. When you are working with an external application that uses the HTTP protocol, an HTTP binding is necessary. The HTTP binding transforms the data that is passed in as a message in native format to a business object in an SCA application. The HTTP binding also can transform the data that is passed out as a business object to the native format expected by the external application.

Note: If you want to interact with clients and services that use the Web services SOAP/HTTP protocol, consider using one of the Web service bindings, which provide additional functionality with respect to handling Web services standard qualities of service.

Some common scenarios for using the HTTP binding are described in the following list:

- SCA-hosted services can invoke HTTP applications using an HTTP import.
- SCA-hosted services can expose themselves as HTTP-enabled applications, so they can be used by HTTP clients, using an HTTP export.
- IBM Business Process Manager and Process Server can communicate between themselves across an HTTP infrastructure, consequently users can manage their communications according to corporate standards.
- IBM Business Process Manager and Process Server can act as mediators of HTTP communications, transforming and routing messages, which improves the integration of applications using a HTTP network.

- IBM Business Process Manager and Process Server can be used to bridge between HTTP and other protocols, such as SOAP/HTTP Web services, Java Connector Architecture (JCA)-based resource adapters, JMS, and so on.

Detailed information about creating HTTP import and export bindings can be found in the Integration Designer information center. See the **Developing integration applications** > **Accessing external services with HTTP**> topics.

HTTP bindings overview:

The HTTP binding provides connectivity to HTTP-hosted applications. It mediates communication between HTTP applications and allows existing HTTP-based applications to be called from a module.

HTTP import bindings

The HTTP import binding provides outbound connectivity from Service Component Architecture (SCA) applications to an HTTP server or applications.

The import invokes an HTTP endpoint URL. The URL can be specified in one of three ways:
- The URL can be set dynamically in the HTTP headers by way of the dynamic override URL.
- The URL can be set dynamically in the SMO target address element.
- The URL can be specified as a configuration property on the import.

This invocation is always synchronous in nature.

Although HTTP invocations are always request-reply, the HTTP import supports both one-way and two-way operations and ignores the response in the case of a one-way operation.

HTTP export bindings

The HTTP export binding provides inbound connectivity from HTTP applications to an SCA application.

A URL is defined on the HTTP export. HTTP applications that want to send request messages to the export use this URL to invoke the export.

The HTTP export also supports pings.

HTTP bindings at runtime

An import with an HTTP binding at runtime sends a request with or without data in the body of the message from the SCA application to the external Web service. The request is made from the SCA application to the external Web service, as shown in Figure 26.

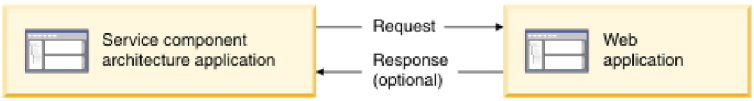


Figure 26. Flow of a request from the SCA application to the Web application

Optionally, the import with the HTTP binding can receive data back from the Web application in a response to the request.

With an export, the request is made by a client application to a Web service, as shown in Figure 27 on page 100.

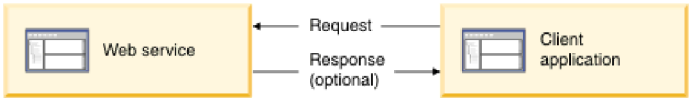


Figure 27. Flow of a request from the Web service to the client application.

The Web service is a Web application running on the server. The export is implemented in that Web application as a servlet so the client sends its request to a URL address. The servlet passes the request to the SCA application in the runtime.

Optionally, the export may send data to the client application in response to the request.

HTTP headers:

HTTP import and export bindings allow configuration of HTTP headers and their values to be used for outbound messages. The HTTP import uses these headers for requests, and the HTTP export uses them for responses.

Statically configured headers and control information take precedence over values dynamically set at runtime. However, the dynamic override URL, Version, and Method control values override the static values, which are otherwise considered defaults.

The binding supports the dynamic nature of the HTTP import URL by determining the value of HTTP target URL, Version, and Method at run time. These values are determined by extracting the value of Endpoint Reference, Dynamic Override URL, Version, and Method.

- For Endpoint Reference, use com.ibm.websphere.sca.addressing.EndpointReference APIs or set the /headers/SMOHeader/Target/address field in the SMO header.
- For Dynamic Override URL, Version, and Method, use the HTTP control parameters section of the Service Component Architecture (SCA) message. Note that the Dynamic Override URL takes precedence over the target Endpoint Reference; however, the Endpoint Reference applies across bindings, so it is the preferred approach and should be used where possible.

The control and header information for outbound messages under HTTP export and import bindings is processed in the following order:

1. Header and control information excluding HTTP dynamic override URL, Version, and Method from the SCA Message (lowest priority)
2. Changes from the administrative console on the export/import level
3. Changes from the administrative console on the method level of the export or import
4. Target address specified by way of the Endpoint Reference or the SMO header
5. Dynamic Override URL, Version, and Method from the SCA message
6. Headers and control information from the data handler or data binding (highest priority)

The HTTP export and import will populate inbound direction headers and control parameters with data from the incoming message (HTTPExportRequest and HTTPImportResponse) only if protocol header propagation is set to **True**. Inversely, the HTTP export and import will read and process outbound headers and control parameters (HTTPExportResponse and HTTPImportRequest) only if protocol header propagation is set to **True**.

Note: Data handler or data binding changes to headers or control parameters in the import response or export request will not alter the processing instructions of the message inside the import or export binding and should be used only to propagate modified values to downstream SCA components.

The context service is responsible for propagating the context (including the protocol headers, such as the HTTP header, and the user context, such as account ID) along an SCA invocation path. During

development in IBM Integration Designer, you can control the propagation of context by way of import and export properties. For more details, see the import and export bindings information in the IBM Integration Designer information center.

Supplied HTTP header structures and support

Table 29 itemizes the request/response parameters for HTTP Import and HTTP Export requests and responses.

Table 29. Supplied HTTP header information

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
URL	Ignored	Not set	Read from the request message. **Note:** Query string is also part of the URL control parameter.	Ignored
Version (possible values: 1.0, 1.1; default is 1.1)	Ignored	Not set	Read from the request message	Ignored
Method	Ignored	Not set	Read from the request message	Ignored
Dynamic Override URL	If set in the data handler or data binding, overrides the HTTP Import URL. Written to the message in the request line. **Note:** Query string is also part of the URL control parameter.	Not set	Not set	Ignored
Dynamic Override Version	If set, overrides the HTTP Import Version. Written to the message in the request line.	Not set	Not set	Ignored
Dynamic Override Method	If set, overrides the HTTP Import Method. Written to the message in the request line.	Not set	Not set	Ignored
Media Type (This control parameter carries part of the value of the Content-Type HTTP header.)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	If present, written to the message as part of Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.

Table 29. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
Character set (default: UTF-8)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	Supported; written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.
Transfer Encoding (Possible values: chunked, identity; default is identity)	If present, written to the message as a header and controls how the message transformation is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the message transformation is encoded.
Content Encoding (Possible values: gzip, x-gzip, deflate, identity; default is identity)	If present, written to the message as a header and controls how the payload is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the payload is encoded.
Content-Length	Ignored	Read from the response message	Read from the request message	Ignored
StatusCode (default: 200)	Not supported	Read from the response message	Not supported	If present, written to the message in the response line
ReasonPhrase (default: OK)	Not supported	Read from the response message	Not supported	Control value ignored. The message response line value is generated from the StatusCode.
Authentication (contains multiple properties)	If present, used to construct the Basic Authentication header. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable	Read from the request message Basic Authentication header. The presence of this header does not indicate the user has been authenticated. Authentication should be controlled in the servlet configuration. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable
Proxy (contains multiple properties: Host, Port, Authentication)	If present, used to establish connection through proxy.	Not applicable	Not applicable	Not applicable

Table 29. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
SSL (contains multiple properties: Keystore, Keystore Password, Trustore, Trustore Password, ClientAuth)	If populated and the destination url is HTTPS, it is used to establish a connection through SSL.	Not applicable	Not applicable	Not applicable

HTTP data bindings:

For each different mapping of data between a Service Component Architecture (SCA) message and an HTTP protocol message, a data handler or an HTTP data binding must be configured. Data handlers provide a binding-neutral interface that allows reuse across transport bindings and represent the recommended approach; data bindings are specific to a particular transport binding. HTTP-specific data binding classes are supplied; you can also write custom data handlers or data bindings.

Note: The three HTTP data binding classes described in this topic (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of IBM Business Process Manager Version 7.0. Instead of using the data bindings described in this topic, consider the following data handlers:

- Use SOAPDataHandler instead of HTTPStreamDataBindingSOAP.
- Use UTF8XMLDataHandler instead of HTTPStreamDataBindingXML
- Use GatewayTextDataHandler instead of HTTPServiceGatewayDataBinding

Data bindings are provided for use with HTTP imports and HTTP exports: binary data binding, XML data binding, and SOAP data binding. A response data binding is not required for one-way operations. A data binding is represented by the name of a Java class whose instances can convert both from HTTP to ServiceDataObject and vice-versa. A function selector must be used on an export which, in conjunction with method bindings, can determine which data binding is used and which operation is invoked. The supplied data bindings are:

- Binary data bindings, which treat the body as unstructured binary data. The implementation of the binary data binding XSD schema is as follows:

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>

  <xsd:complexType name="HTTPBytesBody">
    <xsd:complexContent>
      <xsd:extension base="tns:HTTPBaseBody">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:hexBinary"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

- XML data bindings, which support the body as XML data. The implementation of the XML data binding is similar to the JMS XML data binding and has no restrictions on the interface schema.
- SOAP data bindings, which support the body as SOAP data. The implementation of the SOAP data binding has no restrictions on the interface schema.

Implementing custom HTTP data bindings

This section describes how to implement a custom HTTP data binding.

Note: The recommended approach is to implement a custom data handler because it can be reused across transport bindings.

HTTPStreamDataBinding is the principal interface for handling custom HTTP messages. The interface is designed to allow handling of large payloads. However, in order for such implementation to work, this data binding must return the control information and headers before writing the message into the stream.

The methods and their order of execution, listed below, must be implemented by the custom data binding.

To customize a data binding, write a class that implements HTTPStreamDataBinding. The data binding should have four private properties:

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

The HTTP binding will invoke the customized data binding in the following order:

- Outbound processing (DataObject to Native format):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- Inbound processing (Native format to DataObject):
  1. setControlParameters(...)
  2. setHeaders(...)
  3. convertFromNativeData(...)
  4. isBusinessException()
  5. getDataObject()
  6. getControlParameters()
  7. getHeaders()

You need to invoke setDataObject(...) in convertFromNativeData(...) to set the value of dataObject, which is converted from native data to the private property "pDataObject".

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
```

```
  this.pHeaders = arg0;
}
/*
* Add http header "IsBusinessException" in pHeaders.
* Two steps:
* 1.Remove all the header with name IsBusinessException (case-insensitive) first.
*   This is to make sure only one header is present.
* 2.Add the new header "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //remove all the header with name IsBusinessException (case-insensitive) first.
  //This is to make sure only one header is present.
  //add the new header "IsBusinessException", code example:
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Get header "IsBusinessException" from pHeaders, return its boolean value
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Customer-developed method to
  //Read data from HTTPInputStream
  //Convert it to DataObject
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

EJB bindings:

Enterprise JavaBeans (EJB) import bindings enable Service Component Architecture (SCA) components to invoke services provided by Java EE business logic running on a Java EE server. EJB export bindings allow SCA components to be exposed as Enterprise JavaBeans so that Java EE business logic can invoke SCA components otherwise unavailable to them.

EJB import bindings:

EJB import bindings allow an SCA module to call EJB implementations by specifying the way that the consuming module is bound to the external EJB. Importing services from an external EJB implementation allows users to plug their business logic into the IBM Business Process Manager environment and participate in a business process.

You use Integration Designer to create EJB import bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB import using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB import based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and import files based on the services discovered.
- Creating EJB import using the assembly editor

  You can create an EJB import within an assembly diagram using the Integration Designer assembly editor. From the palette, you can use either an Import or use a Java class to create the EJB binding.

The generated import has data bindings that make the Java-WSDL connection instead of requiring a Java bridge component. You can directly wire a component with a Web Services Description Language (WSDL) reference to the EJB import that communicates to an EJB-based service using a Java interface.

The EJB import can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation to the Java EE business logic can be local (for EJB 3.0 only) or remote.

- Local invocation is used when you want to call Java EE business logic that resides on the same server as the import.

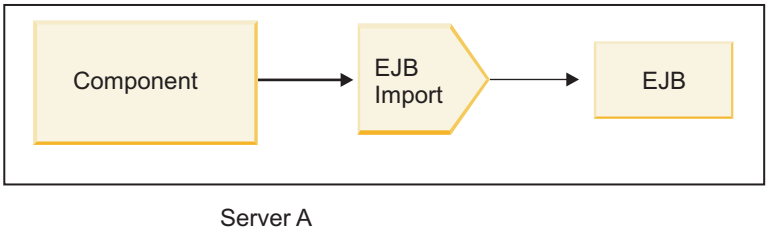


Figure 28. Local invocation of an EJB (EJB 3.0 only)

- Remote invocation is used when you want to call Java EE business logic that does not reside on the same server as the import.

  For example, in the following figure, an EJB import uses the Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) to invoke an EJB method on another server.

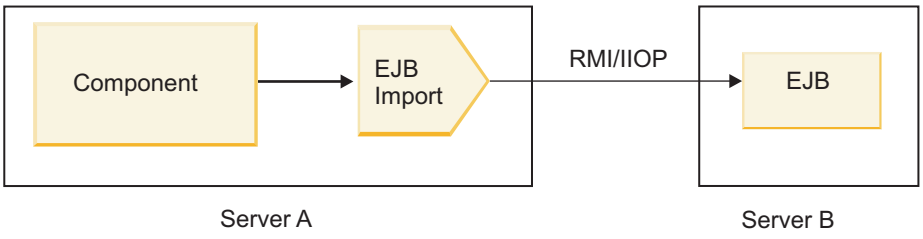


Figure 29. Remote invocation of an EJB

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB import bindings contain the following major components:

- JAX-WS data handler
- EJB fault selector

- EJB import function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler, function selector, and fault selector to perform the tasks otherwise completed by the components that are part of the EJB import bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB export bindings:

External Java EE applications can invoke an SCA component by way of an EJB export binding. Using an EJB export lets you expose SCA components so that external Java EE applications can invoke those components using the EJB programming model.

Note: The EJB export is a stateless bean.

You use Integration Designer to create EJB bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB export bindings using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB export service based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and export files based on the services discovered.
- Creating EJB export bindings using the assembly editor

  You can create an EJB export using the Integration Designer assembly editor.

Important: A Java 2 Platform, Standard Edition (J2SE) client cannot invoke the EJB export client that is generated in Integration Designer.

You can generate the binding from an existing SCA component, or you can generate an export with an EJB binding for a Java interface.

- When you generate an export for an existing SCA component that has an existing WSDL interface, the export is assigned a Java interface.
- When you generate an export for a Java interface, you can select either a WSDL or a Java interface for the export.

  Note: A Java interface used to create an EJB export has the following limitations with regard to the objects (input and output parameters and exceptions) passed as parameters on a remote call:
  - They must be of concrete type (instead of an interface or abstract type).
  - They must conform to the Enterprise JavaBeans specification. They must be serializable and have the default no-argument constructor, and all properties must be accessible through getter and setter methods.

    Refer to the Sun Microsystems, Inc., Web site at http://java.sun.com for information about the Enterprise JavaBeans specification.

  In addition, the exception must be a checked exception, inherited from java.lang.Exception, and it must be singular (that is, it does not support throwing multiple checked exception types).

  Note also that the business interface of a Java EnterpriseBean is a plain Java interface and must not extend javax.ejb.EJBObject or javax.ejb.EJBLocalObject. The methods of the business interface should not throw java.rmi.Remote.Exception.

The EJB export bindings can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation can be local (for EJB 3.0 only) or remote.

- Local invocation is used when the Java EE business logic calls an SCA component that resides on the same server as the export.
- Remote invocation is used when the Java EE business logic does not reside on the same server as the export.

  For example, in the following figure, an EJB uses RMI/IIOP to call an SCA component on a different server.

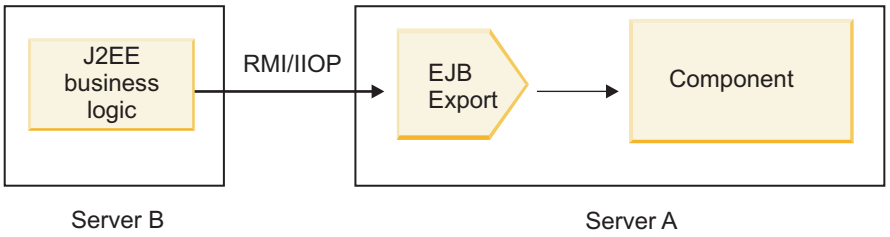


Figure 30. Remote call from a client to an SCA component by way of an EJB export

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB export bindings contain the following major components:

- JAX-WS data handler
- EJB export function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler and function selector to perform the tasks otherwise completed by the components that are part of the EJB export bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB binding properties:

EJB import bindings use their configured JNDI names to determine the EJB programming model level and type of invocation (local or remote). EJB import and export bindings use the JAX-WS data handler for data transformation. The EJB import binding uses an EJB import function selector and an EJB fault selector, and the EJB export binding uses an EJB export function selector.

JNDI names and EJB import bindings:

When it configures the EJB binding on an import, Integration Designer uses the JNDI name to determine the EJB programming model level and type of invocation (local or remote).

If no JNDI name is specified, the default EJB interface binding is used. The default names that are created depend on whether you are invoking EJB 2.1 JavaBeans or EJB 3.0 JavaBeans.

Note: Refer to the "EJB 3.0 application bindings overview" topic in theWebSphere Application Server information center for more detailed information about naming conventions.

- EJB 2.1 JavaBeans

  The default JNDI name preselected by Integration Designer is the default EJB 2.1 binding, which takes the form **ejb/** plus the home interface, separated by slashes.

  For example, for the home interface of EJB 2.1 JavaBeans for com.mycompany.myremotebusinesshome, the default binding is:

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  For EJB 2.1, only remote EJB invocation is supported.
- EJB 3.0 JavaBeans

The default JNDI name preselected by Integration Designer for the local JNDI is the fully qualified class name of the local interface preceded by **ejblocal:**. For example, for the fully qualified interface of the local interface com.mycompany.mylocalbusiness, the preselected EJB 3.0 JNDI is:

```
ejblocal:com.mycompany.mylocalbusiness
```

For the remote interface com.mycompany.myremotebusiness, the preselected EJB 3.0 JNDI is the fully qualified interface:

```
com.mycompany.myremotebusiness
```

The EJB 3.0 default application bindings are described at the following location: EJB 3.0 application bindings overview.

Integration Designer will use the "short" name as the default JNDI location for EJBs using the version 3.0 programming model.

Note: If the deployed JNDI reference of the target EJB is different from the default JNDI binding location because a custom mapping was used or configured, the target JNDI name must be properly specified. You can specify the name in Integration Designer before deployment, or, for the import binding, you can change the name in the administrative console (after deployment) to match the JNDI name of the target EJB.

For more information on creating EJB bindings, see the section devoted to Working with EJB bindings in the Integration Designer information center.

JAX-WS data handler:

The Enterprise JavaBeans (EJB) import binding uses the JAX-WS data handler to turn request business objects into Java object parameters and to turn the Java object return value into the response business object. The EJB export binding uses the JAX-WS data handler to turn request EJBs into request business objects and to turn the response business object into a return value.

This data handler maps data from the SCA-specified WSDL interface to the target EJB Java interface (and vice versa) using the Java API for XML Web Services (JAX-WS) specification and the Java Architecture for XML Binding (JAXB) specification.

Note: Current support is restricted to the JAX-WS 2.1.1 and JAXB 2.1.3 specifications.

The data handler specified at the EJB binding level is used to perform request, response, fault, and runtime exception processing.

Note: For faults, a specific data handler can be specified for each fault by specifying the faultBindingType configuration property. This overrides the value specified at the EJB binding level.

The JAX-WS data handler is used by default when the EJB binding has a WSDL interface. This data handler cannot be used to transform a SOAP message representing a JAX-WS invocation to a data object.

The EJB import binding uses a data handler to transform a data object into a Java Object array (Object[]). During outbound communications, the following processing takes place:

1. The EJB binding sets the expected type, expected element, and targeted method name in the BindingContext to match those specified in the WSDL.
2. The EJB binding invokes the transform method for the data object requiring data transformation.
3. The data handler returns an Object[] representing the parameters of the method (in the order of their definition within the method).
4. The EJB binding uses the Object[] to invoke the method on the target EJB interface.

The binding also prepares an Object[] to process the response from the EJB invocation.

- The first element in the Object[] is the return value from the Java method invocation.

- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For parameters of type Out, the values must be returned in the response data object.

The data handler processes and transforms values found in the Object[] and then returns a response to the data object.

The data handler supports xs:AnyType, xs:AnySimpleType, and xs:Any along with other XSD data types. To enable support for xs:Any, use the **@XmlAnyElement (lax=true)** for the JavaBeans property in the Java code, as shown in the following example:

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

This makes the property object in TestType an xs:any field. The Java class value used in the xs:any field should have the **@XmlAnyElement** annotation. For example, if Address is the Java class being used to populate the object array, the Address class should have the annotation **@XmlRootElement**.

Note: To customize the mapping from the XSD type to Java types defined by the JAX-WS specification, change the JAXB annotations to fit your business need. The JAX-WS data handler supports xs:any, xs:anyType, and xs:anySimpleType.

The following restrictions are applicable for the JAX-WS data handler:

- The data handler does not include support for the header attribute **@WebParam** annotation.
- The namespace for business object schema files (XSD files) does not include default mapping from the Java package name. The annotation **@XMLSchema** in package-info.java also does not work. The only way to create an XSD with a namespace is to use the **@XmlType** and **@XmlRootElement** annotations. **@XmlRootElement** defines the target namespace for the global element in JavaBeans types.
- The EJB import wizard does not create XSD files for unrelated classes. Version 2.0 does not support the **@XmlSeeAlso** annotation, so if the child class is not referenced directly from the parent class, an XSD is not created. The solution to this problem is to run SchemaGen for such child classes.

  SchemaGen is a command line utility (located in the *WPS_Install_Home*/bin directory) provided to create XSD files for a given bean. These XSDs must be manually copied to the module for the solution to work.

EJB fault selector:

The EJB fault selector determines if an EJB invocation has resulted in a fault, a runtime exception, or a successful response.

If a fault is detected, the EJB fault selector returns the native fault name to the binding runtime so the JAX-WS data handler can convert the exception object into a fault business object.

On a successful (non-fault) response, the EJB import binding assembles a Java object array (Object[]) to return the values.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For exception scenarios, the binding assembles an Object[] and the first element represents the exception thrown by the method.

The fault selector can return any of the following values:

Table 30. Return values

Type	Return value	Description
Fault	**ResponseType.FAULT**	Returned when the passed Object[] contains an exception object.
Runtime exception	**ResponseType.RUNTIME**	Returned if the exception object does not match any of the declared exception types on the method.
Normal response	**ResponseType.RESPONSE**	Returned in all other cases.

If the fault selector returns a value of **ResponseType.FAULT**, the native fault name is returned. This native fault name is used by the binding to determine the corresponding WSDL fault name from the model and to invoke the correct fault data handler.

EJB function selector:

The EJB bindings use an import function selector (for outbound processing) or an export function selector (for inbound processing) to determine the EJB method to call.

Import function selector

For outbound processing, the import function selector derives the EJB method type based on the name of the operation invoked by the SCA component that is wired to the EJB import. The function selector looks for the @WebMethod annotation on the Integration Designer-generated JAX-WS annotated Java class to determine the associated target operation name.

- If the @WebMethod annotation is present, the function selector uses the @WebMethod annotation to determine the correct Java method mapping for the WSDL method.
- If the @WebMethod annotation is missing, the function selector assumes that the Java method name is the same as the invoked operation name.

Note: This function selector is valid only for a WSDL-typed interface on an EJB import, not for a Java-typed interface on an EJB import.

The function selector returns a java.lang.reflect.Method object that represents the method of the EJB interface.

The function selector uses a Java Object array (Object[]) to contain the response from the target method. The first element in the Object[] is a Java method with the name of the WSDL, and the second element in the Object[] is the input business object.

Export function selector

For inbound processing, the export function selector derives the target method to be invoked from the Java method.

The export function selector maps the Java operation name invoked by the EJB client into the name of the operation in the interface of the target component. The method name is returned as a String and is resolved by the SCA runtime depending on the interface type of the target component.

EIS bindings:

Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved using EIS exports and EIS imports that support JCA 1.5 resource adapters and Websphere Adapters.

Your SCA components might require that data be transferred to or from an external EIS. When you create an SCA module requiring such connectivity, you will include (in addition to your SCA component) an import or export with an EIS binding for communication with a specific external EIS.

Resource adapters in IBM Integration Designer are used within the context of an import or an export. You develop an import or an export with the external service wizard and, in developing it, include the resource adapter. An EIS import, which lets your application invoke a service on an EIS system, or an EIS export, which lets an application on an EIS system invoke a service developed in IBM Integration Designer, are created with a resource adapter. For example, you would create an import with the JD Edwards adapter to invoke a service on the JD Edwards system.

When you use the external service wizard, the EIS binding information is created for you. You can also use another tool, the assembly editor, to add or modify binding information. See the Accessing external services with adapters for more information.

After the SCA module containing the EIS binding is deployed to the server, you can use the administrative console to view information about the binding or to configure the binding.

EIS bindings overview:

The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.

The following example shows how an SCA module named ContactSyncModule synchronizes contact information between a Siebel system and an SAP system.

1. The SCA component named ContactSync listens (by way of an EIS application export named Siebel Contact) for changes to Siebel contacts.
2. The ContactSync SCA component itself makes use of an SAP application (through an EIS application import) in order to update the SAP contact information accordingly.

Because the data structures used for storing contacts are different in Siebel and SAP systems, the ContactSync SCA component must provide mapping.

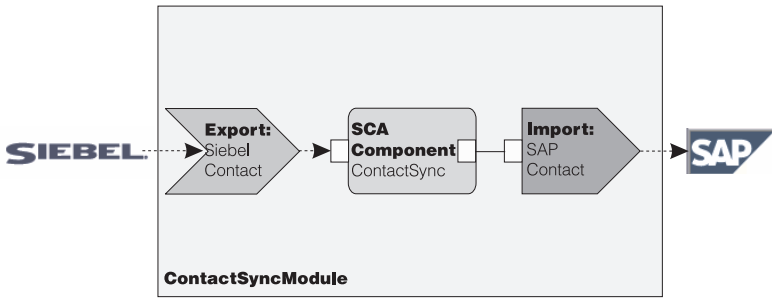


Figure 31. Flow from a Siebel system to an SAP system

The Siebel Contact export and the SAP Contact import have the appropriate resource adapters configured.

Key features of EIS bindings:

An EIS import is a Service Component Architecture (SCA) import that allows components in the SCA module to use EIS applications defined outside the SCA module. An EIS import is used to transfer data from the SCA component to an external EIS; an EIS export is used to transfer data from an external EIS into the SCA module.

Imports

The role of the EIS import is to bridge the gap between SCA components and external EIS systems. External applications can be treated as an EIS import. In this case, the EIS import sends data to the external EIS and optionally receives data in response.

The EIS import provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an external EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

On the client side of the import, there is an interface, exposed by the EIS import application, with one or more methods, each taking data objects as arguments and return values. On the implementation side, there is a Common Client Interface (CCI) implemented by a resource adapter.

The runtime implementation of an EIS import connects the client-side interface and the CCI. The import maps the invocation of the method on the interface to the invocation on the CCI.

Bindings are created at three levels: the interface binding, which then uses the contained method bindings, which in turn use data bindings.

The interface binding relates the interface of the import to the connection to the EIS system providing the application. This reflects the fact that the set of applications, represented by the interface, is provided by the specific instance of the EIS and the connection provides access to this instance. The binding element contains properties with enough information to create the connection (these properties are part of the javax.resource.spi.ManagedConnectionFactory instance).

The method binding associates the method with the specific interaction with the EIS system. For JCA, the interaction is characterized by the set of properties of the javax.resource.cci.InteractionSpec interface implementation. The interaction element of the method binding contains these properties, along with the name of the class, thus providing enough information to perform the interaction. The method binding uses data bindings describing the mapping of the argument and result of the interface method to EIS representation.

The runtime scenario for an EIS import is as follows:

1. The method on the import interface is invoked using the SCA programming model.
2. The request, reaching the EIS import, contains the name of the method and its arguments.
3. The import first creates an interface binding implementation; then, using data from the import binding, it creates a ConnectionFactory and associates the two. That is, the import calls setConnectionFactory on the interface binding.
4. The method binding implementation matching the invoked method is created.
5. The javax.resource.cci.InteractionSpec instance is created and populated; then, data bindings are used to bind the method arguments to a format understood by the resource adapter.
6. The CCI interface is used to perform the interaction.

7. When the call returns, the data binding is used to create the result of the invocation, and the result is returned to the caller.

Exports

The role of the EIS export is to bridge the gap between an SCA component and an external EIS. External applications can be treated as an EIS export. In this case, the external application sends its data in the form of periodic notifications. An EIS export can be thought of as a subscription application listening to an external request from an EIS. The SCA component that uses the EIS export views it as a local application.

The EIS export provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

The export features a listener implementation receiving requests from the EIS. The listener implements a resource adapter-specific listener interface. The export also contains a component implementing interface, exposed to the EIS through the export.

The runtime implementation of an EIS export connects the listener with the component implementing interface. The export maps the EIS request to the invocation of the appropriate operation on the component. Bindings are created at three levels: a listener binding, which then uses a contained native method binding, which in turn uses a data binding .

The listener binding relates the listener receiving requests with the component exposed through the export. The export definition contains the name of the component; the runtime locates it and forwards requests to it.

The native method binding associates the native method or the event type received by the listener to the operation implemented by the component exposed by way of the export. There is no relationship between the method invoked on the listener and the event type; all the events arrive through one or more methods of the listener. The native method binding uses the function selector defined in the export to extract the native method name from the inbound data and data bindings to bind the data format of the EIS to a format understood by the component.

The runtime scenario for an EIS export is as follows:

1. The EIS request triggers invocation of the method on the listener implementation.
2. The listener locates and invokes the export, passing to it all the invocation arguments.
3. The export creates the listener binding implementation.
4. The export instantiates the function selector and sets it on the listener binding.
5. The export initializes native method bindings and adds them to the listener binding. For each native method binding, the data bindings are also initialized.
6. The export invokes the listener binding.
7. The listener binding locates exported components and uses the function selector to retrieve the native method name.
8. This name is used to locate the native method binding, which then invokes the target component.

The adapter interaction style allows for the EIS export binding to invoke the target component either asynchronously (the default) or synchronously.

Resource adapters

You develop an import or an export with the external service wizard and, in developing it, include a resource adapter. The adapters that come with IBM Integration Designer used to access CICS, IMS, JD

Edwards, PeopleSoft, SAP and Siebel systems are intended for development and test purposes only. This means you use them to develop and test your applications.

Once you deploy your application, you will need licensed runtime adapters to run your application. However, when you build your service you can embed the adapter with your service. Your adapter licensing might allow you to use the embedded adapter as the licensed runtime adapter. These adapters are compliant with the Java EE Connector Architecture (JCA 1.5). JCA, an open standard, is the Java EE standard for EIS connectivity. JCA provides a managed framework; that is, Quality of Service (QoS) is provided by the application server, which offers lifecycle management and security to transactions. They are also compliant with the Enterprise Metadata Discovery specification with the exception of IBM CICS ECI Resource Adapter and IBM IMS Connector for Java.

The WebSphere Business Integration Adapters, an older set of adapters, are also supported by the wizard.

Java EE resources

The EIS module, an SCA module that follows the EIS module pattern, can be deployed to the Java EE platform.

The deployment of the EIS module to the Java EE platform results in an application that is ready to execute, packaged as an EAR file and deployed to the server. All Java EE artifacts and resources are created; the application is configured and ready to be run.

JCA Interaction Spec and Connection Spec dynamic properties:

The EIS binding can accept input for the InteractionSpec and ConnectionSpec specified by using a well-defined child data object that accompanies the payload. This allows for dynamic request-response interactions with a resource adapter through the InteractionSpec and component authentication through the ConnectionSpec.

The javax.cci.InteractionSpec carries information on how the interaction request with the resource adapter should be handled. It can also carry information on how the interaction was achieved after the request. These two-way communications through the interactions are sometimes referred to as *conversations*.

The EIS binding expects the payload that will be the argument to the resource adapter to contain a child data object called **properties**. This property data object will contain name/value pairs, with the name of the Interaction Spec properties in a specific format. The formatting rules are:

- Names must begin with the prefix **IS**, followed by the property name. For example, an interaction spec with a JavaBeans property called **InteractionId** would specify the property name as **ISInteractionId**.
- The name/value pair represents the name and the value of the simple type of the Interaction Spec property.

In this example, an interface specifies that the input of an operation is an **Account** data object. This interface invokes an EIS import binding application with the intention to send and receive a dynamic InteractionSpec property called **workingSet** with the value **xyz**.

The business graph or business objects in the server contain an underlying **properties** business object that permits the sending of protocol-specific data with the payload. This **properties** business object is built-in and does not need to be specified in the XML schema when constructing a business object. It only needs to be created and used. If you have your own data types defined based on an XML schema, you need to specify a **properties** element that contains your expected name/value pairs.

```
 BOFactory dataFactory = (BOFactory) \
 serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
  //Wrapper for doc-lit wrapped style interfaces,
  //skip to payload for non doc-lit
  DataObject docLitWrapper = dataFactory.createByElement /
  ("http://mytest/eis/Account", "AccountWrapper");
```

Create the payload.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload


  //Construct properties data for dynamic interaction

  DataObject properties = account.createDataObject("properties");
```

For name workingSet, set the value expected (**xyz**).

```
properties.setString("ISworkingSet", "xyz");


  //Invoke the service with argument

  Service accountImport = (Service) \
  serviceManager.locateService("AccountOutbound");
  DataObject result = accountImport.invoke("createAccount", docLitWrapper);

  //Get returned property
  DataObject retProperties = result.getDataObject("properties");

  String workingset = retProperties.getString("ISworkingSet");
```

You can use ConnectionSpec properties for dynamic component authentication. The same rules apply as above, except that the property name prefix needs to be **CS** (instead of **IS**). ConnectionSpec properties are not two-way. The same **properties** data object can contain both IS and CS properties.

To use ConnectionSpec properties, set the **resAuth** specified on the import binding to **Application**. Also, make sure the resource adapter supports component authorization. See chapter 8 of the J2EE Connector Architecture Specification for more details.

External clients with EIS bindings:

The server can send messages to, or receive messages from, external clients using EIS bindings.

An external client, for example a Web portal or an EIS, needs to send a message to an SCA module in the server or needs to be invoked by a component from within the server.

The client invokes the EIS import as with any other application, using either the Dynamic Invocation Interface (DII) or Java interface.

1. The external client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a service interface implementation.
2. The client creates an input argument, a generic data object, created dynamically using the data object schema. This step is done using the Service Data Object DataFactory interface implementation.
3. The external client invokes the EIS and obtains the required results.

Alternatively, the client can invoke the EIS import using the Java interface.

1. The client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a Java interface of the EIS import.
2. The client creates an input argument and a typed data object.

3. The client invokes EIS and obtains the required results.

The EIS export interface defines the interface of the exported SCA component that is available to the external EIS applications. This interface can be thought of as the interface that an external application (such as SAP or PeopleSoft) will invoke through the implementation of the EIS export application runtime.

The export uses EISExportBinding to bind exported services to the external EIS application. It allows you to subscribe an application contained in your SCA module to listen for EIS service requests. The EIS export binding specifies the mapping between the definition of inbound events as it is understood by the resource adapter (using Java EE Connector Architecture interfaces) and the invocation of SCA operations.

The EISExportBinding requires external EIS services to be based on Java EE Connector Architecture 1.5 inbound contracts. The EISExportBinding requires that a data handler or data binding be specified either at the binding level or the method level.

JMS bindings:

A Java Message Service (JMS) provider enables messaging based on the Java Messaging Service API and programming model. It provides JMS connection factories to create connections for JMS destinations and to send and receive messages.

JMS bindings can be used when you interact with the Service Integration Bus (SIB) provider binding, and are compliant with JMS and JCA 1.5.

You can use the JMS export and import bindings a Service Component Architecture (SCA) module to make calls to, and receive messages from, external JMS systems.

The JMS import and export bindings provide integration with JMS applications using the JCA 1.5-based SIB JMS provider that is part of WebSphere Application Server. Other JCA 1.5–based JMS resource adapters are not supported

JMS bindings overview:

JMS bindings provide connectivity between the Service Component Architecture (SCA) environment and JMS systems.

JMS bindings

The major components of both JMS import and JMS export bindings are:

- Resource adapter: enables managed, bidirectional connectivity between an SCA module and external JMS systems
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Key features of JMS bindings

Special headers

Special header properties are used in JMS imports and exports to tell the target how to handle the message.

For example, TargetFunctionName maps from the native method to the operation method.

Java EE resources

A number of Java EE resources is created when JMS imports and exports are deployed to a Java EE environment.

ConnectionFactory
Used by clients to create a connection to the JMS provider.

ActivationSpec
Imports use this for receiving the response to a request; exports use it when configuring the message endpoints that represent message listeners in their interactions with the messaging system.

Destinations

- Send destination: on an import, this is where the request or outgoing message is sent; on an export, this is the destination where the response message will be sent, if not superseded by the JMSReplyTo header field in the incoming message.
- Receive destination: where the incoming message should be placed; with imports, this is a response; with exports, this is a request.
- Callback destination: SCA JMS system destination used to store correlation information. Do not read or write to this destination.

The installation task creates the ConnectionFactory and three destinations. It also creates the ActivationSpec to enable the runtime message listener to listen for replies on the receive destination. The properties of these resources are specified in the import or export file.

JMS integration and resource adapters:

The Java Message Service (JMS) provides integration through an available JMS JCA 1.5-based resource adapter. Complete support for JMS integration is provided for the Service Integration Bus (SIB) JMS resource adapter.

Use a JMS provider for JCA 1.5 resource adapter when you want to integrate with an external JCA 1.5-compliant JMS system. External services compliant with JCA 1.5 can receive messages and send messages to integrate with your service component architecture (SCA) components using the SIB JMS resource adapter.

The use of other provider-specific JCA 1.5 resource adapters is not supported.

JMS import and export bindings:

You can make SCA modules interact with services provided by external JMS applications using JMS import and export bindings.

JMS import bindings

Connections to the associated JMS provider of JMS destinations are created by using a JMS connection factory. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The JMS import puts a message on the send destination configured in the import binding. Nothing is set in the replyTo field of the JMS header.

- Two-way (request-response): The JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default), or from the request message correlation ID. The import binding can also be configured to use a temporary dynamic response destination to correlate responses with requests. A temporary destination is created for each request and the import uses this destination to receive the response.

  The receive destination is set in the replyTo header property of the outbound message. A message listener is deployed to listen on the receive destination, and when a reply is received, the message listener passes the reply back to the component.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that JMS is an asynchronous binding. If a calling component invokes a JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 32 illustrates how the import is linked to the external service.

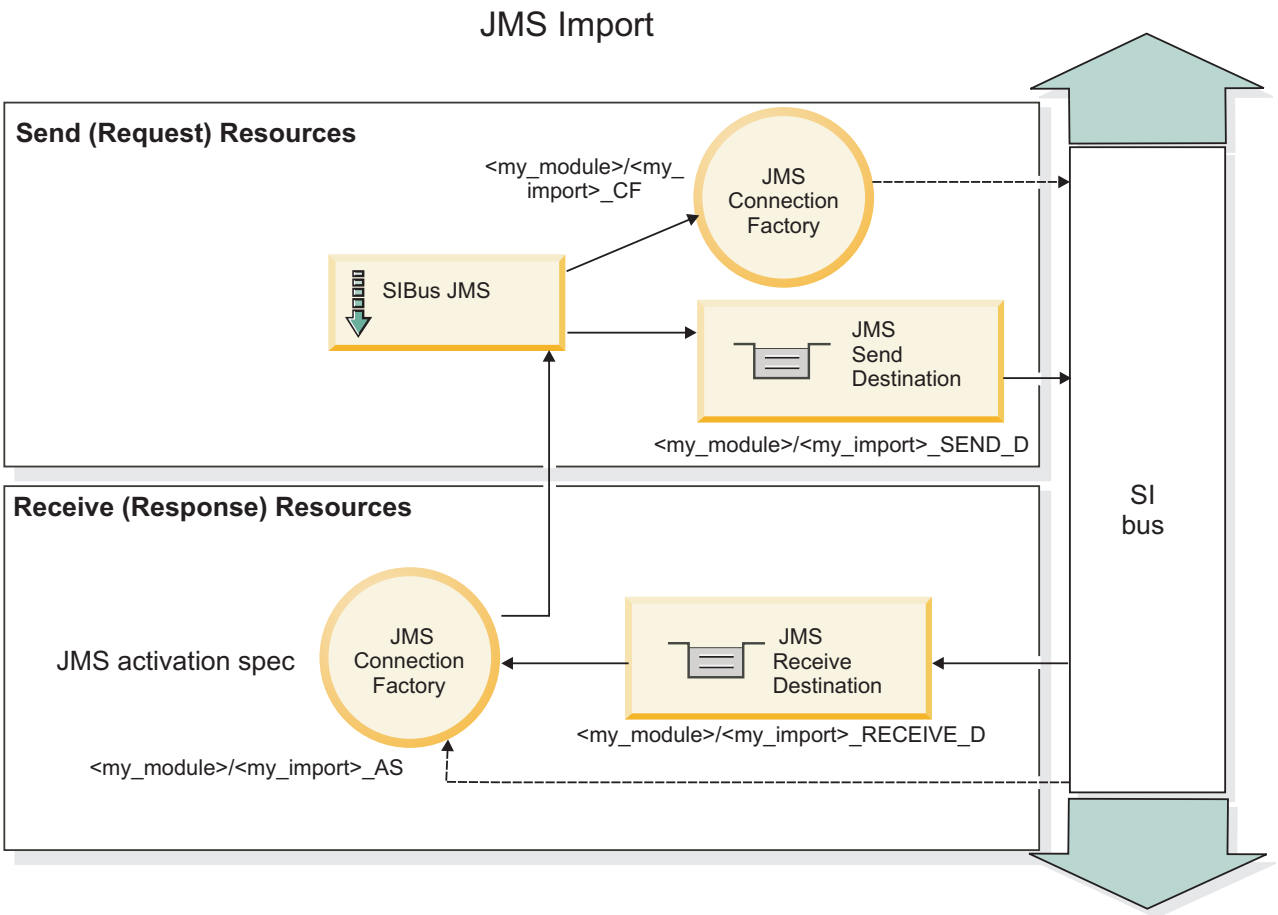


Figure 32. JMS import binding resources

JMS export bindings

JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection that is part of a JMS export is a configurable activation specification.

A JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

A message listener is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the incoming message overrides the destination specified in the send.

Figure 33 illustrates how the external requester is linked to the export.

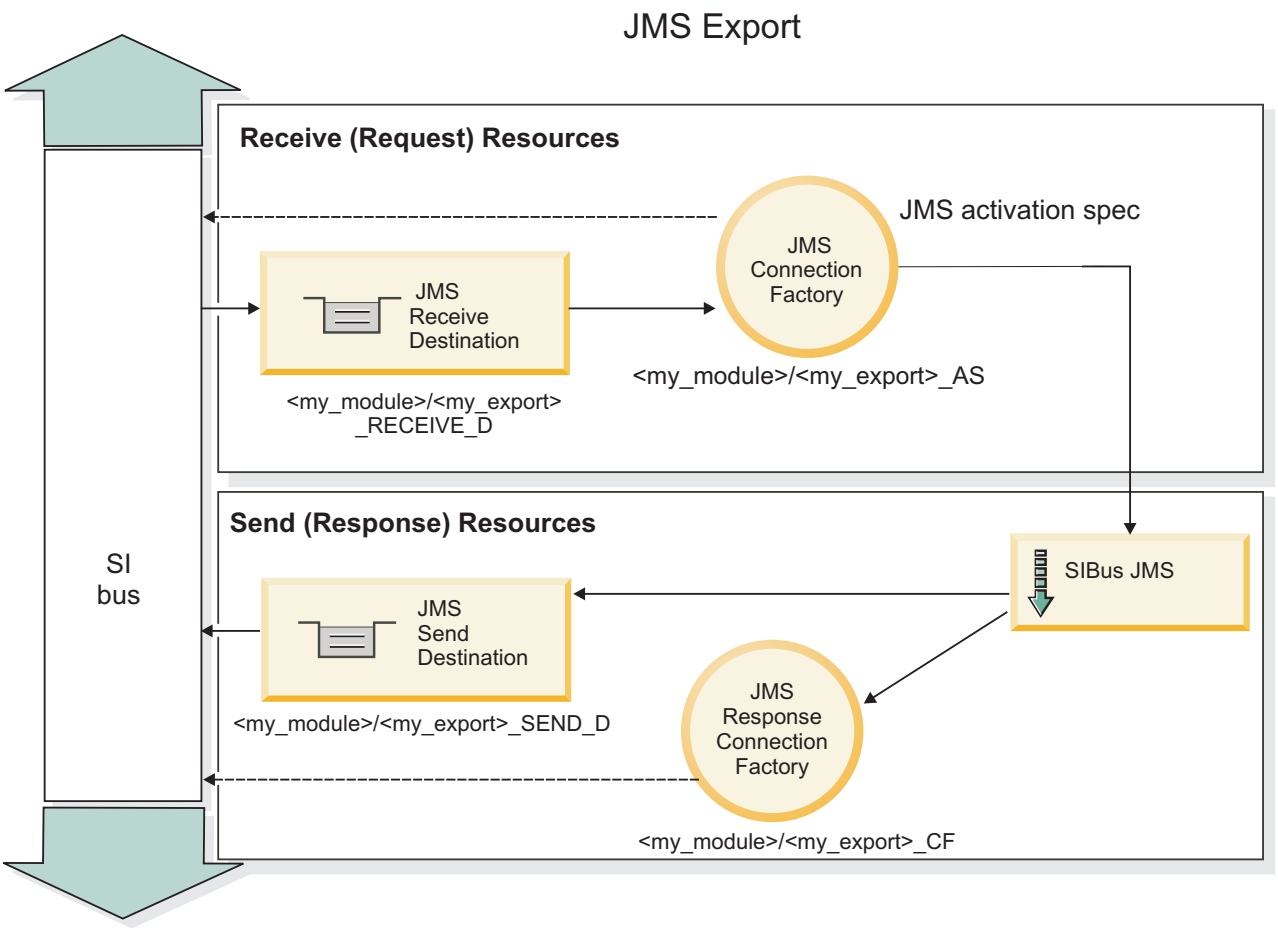


Figure 33. JMS export binding resources

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

JMS temporary dynamic response destination correlation scheme:

The temporary dynamic response destination correlation scheme causes a unique dynamic queue or topic to be created for each request sent.

The static response destination specified in the import is used to derive the nature of the temporary dynamic destination queue or topic. This is set in the **ReplyTo** field of the request, and the JMS import listens for responses on that destination. When the response is received it is requeued to the static response destination for asynchronous processing. The **CorrelationID** field of the response is not used, and does not need to be set.

Transactional issues

When a temporary dynamic destination is being used, the response must be consumed in the same thread as the sent response. The request must be sent outside the global transaction, and must be committed before it is received by the backend service, and a response returned.

Persistence

Temporary dynamic queues are short-lived entities and do not guarantee the same level of persistence associated with a static queue or topic. A temporary dynamic queue or topic will not survive a server restart and neither will messages. After the message has been requeued to the static response destination it retains the persistence defined in the message.

Timeout

The import waits to receive the response on the temporary dynamic response destination for a fixed amount of time. This time interval will be taken from the SCA Response Expiration time qualifier, if it is set, otherwise the time defaults to 60 seconds. If the wait time is exceeded the import throws a `ServiceTimeoutRuntimeException`.

External clients:

The server can send messages to, or receive messages from, external clients using JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA module in the server, or it can be invoked by a component from within the server.

The JMS export components deploy message listeners to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications with the export binding.

JMS imports interact with external clients by sending messages to, and receiving messages from, JMS queues.

Working with external clients:

An external client (that is, outside the server) might need to interact with an application installed in the server.

Consider a very simple scenario in which an external client wants to interact with an application on the server. The figure depicts a typical simple scenario.

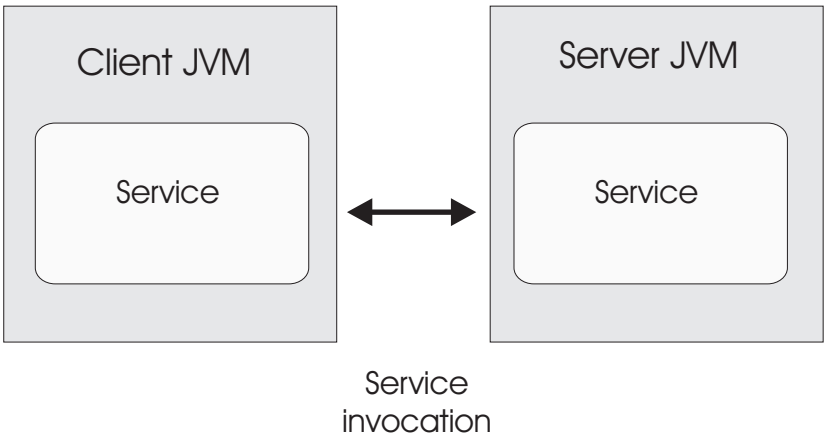


Figure 34. Simple use-case scenario: external client interacts with server application

The SCA application includes an export with a JMS binding; this makes the application available to external clients.

When you have an external client in a Java virtual machine (JVM) separate from your server, there are several steps you must take in order to make a connection and interact with a JMS export. The client obtains an InitialContext with the correct values and then looks up the resources through JNDI. The client then uses the JMS 1.1 specification client to access the destinations and the send and receive messages on the destinations.

The default JNDI names of the resources created automatically by the runtime are listed in the configuration topic of this section. However, if you have pre-created resources, use those JNDI names.

1. Configure JMS destinations and the connection factory to send the message.
2. Make sure that the JNDI context, the port for the SIB resource adapter, and the messaging bootstrapping port are correct.

   The server uses some default ports, but if there are more servers installed on that system, alternate ports are created at installation time to avoid conflicts with other server instances. You can use the administrative console to determine which ports your server is employing . Go to **Servers** > **Application Servers** > *your_server_ name* > **Configuration** and click **Ports** under **Communication**. You can then edit the port being used.
3. The client obtains an initial context with the correct values and then looks up the resources through JNDI.
4. Using JMS 1.1 specifications, the client accesses the destinations and the send and receive messages on the destinations.

Troubleshooting JMS bindings:

You can diagnose and fix problems with JMS bindings.

Implementation exceptions

In response to various error conditions, the JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Primary failure conditions

The primary failure conditions of JMS bindings are determined by transactional semantics, by JMS provider configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the JMS provider or destination.

  A failure to connect to the JMS provider to receive messages will result in the message listener failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the destination until they are successfully retrieved (or expired).

  A failure to connect to the JMS provider to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding or data handler causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected late response messages.

  The import expects only one response message for each request message. Also the valid time period in which a response can be received is determined by the SCA Response Expiration qualifier on the request. When a response arrives or the expiration time is exceeded, the correlation record is deleted. If response messages arrive unexpectedly or arrive late, a Service Runtime Exception is thrown.
- Service timeout runtime exception caused by late response when using the temporary dynamic response destination correlation scheme.

  The JMS import will timeout after a period of time determined by the SCA response expiration qualifier, or if this is not set it will default to 60 seconds.

JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the underlying SIB destination of the JMS destination has a maximum failed deliveries value greater than **1**. Setting this value to **2** or more enables interaction with the failed event manager during SCA invocations for the JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

When any runtime exception, including a **DataBindingException**, is thrown:

- If the mediation flow is configured to be transactional, the JMS message , by default, is stored in the Failed Event Manager for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the Failed Event Manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

Generic JMS bindings:

The Generic JMS binding provides connectivity to third-party JMS 1.1 compliant providers. The operation of the Generic JMS bindings is similar to that of JMS bindings.

The service provided through a JMS binding allows a Service Component Architecture (SCA) module to make calls or receive messages from external systems. The system can be an external JMS system.

The Generic JMS binding provides integration with non-JCA 1.5-compliant JMS providers that support JMS 1.1 and implement the optional JMS Application Server Facility. The Generic JMS binding supports those JMS providers (including Oracle AQ, TIBCO, SonicMQ, WebMethods, and BEA WebLogic) that do not support JCA 1.5 but do support the Application Server Facility of the JMS 1.1 specification. The WebSphere embedded JMS provider (SIBJMS), which is a JCA 1.5 JMS provider, is not supported by this binding; when using that provider, use the "JMS bindings" on page 117.

Use this Generic binding when integrating with a non-JCA 1.5-compliant JMS-based system within an SCA environment. The target external applications can then receive messages and send messages to integrate with an SCA component.

Generic JMS bindings overview:

Generic JMS bindings are non-JCA JMS bindings that provide connectivity between the Service Component Architecture (SCA) environment and JMS systems that are compliant with JMS 1.1 and that implement the optional JMS Application Server Facility.

Generic JMS bindings

The major aspects of Generic JMS import and export bindings include the following:
- Listener port: enables non-JCA-based JMS providers to receive messages and dispatch them to a Message Driven Bean (MDB)
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Generic JMS import bindings

Generic JMS import bindings allow components within your SCA module to communicate with services provided by external non-JCA 1.5-compliant JMS providers.

The connection part of a JMS import is a connection factory. A connection factory, the object a client uses to create a connection to a provider, encapsulates a set of connection configuration parameters defined by an administrator. Each connection factory is an instance of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the Generic JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The Generic JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The Generic JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The receive destination is set in the replyTo header property of the outbound message. A message driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the Generic JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that Generic JMS is an asynchronous binding. If a calling component invokes a Generic JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 35 illustrates how the import is linked to the external service.

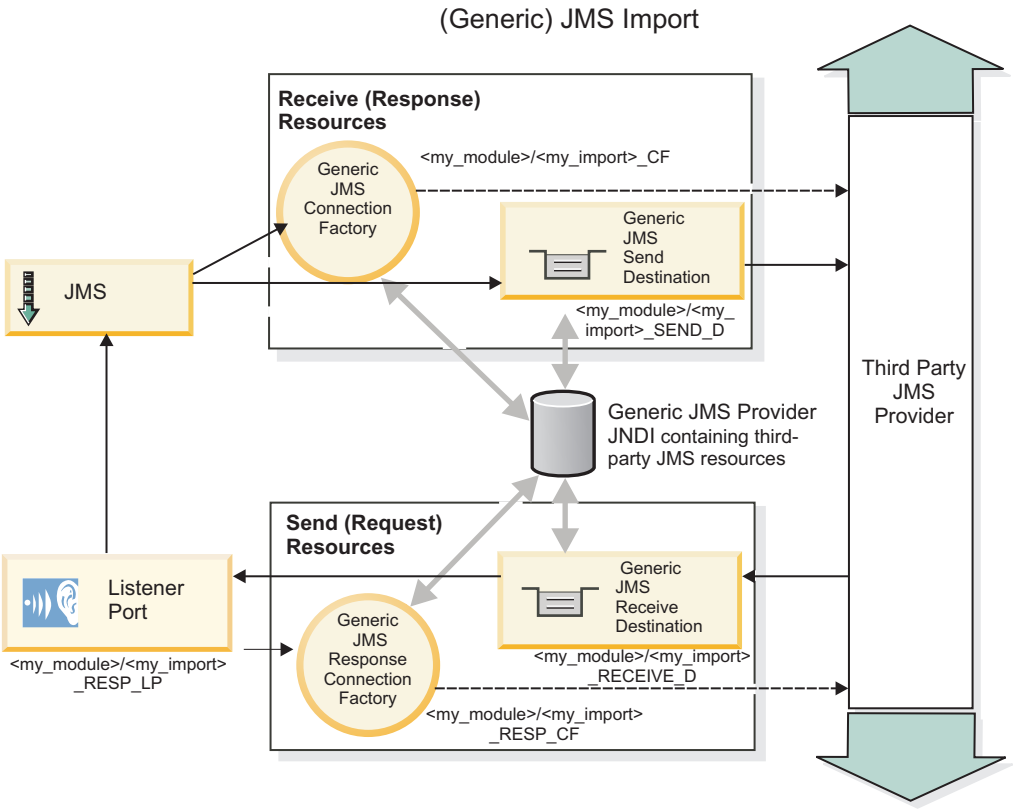


Figure 35. Generic JMS import binding resources

Generic JMS export bindings

Generic JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection part of a JMS export is composed of a ConnectionFactory and a ListenerPort.

A Generic JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding.

- The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply.
- The destination specified in the replyTo field of the incoming message overrides the destination specified in the send field.
- For request/response scenarios, the import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response to copy the request message ID to the correlation ID field of the response message (default), or the response can copy the request correlation ID to the correlation ID field of the response message.

Figure 36 illustrates how the external requester is linked to the export.

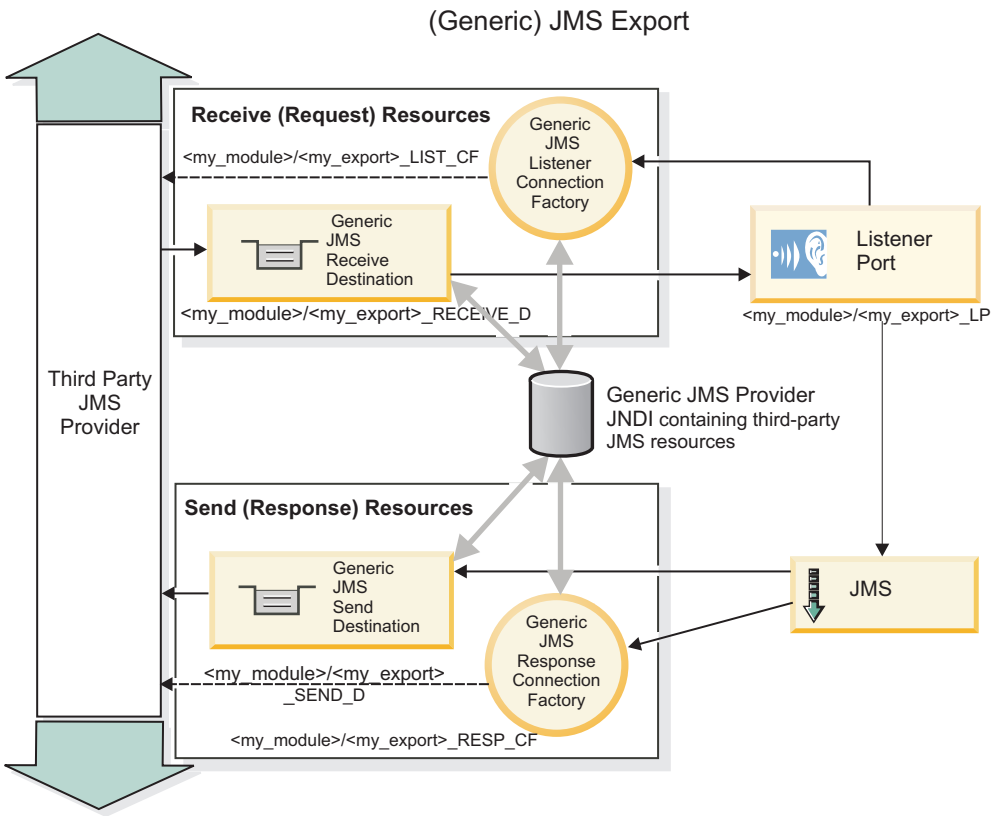


Figure 36. Generic JMS export binding resources

Key features of Generic JMS bindings:

The features of the Generic JMS import and export binding are consistent with those of the WebSphere embedded JMS and MQ JMS import bindings. Key features include header definitions and access to existing Java EE resources. However, because of its generic nature, there are no JMS provider-specific connectivity options, and this binding has limited capability to generate resources at deployment and installation.

Generic imports

Like the MQ JMS import application, the Generic JMS implementation is asynchronous and supports three invocations: one-way, two-way (also known as request-response), and callback.

When the JMS import is deployed, a message driven bean (MDB) provided by the runtime environment is deployed. The MDB listens for replies to the request message. The MDB is associated with (listens on) the destination sent with the request in the replyTo header field of the JMS message.

Generic exports

Generic JMS export bindings differ from EIS export bindings in their handling of the return of the result. A Generic JMS export explicitly sends the response to the replyTo destination specified on the incoming message. If none is specified, the send destination is used.

When the Generic JMS export is deployed, a message driven bean (a different MDB than the one used for Generic JMS imports) is deployed. It listens for the incoming requests on the receive destination and then dispatches the requests to be processed by the SCA runtime.

Special headers

Special header properties are used in Generic JMS imports and exports to tell the target binding how to handle the message.

For example, the TargetFunctionName property is used by the default function selector to identify the name of the operation in the export interface that is being invoked.

Note: The import binding can be configured to set the TargetFunctionName header to the operation name for each operation.

Java EE resources

A number of Java EE resources are created when a JMS binding is deployed to a Java EE environment.

- Listener port for listening on the receive (response) destination (two-way only) for imports and on the receive (request) destination for exports
- Generic JMS connection factory for the outboundConnection (import) and inboundConnection (export)
- Generic JMS destination for the send (import) and receive (export; two-way only) destinations
- Generic JMS connection factory for the responseConnection (two-way only and optional; otherwise, outboundConnection is used for imports, and inboundConnection is used for exports)
- Generic JMS destination for the receive (import) and send (export) destination (two-way only)
- Default messaging provider callback JMS destination used to access the SIB callback queue destination (two-way only)
- Default messaging provider callback JMS connection factory used to access the callback JMS destination (two-way only)
- SIB callback queue destination used to store information about the request message for use during response processing (two-way only)

The installation task creates the ConnectionFactory, the three destinations, and the ActivationSpec from the information in the import and export files.

Generic JMS headers:

Generic JMS headers are Service Data Objects (SDO) that contain all the properties of the Generic JMS message properties. These properties can be from the inbound message or they can be the properties that will be applied to the outbound message.

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

The following properties are set statically on the methodBinding:

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

The Generic JMS binding also supports dynamic modification of JMS headers and properties in the same manner as the JMS and MQ JMS bindings.

Some Generic JMS providers place restrictions on which properties can be set by the application and in what combinations. You must consult your third-party product documentation for more information. However, an additional property has been added to the methodBinding, ignoreInvalidOutboundJMSProperties, which allows any exceptions to be propagated.

The Generic JMS headers and message properties are used only when the base service component architecture SCDL binding switch is turned on. When the switch is turned on, context information is propagated. By default, this switch is on. To prevent context information propagation, change the value to **false**.

When context propagation is enabled, header information is allowed to flow to the message or to the target component. To turn on and off context propagation, specify **true** or **false** for the contextPropagationEnabled attribute of the import and export bindings. For example:

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

The default is **true**.

Troubleshooting Generic JMS bindings:

You can diagnose and fix problems with Generic JMS binding.

Implementation exceptions

In response to various error conditions, the Generic JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

Troubleshooting Generic JMS message expiry

A request message by the JMS provider is subject to expiration.

Request expiry refers to the expiration of a request message by the JMS provider when the JMSExpiration time on the request message is reached. As with other JMS bindings, the Generic JMS binding handles the request expiry by setting expiration on the callback message placed by the import to be the same as for the outgoing request. Notification of the expiration of the callback message will indicate that the request message has expired and the client should be notified by means of a business exception.

If the callback destination is moved to the third-party provider, however, this type of request expiry is not supported.

Response expiry refers to the expiration of a response message by the JMS provider when the JMSExpiration time on the response message is reached.

Response expiry for the generic JMS binding is not supported, because the exact expiry behavior of a third-party JMS provider is not defined. You can, however, check that the response is not expired if and when it is received.

For outbound request messages, the JMSExpiration value will be calculated from the time waited and from the requestExpiration values carried in the asyncHeader, if set.

Troubleshooting Generic JMS connection factory errors

When you define certain types of connection factories in your Generic JMS provider, you might receive an error message when you try to start an application. You can modify the external connection factory to avoid this problem.

When launching an application, you might receive the following error message:

```
MDB Listener Port JMSConnectionFactory type does not match
JMSDestination type
```

This problem can arise when you are defining external connection factories. Specifically, the exception can be thrown when you create a JMS 1.0.2 Topic Connection Factory, instead of a JMS 1.1 (unified) Connection Factory (that is, one that is able to support both point-to-point and publish/subscribe communication).

To resolve this issue, take the following steps:

1. Access the Generic JMS provider that you are using.
2. Replace the JMS 1.0.2 Topic Connection Factory that you defined with a JMS 1.1 (unified) Connection Factory.

When you launch the application with the newly defined JMS 1.1 Connection Factory, you should no longer receive an error message.

Generic JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a generic JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the generic JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

WebSphere MQ JMS bindings:

The WebSphere MQ JMS binding provides integration with external applications that use a WebSphere MQ JMS-based provider.

Use the WebSphere MQ JMS export and import bindings to integrate directly with external JMS or MQ JMS systems from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ JMS-based service by way of an import, the WebSphere MQ JMS import binding utilizes a destination to which data will be sent and a destination where the reply can be received. Conversion of the data to and from a JMS message is accomplished through the JMS data handler or data binding edge component.

When an SCA module provides a service to WebSphere MQ JMS clients, the WebSphere MQ JMS export binding utilizes a destination where the request can be received and the response can be sent. The conversion of the data to and from a JMS message is done through the JMS data handler or data binding.

The function selector provides a mapping to the operation on the target component to be invoked.

WebSphere MQ JMS bindings overview:

The WebSphere MQ JMS binding provides integration with external applications that use the WebSphere MQ JMS provider.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ JMS bindings will use, before running an application containing these bindings.

WebSphere MQ JMS import bindings

The WebSphere MQ JMS import allows components within your SCA module to communicate with services provided by WebSphere MQ JMS-based providers. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Two types of usage scenarios for WebSphere MQ JMS import bindings are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The WebSphere MQ JMS import puts a message on the send destination.

  The receive destination is set in the replyTo header field. A message-driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified.Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that WebSphere MQ JMS is an asynchronous binding. If a calling component invokes a WebSphere MQ JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 37 on page 133 illustrates how the import is linked to the external service.

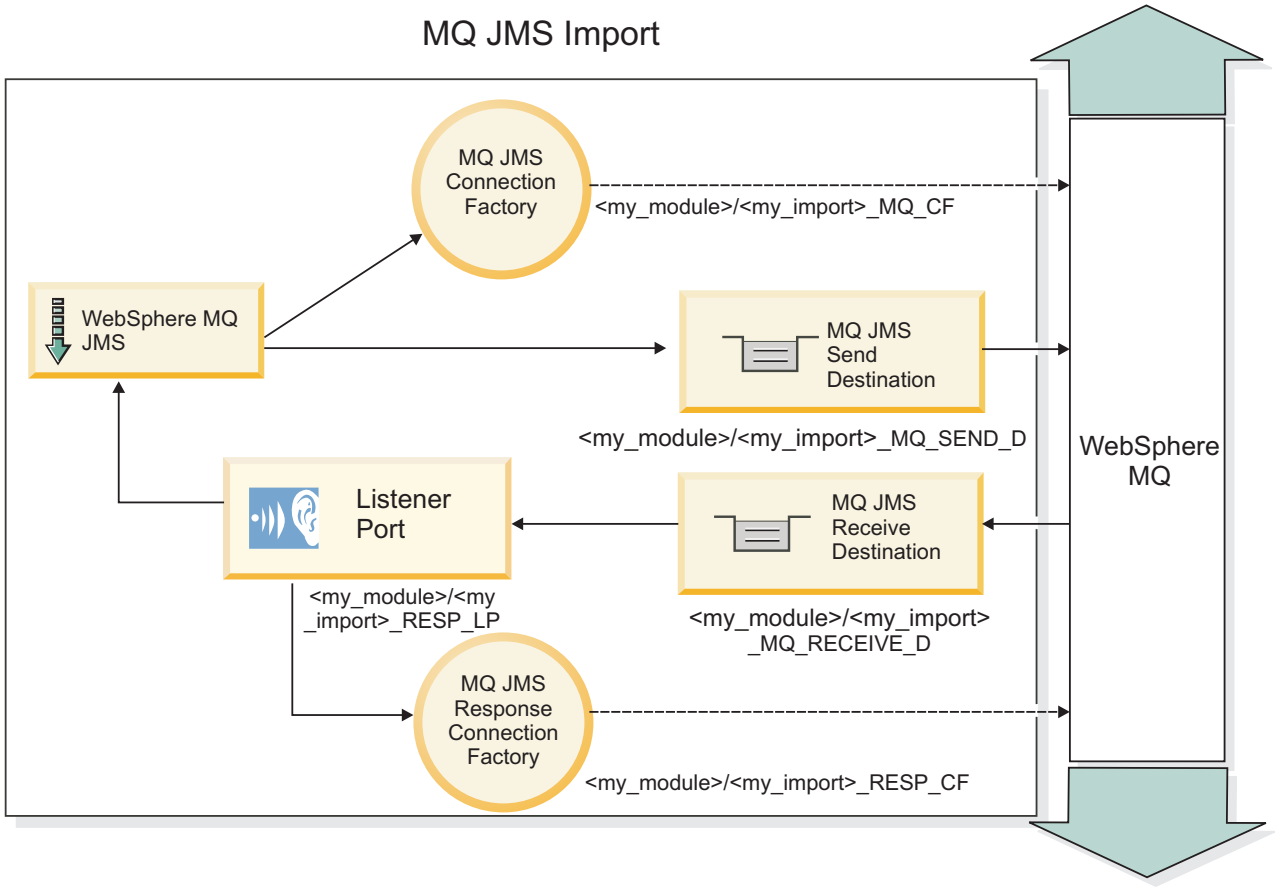


Figure 37. WebSphere MQ JMS import binding resources

WebSphere MQ JMS export bindings

The WebSphere MQ JMS export binding provides the means for SCA modules to provide services to external JMS applications on the WebSphere MQ-based JMS provider.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the response message overrides the destination specified in the send field.

Figure 38 on page 134 illustrates how the external requester is linked to the export.

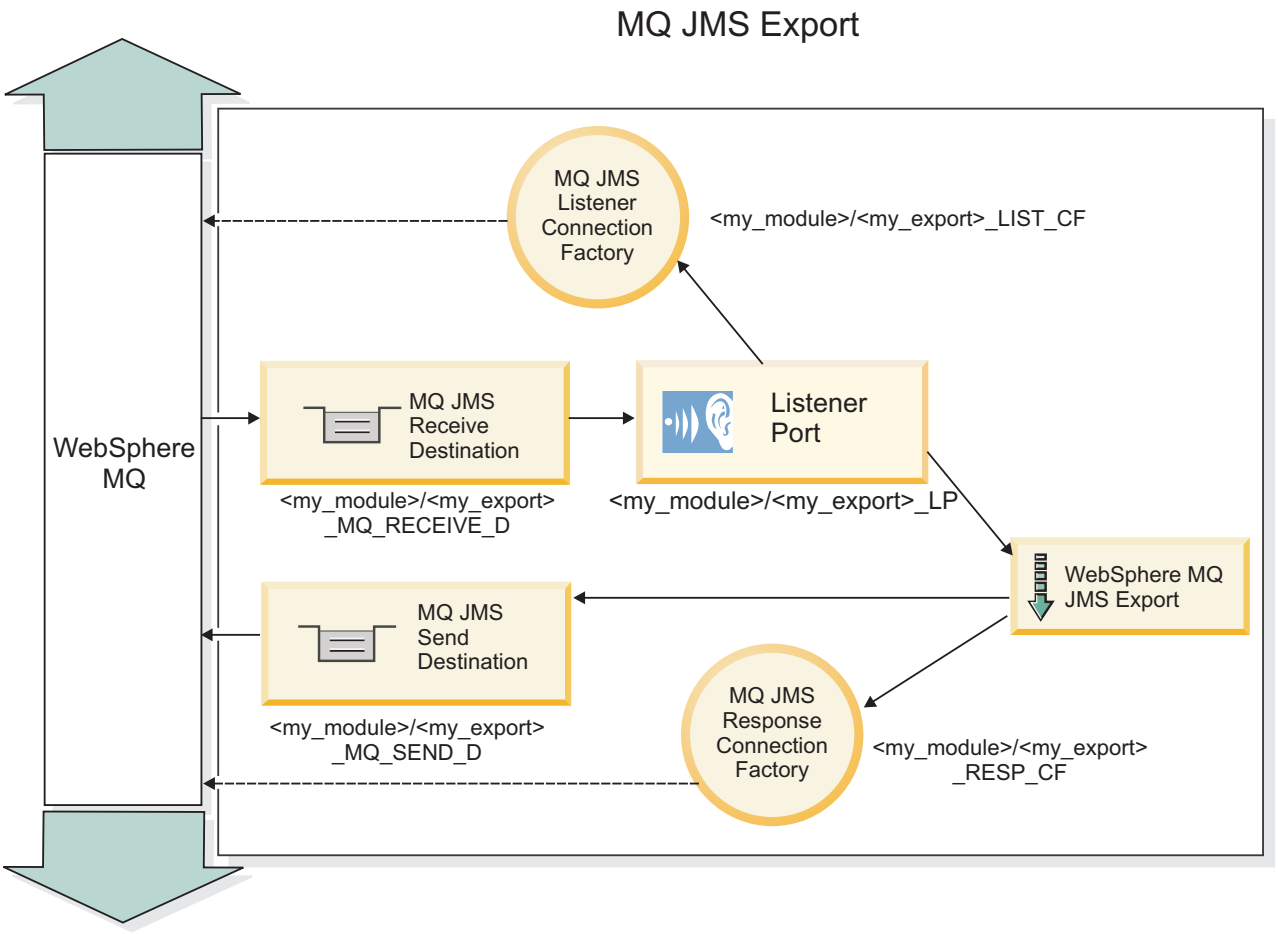


Figure 38. WebSphere MQ JMS export binding resources

Note: Figure 37 on page 133 and Figure 38 illustrate how an application from a previous version of IBM Business Process Manager is linked to an external service. For applications developed for IBM Business Process Manager Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of WebSphere MQ JMS bindings:

Key features of WebSphere MQ JMS bindings include headers, Java EE artifacts, and created Java EE resources.

Headers

A JMS message header contains a number of predefined fields containing values used by both clients and providers to identify and to route messages. You can use binding properties to configure these headers with fixed values, or the headers can be specified dynamically at runtime.

`JMSCorrelationID`
Links to a related message. Typically, this field is set to the message identifier string of the message that is being replied to.

`TargetFunctionName`
This header is used by one of the supplied function selectors to identify the operation being invoked. Setting the TargetFunctionName JMS header property in messages sent to a JMS export allows this function selector to be used. The property can be set directly in JMS client applications or when

connecting an import with a JMS binding to such an export. In this case, the JMS import binding should be configured to set the TargetFunctionName header for each operation in the interface to the name of the operation.

Correlation schemes

The WebSphere MQ JMS bindings provide various correlation schemes that are used to determine how to correlate request messages with response messages.

RequestMsgIDToCorrelID
The JMSMessageID is copied to the JMSCorrelationID field. This is the default setting.

RequestCorrelIDToCorrelID
The JMSCorrelationID is copied to the JMSCorrelationID field.

Java EE resources

A number of Java EE resources are created when an MQ JMS import is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the MQ JMS provider.

Response Connection Factory
Used by the SCA MQ JMS runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination:
  - Imports: Where the request or outgoing message is sent.
  - Exports: Where the response message will be sent if it is not superseded by the JMSReplyTo header field of the incoming message.
- Receive destination:
  - Imports: Where the response or incoming message should be placed.
  - Exports: Where the incoming or request message should be placed.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties

- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

External clients:

The server can send messages to, or receive messages from, external clients using WebSphere MQ JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ JMS export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications via the export binding.

WebSphere MQ JMS imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ JMS bindings:

You can diagnose and fix problems with WebSphere MQ JMS bindings.

Implementation exceptions

In response to various error conditions, the MQ JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

WebSphere MQ JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a WebSphere MQ JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than **1**. Setting this value to **1** or more enables interaction with the failed event manager during SCA invocations for the MQ JMS bindings.

Misusage scenarios: comparison with WebSphere MQ bindings

The WebSphere MQ JMS binding is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model. The WebSphere MQ import and export, however, are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data handler or data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

WebSphere MQ bindings:

The WebSphere MQ binding provides Service Component Architecture (SCA) connectivity with WebSphere MQ applications.

Use the WebSphere MQ export and import bindings to integrate directly with a WebSphere MQ-based system from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ service by way of an import, the WebSphere MQ import binding uses a queue to which data will be sent and a queue where the reply can be received.

When an SCA module provides a service to WebSphere MQ clients, the WebSphere MQ export binding uses a queue where the request can be received and the response can be sent. The function selector provides a mapping to the operation on the target component to be invoked.

Conversion of the payload data to and from an MQ message is done through the MQ body data handler or data binding. Conversion of the header data to and from an MQ message is done through the MQ header data binding.

For information about the WebSphere MQ versions supported, see the detailed system requirements web page.

WebSphere MQ bindings overview:

The WebSphere MQ binding provides integration with native MQ-based applications.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ bindings will use, before running an application containing these bindings.

WebSphere administrative tasks

You must set the **Native library path** property of the MQ resource adapter in Websphere to the WebSphere MQ version supported by the server, and restart the server. This ensures that the libraries of a supported version of WebSphere MQ are being used. Detailed hardware and software requirements can be found on the IBM support pages.

WebSphere MQ import bindings

The WebSphere MQ import binding allows components within your SCA module to communicate with services provided by external WebSphere MQ-based applications. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Interaction with external WebSphere MQ systems includes the use of queues for sending requests and receiving replies.

Two types of usage scenarios for the WebSphere MQ import binding are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field of the import binding. Nothing is sent to the replyTo field of the MQMD header.
- Two-way (request-response): The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field

  The receive queue is set in the replyTo MQMD header field. A message driven bean (MDB) is deployed to listen on the receive queue, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

It is important to note that WebSphere MQ is an asynchronous binding. If a calling component invokes a WebSphere MQ import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the WebSphere MQ service.

Figure 39 on page 140 illustrates how the import is linked to the external service.

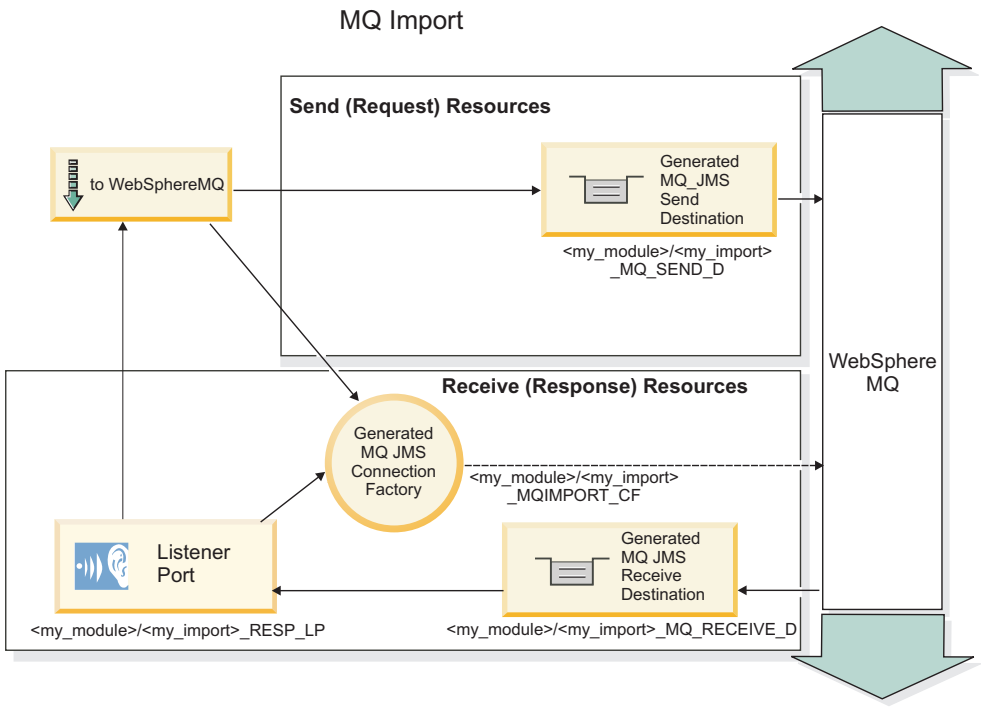


Figure 39. WebSphere MQ import binding resources

WebSphere MQ export bindings

The WebSphere MQ export binding provides the means for SCA modules to provide services to external WebSphere MQ-based applications.

An MDB is deployed to listen to requests incoming to the **Receive destination queue** specified in the export binding. The queue specified in the **Send destination queue** field is used to send the reply to the inbound request if the invoked component provides a reply. The queue specified in the replyTo field of the response message overrides the queue specified in the **Send destination queue** field.

Figure 40 on page 141 illustrates how the external requester is linked to the export.

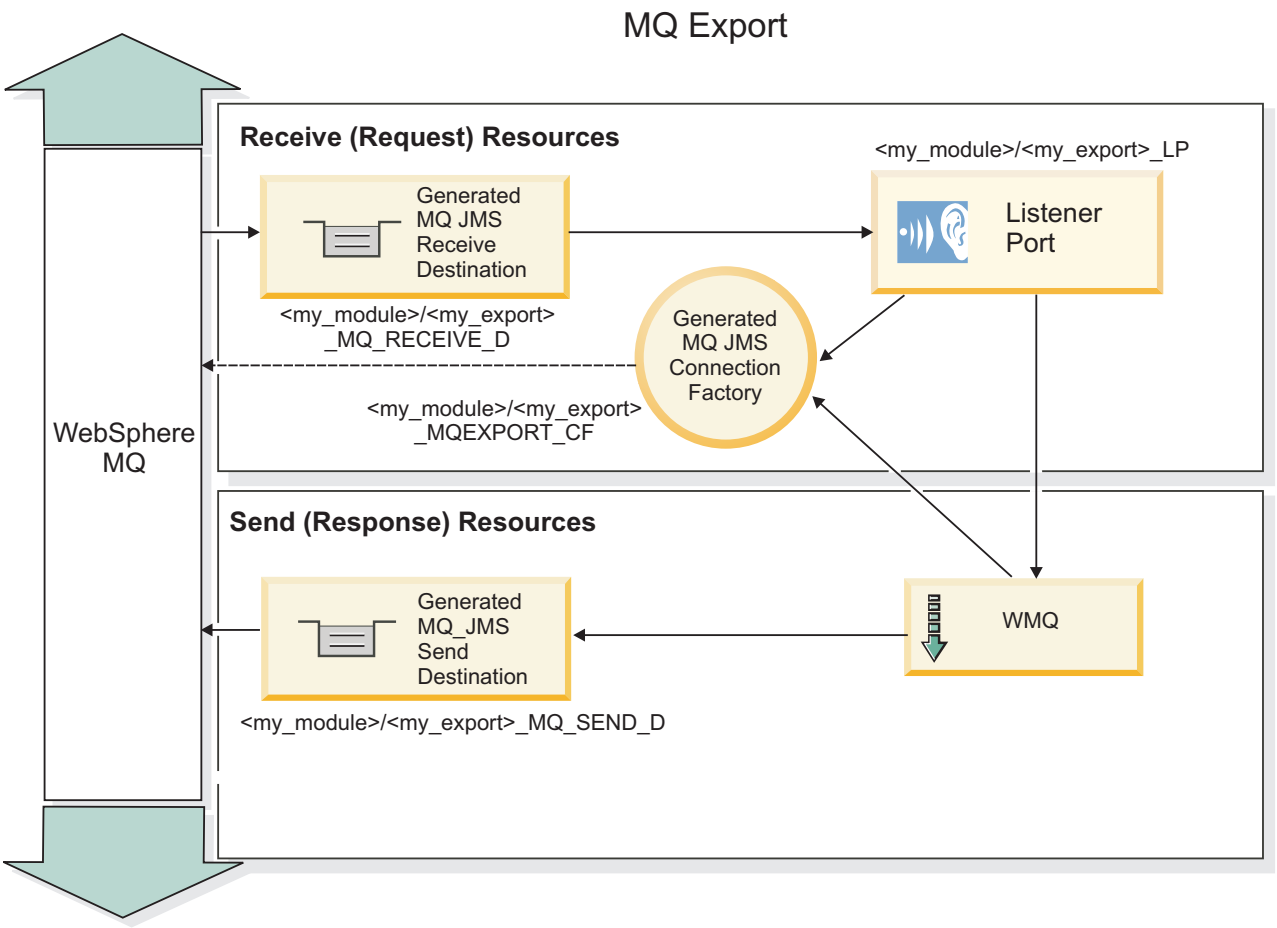


Figure 40. WebSphere MQ export binding resources

Note: Figure 39 on page 140 and Figure 40 illustrate how an application from a previous version of IBM Business Process Manager is linked to an external service. For applications developed for IBM Business Process Manager Version 7.x or later, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of a WebSphere MQ binding:

Key features of a WebSphere MQ binding include headers, Java EE artifacts, and created Java EE resources.

Correlation schemes

A WebSphere MQ request/reply application can use one of a number of techniques to correlate response messages with requests, built around the MQMD MessageID and CorrelID fields. In the vast majority of cases, the requester lets the queue manager select a MessageID and expects the responding application to copy this into the CorrelID of the response. In most cases, the requester and responding application implicitly know which correlation technique is in use. Occasionally the responding application will honor various flags in the Report field of the request that describe how to handle these fields.

Export bindings for WebSphere MQ messages can be configured with the following options:

Response MsgId options:

`New MsgID`
Allows the queue manager to select a unique MsgId for the response (default).

Copy from Request MsgID
Copies the MsgId field from the MsgId field in the request.

Copy from SCA message
Sets the MsgId to be that carried in WebSphere MQ headers in the SCA response message, or lets the queue manager define a new Id if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the MsgId. The MQRO_NEW_MSG_ID and MQRO_PASS_MSG_ID options are supported and behave like New MsgId and Copy from Request MsgID, respectively.

Response CorrelId options:

Copy from Request MsgID
Copies the CorrelId field from the MsgId field in the request (default).

Copy from Request CorrelID
Copies the CorrelId field from the CorrelId field in the request.

Copy from SCA message
Sets the CorrelId to be carried in WebSphere MQ headers in the SCA response message or leaves it blank if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the CorrelId. The MQRO_COPY_MSG_ID_TO_CORREL_ID and MQRO_PASS_CORREL_ID options are supported and behave like Copy from Request MsgID and Copy from Request CorrelID, respectively.

Import bindings for WebSphere MQ messages can be configured with the following options:

Request MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the request (default).

Copy from SCA message
Sets the MsgId to be carried in WebSphere MQ headers in the SCA request message or lets the queue manager define a new Id if the value does not exist.

Response correlation options:

Response has CorrelID copied from MsgId
Expects the response message to have a CorrelId field set, per the MsgId of the request (default).

Response has MsgID copied from MsgId
Expects the response message to have a MsgId field set, per the MsgId of the request.

Response has CorrelID copied from CorrelId
Expects the response message to have a CorrelId field set, per the CorrelId of the request.

Java EE resources

A number of Java EE resources are created when a WebSphere MQ binding is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the WebSphere MQ provider.

Response Connection Factory
Used by the SCA MQ runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations
- Send destination: where the request or outgoing message is sent (import); where the response message will be sent (export), if not superseded by the MQMD ReplyTo header field in the incoming message.
- Receive destination: where the response/request or incoming message should be placed.

WebSphere MQ headers:

WebSphere MQ headers incorporate certain conventions for conversion to the service component architecture (SCA) messages.

WebSphere MQ messages consist of a system header (the MQMD), zero or more other MQ headers (system or custom), and a message body. If multiple message headers exist in the message, the order of the headers is significant.

Each header contains information describing the structure of the following header. The MQMD describes the first header.

How MQ headers are parsed

An MQ Header data binding is used to parse MQ headers. The following headers are supported automatically:
- MQRFH
- MQRFH2
- MQCIH
- MQIIH

Headers that start with **MQH** are handled differently. Specific fields of the header are not parsed; they remain as unparsed bytes.

For other MQ headers, you can write custom MQ header data bindings to parse those headers.

How MQ headers are accessed

MQ headers can be accessed in the product in one of two ways:
- Through the service message object (SMO) in a mediation
- Through the ContextService API

MQ headers are represented internally with the SMO MQHeader element. MQHeader is a container of header data that extends MQControl but contains a value element of anyType. It contains the MQMD, MQControl (MQ message body control information), and a list of other MQ headers.
- MQMD represents the contents of the WebSphere MQ message description, except for information determining the structure and encoding of the body.
- MQControl contains information determining the structure and encoding of a message body.
- MQHeaders contain a list of MQHeader objects.

The MQ header chain is unwound so that, inside the SMO, each MQ header carries its own control information (CCSID, Encoding, and Format). Headers can be added or deleted easily, without altering other header data.

Setting fields in the MQMD

You can update the MQMD using the Context API or through the service message object (SMO) in a mediation. The following fields are automatically propagated to the outbound MQ message:

- Encoding
- CodedCharacterSet
- Format
- Report
- Expiry
- Feedback
- Priority
- Persistence
- CorrelId
- MsgFlags

Configure the MQ binding on an Import or Export to propagate the following properties to the outbound MQ message:

MsgID
Set **Request Message ID** to `copy from SCA message`.

MsgType
Clear the **Set message type to MQMT_DATAGRAM or MQMT_REQUEST for request-response operation** check box.

ReplyToQ
Clear the **Override reply to queue of request message** check box.

ReplyToQMgr
Clear the **Override reply to queue of request message** check box.

From version 7.0 onwards, context fields can be overridden using a custom property on the JNDI destination definition. Set the custom property MDCTX with value `SET_IDENTITY_CONTEXT` on the send destination to propagate the following fields to the outbound MQ message:

- UserIdentifier
- AppIdentityData

Set the custom property MDCTX with value `SET_ALL_CONTEXT` on the send destination to propagate the following properties to the outbound MQ message:

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Some fields are not propagated to the outbound MQ message. The following fields are overridden during the send of the message:

- BackoutCount
- AccountingToken

- PutDate
- PutTime
- Offset
- OriginalLength

Adding MQCIH statically in a WebSphere MQ binding:

IBM Business Process Manager supports adding MQCIH header information statically without using a mediation module.

There are various ways to add MQCIH header information to a message (for example, by using the Header Setter mediation primitive). It might be useful to add this header information statically, without the use of an additional mediation module. Static header information, including the CICS® program name, the transaction ID, and other data format header details, can be defined and added as part of the WebSphere MQ binding.

WebSphere MQ, the MQ CICS Bridge, and CICS must be configured for MQCIH header information to be added statically.

You can use Integration Designer to configure the WebSphere MQ import with the static values that are required for the MQCIH header information.

When a message arrives and is processed by the WebSphere MQ import, a check is made to see if MQCIH header information is already present in the message. If the MQCIH is present, the static values defined in the WebSphere MQ import are used to override the corresponding dynamic values in the message. If the MQCIH is not present, one is created in the message and the static values defined in the WebSphere MQ import are added.

The static values defined in the WebSphere MQ import are specific to a method. You can specify different static MQCIH values for different methods within the same WebSphere MQ import.

This facility is not used to provide default values if the MQCIH does not contain specific header information because a static value defined in the WebSphere MQ import will override a corresponding value provided in the incoming message.

External clients:

IBM Business Process Manager can send messages to, or receive messages from, external clients using WebSphere MQ bindings.

An external client (for example, a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications by way of the export binding.

WebSphere MQ imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ bindings:

You can diagnose and fix faults and failure conditions that occur with WebSphere MQ bindings.

Primary failure conditions

The primary failure conditions of WebSphere MQ bindings are determined by transactional semantics, by WebSphere MQ configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the WebSphere MQ queue manager or queue.

  A failure to connect to WebSphere MQ to receive messages will result in the MDB Listener Port failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the WebSphere MQ queue until they are successfully retrieved (or expired by WebSphere MQ).

  A failure to connect to WebSphere MQ to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected response messages.

  The import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Misusage scenarios: comparison with WebSphere MQ JMS bindings

The WebSphere MQ import and export are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations. The WebSphere MQ JMS binding, however, is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Undelivered messages

If WebSphere MQ cannot deliver a message to its intended destination (because of configuration errors, for example), it sends the messages instead to a nominated dead-letter queue.

In doing so, it adds a dead-letter header to the start of the message body. This header contains the failure reasons, the original destination, and other information.

MQ-based SCA messages not appearing in the failed event manager

If SCA messages originated because of a WebSphere MQ interaction failure, you would expect to find these messages in the failed event manager. If these messages are not showing in the failed event manager, check that the underlying WebSphere MQ destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more allows interaction with the failed event manager during SCA invocations for the WebSphere MQ bindings.

MQ failed events are replayed to the wrong queue manager

When a predefined connection factory is to be used for outbound connections, the connection properties must match those defined in the activation specification used for inbound connections.

The predefined connection factory is used to create a connection when replaying a failed event and must therefore be configured to use the same queue manager from which the message was originally received.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

Limitations of bindings:

The bindings have some limitations in their use that are listed here.

Limitations of the MQ binding:

The MQ binding has some limitations in its use that are listed here.

No publish-subscribe message distribution

The publish-subscribe method of distributing messages is not currently supported by the MQ binding though WMQ itself supports publish-subscribe. However, the MQ JMS binding does support this method of distribution.

Shared receive queues

Multiple WebSphere MQ export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each MQ import must have a different receive queue because the MQ import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each MQ export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- MQ imports and exports can point to the same send queue.

Limitations of the JMS, MQ JMS, and generic JMS bindings:

The JMS and MQ JMS bindings have some limitations.

Implications of generating default bindings

The limitations of using the JMS, MQ JMS, and generic JMS bindings are discussed in the following sections:

- Implications of generating default bindings
- Response correlation scheme
- Bidirectional support

When you generate a binding, several fields will be filled in for you as defaults, if you do not choose to enter the values yourself. For example, a connection factory name will be created for you. If you know that you will be putting your application on a server and accessing it remotely with a client, you should at binding creation time enter JNDI names rather than take the defaults since you will likely want to control these values through the administrative console at run time.

However, if you did accept the defaults and then find later that you cannot access your application from a remote client, you can use the administrative console to explicitly set the connection factory value. Locate the provider endpoints field in the connection factory settings and add a value such as <server_hostname>:7276 (if using the default port number).

Response correlation scheme

If you use the CorrelationId To CorrelationId response correlation scheme, used to correlate messages in a request-response operation, you must have a dynamic correlation ID in the message.

To create a dynamic correlation ID in a mediation module using the mediation flow editor, add a Mapping mediation primitive before the import with the JMS binding. Open the mapping editor. The known service component architecture headers will be available in the target message. Drag a field containing a unique ID in the source message onto the correlation ID in the JMS header in the target message.

Bidirectional support

Only ASCII characters are supported for Java Naming and Directory Interface (JNDI) names at runtime.

Shared receive queues

Multiple export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each import binding must have a different receive queue because the import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- Imports and exports can point to the same send queue.

Business objects

The computer software industry has developed several programming models and frameworks in which *business objects* provide a natural representation of the business data for application processing.

In general, these business objects:

- Are defined using industry standards
- Transparently map data to database tables or enterprise information systems
- Support remote invocation protocols
- Provide the data programming model foundation for application programming

Process Designer and Integration Designer provide developers with one such common business object model for representing different kinds of business entities from different domains. At development time, this model enables developers to define business objects as XML schema definitions.

At run time, the business data defined by the XML schema definitions is represented as Java Business Objects. In this model, business objects are loosely based on early drafts of the Service Data Object (SDO) specification and provide the complete set of programming model application interfaces required to manipulate business data.

Defining business objects

You define business objects using the business object editor in Integration Designer. The business object editor stores the business objects as XML schema definitions.

Using XML schema to define business objects provides several advantages:

- XML schema provide a standards-based data definition model and a foundation for interoperability between disparate heterogeneous systems and applications. XML schema are used in conjunction with the Web Services Description Language (WSDL) to provide standards-based interface contracts among components, applications, and systems.
- XML schema define a rich data definition model for representing business data. This model includes complex types, simple types, user-defined types, type inheritance, and cardinality, among other features.
- Business objects can be defined by business interfaces and data defined in the Web Services Description Language, as well as by XML schema from industry standards organizations or from other systems and applications. Integration Designer can import these business objects directly.

Integration Designer also provides support for discovering business data in databases and enterprise information systems and then generating the standards-based XML schema business object definition of that business data. Business objects generated in this fashion are often referred to as *application specific business objects* because they mimic the structure of the business data defined in the enterprise information system.

When a process is manipulating data from many different information systems, it can be valuable to transform the disparate representation of business data (for example, CustomerEIS1 and CustomerEIS2 or OrderEIS1 and OrderEIS2) into a single canonical representation (for example, Customer or Order). The canonical representation is often referred to as the *generic business object*.

Business object definitions, particularly for generic business objects, are frequently used by more than one application. To support this reuse, Integration Designer allows business objects to be created in libraries that can then be associated with multiple application modules.

The Web Services Description Language (WSDL) defines the he contracts for the services provided and consumed by a Service Component Architecture (SCA) application module, as well as the contracts used to create the components within an application module. In a contract, a WSDL can represent both the operations and business objects (which are defined by XML schema to represent the business data).

Working with business objects

Service Component Architecture (SCA) provides the framework for defining an application module, the services it provides, the services it consumes, and the composition of components that provide the business logic of the application module. Business objects play an important role in the application, defining the business data that is used to describe the service and component contracts and the business data that the components manipulate.

The following diagram depicts an SCA application module and illustrates many of the places in which the developer works with business objects.

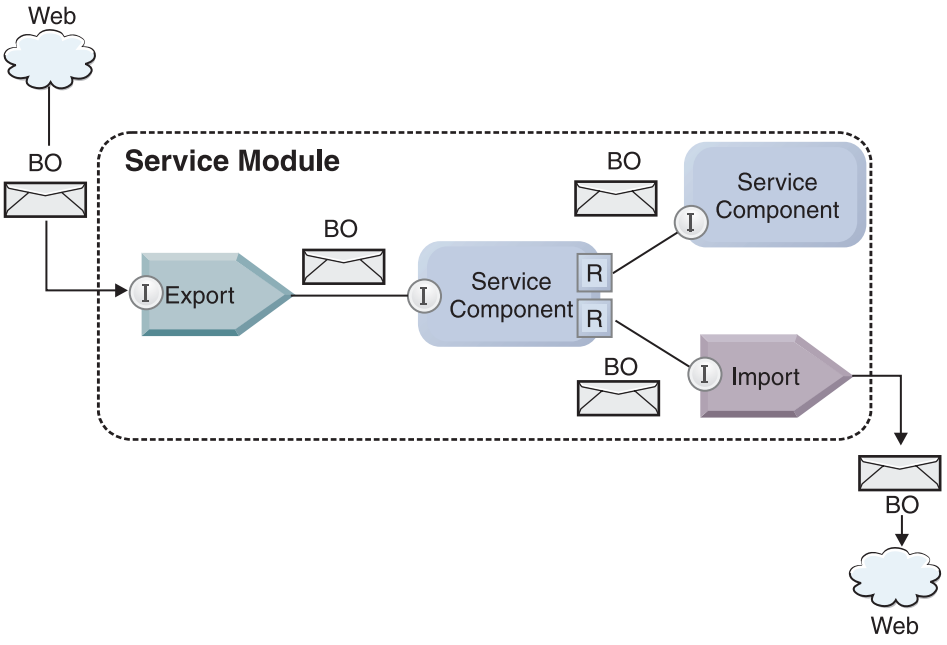


Figure 41. Business objects represent the data that flows between services in an application

Note: This topic describes how business objects are used by SCA application modules. If you are using Java interfaces, the SCA application modules can also process Java objects.

Business object programming model

The business object programming model consists of a set of Java interfaces that represent:

- The business object definition and instance data
- A set of services that support the operations on the business objects

Business object type definitions are represented by the commonj.sdo.Type and commonj.sdo.Property interfaces. The business object programming model provides a set of rules for mapping the XML schema complex type information to the Type interface and each of the elements in the complex type definition to the Property interface.

Business object instances are represented by the commonj.sdo.DataObject interface. The business object programming model is untyped, which means that the same commonj.sdo.DataObject interface can be used to represent different business object definitions, such as Customer and Order. The definition of which properties can be set and retrieved from each business object is determined by type information defined in the XML schema associated with each business object.

The business object programming model behavior is based on the Service Data Object 2.1 specification.

Business object services support various lifecycle operations (such as creation, equality, parsing, and serialization) on business objects.

For specifics on the business object programming model, see Programming using business object services and the Generated API and SPI documentation on business objects.

Bindings, data bindings, and data handlers

As shown in Figure 41 on page 150, business data that is used to invoke services provided by SCA application modules is transformed into business objects so that the SCA components can manipulate the business data. Similarly, the business objects manipulated by SCA components are converted into the data format required by the external services.

In some cases, such as the web service binding, the binding used to export and import services automatically transforms the data into the appropriate format. In other cases, such as the JMS binding, developers can provide a data binding or data handler that converts non-native formats into business objects represented by the DataObject interface.

For more information on developing data bindings and data handlers, refer to “Data handlers” on page 60 and “Data bindings” on page 61.

Components

SCA components define their provision and consumption service contracts using a combination of the Web Services Description Language and XML schema. The business data that SCA passes between components is represented as business objects using the DataObject interface. SCA verifies that these business object types are compatible with the interface contract defined by the component to be invoked.

The programming model abstractions for manipulating business objects vary from component to component. The POJO component and the mediation flow component Custom primitive provide direct manipulating of the business objects by enabling Java programming directly using the business object programming interfaces and services. Most components provide higher level abstractions for manipulating business objects, but also provide snippets of Java code for defining custom behavior in the business object interfaces and services.

Business objects can be transformed using either the combination of the Interface Flow Mediation and Business Object Map component or the mediation flow component and its XML Map primitive. These business object transformation capabilities are useful for converting application specific business objects to and from generic business objects.

Special business objects

Service message objects and business graphs are two specialized types of business objects that are used for specific application purposes.

Service message object

A service message object (SMO) is a specialized business object that is used by mediation flow components to represent the collection of data associated with a service invocation.

A SMO has a fixed top-level structure consisting of headers, context, body, and attachments (if present).

- Headers carry information related to the service invocation over a particular protocol or binding. Examples are SOAP headers and JMS headers.
- Context data carries additional logical information associated with the invocation while it is being processed by the mediation flow component. This information is typically not part of the application data sent or received by clients.
- The body of the SMO carries the payload business data, which represents the core application message or invocation data in the form of a standard business object.

The SMO can also carry attachment data for Web service invocations using SOAP with attachments.

Mediation flows perform such tasks as request routing and data transformation, and the SMO provides the combined view of header and payload contents in a single unified structure.

Business graph

A business graph is a special business object used to provide support for data synchronization in integration scenarios.

Consider an example in which two enterprise information systems have a representation of a specific order. When the order changes in one system, a message can be sent to the other system to synchronize the order data. Business graphs support the notion of sending just the portion of the order that changed to the other system and annotating it with change-summary information to define the type of change.

In this example, an Order business graph would convey to the other system that one of the line items in the order was deleted and that the projected ship date property of the order was updated.

Business graphs can easily be added to existing business objects in Integration Designer. They are most frequently found in scenarios in which WebSphere adapters are being used and to support the migration of WebSphere InterChange Server applications.

Business object parsing mode

Integration Designer provides a property on modules and libraries you can use to configure XML parsing mode for business objects to either eager or lazy.

- If the option is set to *eager*, XML byte streams are eagerly parsed to create the business object.
- If the option is set to *lazy*, the business object is created normally, but the actual parsing of the XML byte stream is deferred and partially parsed only when the business object properties are accessed.

In either XML parsing mode, non-XML data is always eagerly parsed to create the business object.

Considerations when choosing the business object parsing mode:

The business object parsing mode determines how XML data is parsed at runtime. A business object parsing mode is defined on a module or library when it is created. You can change the parsing mode of the module or library, however you should be aware of the implications.

The business object parsing mode is set at the module and library level. Modules that were created in a version of IBM Integration Designer prior to version 7 will run in the eager parsing mode without any changes required. By default, modules and libraries that are created in IBM Integration Designer version 7 and later versions will be given the most suitable parsing mode depending on a number of factors such as the parsing mode of existing projects in your workspace, or the parsing mode of dependent projects or other projects in the same solution and so on. You can change the business object parsing mode of a module or library to suit your implementation, however you should be aware of the following considerations.

Considerations

- The lazy business object parsing mode processes XML data faster; however there are compatibility differences between the eager mode and the lazy mode that you need to be aware of before changing the configuration of a module or library. These differences will affect the runtime behavior of the modules. For information on which parsing mode is optimal for your application, see "Benefits of using lazy versus eager parsing mode" in the related links.
- A module can only be configured to run in one parsing mode. Libraries can be configured to support either parsing modes or both parsing modes. A library that is configured to support both parsing modes may be referenced by both a module using the eager parsing mode and a module using the lazy parsing mode. The parsing mode of a library at run time is determined by the modules that reference the library. At runtime, a module declares its parsing mode, and that parsing mode is used by the module and any libraries that the module uses.
- Modules and libraries that are configured for different parsing modes are compatible in the following cases:
  - Modules and libraries configured with the lazy parsing mode are compatible with libraries that use either the lazy parsing mode, or both the eager and the lazy parsing modes.
  - Modules and libraries configured with the eager parsing mode are compatible with libraries that use either the eager parsing mode, or both the eager and the lazy parsing modes.
  - Libraries configured with the lazy and eager parsing modes are compatible only with libraries that use both lazy and eager parsing modes.
- Use the same parsing mode for interacting modules that communicate using the SCA binding. If modules communicate using different parsing modes, performance problems may result.

Related concepts:

"Benefits of using lazy versus eager parsing mode"
Some applications benefit from lazy XML parsing mode, while others see improved performance with eager parsing mode. It is recommended that you benchmark your application in both parsing modes to determine which mode best suites the specific characteristics of your application.

Benefits of using lazy versus eager parsing mode:

Some applications benefit from lazy XML parsing mode, while others see improved performance with eager parsing mode. It is recommended that you benchmark your application in both parsing modes to determine which mode best suites the specific characteristics of your application.

Applications that parse large XML data streams are likely to see performance improvements when the lazy XML parsing mode is used. The performance benefits increase as the size of the XML byte stream increases, and the amount of data from the byte stream that is accessed by the application decreases.

Note: The business object lazy parsing mode is supported in WebSphere Process Server version 7.0.0.3, and later versions. It is also supported in IBM Process Server. Modules and mediation modules that include mediation flow components are not supported.

The following applications are likely to perform better using an eager parsing mode:

- Applications that parse non-XML data streams
- Applications that use messages that are created using the BOFactory service
- Applications that parse very small XML messages

Related reference:

"Considerations when choosing the business object parsing mode" on page 153
The business object parsing mode determines how XML data is parsed at runtime. A business object parsing mode is defined on a module or library when it is created. You can change the parsing mode of the module or library, however you should be aware of the implications.

Application migration and development considerations:

If you are configuring an application that was originally developed using eager parsing mode to now use lazy parsing mode, or if you are planning to switch an application between lazy and eager parsing mode, be aware of the differences between modes and the considerations when switching modes.

Error handling

If the XML byte stream being parsed is ill-formed, parsing exceptions occur.

- In eager XML parsing mode, those exceptions occur as soon as the business object is parsed from the inbound XML stream.
- If lazy XML parsing mode is configured, the parsing exceptions occur latently when the business object properties are accessed and the portion of the XML that is ill-formed is parsed.

To deal with ill-formed XML, select one of the following options:

- Deploy an enterprise service bus on the edges to validate inbound XML
- Author lazy error-detection logic at the point where business object properties are accessed

Exception stacks and messages

Because the eager and lazy XML parsing modes have different underlying implementations, stack traces thrown by the business object programming interfaces and services have the same exception class name, but they might not contain the same exception message or wrapped set of implementation-specific exception classes.

XML serialization format

The lazy XML parsing mode provides a performance optimization that attempts to copy unmodified XML from the inbound byte stream to the outbound byte stream upon serialization. The result is increased performance, but the serialization format of the outbound XML byte stream might be different if the entire business object was updated in lazy XML parsing mode or if it was running in eager XML parsing mode.

Although the XML serialization format might not be precisely syntactically equivalent, the semantic value provided by the business object is equivalent independent of the parsing modes, and XML can be safely passed between applications running in different parsing modes with semantic equivalence.

Business object instance validator

The lazy XML parsing business object mode instance validator provides a higher fidelity validation of business objects, particularly facet validation of property values. Because of these improvements, the lazy parsing mode instance validator catches additional issues that are not caught in eager parsing mode and provides more detailed error messages.

Version 602 XML Maps

Mediation flows originally developed before WebSphere Integration Developer Version 6.1 might contain Mapping primitives that use a map or stylesheet that cannot execute directly in lazy XML parsing mode. When an application is migrated for use in lazy XML parsing mode, map files associated with Mapping primitives can be automatically updated by the migration wizard to run in the new mode. However, if a Mapping primitive refers directly to a stylesheet that has been edited manually, the stylesheet is not migrated and cannot execute in lazy XML parsing mode.

Private unpublished APIs

If an application is taking advantage of unpublished, private, implementation-specific business object programming interfaces, the application is likely to fail compilation when the parsing mode is switched. In eager parsing mode, these private interfaces are typically business object implementation classes defined by the Eclipse Modeling Framework (EMF).

In all cases, it is strongly recommend that private APIs be removed from the application.

Service Message Object EMF APIs

A mediation component in IBM Integration Designer provides the ability to manipulate message content using the Java classes and interfaces provided in the com.ibm.websphere.sibx.smobo package. In lazy XML parsing mode, the Java interfaces in the com.ibm.websphere.sibx.smobo package can still be used, but methods that refer directly to Eclipse Modeling Framework (EMF) classes and interfaces or that are inherited from EMF interfaces are likely to fail.

The ServiceMessageObject and its contents cannot be cast to EMF objects in lazy XML parsing mode.

BOMode service

The BOMode service is used to determine whether the currently executing XML parsing mode is eager or lazy.

Migration

All applications before version 7.0.0.0 are running in eager XML parsing mode. When they are runtime migrated using the BPM runtime migration tools, they continue to run in eager XML parsing mode.

To enable an application earlier than version 7.0.0.0 to be configured to use the lazy XML parsing mode, you first use Integration Designer to migrate the artifacts of the application. After migration, you then configure the application to use lazy XML parsing.

See Migrating source artifacts for information on migrating artifacts in Integration Designer, and see Configuring the business object parsing mode of modules and libraries for information on setting the parsing mode.

Relationships

A relationship is an association between two or more data entities, typically business objects. In IBM Business Process Manager Advanced, relationships can be used to transform data that is equivalent across business objects and other data but that is represented differently, or they can be used to draw associations across different objects found in different applications. They can be shared across applications, across solutions, and even across products.

The relationship service in IBM Business Process Manager Advanced provides the infrastructure and operations for managing relationships. Because it enables you to deal with business objects regardless of where they reside, it can provide a unified holistic view across all applications in an enterprise, and serve as a building block for BPM solutions. Because relationships are extensible and manageable, they can be used in complex integration solutions.

What are relationships?

A relationship is an association between business objects. Each business object in a relationship is called a *participant* in the relationship. Each participant in the relationship is distinguished from other participants based on the function, or *role*, it serves in that relationship. A relationship contains a list of roles.

The relationship *definition* describes each role and specifies how the roles are related. It also describes the overall "shape" of the relationship. For example, this role can have only one participant, but this other role can have as many participants as necessary. You might define a *car-owner* relationship, for instance, where one owner might own multiple cars. For example, one instance could have the following participants for each of these roles:

- Car (Ferrari)
- Owner (John)

The relationship definition is a template for the relationship *instance*. The instance is the run-time instantiation of the relationship. In the *car-owner* example, an instance might describe any of the following associations:

- John owns Ferrari
- Sara owns Mazda
- Bob owns Ferrari

Using relationships frees you from the need to custom build relationship tracking persistence within your business logic. For certain scenarios, the relationship service does all the work for you. See the example described in the section on Identity relationships.

Scenarios

Here is a typical example of a situation in which an integration solution might use relationships. A large corporation buys multiple companies, or business units. Each business unit uses different software to monitor personnel and laptops. The company needs a way to monitor its employees and their laptops. It wants a solution that enables them to:

- View all the employees in the various business units as if they were in one database
- Have a single view of all their laptops
- Allow employees to log on to the system and buy a laptop
- Accommodate the different enterprise application systems in the various business units

To accomplish this, the company needs a way to ensure, for example, that John Smith and John A. Smith in different applications are seen as the same employee. For Example, they need a way to consolidate a single entity across multiple application spaces.

More complex relationship scenarios involve building BPEL processes that draw relationships across different objects found in multiple applications. With complex relationship scenarios, the business objects reside in the integration solution, and not in the applications. The relationship service provides a platform for managing relationships persistently. Before the relationship service, you would have to build your own object persistence service. Two examples of complex relationship scenarios are:

- You have a **car** business object with a VIN number in an SAP application, and you want to track the fact that this car is owned by someone else. However, the ownership relationship is with someone in a PeopleSoft application. In this pattern of relationships, you have two solutions and you need to build a cross-bridge between them.
- A large retail company wants to be able to monitor merchandise returned for cash back or credit. There are two different applications involved: an order management system (OMS) for purchases, and a returns management system (RMS) for returns. The business objects reside in more than one application, and you need a way to show the relationships that exist between them.

Common usage patterns

The most common relationship patterns are *equivalence* patterns. These are based on cross-referencing, or correlation. There are two types of relationships that fit this pattern: *non-identity* and *identity*.

- **Non-identity relationships** establish associations between business objects or other data on a one-to-many or many-to-many basis. For each relationship instance, there can be one or more instances of each participant. One type of non-identity relationship is a static lookup relationship. An example of this is a relationship in which **CA** in an SAP application is related to **California** in a Siebel application.
- **Identity relationships** establish associations between business objects or other data on a one-to-one basis. For each relationship instance, there can be only one instance of each participant. Identity relationships capture cross-references between business objects that are semantically equivalent, but that are identified differently within different applications. Each participant in the relationship is associated with a business object that has a value (or combination of values) that uniquely identifies the object. Identity relationships typically transform the key attributes of business objects, such as ID numbers and product codes.

  For example, if you have **car** business objects in SAP, PeopleSoft, and Siebel applications, and you want to build a solution that synchronizes them, you would normally need to introduce hand-built relationship synchronization logic in six maps:

  SAP -> generic

  generic -> SAP

  PeopleSoft-> generic

  generic-> PeopleSoft

  Siebel-> generic

  generic-> Siebel

  However, if you use relationships in your solution, the relationship service provides prebuilt pattern implementations that maintains all these relationship instances for you.

Tools for working with relationships

The *relationship editor* in Integration Designer is the tool you use to model and design business integration relationships and roles. For detailed background and task information about creating relationships and using the relationship editor, see Creating relationships.

The *relationship service* is an infrastructure service in IBM Business Process Manager that maintains relationships and roles in the system and provides operations for relationship and role management.

The *relationship manager* is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

Relationships can be invoked programmatically through the relationship service APIs.

Relationship service

The relationship service stores relationship data in relationship tables, where it keeps track of application-specific values across applications and across solutions. The relationship service provides operations for relationship and role management.

How relationships work

Relationships and roles are defined using the graphical interface of the relationship editor tool in Integration Designer. The relationship service stores the correlation data in tables in the relationship database in the default data source that you specify when you configure the relationship service. A separate table (sometimes called a participant table) stores information for each participant in the relationship. The relationship service uses these relationship tables to keep track of the related application-specific values and propagate updated information across all the solutions.

Relationships, which are business artifacts, are deployed within a project or in a shared library. At the first deployment, the relationship service populates the data.

At run time, when maps or other IBM Business Process Manager components need a relationship instance, the instances of the relationship are either updated or retrieved, depending on the scenario.

Relationship and role instance data can be manipulated through three means:

- IBM Business Process Manager component Java snippet invocations of the relationship service APIs
- Relationship transformations in the IBM Business Process Manager business object mapping service
- The relationship manager tool

For detailed background and task information on creating relationships, identifying relationship types, and using the relationship editor, see Creating relationships topic.

Relationship manager

The relationship manager is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

The relationship manager provides a graphical user interface for creating and manipulating relationship and role data at run time. You can manage relationship entities at all levels: relationship instance, role instance, and attribute data and property data levels. With the relationship manager, you can:

- View a list of the relationships in the system and detailed information for individual relationships
- Manage relationship instances:
  - Query relationship data to view subsets of instance data
  - Query relationship data to view subsets of instance data using database views
  - View a list of relationship instances that match a relationship query and detailed information about an instance
  - Edit the property values for a relationship instance
  - Create and delete relationship instances
- Manage roles and role instances:
  - View details about a role or a role instance
  - Edit role instance properties
  - Create and delete role instances for a relationship
  - Roll back relationship instance data to a point in time when you know the data is reliable
- Import data from an existing static relationship into your system, or export data from an existing static relationship to an RI or CSV file

- Remove relationship schema and data from the repository when the application that uses it is uninstalled

Relationships in Network Deployment environments

Relationships can be used in Network Deployment (ND) environments without any extra configuration.

In Network Deployment (ND) environments, relationships are installed in an application cluster. Relationships are then visible within the cluster, and all servers in the cluster have access to the instance data stored in the relationship database. The ability to run the relationship service in an ND environment makes it scalable and highly available.

The relationship manager allows relationships to be managed across different clusters through a centralized administrative interface. You connect the relationship manager to a server in a cluster by selecting its relationship MBean.

Relationship service APIs

Relationships can be invoked programmatically through the relationship service APIs, within or outside of business object maps.

Three API types are available:

- Relationship instance manipulation APIs (including create, update, delete instance data directly)
- Relationship pattern support APIs (including correlate(), correlateforeignKeyLookup)
- Relationship lookup patterns (lookup APIs)

The enterprise service bus in IBM Business Process Manager

IBM Business Process Manager supports the integration of application services, including the same capabilities as WebSphere Enterprise Service Bus.

Connecting services through an enterprise service bus

With an enterprise service bus (ESB), you can maximize the flexibility of an SOA. Participants in a service interaction are connected to the ESB, rather than directly to one another.

When the service requester connects to the ESB, the ESB takes responsibility for delivering its requests, using messages, to a service provider offering the required function and quality of service. The ESB facilitates requester-provider interactions and addresses mismatched protocols, interaction patterns, or service capabilities. An ESB can also enable or enhance monitoring and management. The ESB provides virtualization and management features that implement and extend the core capabilities of SOA.

The ESB abstracts the following features:

Location and identity
Participants do not have to know the location or identity of other participants. For example, requesters do not have to be aware that a request could be serviced by any of several providers; service providers can be added or removed without disruption.

Interaction protocol
Participants do not have to share the same communication protocol or interaction style. For example, a request expressed as SOAP over HTTP can be serviced by a provider that only understands SOAP over Java Message Service (JMS).

Interface
Requesters and providers do not have to agree on a common interface. An ESB reconciles differences by transforming request and response messages into a form expected by the provider.

Qualities of (interaction) service
Participants, or systems administrators, declare their quality-of-service requirements, including

authorization of requests, encryption and decryption of message contents, automatic auditing of service interactions, and how their requests should be routed (for example, optimizing for speed or cost).

Interposing the ESB between participants enables you to modulate their interaction through a logical construct called a *mediation*. Mediations operate on messages in-flight between requesters and providers. For example, mediations can be used to find services with specific characteristics that a requester is asking for, or to resolve interface differences between requesters and providers. For complex interactions, mediations can be chained sequentially.

Using mediations, an enterprise service bus carries out the following actions between requester and service:

- *Routing* messages between services. An enterprise service bus offers a common communication infrastructure that can be used to connect services, and thereby the business functions they represent, without the need for programmers to write and maintain complex connectivity logic.
- *Converting* transport protocols between requester and service. An enterprise service bus provides a consistent, standards-based way to integrate business functions that use different IT standards. This enables integration of business functions that could not normally communicate, such as to connect applications in departmental silos or to enable applications in different companies to participate in service interactions.
- *Transforming* message formats between requester and service. An enterprise service bus enables business functions to exchange information in different formats, with the bus ensuring that the information delivered to a business function is in the format required by that application.
- *Handling* business events from disparate sources. An enterprise service bus supports event-based interactions in addition to the message exchanges to handle service requests.

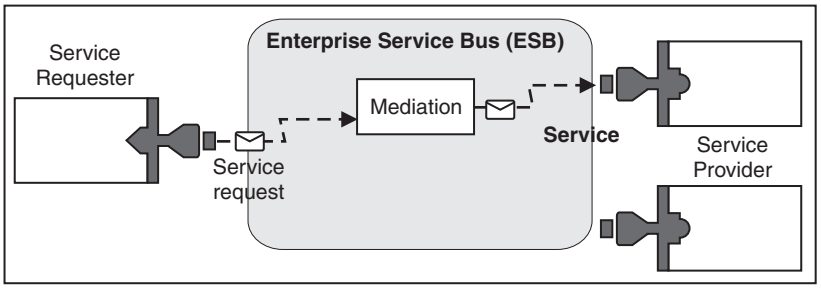


Figure 42. An enterprise service bus. The enterprise service bus is routing messages between applications, which are requesters or providers of services. The bus is converting transport protocols and transforming message formats between requesters and providers. In this figure, each application uses a different protocol (represented by the different geometric shapes of their connectors) and uses different message formats.

By using the enterprise service bus you can focus on your core business rather than your computer systems. You can change or add to the services if required; for example, to respond to changes in the business requirement, to add extra service capacity, or to add new capabilities. You can make the required changes by reconfiguring the bus, with little or no impact to existing services and applications that use the bus.

Enterprise service bus messaging infrastructure

IBM Business Process Manager includes enterprise service bus capabilities. IBM Business Process Manager supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based, messaging infrastructure in an integrated enterprise service bus.

The enterprise service capabilities that you can use for your enterprise applications provide not only a transport layer but mediation support to facilitate service interactions. The enterprise service bus is built

around open standards and service-oriented architecture (SOA). It is based on the robust Java EE infrastructure and associated platform services provided by IBM WebSphere Application Server Network Deployment.

IBM Business Process Manager is powered by the same technology available with IBM WebSphere Enterprise Service Bus. This capability is part of the underlying functionality of IBM Business Process Manager, and no additional license for WebSphere Enterprise Service Bus is required to take advantage of these capabilities.

However, you can deploy additional stand-alone licenses of WebSphere Enterprise Service Bus around your enterprise to extend the connectivity reach of the process integration solutions powered by IBM Business Process Manager. For example, WebSphere Enterprise Service Bus can be installed closer to an SAP application to host an IBM WebSphere Adapter for SAP and to transform SAP messages before sending that information across the network to a business process choreographed by IBM Business Process Manager.

Messaging or queue destination hosts:

A messaging or queue destination host provides the messaging function within a server. A server becomes the messaging destination host when you configure it as the messaging target.

A messaging engine runs within a server. The messaging engine provides messaging functions and a connection point for applications to connect to the bus. Service Component Architecture (SCA) asynchronous communication, JMS imports and exports, asynchronous internal processing use message queues on the messaging engine.

The deployment environment connects the message source to the message target through the bus when the application modules are deployed. Knowing the message source and message target helps you determine what type of deployment environment you need.

Applications can store persistent data in a data store, which is a set of tables in a database or schema, or in a file store. The messaging engine uses an instance of a JDBC data source to interact with that database.

Configure the messaging destination host when you define your deployment environment by using **Server** from the administrative console or designate the server as the destination host during software installation.

Data stores:

Every messaging engine can use a data store, which is a set of tables in a database or schema that store persistent data.

All of the tables in the data store are held in the same database schema. You can create each data store in a separate database. Alternatively, you can create multiple data stores in the same database, with each data store using a different schema.

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

JDBC providers:

You can use JDBC providers to interact applications with relational databases.

Applications use JDBC providers to interact with relational databases. The JDBC provider supplies the specific JDBC driver implementation class for access to a specific type of database. To create a pool of

connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the Java EE Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

Refer to the examples of both Typical stand-alone environment setup and Typical deployment environment setup in the previous topic.

For more information on JDBC providers, see “JDBC providers” in the WebSphere Application Server information center.

Service integration buses for IBM Business Process Manager:

A service integration bus is a managed communication mechanism that supports service integration through synchronous and asynchronous messaging. A bus consists of interconnecting messaging engines that manage bus resources. It is one of the WebSphere Application Server technologies on which IBM Business Process Manager is based.

Some buses are automatically created for use by the system, the Service Component Architecture (SCA) applications that you deploy, and by other components. You can also create buses to support service integration logic or other applications, for example, to support applications that act as service requesters and providers within IBM Business Process Manager, or to link to WebSphere MQ.

A bus destination is a logical address to which applications can attach as a producer, consumer, or both. A queue destination is a bus destination that is used for point-to-point messaging.

Each bus can have one or more bus members, each of which is either a server or a cluster.

The *bus topology* is the physical arrangement of application servers, messaging engines, and WebSphere MQ queue managers, and the pattern of bus connections and links between them, that makes up your enterprise service bus.

Some service integration buses are created automatically to support IBM Business Process Manager. Up to six buses are created when you create your deployment environment or configure a server or cluster to support SCA applications. These buses each have five authentication aliases that you must configure.

SCA system bus:

The *SCA system bus* is a service integration bus that is used to host queue destinations for Service Component Architecture (SCA) modules. The SCA run time, which supports mediation modules, uses queue destinations on the system bus as an infrastructure to support asynchronous interactions between components and modules.

The system bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The system bus provides a scope within which resources, such as queue destinations, are configured for mediation modules and interaction endpoints. The bus enables message routing between endpoints. You can specify the quality of service for the bus, including priority and reliability.

The bus name is SCA.SYSTEM.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

SCA application bus:

The application bus destinations support the asynchronous communication of WebSphere Business Integration Adapters and other System Component Architecture components.

The application bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The application bus is similar to service integration buses you might create to support service integration logic or other applications.

The bus name is SCA.APPLICATION.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

The Common Event Infrastructure bus:

The Common Event Infrastructure bus is used for transmitting common base events, asynchronously, to the configured Common Event Infrastructure server.

The bus name is CommonEventInfrastructure_Bus. The authentication alias used for securing this bus is CommonEventInfrastructureJMSAuthAlias

The Business Process Choreographer bus:

Use the Business Process Choreographer bus name and authentication for internal message transmission.

The Business Process Choreographer bus is used for transmitting messages internally and for business flow manager's Java Messaging Service (JMS) API.

The bus name is BPC.cellName.Bus. The authentication alias is BPC_Auth_Alias

Performance Data Warehouse bus:

The Performance Data Warehouse bus is used for transmitting messages internally by the infrastructure and for communicating with IBM Business Process Manager clients.

The Performance Data Warehouse bus is automatically created when you create a deployment environment.

The bus name is PERFDW.busID.Bus. The authentication alias used for securing this bus is PERFDWME_Auth_Alias.

Process Server bus:

The Process Server bus is used for transmitting messages internally by the infrastructure and for communicating with IBM Business Process Manager clients.

The Process Server bus is automatically created when you create a deployment environment.

The bus name is PROCSVR.busID.Bus. The authentication alias used for securing this bus is PROCSVRME_Auth_Alias.

Service applications and service modules

A service module is a Service Component Architecture (SCA) module that provides services in the run time. When you deploy a service module to IBM Business Process Manager, you build an associated service application that is packaged as an enterprise archive (EAR) file.

Service modules are the basic units of deployment and can contain components, libraries, and staging modules used by the associated service application. Service modules have exports and, optionally, imports to define the relationships between modules and service requesters and providers. WebSphere Process Server supports modules for business services and mediation modules. Both modules and mediation modules are types of SCA modules. 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.

Deploying a service application

The process of deploying an EAR file containing a service application is the same as the process of deploying any EAR file. You can modify values for mediation parameters at deployment time. After you have deployed an EAR file containing an SCA module, you can view details about the service application and its associated module. You can see how a service module is connected to service requesters (through exports) and service providers (through imports).

Viewing SCA module details

The service module details that you can view depend upon the SCA module. They include the following attributes.

- SCA module name
- SCA module description
- Associated application name
- SCA module version information, if the module is versioned
- SCA module imports:
  - Import interfaces are abstract definitions that describe how an SCA module accesses a service.
  - Import bindings are concrete definitions that specify the physical mechanism by which an SCA module accesses a service. For example, using SOAP/HTTP.
- SCA module exports:
  - Export interfaces are abstract definitions that describe how service requesters access an SCA module.
  - Export bindings are concrete definitions that specify the physical mechanism by which a service requester accesses an SCA module, and indirectly, a service.
- SCA module properties

Imports and import bindings:

Imports define interactions between SCA modules and service providers. SCA modules use imports to permit components to access external services (services that are outside the SCA module) using a local representation. Import bindings define the specific way that an external service is accessed.

If SCA modules do not need to access external services, they are not required to have imports. Mediation modules usually have one or more imports that are used to pass messages or requests on to their intended targets.

Interfaces and bindings

An SCA module import needs at least one interface, and an SCA module import has a single binding.

- Import interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many import interfaces.
- Import bindings are concrete definitions that specify the physical mechanism that SCA modules use to access an external service.

Supported import bindings

IBM Business Process Manager supports the following import bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit components to invoke Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

  You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.
- HTTP bindings permit you to access applications using the HTTP protocol.
- Enterprise JavaBeans (EJB) import bindings enable SCA components to invoke services provided by Java EE business logic running on a Java EE server.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.
- WebSphere MQ bindings permit interoperability with WebSphere MQ. You can use WebSphere MQ bindings only with remote queue managers by way of a WebSphere MQ client connection; you cannot use them with local queue managers. Use WebSphere MQ bindings if you want to communicate with native WebSphere MQ applications.

Dynamic invocation of services

Services can be invoked through any supported import binding. A service is normally found at an endpoint specified in the import. This endpoint is called a static endpoint. It is possible to invoke a different service by overriding the static endpoint. Dynamic override of static endpoints lets you invoke a service at another endpoint, through any supported import binding. Dynamic invocation of services also permits you to invoke a service where the supported import binding does not have a static endpoint.

An import with an associated binding is used to specify the protocol and its configuration for dynamic invocation. The import used for the dynamic invocation can be wired to the calling component, or it can be dynamically selected at runtime.

For Web service and SCA invocations, it is also possible to make a dynamic invocation without an import, with the protocol and configuration deduced from the endpoint URL. The invocation target type is identified from the endpoint URL. If an import is used, the URL must be compatible with the protocol of the import binding.

- An SCA URL indicates invocation of another SCA module.
- An HTTP or a JMS URL by default indicates invocation of a Web service; for these URLs, it is possible to provide an additional binding type value that indicates that the URL represents an invocation by way of an HTTP or JMS binding.
- For a Web service HTTP URL, the default is to use SOAP 1.1, and a binding type value can be specified that indicates the use of SOAP 1.2.

Exports and export bindings:

Exports define interactions between SCA modules and service requesters. SCA modules use exports to offer services to others. Export bindings define the specific way that an SCA module is accessed by service requesters.

Interfaces and bindings

An SCA module export needs at least one interface.

- Export interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many export interfaces.
- Export bindings are concrete definitions that specify the physical mechanism that service requesters use to access a service. Usually, an SCA module export has one binding specified. An export with no binding specified is interpreted by the run time as an export with an SCA binding.

Supported export bindings

IBM Business Process Manager supports the following export bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit exports to be invoked as Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

  You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.
- HTTP bindings permit exports to be accessed using the HTTP protocol.
- Enterprise JavaBeans (EJB) export bindings allow SCA components to be exposed as EJBs so that Java EE business logic can invoke SCA components otherwise unavailable to them.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.
- WebSphere MQ bindings permit interoperability with WebSphere MQ. You use a remote (or client) connection to connect to an MQ queue manager on a remote machine. A local (or bindings) connection is a direct connection to WebSphere MQ. This can be used only for a connection to an MQ queue manager on the same machine. WebSphere MQ will permit both types of connection, but MQ bindings only support the "remote" (or "client") connection.

Mediation modules:

Mediation modules are Service Component Architecture (SCA) modules that can change the format, content, or target of service requests.

Mediation modules operate on messages that are in-flight between service requesters and service providers. You can route messages to different service providers and to amend message content or form. Mediation modules can provide functions such as message logging, and error processing that is tailored to your requirements.

You can change certain aspects of mediation modules, from the administrative console, without having to redeploy the module.

Components of mediation modules

Mediation modules contain the following items:

- Imports, which define interactions between SCA modules and service providers. They allow SCA modules to call external services as if they were local. You can view mediation module imports and modify the binding.
- Exports, which define interactions between SCA modules and service requesters. They allow an SCA module to offer a service and define the external interfaces (access points) of an SCA module. You can view mediation module exports.
- SCA components, which are building blocks for SCA modules such as mediation modules. You can create and customize SCA modules and components graphically, using Integration Designer. After you deploy a mediation module you can customize certain aspects of it from the administrative console, without having to redeploy the module.

  Usually, mediation modules contain a specific type of SCA component called a *mediation flow component*. Mediation flow components define mediation flows.

  A mediation flow component can contain none, one, or a number of mediation primitives. IBM Business Process Manager supports a supplied set of mediation primitives that provide functionality for message routing and transformation. For additional mediation primitive flexibility, use the Custom Mediation primitive to call custom logic.

  The purpose of a mediation module that does not contain a mediation flow component is to transform service requests from one protocol to another. For example, a service request might be made using SOAP/JMS but might need transforming to SOAP/HTTP before sending on.

  Note: You can view and make certain changes to mediation modules from IBM Business Process Manager. However, you cannot view or change the SCA components inside a module from IBM Business Process Manager. Use Integration Designer to customize SCA components.

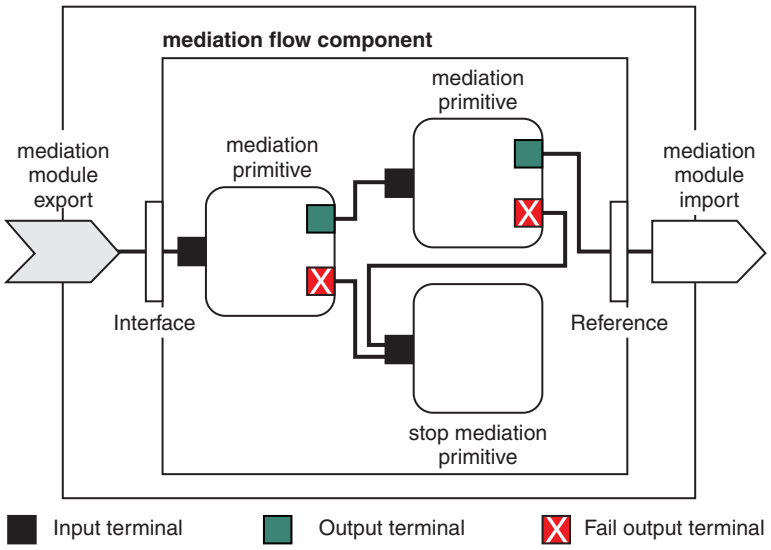


Figure 43. Simplified example of a mediation module. The mediation module contains one mediation flow component, which contains mediation primitives.

- Properties

  Mediation primitives have properties, some of which can be displayed in the administrative console as additional properties of an SCA module.

  For mediation primitive properties to be visible from the IBM Business Process Manager administrative console, the integration developer must promote the properties. Certain properties lend themselves to being administratively configured and Integration Designer describes these as promotable properties, because they can be promoted from the integration cycle to the administrative cycle. Other properties are not suitable for administrative configuration, because modifying them can affect the mediation flow in such a way that the mediation module needs to be redeployed. Integration Designer lists the properties that you can choose to promote under the promoted properties of a mediation primitive.

  You can use the IBM Business Process Manager administrative console to change the value of promoted properties without having to redeploy a mediation module, or restart the server or module.

  Generally, mediation flows use property changes immediately. However, if property changes occur in a deployment manager cell, they take effect on each node as that node is synchronized. Also, mediation flows that are in-flight continue to use previous values.

  Note: From the administrative console, you can only change property values, not property groups, names or types. If you want to change property groups, names or types, you must use Integration Designer.
- A mediation module or dependent library may also define subflows. A subflow encapsulates a set of mediation primitives wired together as a reusable piece of integration logic. A primitive can be added to a mediation flow to invoke a subflow.

Deploying mediation modules

Mediation modules are created using Integration Designer, and are generally deployed to IBM Business Process Manager inside an enterprise archive (EAR) file.

You can change the value of promoted properties at deployment time.

You can export a mediation module from Integration Designer, and cause Integration Designer to package the mediation module inside a Java archive (JAR) file, and the JAR file inside an EAR file. You can then deploy the EAR file, by installing a new application from the administrative console.

Mediation modules can be thought of as one entity. However, SCA modules are defined by a number of XML files stored in a JAR file.

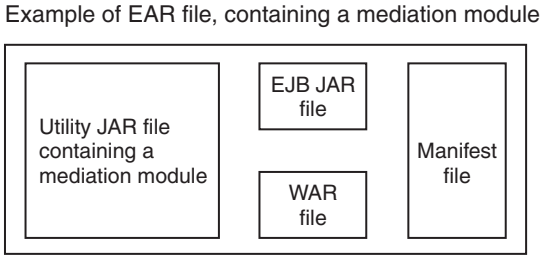


Figure 44. Simplified example of an EAR file containing a mediation module. The EAR file contains JARs. The utility JAR file contains a mediation module.

Mediation primitives:

Mediation flow components operate on message flows between service components. The capabilities of a mediation component are implemented by *mediation primitives*, which implement standard service implementation types.

A mediation flow component has one or more flows. For example, one for request and one for reply.

IBM Business Process Manager supports a supplied set of mediation primitives, which implement standard mediation capabilities for mediation modules or modules deployed into IBM Business Process Manager. If you need special mediation capabilities, you can develop your own custom mediation primitives.

A mediation primitive defines an "in" operation that processes or handles messages that are represented by service message objects (SMOs). A mediation primitive can also define "out" operations that send messages to another component or module.

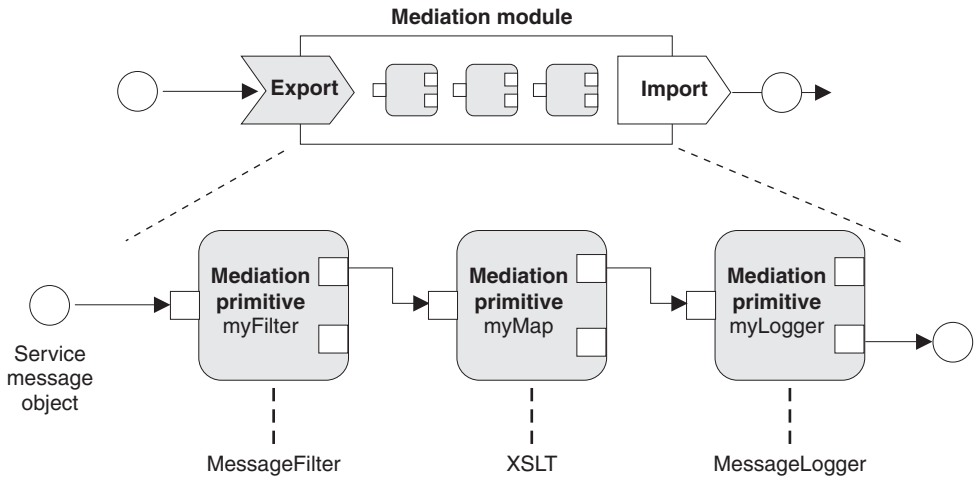


Figure 45. Mediation module containing three mediation primitives

You can use Integration Designer to configure mediation primitives and set their properties. Some of these properties can be made visible to the runtime administrator by promoting them. Any mediation primitive property that can be promoted can also be a dynamic property. A dynamic property can be overridden, at run time, using a policy file.

Integration Designer also allows you to graphically model and assemble mediation flow components from mediation primitives, and assemble mediation modules or modules from mediation flow components. The administrative console refers to mediation modules and modules as SCA modules.

Integration Designer also allows the definition of subflows in modules or their dependent libraries. A subflow can contain any mediation primitive except for the Policy Resolution mediation primitive. A subflow is invoked from a request or response flow, or from another subflow using the Subflow mediation primitive. Properties promoted from mediation primitives in a subflow are exposed as properties on the Subflow mediation primitives. These may then be promoted again until they reach the module level at which point they can then be modified by the runtime administrator.

Supported mediation primitives

The following set of mediation primitives are supported by IBM Business Process Manager:

Business Object Map

Transforms messages.

- Defines message transformations using a business object map, which can be reused.
- Allows you to define message transformations graphically, using the business object map editor.
- Can alter the content of a message.
- Can transform an input message type to a different output message type.

Custom Mediation

Allows you to implement your own mediation logic in Java code. The Custom Mediation primitive combines the flexibility of a user-defined mediation primitive, with the simplicity of a pre-defined mediation primitive. You can create complex transformations and routing patterns by:

- Creating Java code.
- Creating your own properties.
- Adding new terminals.

You can call a service from a Custom Mediation primitive, but the Service Invoke mediation primitive is designed to call services and provides additional functionality, such as retry.

Data Handler

Allows you to transform a part of a message. It is used to convert an element of a message from a physical format to a logical structure or a logical structure to a physical format. The primary usage of the primitive is to convert a physical format, such as a Text string within a JMS Text Message object, into a logical Business Object structure and back again. This mediation is commonly used to:

- Transform a section of the input message from a defined structure to another - an example of this would be where the SMO includes a string value that is comma delimited and you want to parse this into a specific Business Object.
- Alter the message type – an example would be when a JMS export has been configured to use a JMS basic typed data binding and within the mediation module the integration developer decides that the content should be inflated to a specific BO structure.

Database Lookup

Modifies messages, using information from a user-supplied database.

- You must set up a database, data source, and any server authentication settings for the Database Lookup mediation primitive to use. Use the administrative console to help you do this.
- The Database Lookup mediation primitive can read from only one table.
- The specified key column must contain a unique value.

- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Endpoint Lookup

Allows for the dynamic routing of requests, by searching for service endpoints in a repository.

- Service endpoint information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- IBM Business Process Manager needs to know which registry to use, therefore, you must create WSRR access definitions using the IBM Business Process Manager administrative console.

Event Emitter

Enhances monitoring by letting you send events from inside a mediation flow component.

- You can suspend the mediate action by deselecting the check box.
- You can view Event Emitter events using the Common Base Events browser on IBM Business Process Manager.
- You should only send events at a significant point in a mediation flow, for performance reasons.
- You can define the parts of the message that the event contains.
- The events are sent in the form of Common Base Events and are sent to a Common Event Infrastructure server.
- To fully use the Event Emitter information, event consumers need to understand the structure of the Common Base Events. The Common Base Events has an overall schema, but this does not model the application specific data, which is contained in the extended data elements. To model the extended data elements, the Integration Designer tools generate a Common Event Infrastructure event catalog definition file for each of the configured Event Emitter mediation primitives. Event catalog definition files are export artifacts that are provided to help you; they are not used by Integration Designer or by the IBM Business Process Manager runtime. You should refer to the event catalog definition files when you create applications to consume Event Emitter events.
- You can specify other monitoring from IBM Business Process Manager. For example, you can monitor events to be emitted from imports and exports.

Fail Stops a particular path in the flow, and generates an exception.

Fan In Helps aggregate (combine) messages.

- Can only be used in combination with the Fan Out mediation primitive.
- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- The Fan In mediation primitive receives messages until a decision point is reached, then one message is output.
- The shared context should be used to hold aggregation data.

Fan Out

Helps split and aggregate (combine) messages.

- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- In iterate mode, the Fan Out mediation primitive lets you iterate through a single input message that contains a repeating element. For each occurrence of the repeating element, a message is sent.
- The shared context should be used to hold aggregation data.

HTTP Header Setter

Provides a mechanism for managing headers in HTTP messages.

- Can create, set, copy, or delete HTTP message headers.
- Can set multiple actions to change multiple HTTP headers.

Mapping

Transforms messages.

- Allows you to perform Extensible Stylesheet Language (XSL) transformations or Business Object Map transformations.
- You transform messages using an XSLT 1.0 or an XSLT 2.0 transformation or a Business Object Map transformation. The XSL transformations operate on an XML serialization of the message, whereas the Business Object Map transformation operates on the Service Data Objects (SDO).

Message Element Setter

Provides a simple mechanism for setting the content of messages.

- Can change, add or delete message elements.
- Does not change the type of the message.
- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Message Filter

Routes messages down different paths, based on the message content.

- You can suspend the mediate action by deselecting the check box.

Message Logger

Logs messages in a relational database or through your own custom logger. The messages are stored as XML, therefore, data can be post-processed by XML-aware applications.

- You can suspend the mediate action by deselecting the check box.
- The rational database schema (table structure) is defined by IBM.
- By default, the Message Logger mediation primitive uses the Common database. The runtime maps the data source at **jdbc/mediation/messageLog** to the Common database.
- You can set Handler implementation classes to customize the behavior of the custom logger. Optionally, you can provide Formatter implementation classes, Filter implementation classes, or both to customize the behavior of the custom logger.

MQ Header Setter

Provides a mechanism for managing headers in MQ messages.

- Can create, set, copy, or delete MQ message headers.
- Can set multiple actions to change multiple MQ headers.

Policy Resolution

Allows for the dynamic configuration of requests, by searching for service endpoints, and associated policy files, in a repository.

- You can use a policy file to dynamically override the promoted properties of other mediation primitives.
- Service endpoint information and policy information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- IBM Business Process Manager needs to know which registry to use, therefore, you must create WSRR access definitions using the IBM Business Process Manager administrative console.

Service Invoke

Calls a service from inside a mediation flow, rather than waiting until the end of the mediation flow and using the callout mechanism.

- If the service returns a fault, you can retry the same service or call another service.

- The Service Invoke mediation primitive is a powerful mediation primitive that can be used on its own for simple service calls, or in combination with other mediation primitives for complex mediations.

Set Message Type
During integration development, lets you treat weakly-typed message fields as though they are strongly-typed. A field is weakly-typed if it can contain more than one type of data. A field is strongly-typed if its type and internal structure are known.

- At runtime, the Set Message Type mediation primitive lets you check that the content of a message matches the data types you expect.

SOAP Header Setter
Provides a mechanism for managing headers in SOAP messages.

- Can create, set, copy, or delete SOAP message headers.
- Can set multiple actions to change multiple SOAP headers.

Stop Stops a particular path in the flow, without generating an exception.

Type Filter
Allows you to direct messages down a different path of a flow, based on their type.

WebSphere eXtreme Scale Retrieve
You can retrieve information from an eXtreme Scale server cache environment.

- You can look up values in the cache, and store them as elements in the message using a key.
- Combining the eXtreme Scale Store and Retrieve mediation primitives, you can cache the response from a back-end system. Future requests will not require access to that back-end system.
- You must create eXtreme Scale definitions using the WebSphere ESB administrative console, so you can specify which eXtreme Scale server to use.

WebSphere eXtreme Scale Store
You can store information into an eXtreme Scale server cache environment.

- You can store information in a eXtreme Scale cache using a key and an object.
- Combining the eXtreme Scale Store and Retrieve mediation primitives, you can use the Store mediation primitive to store data within the cache, and use the Retrieve mediation primitive to retrieve data previously stored within the cache.
- You must create eXtreme Scale definitions using the WebSphere ESB administrative console, so you can specify which eXtreme Scale server to use.

Dynamic routing:

You can route messages in various ways using endpoints defined at integration time or endpoints determined, dynamically, at run time.

Dynamic routing covers two message routing cases:

- Message routing where the flow is dynamic, but all possible endpoints are predefined in a Service Component Architecture (SCA) module.
- Message routing where the flow is dynamic, and the endpoint selection is also dynamic. The service endpoints are selected from an external source at run time

Dynamic endpoint selection

The run time has the capability to route request and response messages to an endpoint address identified by a message header element. This message header element can be updated by mediation primitives, in a

mediation flow. The endpoint address could be updated with information from a registry, a database, or with information from the message itself. Routing of response messages applies only when the response is being sent by a Web service JAX-WS export.

In order for the run time to implement dynamic routing on a request or response, the SCA module must have the `Use dynamic endpoint if set in the message header` property set. Integration developers can set the `Use dynamic endpoint if set in the message header` property or they can promote it (make it visible at run time), so that the runtime administrator can set it. You can view module properties in the Module Properties window. To see the window, click **Applications** > **SCA Modules** > **Module Properties**. The integration developer gives promoted properties alias names, and these are the names displayed on the administrative console.

Registry

You can use IBM WebSphere Service Registry and Repository (WSRR) to store service endpoint information, and then create SCA modules to retrieve endpoints from the WSRR registry.

When you develop SCA modules, you use the Endpoint Lookup mediation primitive to allow a mediation flow to query a WSRR registry for a service endpoint, or a set of service endpoints. If an SCA module retrieves a set of endpoints then it must use another mediation primitive to select the preferred one.

Mediation policy control of service requests:

You can use mediation policies to control mediation flows between service requesters and service providers.

You can control mediation flows using mediation policies stored in IBM WebSphere Service Registry and Repository (WSRR). The implementation of service policy management in WSRR is based on the Web Services Policy Framework (WS-Policy).

In order to control service requests using mediation policies, you must have suitable Service Component Architecture (SCA) modules and mediation policy documents in your WSRR registry.

How to attach a mediation policy to a service request

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry. Therefore, an SCA module must contain a mediation flow component in order to support mediation policy control of service requests.

In the registry, you can attach one or more mediation policies to an SCA module, or to a target service used by the SCA module. Attached mediation policies could be used (are in scope) for all service messages processed by that SCA module. The mediation policies can have policy attachments that define conditions. Mediation policy conditions allow different mediation policies to apply in different contexts. In addition, mediation policies can have classifications, which can be used to specify a governance state.

WebSphere Service Registry and Repository:

The WebSphere Service Registry and Repository (WSRR) product allows you to store, access, and manage information about service endpoints and mediation policies. You can use WSRR to make your service applications more dynamic, and more adaptable to changing business conditions.

Introduction

Mediation flows can use WSRR as a dynamic lookup mechanism, providing information about service endpoints or mediation policies.

To configure access to WSRR, you create WSRR definition documents using the administrative console. Alternatively, you can use the WSRR administration commands from the wsadmin scripting client. WSRR definitions and their connection properties are the mechanism used to connect to a registry instance, and retrieve a service endpoint or mediation policy.

Service endpoints

You can use WSRR to store information about services that you already use, that you plan to use, or that you want to be aware of. These services might be in your systems, or in other systems. For example, an application could use WSRR to locate the most appropriate service to satisfy its functional and performance needs.

When you develop an SCA module that needs to access service endpoints from WSRR, you must include an Endpoint Lookup mediation primitive in the mediation flow. At run time, the Endpoint Lookup mediation primitive obtains service endpoints from the registry.

Mediation policies

You can also use WSRR to store mediation policy information. Mediation policies can help you to control service requests, by dynamically overriding module properties. If WSRR contains mediation policies that are attached to an object representing either your SCA module or your target service, then the mediation policies could override the module properties. If you want different mediation policies to apply in different contexts, you can create mediation policy conditions.

Note: Mediation policies are concerned with the control of mediation flows, and not with security.

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry.

WebSphere eXtreme Scale:

By using the WebSphere eXtreme Scale (eXtreme Scale) product you can provide a caching system that you can integrate with a IBM Business Process Manager application. Using eXtreme Scale with IBM Business Process Manager can improve service response times and reliability, and provide additional integration functionality.

eXtreme Scale acts as an elastic, scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. With eXtreme Scale, you can also get qualities of service such as transactional integrity, high availability, and predictable response times.

You can use mediation flows to access the eXtreme Scale caching function by including the WebSphere eXtreme Scale mediation primitives within your flow. When you develop a Service Component Architecture (SCA) module that needs to store information in an eXtreme Scale cache, you must include the WebSphere eXtreme Scale Store mediation primitive in the mediation flow. If you want to retrieve information from an eXtreme Scale cache, you must include the WebSphere eXtreme Scale Retrieve mediation primitive. By combining the two mediation primitives in a mediation flow you can cache the response from a back-end system, so that future requests can retrieve the response from the cache.

To configure access to eXtreme Scale , you must create a WebSphere eXtreme Scale definition using the administrative console. Alternatively, you can use the WebSphere eXtreme Scale administration commands from the wsadmin scripting client. An eXtreme Scale definition is the mechanism used by the WebSphere eXtreme Scale Retrieve and Store mediation primitives to connect to an eXtreme Scale server.

Message Service clients

Message Service clients is available for C/C++ and .NET to enable non-Java applications to connect to the enterprise service bus.

Message Service Clients for C/C++ and .NET provides an API called XMS that has the same set of interfaces as the Java Message Service (JMS) API. Message Service Client for C/C++ contains two implementations of XMS, one for use by C applications and another for use by C++ applications. Message Service Client for .NET contains a fully-managed implementation of XMS, which can be used by any .NET compliant language.

You can obtain Message Service Clients for .NET from http://www.ibm.com/support/docview.wss?uid=swg24011756.

You can obtain Message Service Clients for C/C++ from http://www.ibm.com/support/docview.wss?uid=swg24007092.

You can also install and use the Java EE client support from WebSphere Application Server Network Deployment, including Web services Client, EJB Client, and JMS Client.

Chapter 2. Learn more about key concepts

Use this section as a starting point to investigate the technologies used in and by IBM Business Process Manager.

Authoring scenarios

Use scenarios to understand and work with components and products from the business process management family.

Versioning

The lifecycle of a process application begins with the creation of the process application and continues through a cycle of updating, deploying, co-deploying, undeploying, and archiving the process application. *Versioning* is a mechanism used to manage the lifecycle of the process application by uniquely identifying the individual versions of the process application.

The way that versioning works in IBM Business Process Manager depends on what you are deploying–a process application, deployed from the repository in IBM Process Center, or an enterprise application deployed directly from IBM Integration Designer.

The process applications and toolkits that you deploy to a runtime environment from the Process Center are, by default, versioned. For enterprise applications, you can choose to version modules and libraries in IBM Integration Designer.

In addition, you can create versions of a human task or state machine, so that multiple versions of the task or state machine can coexist in the runtime environment.

Versioning process applications

Versioning provides the ability for the runtime environment to identify snapshots in the lifecycle of a process application, and to be able to concurrently run multiple snapshots on a process server.

To understand how process applications are versioned, it is important to remember that a process application is a container that holds various artifacts used in or by the process application (for example, process models or BPDs, toolkit references, services, tracks, or monitor models). Any versioning is done at this container level, not at the level of the individual artifacts. For process applications, that means that versioning happens when you take a snapshot.

You can compare snapshots to determine differences between the versions. For example, if a developer fixed a problem with a service and took a snapshot of its containing process application or toolkit at that point, and then a different developer made several additional changes to the same service and took a new snapshot, the project manager can compare the two snapshots to determine which changes were made when and by whom. If the project manager decided that the additional changes to the service were not worthwhile, the project manager can revert to the snapshot of the original fix.

You can run different versions (snapshots) of a process application concurrently on a server; when you install a new snapshot, either remove the original or leave it running.

Version context

Each snapshot has unique metadata to identify the version (referred to as version context). You assign that identifier, but IBM recommends using a three-digit numeric version system in the format <major>.<minor>.<service>. See the topics about naming conventions for a more detailed description of this versioning scheme.

IBM Business Process Manager assigns a global namespace for each process application. The global namespace is specifically either the process application's tip or a particular process application snapshot. The version name used by the server cannot be longer than seven characters, so the assigned name is an acronym that uses characters from the snapshot name that you assigned. Snapshot acronyms areidentical to their snapshot names if the snapshot names conform to the recommended IBM VRM style and are not more that seven characters. For example, a snapshot name of 1.0.0 will have an acronym of 1.0.0, and a snapshot name of 10.3.0 will have the acronym of 10.3.0. The snapshot acronym will be guaranteed to be unique within the context of the process application within the scope of the Process Center server. For that reason, you cannot edit the snapshot acronym.

Versioning considerations for process applications in multiple clusters

You can install the same version of a process application to multiple clusters within the same cell. To differentiate among these multiple installations of the same version of the process application, create a snapshot for each installation and include a cell-unique ID in the snapshot name (for example, v1.0_cell1_1 and v1.0_cell1_2). Each snapshot is a new version of the process application (from a pure lifecycle management perspective), but the content and function are the same.

When you install a process application in a cluster, an automatic synchronization of the nodes is performed.

Versioning considerations for Process Designer toolkits

Remember that process application snapshots are typically taken when you are ready to test or install. Toolkit snapshots, however, are typically taken when you are ready for that toolkit to be used by process applications. Afterward, if you want to update the toolkit, you must take another snapshot of "tip" when you are ready, and then the owners of process applications and toolkits can decide whether they want to move up to the new snapshot.

Versioning modules and libraries

If a module or library is in a process application or toolkit, it takes on the lifecycle of the process application or toolkit (versions, snapshots, tracks, and so on). Module and library names must be unique within the scope of a process application or toolkit.

This topic describes the versioning of modules and libraries that are used with process applications. Note, however, that if you deploy modules directly from IBM Integration Designer to Process Server, you can continue to follow the procedure of assigning version numbers to modules during deployment, as described in “Creating versioned modules and libraries”.

A module or library that is associated with the IBM Process Center must have its dependent libraries in the same process application or in a dependent toolkit.

The following table lists the selections you can make in the dependency editor in IBM Integration Designer when a library is associated with a process application or toolkit:

Table 31. Dependencies for Module, Process App or Toolkit, and Global libraries

Library scope	Description	Can depend on . . .
Module	A copy of this library exists on the server for each module that uses it.	A module-scoped library can depend on all types of libraries.
Process App or Toolkit	The library is shared among all modules in the scope of the process application or toolkit. This setting takes effect if deployment is done through the IBM Process Center. If deployment occurs outside of the IBM Process Center, the library is copied into each module. **Note:** Libraries created in IBM Integration Designer version 8 have a sharing level of **Process App or Toolkit** by default.	A library of this type can depend only on global libraries.
Global	The library is shared among all modules that are running.	A global library can depend only on other global libraries. **Note:** You must configure a WebSphere shared library in order to deploy the global library. See "Module and library dependencies" for more information.

Modules and libraries associated with process applications or toolkits

You do not need to version modules and libraries associated with process applications or toolkits.

Modules and libraries that are associated with a process application or toolkit do not need to be versioned. In fact, you cannot create a version of a module or library associated with a process application or toolkit in the dependency editor. Modules and libraries associated with a process application or toolkit use snapshots, a function in the Process Center, to achieve the same result as a version.

Libraries associated with a process application or toolkit will not have a required version number in the Libraries section of the dependency editor because no version is needed.

Naming conventions

A naming convention is used to differentiate the various versions of a process application as it moves through the lifecycle of updating, deploying, co-deploying, undeploying, and archiving.

This section provides you with the conventions that are used to uniquely identify versions of a process application.

A *version context* is a combination of acronyms that uniquely describes a process application or toolkit. Each type of acronym has a naming convention. The acronym is limited to a maximum length of seven characters from the [A-Z0-9_] character set, except for the snapshot acronym, which can also include a period.

- The process application acronym is created when the process application is created. It can be a maximum of seven characters in length.
- The snapshot acronym is created automatically when the snapshot is created. It can be a maximum of seven characters in length.

  If the snapshot name meets the criteria for a valid snapshot acronym, the snapshot name and acronym will be the same.

  Note: When using the mediation flow component version-aware routing function, name your snapshot so that it conforms to the *<version>.<release>.<modification>* scheme (for example, **1.0.0**). Because the

snapshot acronym is limited to seven characters, the digit values are limited to a maximum of five total digits (five digits plus two periods). Therefore, care should be taken when the digit fields are incremented, because anything beyond the first seven characters is truncated.

For example, a snapshot name **11.22.33** results in a **11.22.3** snapshot acronym.

- The track acronym is automatically generated from the first character of each word of the track name. For example, a new track created with the name **My New Track** would result in an acronym value of **MNT**.

  The default track name and acronym are **Main**. Deployment to a IBM Process Center server includes the track acronym in the versioning context if the track acronym is not **Main**.

A business process definition in a process application is typically identified by the process application name acronym, the snapshot acronym, and the name of the business process definition. Choose unique names for your business process definitions whenever possible. When duplicate names exist, you might encounter the following problems:

- You might be unable to expose the business process definitions as web services without some form of mediation.
- You might be unable to invoke a business process definition created in IBM Process Designer from a BPEL process created in IBM Integration Designer.

The version context varies, depending on how the process application is deployed.

Naming conventions for Process Center server deployments

On the IBM Process Center server, you can deploy a snapshot of a process application as well as a snapshot of a toolkit. In addition, you can deploy the tip of a process application or the tip of a toolkit. (A *tip* is the current working version of your process application or toolkit.) The version context varies, depending on the type of deployment.

For process applications, the process application tip or the specific process application snapshot is used to uniquely identify the version.

Toolkits can be deployed with one or more process applications, but the lifecycle of each toolkit is bound to the lifecycle of the process application. Each process application has its own copy of the dependent toolkit or toolkits deployed to the server. A deployed toolkit is not shared between process applications.

If the track associated with the process application is named something other than the default of **Main**, the track acronym is also part of the version context.

For more information, see the "Examples" on page 37 section, later in this topic.

Process application snapshots

For process application snapshot deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application track acronym (if a track other than **Main** is used)
- Process application snapshot acronym

Stand-alone toolkits

For toolkit snapshot deployments, the version context is a combination of the following items:

- Toolkit name acronym
- Toolkit track acronym (if a track other than **Main** is used)
- Toolkit snapshot acronym

Tips

Process application tips are used during iterative testing in Process Designer. They can be deployed to Process Center servers only.

For process application tip deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application track acronym (if a track other than **Main** is used)
- "Tip"

Toolkit tips are also used during iterative testing in Process Designer. They are not deployed to a production server.

For toolkit tip deployments, the version context is a combination of the following items:

- Toolkit name acronym
- Toolkit track acronym (if a track other than **Main** is used)
- "Tip"

Examples

Resources should be uniquely named and identified externally using the version context.

- The following table shows an example of names that are uniquely identified. In this example, a process application tip uses the default track name (**Main**):

Table 32. Process application tip with default track name

Type of name	**Example**
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Main**
Process application track acronym	"" (when the track is **Main**)
Process application snapshot	
Process application snapshot acronym	**Tip**

Any SCA modules associated with this process application tip include the version context, as shown in the following table:

Table 33. SCA modules and version-aware EAR files

SCA module name	**Version-aware name**	**Version-aware EAR/application name**
M1	PA1-Tip-M1	PA1-Tip-M1.ear
M2	PA1-Tip-M2	PA1-Tip-M2.ear

- The following table shows an example of a process application tip that uses a non-default track name:

Table 34. Process application tip with non-default track name

Type of name	**Example**
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Track1**
Process application track acronym	**T1**

Table 34. Process application tip with non-default track name (continued)

Type of name	Example
Process application snapshot	
Process application snapshot acronym	**Tip**

Any SCA modules associated with this process application tip include the version context, as shown in the following table:

Table 35. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-T1-Tip-M1	PA1-T1-Tip-M1.ear
M2	PA1-T1-Tip-M2	PA1-T1-Tip-M2.ear

Similar naming conventions apply to advanced Toolkit tip and snapshot deployments. They also apply to advanced snapshots installed to Process Server.

- The following table shows an example of names that are uniquely identified. In this example, a process application snapshot uses the default track name (**Main**):

Table 36. Process application snapshot with default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Main**
Process application track acronym	"" (when the track is **Main**)
Process application snapshot	**Process Shapshot V1**
Process application snapshot acronym	**PSV1**

Any SCA modules associated with this process application snapshot include the version context, as shown in the following table:

Table 37. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-PSV1-M1	PA1-PSV1-M1.ear
M2	PA1-PSV1-M2	PA1-PSV1-M2.ear

- The following table shows an example of a process application snapshot that uses a non-default track name:

Table 38. Process application snapshot with non-default track name

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application track	**Track1**
Process application track acronym	**T1**
Process application snapshot	**Process Snapshot V1**
Process application snapshot acronym	**PSV1**

Any SCA modules associated with this process application snapshot include the version context, as shown in the following table:

Table 39. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-T1-PSV1-M1	PA1-T1-PSV1-M1.ear
M2	PA1-T1-PSV1-M2	PA1-T1-PSV1-M2.ear

Naming conventions for Process Server deployments

On the Process Server, you can deploy the snapshot of a process application. The process application snapshot acronym is used to uniquely identify the version.

For process application snapshot deployments, the version context is a combination of the following items:

- Process application name acronym
- Process application snapshot acronym

Resources should be uniquely named and identified externally using the version context. The following table shows an example of names that are uniquely identified:

Table 40. Example of names and acronyms

Type of name	Example
Process application name	**Process Application 1**
Process application name acronym	**PA1**
Process application snapshot	**1.0.0**
Process application snapshot acronym	**1.0.0**

A resource, such as a module or library, has the version context as part of its identify.

The following table shows an example of two modules and how the associated EAR files include the version context:

Table 41. SCA modules and version-aware EAR files

SCA module name	Version-aware name	Version-aware EAR/application name
M1	PA1-1.0.0-M1	PA1-1.0.0-M1.ear
M2	PA1-1.0.0-M2	PA1-1.0.0-M2.ear

The following table shows an example of two process-application-scoped libraries and how the associated JAR files include the version context:

Table 42. Process-application-scoped libraries and version-aware JAR files

SCA process-application-scoped library name	Version-aware name	Version-aware JAR name
Lib1	PA1-1.0.0-Lib1	PA1-1.0.0-Lib1.jar
Lib2	PA1-1.0.0-Lib2	PA1-1.0.0-Lib2.jar

Version-aware bindings

Process applications can contain SCA modules that include import and export bindings. When you co-deploy applications, the binding for each version of the application must be unique. Some bindings

are automatically updated during deployment to ensure the uniqueness between versions. In other cases, you have to update the binding after deployment to ensure its uniqueness.

A *version-aware* binding is scoped to a particular version of a process application, which guarantees its uniqueness between process applications. The following sections describe the bindings that are automatically updated to be version-aware as well as any actions you need to take at run time when a binding is not version-aware. For information about things to consider when you create modules, see "Considerations when using bindings".

SCA

The target of an SCA binding is automatically renamed to be version-aware during deployment if the import and export bindings of the module are defined in the same process application scope.

If the bindings are not defined in the same process application scope, an information message is logged. You must modify the import binding after deployment to change the endpoint target address. You can use the administrative console to change the endpoint target address.

Web service (JAX-WS or JAX-RPC)

The endpoint target address of a web service binding is automatically renamed to be version-aware during deployment if all the following conditions are true:

- You followed the default naming convention for the address:
  http://*ip*:*port*/*ModuleName*Web/sca/*ExportName*
- The endpoint address is SOAP/HTTP.
- The import and export bindings of the module are defined in the same process application scope.

If these conditions are not true, an information message is logged. The action you then take depends on how you are deploying your process application:

- If you are co-deploying your process application, you must manually rename the SOAP/HTTP endpoint URL or the SOAP/JMS destination queue to be unique between versions of the process application. You can use the administrative console after deployment to change the endpoint target address.
- If you are deploying only a single version of the process application, you can ignore this message

For SOAP/ JMS web service binding snapshot co-deployment, the action you take depends on how you are deploying your process application:

- If the import and target export are in the same process application, perform the following steps before you publish the process application to the Process Center and create the snapshot:
  1. Change the endpoint URL of the export. Make sure the destination and connection factory are unique.
  2. Change the endpoint URL of the import so that it is the same as the one you specified for the export in the previous step.
- If the import and target export are in different process applications, perform the following steps:
  1. Change the endpoint URL of the export. Make sure the destination and connection factory are unique.
  2. Publish the process application to the Process Center.
  3. Create the snapshot.
  4. Deploy the process application to the Process Server.
  5. Use the WebSphere administrative console to change the endpoint URL of the corresponding import so that it is the same as the one you specified for the export.

HTTP

The endpoint URL address of an HTTP binding is automatically renamed to be version-aware during deployment if all the following conditions are true:

- You followed the default naming convention for the address:
  http(s)://*ip*:*port*/*ModuleName*Web/*contextPathinExport*
- The import and export bindings of the module are defined in the same process application scope.

If these conditions are not true, an information message is logged. The action you then take depends on how you are deploying your process application:

- If you are co-deploying your process application, you must manually rename the endpoint URL to be unique between versions of the process application. You can use the administrative console after deployment to change the endpoint target address.
- If you are deploying only a single version of the process application, you can ignore this message

JMS and generic JMS

System-generated JMS and generic JMS bindings are automatically version-aware.

Note: For user-defined JMS and generic JMS bindings, no automatic renaming occurs during deployment to enable the bindings to be version-aware. If the binding is user-defined, you must rename the following attributes to be unique between versions of the process application:

- Endpoint configuration
- Receive destination queue
- Listener port name (if defined)

Set the matching Send destination if you change the target module endpoint.

MQ/JMS and MQ

No automatic renaming occurs during deployment to enable bindings of type MQ/JMS or MQ to be version-aware.

You must rename the following attributes to be unique between versions of the process application:

- Endpoint configuration
- Receive destination queue

Set the matching Send destination if you change the target module endpoint.

EJB

No automatic renaming occurs during deployment to enable bindings of type EJB to be version-aware.

You must rename the JNDI names attribute to be unique between versions of the process application.

Note that client applications also need to be updated to use the new JNDI names.

EIS

A resource adapter is automatically renamed to be version-aware during deployment as long as the default resource name (***ModuleName*App:Adapter Description**) was not modified.

If the default resource name was modified, the resource adapter names must be unique between versions of the process application.

If the resource adapter names are not unique, an informational message is logged during deployment to alert you. You can manually rename the resource adapters after deployment using the administrative console.

Version-aware dynamic invocation

You can configure mediation flow components to route messages to endpoints that are determined dynamically at run time. When you create the mediation module, you configure the endpoint lookup to use version-aware routing.

If you use the IBM_VRM style (<*version*>.<*release*>.<*modification*) for the snapshot, you can export the process application EAR file to WebSphere Service Registry and Repository (WSRR). When you create the mediation module, you then configure the endpoint lookup to use version-aware routing. For example, you select **Return endpoint matching latest compatible version of SCA module-based services** from the **Match Policy field**, and you select **SCA** for **Binding Type**.

Future versions of the process application are deployed to the server and published to WSRR, and the mediation module endpoint lookup dynamically invokes the latest compatible version of the service endpoint.

Note that, as an alternative, you can set the target in the SMOHeader, and the value can be carried by the request message.

Deploying process applications with Java modules and projects

Process applications can contain custom Java EE modules and Java projects. When you co-deploy applications, the custom Java EE module for each version of the application must be unique.

Note that custom Java EE modules and Java projects are deployed to a server if they are deployed with an SCA module that has a dependency declared on them. If you do not select **Deploy with module** (which is the default) when you declare the dependency, you need to deploy the module or project manually.

Deploying process applications with business rules and selectors

If you are deploying multiple versions of a process application that includes a business rule or selector component, be aware of the way that the associated metadata is used by the versions.

The dynamic metadata for a business rule or selector component is defined at run time by the component name, component target namespace, and component type. If two or more versions of a process application containing a business rule or selector are deployed to the same runtime environment, they will share the same rule logic (business rule) or routing (selector) metadata.

To enable each version of the business rule or selector component of the process application to use its own dynamic metadata (rule logic or routing), refactor the target namespace so that it is unique for each version of the process application.

Deployment architecture

The IBM Business Process Manager deployment architecture consists of software processes called servers, topological units referenced as nodes and cells, and the configuration repository used for storing configuration information.

Cells

In IBM Business Process Manager, *cells* are logical groupings of one or more nodes in a distributed network.

A cell is a configuration concept, a way for administrators to logically associate nodes with one another. Administrators define the nodes that make up a cell, according to the specific criteria that make sense in their organizational environments.

Administrative configuration data is stored in XML files. A cell retains master configuration files for each server in every node in the cell. Each node and server also have their own local configuration files. Changes to a local node or to a server configuration file are temporary, if the server belongs to the cell. While in effect, local changes override cell configurations. Changes to the master server and master node configuration files made at the cell level replace any temporary changes made at the node when the cell configuration documents are synchronized to the nodes. Synchronization occurs at designated events, such as when a server starts.

Servers

Servers provide the core functionality of IBM Business Process Manager. Process servers extend, or augment, the ability of an application server to handle Service Component Architecture (SCA) modules. Other servers (deployment managers and node agents) are used for managing process servers.

A process server can be either a *stand-alone server* or a *managed server*. A managed server can optionally be a member of a *cluster*. A collection of managed servers, clusters of servers, and other middleware is called a *deployment environment*. In a deployment environment, each of the managed servers or clusters is configured for a specific function within the deployment environment (for example, destination host, application module host, or Common Event Infrastructure server). A stand-alone server is configured to provide all of the required functions.

Servers provide the runtime environment for SCA modules, for the resources that are used by those modules (data sources, activation specifications, and JMS destinations), and for IBM-supplied resources (message destinations, Business Process Choreographer containers, and Common Event Infrastructure servers).

A *node agent* is an administrative agent that represents a node to your system and manages the servers on that node. Node agents monitor servers on a host system and route administrative requests to servers. The node agent is created when a node is federated to a deployment manager.

A *deployment manager* is an administrative agent that provides a centralized management view for multiple servers and clusters.

A stand-alone server is defined by a stand-alone profile; a deployment manager is defined by a deployment manager profile; managed servers are created within a *managed node*, which is defined by a custom profile.

Stand-alone servers

A stand-alone server provides an environment for deploying SCA modules in one server process. This server process includes, but is not limited to, an administrative console, a deployment target, the messaging support, the business process rules manager, and a Common Event Infrastructure server.

A stand-alone server is simple to set up, and has a First steps console from which you can start and stop the server and open the samples gallery and the administrative console. If you install the IBM Business Process Manager samples, and then open the samples gallery, a sample solution is deployed to the stand-alone server. You can explore the resources used for this sample in the administrative console.

You can deploy your own solutions to a stand-alone server, but a stand-alone server cannot provide the capacity, scalability, or robustness that is required of a production environment. For your production environment, it is better to use a network deployment environment.

It is possible to start off with a stand-alone server and later include it in a network deployment environment, by federating it to a deployment manager cell, *provided that no other nodes have been federated*

to that cell. It is not possible to federate multiple stand-alone servers into one cell. To federate the stand-alone server, use the administrative console of the deployment manager or the **addNode** command. The stand-alone server must not be running when you federate it using the **addNode** command.

A stand-alone server is defined by a stand-alone server profile.

Clusters

Clusters are groups of servers that are managed together and participate in workload management.

A cluster can contain nodes or individual application servers. A node is usually a physical computer system with a distinct host IP address that is running one or more application servers. Clusters can be grouped under the configuration of a cell, which logically associates many servers and clusters with different configurations and applications with one another depending on the discretion of the administrator and what makes sense in their organizational environments.

Clusters are responsible for balancing workload among servers. Servers that are a part of a cluster are called cluster members. When you install an application on a cluster, the application is automatically installed on each cluster member.

Because each cluster member contains the same applications, you can distribute client tasks according to the capacities of the different machines by assigning weights to each server.

Assigning weights to the servers in a cluster improves performance and failover. Tasks are assigned to servers that have the capacity to perform the task operations. If one server is unavailable to perform the task, it is assigned to another cluster member. This reassignment capability has obvious advantages over running a single application server that can become overloaded if too many requests are made.

Profiles

A profile defines a unique runtime environment, with separate command files, configuration files, and log files. Profiles define three different types of environments on IBM Business Process Manager systems: stand-alone server, deployment manager, and managed node.

Using profiles, you can have more than one runtime environment on a system, without having to install multiple copies of the IBM Business Process Manager binary files.

Use the Profile Management Tool or the **manageprofiles** command-line utility to create profiles.

Note: On distributed platforms, each profile has a unique name. On the z/OS platform, all profiles are named “default”.

The profile directory

Every profile in the system has its own directory containing all of its files. You specify the location of the profile directory when you create the profile. By default, it is in the `profiles` directory in the directory where IBM Business Process Manager is installed. For example, the Dmgr01 profile is in `C:\Program Files\IBM\WebSphere\ProcServer\profiles\Dmgr01`.

The First steps console

Every profile in the system has a First steps console. You can use this interface to familiarize yourself with the stand-alone server, deployment manager, or managed node.

The default profile

The first profile that you create within one installation of IBM Business Process Manager is the *default profile*. The default profile is the default target for commands issued from the `bin` directory in the

directory where IBM Business Process Manager was installed. If only one profile exists on a system, every command operates on that profile. If you create another profile, you can make it the default.

Note: The default profile is not necessarily a profile whose name is “default”.

Augmenting profiles

If you already have a deployment manager profile, a custom profile, or a stand-alone server profile created for WebSphere Application Server Network Deployment or WebSphere ESB, you can *augment* it to support IBM Business Process Manager in addition to existing function. To augment a profile, first install IBM Business Process Manager. Then use the Profile Management Tool or the **manageprofiles** command-line utility.

Restriction: You cannot augment a profile if it defines a managed node that is already federated to a deployment manager.

Deployment managers

A deployment manager is a server that manages operations for a logical group, or cell, of other servers. The deployment manager is the central location for administering the servers and clusters.

When creating a deployment environment, the deployment manager profile is the first profile that you create. The deployment manager has a First steps console, from which you can start and stop the deployment manager and start its administrative console. You use the administrative console of the deployment manager to manage the servers and clusters in the cell. This includes configuring servers and clusters, adding servers to clusters, starting and stopping servers and clusters, and deploying SCA modules.

Although the deployment manager is a type of server, you cannot deploy modules to the deployment manager itself.

Nodes

A *node* is a logical grouping of managed servers.

A node usually corresponds to a logical or physical computer system with a distinct IP host address. Nodes cannot span multiple computers. Node names usually are identical to the host name for the computer.

Nodes in the network deployment topology can be managed or unmanaged. A managed node has a node agent process that manages its configuration and servers. Unmanaged nodes do not have a node agent.

Managed nodes

A *managed node* is a node that is federated to a deployment manager and contains a node agent and can contain managed servers. In a managed node, you can configure and run managed servers.

The servers that are configured on a managed node make up the resources of your deployment environment. These servers are created, configured, started, stopped, managed and deleted using the administrative console of the deployment manager.

A managed node has a node agent that manages all servers on a node.

When a node is federated, a node agent process is created automatically. This node agent must be running to be able to manage the configuration of the profile. For example, when you do the following tasks:

- Start and stop server processes.
- Synchronize configuration data on the deployment manager with the copy on the node.

However, the node agent does not need to be running in order for the applications to run or to configure resources in the node.

A managed node can contain one or more servers, which are managed by a deployment manager. You can deploy solutions to the servers in a managed node, but the managed node does not contain a sample applications gallery. The managed node is defined by a custom profile and has a First steps console.

Unmanaged nodes

An unmanaged node does not have a node agent to manage its servers.

Unmanaged nodes in the Network Deployment topology can have server definitions such as Web servers , but not Application Server definitions. Unmanaged nodes can never be federated. That is, a node agent can never be added to an unmanaged node. Another type of unmanaged node is a stand-alone server. The deployment manager cannot manage this stand-alone server because it is not known to the cell. A stand-alone server can be federated. When it is federated, a node agent is automatically created. The node becomes a managed node in the cell.

Node agents

Node agents are administrative agents that route administrative requests to servers.

A node agent is a server that runs on every host computer system that participates in the Network Deployment configuration. It is purely an administrative agent and is not involved in application-serving functions. A node agent also hosts other important administrative functions such as file transfer services, configuration synchronization, and performance monitoring.

Naming considerations for profiles, nodes, servers, hosts, and cells

This topic discusses reserved terms and issues you must consider when naming your profile, node, server, host, and cell (if applicable). This topic applies to distributed platforms.

Profile naming considerations

The profile name can be any unique name with the following restrictions. Do not use any of the following characters when naming your profile:

- Spaces
- Special characters that are not allowed within the name of a directory on your operating system, such as *, &, or ?.
- Slashes (/) or back slashes (\)

Double-byte characters are allowed.

Windows **Directory path considerations:** The installation directory path must be less than or equal to 60 characters. The number of characters in the *profiles_directory_path**profile_name* directory must be less than or equal to 80 characters.

Node, server, host, and cell naming considerations

Reserved names: Avoid using reserved names as field values. The use of reserved names can cause unpredictable results. The following words are reserved:

- cells
- nodes
- servers
- clusters
- applications

- deployments

Descriptions of fields on the Node and Hosts Names and Node, Host, and Cell Names pages: Table 17 on page 46 describes the fields found on the Node and Host Names and Node, Host, and Cell Names pages of the Profile Management Tool, including the field names, default values, and constraints. Use this information as a guide when you are creating profiles.

Table 43. Naming guidelines for nodes, servers, hosts, and cells

Field name	Default value	Constraints	Description
Stand-alone server profiles			
Node name	Linux UNIX Windows *shortHostName* Node *NodeNumber* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Avoid using the reserved names.	Select any name you want. To help organize your installation, use a unique name if you plan to create more than one server on the system.
Server name	Linux UNIX Windows server1	Use a unique name for the server.	The logical name for the server.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.

Table 43. Naming guidelines for nodes, servers, hosts, and cells (continued)

Field name	Default value	Constraints	Description
Cell name	Linux UNIX Windows *shortHostName* Node *NodeNumber* Cell where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Use a unique name for the cell. A cell name must be unique in any circumstance in which the product is running on the same physical workstation or cluster of workstations, such as a Sysplex. Additionally, a cell name must be unique in any circumstance in which network connectivity between entities is required either between the cells or from a client that must communicate with each of the cells. Cell names also must be unique if their name spaces are going to be federated. Otherwise, you might encounter symptoms such as a javax.naming.Name NotFoundException exception, in which case, you need to create uniquely named cells.	All federated nodes become members of a deployment manager cell.
Deployment manager profiles			
Node name	Linux UNIX Windows *shortHostName* Cell Manager*Node Number* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Use a unique name for the deployment manager. Avoid using the reserved names.	The name is used for administration within the deployment manager cell.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. Avoid using the reserved names. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.

Table 43. Naming guidelines for nodes, servers, hosts, and cells (continued)

Field name	Default value	Constraints	Description
Cell name	Linux UNIX Windows *shortHostName* Cell *CellNumber* where: • *shortHost Name* is the short host name. • *CellNumber* is a sequential number starting at 01.	Use a unique name for the deployment manager cell. A cell name must be unique in any circumstance in which the product is running on the same physical workstation or cluster of workstations, such as a Sysplex. Additionally, a cell name must be unique in any circumstance in which network connectivity between entities is required either between the cells or from a client that must communicate with each of the cells. Cell names also must be unique if their name spaces are going to be federated. Otherwise, you might encounter symptoms such as a javax.naming.Name NotFoundException exception, in which case, you need to create uniquely named cells.	All federated nodes become members of the deployment manager cell , which you name in the Node, Host, and Cell Names page of the Profile Management Tool.
Custom profiles			
Node name	Linux UNIX Windows *shortHostName* Node *NodeNumber* where: • *shortHost Name* is the short host name. • *NodeNumber* is a sequential number starting at 01.	Avoid using the reserved names. Use a unique name within the deployment manager cell.	The name is used for administration within the deployment manager cell to which the custom profile is added. Use a unique name within the deployment manager cell.
Host name	Linux UNIX Windows The long form of the domain name server (DNS) name.	The host name must be addressable through your network. If you are planning to use Business Space, use a fully qualified host name.	Use the actual DNS name or IP address of your workstation to enable communication with it. See additional information about the host name following this table.

Host name considerations:

The host name is the network name for the physical workstation on which the node is installed. The host name must resolve to a physical network node on the server. When multiple network cards exist in the server, the host name or IP address must resolve to one of the network cards. Remote nodes use the host name to connect to and to communicate with this node.

IBM Business Process Manager is compliant to both Internet Protocol version 4 (IPv4) and version 6 (IPv6). Wherever you can enter IP addresses in the administrative console, or elsewhere, you can do so in either format. Note that if IPv6 is implemented on your system you must enter the IP address in IPv6 format, and conversely, if IPv6 is not yet available to you, enter IP addresses in IPv4 format. For more information on IPv6 refer to the following description: IPv6.

The following guidelines can help in determining the appropriate host name for your workstation:

- Select a host name that other workstations can reach within your network.
- Do not use the generic identifier, localhost, for this value.
- Do not attempt to install IBM Business Process Manager products on a server with a host name that uses characters from the double-byte character set (DBCS). DBCS characters are not supported when used in the host name.
- Avoid using the underscore (_) character in server names. Internet standards dictate that domain names conform to the host name requirements described in Internet Official Protocol Standards RFC 952 and RFC 1123. Domain names must contain only letters (upper or lower case) and digits. Domain names can also contain dash characters (-) as long as the dashes are not on the ends of the name. Underscore characters (_) are not supported in the host name. If you have installed IBM Business Process Manager on a server with an underscore character in the server name, access the server with its IP address until you rename it.

If you define coexisting nodes on the same computer with unique IP addresses, define each IP address in a domain name server (DNS) look-up table. Configuration files for servers do not provide domain name resolution for multiple IP addresses on a workstation with a single network address.

The value that you specify for the host name is used as the value of the hostName property in configuration documents. Specify the host name value in one of the following formats:

- Fully qualified domain name servers (DNS) host name string, such as xmachine.manhattan.ibm.com
- The default short DNS host name string, such as xmachine
- Numeric IP address, such as 127.1.255.3

The fully qualified DNS host name has the advantages of being totally unambiguous and flexible. You have the flexibility of changing the actual IP address for the host system without having to change the server configuration. This value for host name is particularly useful if you plan to change the IP address frequently when using Dynamic Host Configuration Protocol (DHCP) to assign IP addresses. A disadvantage of this format is being dependent on DNS. If DNS is not available, then connectivity is compromised.

The short host name is also dynamically resolvable. A short name format has the added ability of being redefined in the local hosts file so that the system can run the server even when disconnected from the network. Define the short name to 127.0.0.1 (local loopback) in the hosts file to run disconnected. A disadvantage of the short name format is being dependent on DNS for remote access. If DNS is not available, then connectivity is compromised.

A numeric IP address has the advantage of not requiring name resolution through DNS. A remote node can connect to the node you name with a numeric IP address without DNS being available. A disadvantage of this format is that the numeric IP address is fixed. You must change the setting of the hostName property in configuration documents whenever you change the workstation IP address. Therefore, do not use a numeric IP address if you use DHCP, or if you change IP addresses regularly.

Another disadvantage of this format is that you cannot use the node if the host is disconnected from the network.

BPMN 2.0

IBM Business Process Manager business process definitions support the Common Executable subclass of the BPMN 2.0 Process Modeling conformance class, which deals with models that you can run.

BPMN (Business Process Model and Notation) is the foundational standard for the processes in IBM Process Designer and IBM Process Center. Business process definition (BPD) diagrams are based on the BPMN specification. This topic introduces some of the ways BPMN 2.0 is applied in IBM Business Process Manager. For detailed information about BPMN, see the BPMN Specification page at http://www.bpmn.org/.

IBM Business Process Manager supports the following BPMN 2.0 task types:

- None (abstract task in the BPMN 2.0 specification)
- System task (service task in the BPMN 2.0 specification)
- User task
- Script
- Decision task (business rule task in the BPMN 2.0 specification)

The IBM BPM intermediate message events provide similar functions to the BPMN send task and receive task.

BPMN 2.0 notation

Starting in V7.5.1, Process Designer BPMN 2.0 task icons in the BPD diagrams are collected on a simplified palette and displayed in process diagrams. The icons show whether your activity is a system task, user task, decision task, script, or linked process. Activities in models that were created in earlier versions also show appropriate BPMN 2.0 tasks types and task icons when you view them in version 7.5.1 or later.

Activities and tasks

There are some terminology changes from previous versions of Process Designer. A number of those changes involve activity types that have been renamed.

- Service (automated) activities are now system tasks.
- Service (task) activities in a non-system swimlane are now user tasks.
- Service (task) activities in a system swimlane are now decision tasks if they reference a decision service.
- Service (task) activities in a system swimlane are now system tasks if they reference any kind of service other than a decision service.
- Javascript activities are now script tasks.
- Nested process activities are now linked processes.
- External activities from previous versions of Process Designer are available as external implementations for user tasks or system tasks.

Gateways

There are no notation changes to the gateways of previous versions. However, there are three terminology changes. The decision gateway is now the *exclusive gateway*, the simple split or join gateway is now the *parallel gateway*, and the conditional split or join gateway is now the *inclusive gateway*.

There is also a new gateway type, the *event gateway*. An event gateway represents a branching point in a process where the alternative paths that follow the gateway are based on events that occur, rather than the evaluation of expressions using process data (as with an exclusive or inclusive gateway). A specific event, usually the receipt of a message, determines the path that will be taken.

Non-interrupting events

BPMN 2.0 added notation for non-interrupting events. By default, a boundary event interrupts the activity that it is attached to. When the event is triggered, the activity stops and the token continues down the outgoing sequence flow of the event. If the event is set as non-interrupting, when the event is triggered the attached activity continues in parallel, and a new token is generated and is passed down the outgoing sequence flow of the event. The event boundary changes to a dashed line for non-interrupting events.

Intermediate events that are attached to activities are interrupting intermediate events if they close their attached activities or non-interrupting intermediate events if they do not close their attached activities.

Start event

The BPMN specification permits process models to omit start and end event symbols. Process Designer requires that process models use start and stop events.

There are various types of start events available in Process Designer:

processes

- none
- message
- ad hoc

subprocesses

- none

event subprocesses

- error
- message
- timer

You can change the type of a start event by editing the properties of the event. You can have numerous message start events in a process, but you can use only one none start event.

End events

Four types of end events are available: *message*, *terminate*, *error* and *none*. You can change the type of an end event.

When a parent process calls a child process and the child process runs a terminate event action, the BPMN semantics say that the child process immediately stops and the parent process then continues to its next steps. In Process Designer, if a child process runs a terminate event activity, both the child process and its parent process stop.

Subprocesses

The BPMN specification defines two types of subprocesses, embedded and reusable. You can reate both types in Process Designer. Embedded subprocesses are just called *subprocesses* in Process Designer and are new in version 7.5.1. The BPMN reusable subprocess is called a *linked process* in Process Designer.

A subprocess exists within the containing process and is a way of grouping process steps to reduce diagram complexity and clutter. Subprocesses collapse multiple steps into one activity. The subprocess can be seen only by the process in which it is defined. A subprocess exists within the scope of its caller and has access to all the variables within that environment. There is no parameter passing in and out of the embedded subprocess.

Separate from the subprocess and the linked process, Process Designer has an event subprocess, which is a specialized subprocess that is used for event handling. It is not connected to other activities through sequence flow, and it occurs only if its start event is triggered.

Linked processes

A BPMN reusable subprocess is called a *linked process* in Process Designer. It is a process created outside the current process that can be called by the current process. It is reusable because other process definitions can call this process as well. The linked process defines its input and output parameters and has no access to the caller's scope or environment. The linked process is similar to the nested process available in former versions; there is no change to the behavior of the activity. Previous nested processes are migrated to linked processes. The linked process looks like a subprocess with a thick boundary and is highlighted in the Inspector window.

Loops

BPMN provides the concept of an activity that can be repeated. The activity can be atomic, meaning that the activity repeats, or it can be a subprocess, encapsulating a series of steps that are repeated. If you expand the repeated activity, you see the contained activities that are to be run repeatedly. The condition is always evaluated at the start of each loop iteration. There is no ability to evaluate at the end of each loop iteration.

IBM Business Process Manager has a *multi-instance loop*, which is run some finite number of times with the activities contained within run sequentially or in parallel.

Importing non-BPMN processes

You can import models that were created in IBM WebSphere Business Modeler and use them in Process Designer. For information about the BPMN 2.0 import, see Mapping IBM WebSphere Business Modeler elements to IBM Business Process Manager constructs. You also can import BPMN 2.0 models that were created in IBM WebSphere Business Compass, Rational Software Architect, or other modeling environments.

Business process definitions (BPDs)

To model a process in IBM Process Designer, you must create a business process definition (BPD). The business process definition can be based on an imported BPMN model.

A BPD is a reusable model of a process, defining what is common to all runtime instances of that process model. A BPD must contain a start event, an end event, at least one lane, and one or more activities. See "IBM Process Designer naming conventions" in the related links for details about the character limitations that apply to BPDs.

A Business Process Definition (BPD) needs to include a lane for each system or group of users who participates in a process. A lane can be a participant lane or a system lane. However, you can create a BPD that groups the activities of a group and a system into a single lane if that is your preference. See "Creating a business process definition (BPD)" in the related links for information about how to create a BPD.

You can designate any specific person or group to be responsible for the activities in a participant lane. Each lane that you create is assigned to the All Users participant group by default. You can use this default participant group for running and testing your BPD in the Inspector. The All Users participant group includes all users who are members of the tw_allusers security group, which is a special security group that automatically includes all users in the system.

A system lane contains activities handled by a specific IBM Process Center system. Each activity needs an implementation, which defines the activity and sets the properties for the task. During implementation, a developer creates a service or writes the JavaScript necessary to complete the activities in a system lane. See "Understanding service types" in the related links for information about services.

For each BPD that you create, you need to declare variables to capture the business data that is passed from activity to activity in your process. See "Managing and mapping variables" in the related links to learn about implementing variables.

You can also add events to a BPD. Events in IBM BPM can be triggered by a due date passing, an exception, or a message arriving. The trigger that you want determines the type of event you choose to implement. For detailed information about available event types and their triggers, see "Modeling events".

Bindings

At the core of a service-oriented architecture is the concept of a *service*, a unit of functionality accomplished by an interaction between computing devices. An *export* defines the external interface (or access point) of a module, so that Service Component Architecture (SCA) components within the module can provide their services to external clients. An *import* defines an interface to services outside a module, so the services can be called from within the module. You use protocol-specific *bindings* with imports and exports to specify the means of transporting the data into or out of the module.

Exports

External clients can invoke SCA components in an integration module over a variety of protocols (such as HTTP, JMS, MQ, and RMI/IIOP) with data in a variety of formats (such as XML, CSV, COBOL, and JavaBeans). Exports are components that receive these requests from external sources and then invoke IBM Business Process Manager components using the SCA programming model.

For example, in the following figure, an export receives a request over the HTTP protocol from a client application. The data is transformed into a business object, the format used by the SCA component. The component is then invoked with that data object.

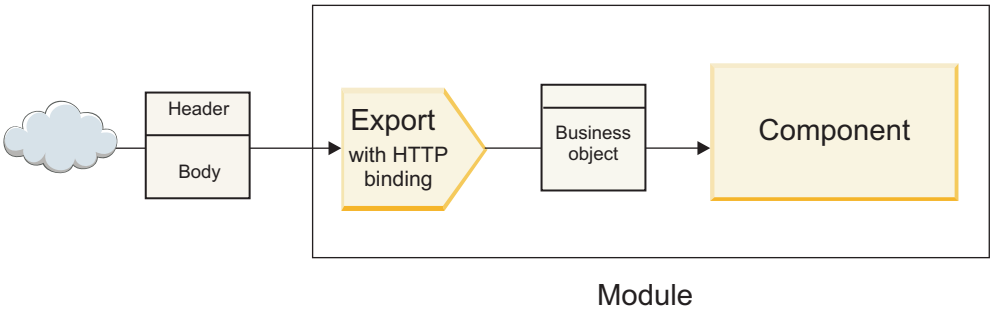


Figure 46. An export with HTTP binding

Imports

An SCA component might want to invoke a non-SCA external service that expects data in a different format. An import is used by the SCA component to invoke the external service using the SCA programming model. The import then invokes the target service in the way that the service expects.

For example, in the following figure, a request from an SCA component is sent, by the import, to an external service. The business object, which is the format used by the SCA component, is transformed to the format expect by the service, and the service is invoked.

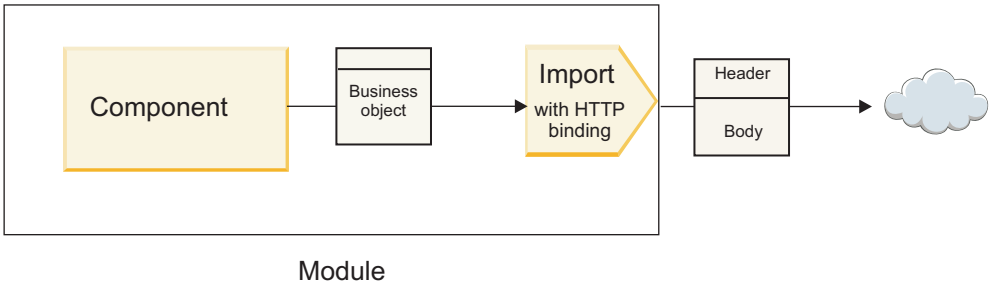


Figure 47. An import with HTTP binding

List of bindings

You use Integration Designer to generate a binding for an import or export and to configure the binding. The types of bindings that are available are described in the following list.

- SCA

  The SCA binding, which is the default, lets your service communicate with services in other SCA modules. You use an import with an SCA binding to access a service in another SCA module. You use an export with an SCA binding to offer a service to other SCA modules.
- Web service

  A Web service binding lets you access an external service using interoperable SOAP messages and qualities of service. You can also use Web service bindings to include attachments as part of the SOAP message.

  The Web service binding can use a transport protocol of either SOAP/HTTP (SOAP over HTTP) or SOAP/JMS (SOAP over JMS). Regardless of the transport (HTTP or JMS) used to convey the SOAP messages, Web service bindings always handle request/response interactions synchronously.
- HTTP

  The HTTP binding lets you access an external service using the HTTP protocol, where non-SOAP messages are used, or where direct HTTP access is required. This binding is used when you are working with Web services that are based on the HTTP model (that is, services that use well-known HTTP interface operations such as GET, PUT, DELETE, and so on).
- Enterprise JavaBeans (EJB)

  The EJB bindings let SCA components interact with services provided by Java EE business logic running on a Java EE server.
- EIS

  The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.
- JMS bindings

  Java Message Service (JMS), generic JMS, and WebSphere MQ JMS (MQ JMS) bindings are used for interactions with messaging systems, where asynchronous communication through message queues is critical for reliability.

An export with one of the JMS bindings watches a queue for the arrival of a message and asynchronously sends the response, if any, to the reply queue. An import with one of the JMS bindings builds and sends a message to a JMS queue and watches a queue for the arrival of the response, if any.

- JMS

  The JMS binding lets you access the WebSphere-embedded JMS provider.
- Generic JMS

  The generic JMS binding lets you access a non-IBM vendor messaging system.
- MQ JMS

  The MQ JMS binding lets you access the JMS subset of a WebSphere MQ messaging system. You would use this binding when the JMS subset of functions is sufficient for your application.

- MQ

  The WebSphere MQ binding lets you communicate with MQ native applications, bringing them into the service oriented architecture framework and providing access to MQ-specific header information. You would use this binding when you need to use MQ native functions.

Export and import binding overview

An export lets you make services in an integration module available to external clients, and an import makes it possible for your SCA components in an integration module to call external services. The binding associated with the export or import specifies the relationship between protocol messages and business objects. It also specifies the way that operations and faults are selected.

Flow of information through an export

An export receives a request, which is intended for the component to which the export is wired, over a specific transport determined by the associated binding (for example, HTTP).

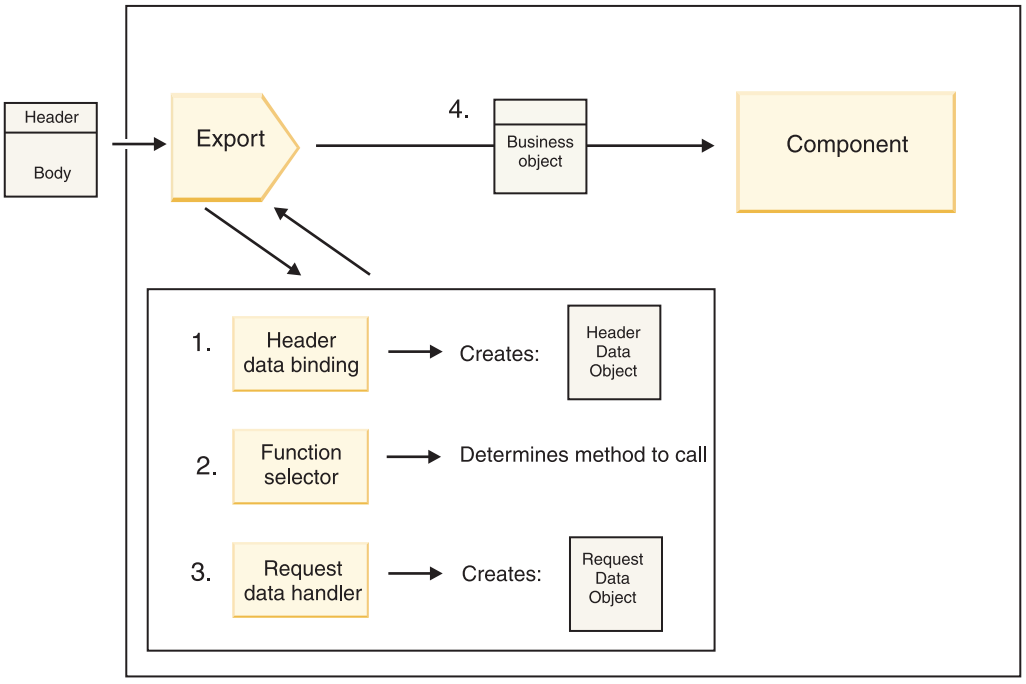


Figure 48. Flow of a request through the export to a component

When the export receives the request, the following sequence of events occurs:

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.

2. The function selector determines the native method name from the protocol message. The native method name is mapped by the export configuration to the name of an operation on the interface of the export.
3. The request data handler or data binding on the method transforms the request to a request business object.
4. The export invokes the component method with the request business object.
   - The HTTP export binding, the Web service export binding, and the EJB export binding invoke the SCA component synchronously.
   - The JMS, Generic JMS, MQ JMS, and WebSphere MQ export bindings invoke the SCA component asynchronously.

Note that an export can propagate the headers and user properties it receives over the protocol, if context propagation is enabled. Components that are wired to the export can then access these headers and user properties. See the "Propagation" topic in the WebSphere Integration Developer information center for more information.

If this is a two-way operation, the component returns a response.

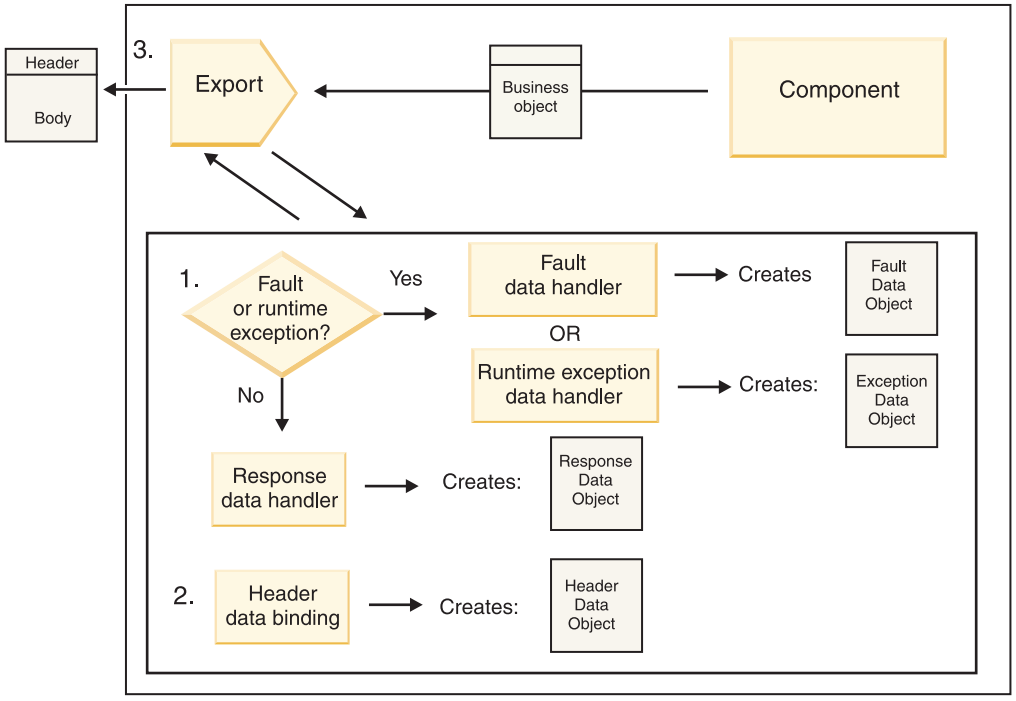


Figure 49. Flow of a response back through the export

The following sequence of steps occurs:

1. If a normal response message is received by the export binding, the response data handler or data binding on the method transforms the business object to a response.

   If the response is a fault, the fault data handler or data binding on the method transforms the fault to a fault response.

   For HTTP export bindings only, if the response is a runtime exception, the runtime exception data handler, if configured, is called.
2. For WebSphere MQ bindings only, the header data binding transforms the header data objects into protocol headers.
3. The export sends the service response over the transport.

Flow of information through an import

Components send requests to services outside the module using an import. The request is sent, over a specific transport determined by the associated binding.

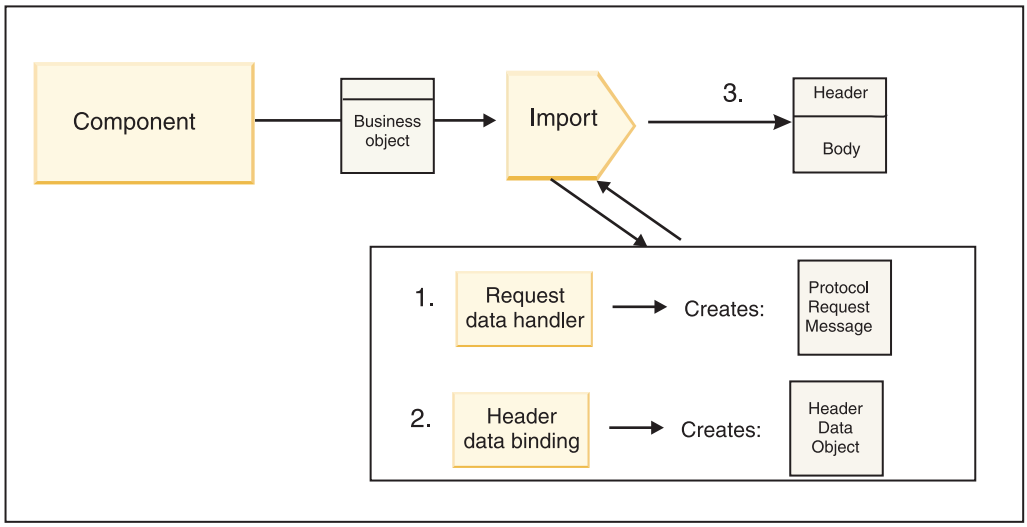


Figure 50. Flow from a component through the import to a service

The component invokes the import with a request business object.

Note:

- The HTTP import binding, the Web service import binding, and the EJB import binding should be invoked synchronously by the calling component.
- The JMS, Generic JMS, MQ JMS, and WebSphere MQ import binding should be invoked asynchronously.

After the component invokes the import, the following sequence of events occurs:

1. The request data handler or data binding on the method transforms the request business object into a protocol request message.
2. For WebSphere MQ bindings only, the header data binding on the method sets the header business object in the protocol header.
3. The import invokes the service with the service request over the transport.

If this is a two-way operation, the service returns a response, and the following sequence of steps occurs:

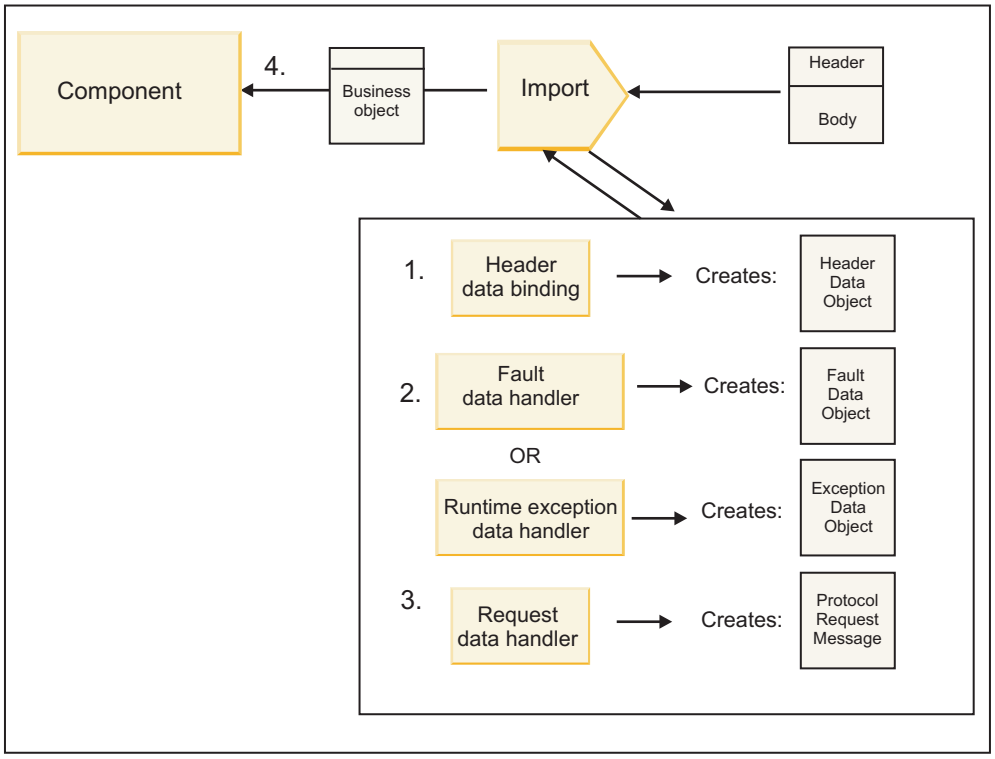


Figure 51. Flow of a response back through the import

1. For WebSphere MQ bindings only, the header data binding transforms the protocol header into a header data object.
2. A determination is made about whether the response is a fault.
   - If the response is a fault, the fault selector inspects the fault to determine which WSDL fault it maps to. The fault data handler on the method then transforms the fault to a fault response.
   - If the response is a runtime exception, the runtime exception data handler, if configured, is called.
3. The response data handler or binding on the method transforms the response to a response business object.
4. The import returns the response business object to the component.

Export and import binding configuration

One of the key aspects of export and import bindings is data format transformation, which indicates how data is mapped (deserialized) from a native wire format to a business object or how it is mapped (serialized) from a business object to a native wire format. For bindings associated with exports, you can also specify a function selector to indicate which operation should be performed on the data. For bindings associated with exports or imports, you can indicate how faults that occur during processing should be handled.

In addition, you specify transport-specific information on bindings. For example, for an HTTP binding, you specify the endpoint URL. For the HTTP binding, the transport-specific information is described in the "Generating an HTTP import binding" and "Generating an HTTP export binding" topics. You can also find information about other bindings in the information center.

Data format transformation in imports and exports

When an export or import binding is configured in IBM Integration Designer, one of the configuration properties that you specify is the data format used by the binding.

- For export bindings, in which a client application sends requests to and receives responses from an SCA component, you indicate the format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the native data to a business object (which is used by the SCA component) and conversely to transform the business object to native data (which is the response to the client application).
- For import bindings, in which an SCA component sends requests to and receives responses from a service outside the module, you indicate the data format of the native data. Depending on the format, the system selects the appropriate data handler or data binding to transform the business object to native data and vice versa.

IBM Business Process Manager provides a set of predefined data formats and corresponding data handlers or data bindings that support the formats. You can also create your own custom data handlers and register the data format for those data handlers. For more information, see the "Developing data handlers" topic in the IBM Integration Designer information center.

- *Data handlers* are protocol-neutral and transform data from one format to another. In IBM Business Process Manager, data handlers typically transform native data (such as XML, CSV, and COBOL) to a business object and a business object to native data. Because they are protocol-neutral, you can reuse the same data handler with a variety of export and import bindings. For example, you can use the same XML data handler with an HTTP export or import binding or with a JMS export or import binding.
- *Data bindings* also transform native data to a business object (and vice versa), but they are protocol-specific. For example, an HTTP data binding can be used with an HTTP export or import binding only. Unlike data handlers, an HTTP data binding cannot be reused with an MQ export or import binding.

  Note: Three HTTP data bindings (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of IBM Business Process Manager Version 7.0. Use data handlers whenever possible.

As noted earlier, you can create custom data handlers, if necessary. You can also create custom data bindings; however, it is recommended that you create custom data handlers because they can be used across multiple bindings.

Data handlers:

Data handlers are configured against export and import bindings to transform data from one format to another in a protocol-neutral fashion. Several data handlers are provided as part of the product, but you can also create your own data handler, if necessary. You can associate a data handler with an export or import binding at one of two levels: you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

Predefined data handlers

You use IBM Integration Designer to specify the data handler that you want to use.

The data handlers that are predefined for your use are listed in the following table, which also describes how each data handler transforms inbound and outbound data.

Note: Except where noted, these data handlers can be used with JMS, Generic JMS, MQ JMS, WebSphere MQ, and HTTP bindings.
See the "Data handlers" topic in the Integration Designer information center for more detailed information.

Table 44. Predefined data handlers

Data handler	Native data to business object	Business object to native data
ATOM	Parses ATOM feeds into an ATOM feed business object.	Serializes an ATOM feed business object to ATOM feeds.
Delimited	Parses delimited data into a business object.	Serializes a business object to delimited data, including CSV.
Fixed Width	Parses fixed-width data into a business object.	Serializes a business object to fixed-width data.
Handled by WTX	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.	Delegates data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is derived by the data handler.
Handled by WTX Invoker	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.	Delegates the data format transformation to the WebSphere Transformation Extender (WTX). The WTX map name is supplied by the user.
JAXB	Serializes Java beans to a business object using the mapping rules defined by the Java Architecture for XML Binding (JAXB) specification.	Deserializes a business object to Java beans using the mapping rules defined by the JAXB specification.
JAXWS **Note:** The JAXWS data handler can be used only with the EJB binding.	Used by an EJB binding to transform a response Java object or exception Java object to a response business object using the mapping rules defined by the Java API for XML Web Services (JAX-WS) specification.	Used by an EJB binding to transform a business object to the outgoing Java method parameters using the mapping rules defined by the JAX-WS specification.
JSON	Parses JSON data into a business object.	Serializes a business object to JSON data.
Native body	Parses the native bytes, text, map, stream, or object into one of five base business objects (text, bytes, map, stream, or object).	Transforms the five base business objects into byte, text, map, stream, or object.
SOAP	Parses the SOAP message (and the header) into a business object.	Serializes a business object to a SOAP message.
XML	Parses XML data into a business object.	Serializes a business object to XML data.
UTF8XMLDataHandler	Parses UTF-8 encoded XML data into a business object.	Serializes a business object into UTF-8 encoded XML data when sending a message.

Creating a data handler

Detailed information about creating a data handler can be found in the “Developing data handlers” topic in the Integration Designer information center.

Data bindings:

Data bindings are configured against export and import bindings to transform data from one format to another. Data bindings are specific to a protocol. Several data bindings are provided as part of the product, but you can also create your own data binding, if necessary. You can associate a data binding

with an export or import binding at one of two levels–you can associate it with all operations in the interface of the export or import, or you can associate it with a specific operation for the request or response.

You use IBM Integration Designer to specify which data binding you want to use or to create your own data binding. A discussion of creating data bindings can be found in the "Overview of JMS, MQ JMS and generic JMS bindings" section of the IBM Integration Designer information center.

JMS bindings

The following table lists the data bindings that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

The table also includes a description of the tasks that the data bindings perform.

Table 45. Predefined data bindings for JMS bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object into a business object (which is mapped as the input or output type in the WSDL).	Serializes a business object to the Java serialized object in the JMS object message.
Wrapped bytes	Extracts the bytes from the incoming JMS bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from the JMSBytesBody business object and wraps them into the outgoing JMS bytes message
Wrapped map entry	Extracts the name, value, and type information for every entry in the incoming JMS map message and creates a list of MapEntry business objects. It then wraps the list into the JMSMapBody business object	Extracts the name, value, and type information from the MapEntry list in the JMSMapBody business object and creates the corresponding entries in the outgoing JMS map message.
Wrapped object	Extracts the object from the incoming JMS object message and wraps it into the JMSObjectBody business object.	Extracts the object from the JMSObjectBody business object and wraps it into the outgoing JMS object message.
Wrapped text	Extracts the text from the incoming JMS text message and wraps it into the JMSTextBody business object.	Extracts the text from the JMSTextBody business object and wraps it into the outgoing JMS text message.

WebSphere MQ bindings

The following table lists the data bindings that can be used with WebSphere MQ and describes the tasks that the data bindings perform.

Table 46. Predefined data bindings for WebSphere MQ bindings

Data binding	Native data to business object	Business object to native data
Serialized Java object	Transforms the Java serialized object from the incoming message into a business object (which is mapped as the input or output type in the WSDL).	Transforms a business object to the Java serialized object in the outgoing message

Table 46. Predefined data bindings for WebSphere MQ bindings (continued)

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the unstructured MQ bytes message and wraps them into the JMSBytesBody business object.	Extracts the bytes from a JMSBytesBody business object and wraps the bytes into the outgoing unstructured MQ bytes message.
Wrapped text	Extracts the text from an unstructured MQ text message and wraps it into a JMSTextBody business object.	Extracts text from a JMSTextBody business object and wraps it in an unstructured MQ text message.
Wrapped stream entry	Extracts the name and type information for every entry in the incoming JMS stream message and creates a list of the StreamEntry business objects. It then wraps the list into the JMSStreamBody business object.	Extracts the name and type information from the StreamEntry list in the JMSStreamBody business object and creates corresponding entries in the outgoing JMSStreamMessage.

In addition to the data bindings listed in Table 20 on page 62, WebSphere MQ also uses header data bindings. See the IBM Integration Designer information center for details.

HTTP bindings

The following table lists the data bindings that can be used with HTTP and describes the tasks that the data bindings perform.

Table 47. Predefined data bindings for HTTP bindings

Data binding	Native data to business object	Business object to native data
Wrapped bytes	Extracts the bytes from the body of the incoming HTTP message and wraps them into the HTTPBytes business object.	Extracts the bytes from the HTTPBytes business object and adds them to the body of the outgoing HTTP message.
Wrapped text	Extracts the text from the body of the incoming HTTP message and wraps it into the HTTPText business object.	Extracts the text from the HTTPText business object and adds it to the body of the outgoing HTTP message.

Function selectors in export bindings

A function selector is used to indicate which operation should be performed on the data for a request message. Function selectors are configured as part of an export binding.

Consider an SCA export that exposes an interface. The interface contains two operations–Create and Update. The export has a JMS binding that reads from a queue.

When a message arrives on the queue, the export is passed the associated data, but which operation from the export's interface should be invoked on the wired component? The operation is determined by the function selector and the export binding configuration.

The function selector returns the native function name (the function name in the client system that sent the message). The native function name is then mapped to the operation or function name on the interface associated with the export. For example, in the following figure, the function selector returns the native function name (CRT) from the incoming message, the native function name is mapped to the Create operation, and the business object is sent to the SCA component with the Create operation.

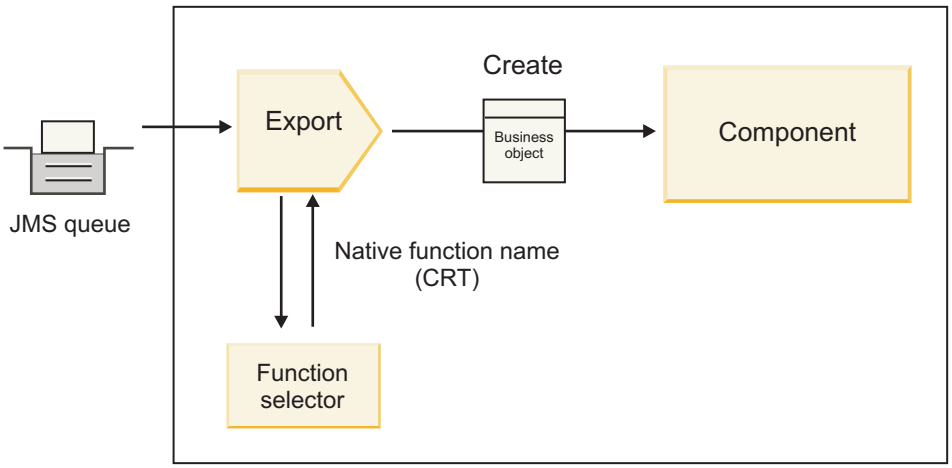


Figure 52. The function selector

If the interface has only one operation, there is no need to specify a function selector.

Several prepackaged function selectors are available and are listed in the sections that follow.

JMS bindings

The following table lists the function selectors that can be used with:

- JMS bindings
- Generic JMS bindings
- WebSphere MQ JMS bindings

Table 48. Predefined function selectors for JMS bindings

Function selector	Description
JMS function selector for simple JMS data bindings	Uses the JMSType property of the message to select the operation name.
JMS header property function selector	Returns the value of the JMS String Property, TargetFunctionName, from the header.
JMS service gateway function selector	Determines if the request is a one-way or two-way operation by examining the JMSReplyTo property set by the client.

WebSphere MQ bindings

The following table lists the function selectors that can be used with WebSphere MQ bindings.

Table 49. Predefined function selectors for WebSphere MQ bindings

Function selector	Description
MQ handleMessage function selector	Returns handleMessage as a value, which is mapped using the export method bindings to the name of an operation on the interface.
MQ uses JMS default function selector	Reads the native operation from the TargetFunctionName property of the folder of an MQRFH2 header.
MQ uses message body format as native function	Finds the Format field of the last header and returns that field as a String.

Table 49. Predefined function selectors for WebSphere MQ bindings (continued)

Function selector	Description
MQ type function selector	Creates a method in your export binding by retrieving a URL containing the Msd, Set, Type and Format properties found in the MQRFH2 header.
MQ service gateway function selector	Uses the MsgType property in the MQMD header to determine the operation name.

HTTP bindings

The following table lists the function selectors that can be used with HTTP bindings.

Table 50. Predefined function selectors for HTTP bindings

Function selector	Description
HTTP function selector based on the TargetFunctionName header	Uses the TargetFunctionName HTTP header property from the client to determine which operation to invoke at runtime from the export.
HTTP function selector based on the URL and HTTP method	Uses the relative path from the URL appended with the HTTP method from the client to determine the native operation defined on the export.
HTTP service gateway function selector based on URL with an operation name	Determines the method to invoke based on the URL if "operationMode = oneway" has been appended to the request URL.

Note: You can also create your own function selector, using IBM Integration Designer. Information about creating a function selector is provided in the IBM Integration Designer information center. For example, a description of creating a function selector for WebSphere MQ bindings can be found in “Overview of the MQ function selectors”.

Fault handling

You can configure your import and export bindings to handle faults (for example, business exceptions) that occur during processing by specifying fault data handlers. You can set up a fault data handler at three levels–you can associate a fault data handler with a fault, with an operation, or for all operations with a binding.

A fault data handler processes fault data and transforms it into the correct format to be sent by the export or import binding.

- For an export binding, the fault data handler transforms the exception business object sent from the component to a response message that can be used by the client application.
- For an import binding, the fault data handler transforms the fault data or response message sent from a service into an exception business object that can be used by the SCA component.

For import bindings, the binding calls the fault selector, which determines whether the response message is a normal response, a business fault, or a runtime exception.

You can specify a fault data handler for a particular fault, for an operation, and for all operations with a binding.

- If the fault data handler is set at all three levels, the data handler associated with a particular fault is called.
- If fault data handlers are set at the operation and binding levels, the data handler associated with the operation is called.

Two editors are used in IBM Integration Designer to specify fault handling. The interface editor is used to indicate whether there will be a fault on an operation. After a binding is generated with this interface, the editor in the properties view lets you configure how the fault will be handled. For more information, see the “Fault selectors” topic in the IBM Integration Designer information center.

How faults are handled in export bindings:

When a fault occurs during the processing of the request from a client application, the export binding can return the fault information to the client. You configure the export binding to specify how the fault should be processed and returned to the client.

You configure the export binding using IBM Integration Designer.

During request processing, a client invokes an export with a request, and the export invokes the SCA component. During the processing of the request, the SCA component can either return a business response or can throw a service business exception or a service runtime exception. When this occurs, the export binding transforms the exception into a fault message and sends it to the client, as shown in the following figure and described in the sections that follow.

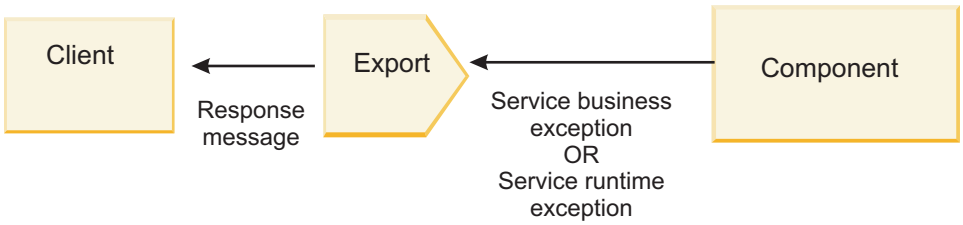


Figure 53. How fault information is sent from the component through the export binding to the client

You can create a custom data handler or data binding to handle faults.

Business faults

Business faults are business errors or exceptions that occur during processing.

Consider the following interface, which has a createCustomer operation on it. This operation has two business faults defined: CustomerAlreadyExists and MissingCustomerId.

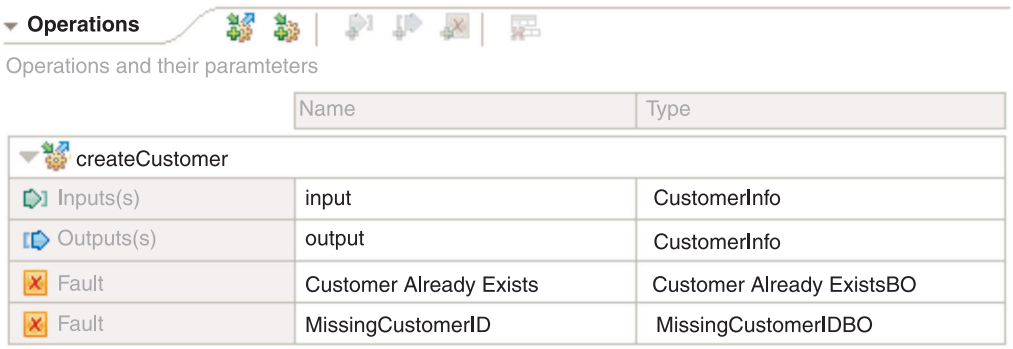


Figure 54. Interface with two faults

In this example, if a client sends a request to create a customer (to this SCA component) and that customer already exists, the component throws a CustomerAlreadyExists fault to the export. The export needs to propagate this business fault back to the calling client. To do so, it uses the fault data handler that is set up on the export binding.

When a business fault is received by the export binding, the following processing occurs:

1. The binding determines which fault data handler to invoke for handling the fault. If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.
2. The binding calls the fault data handler with the data object from the service business exception.
3. The fault data handler transforms the fault data object to a response message and returns it to the export binding.
4. The export returns the response message to the client.

If the service business exception contains the fault name, the data handler that is set up on the fault is called. If the service business exception does not contain the name of the fault, the fault name is derived by matching the fault types.

Runtime exceptions

A runtime exception is an exception that occurs in the SCA application during the processing of a request that does not correspond to a business fault. Unlike business faults, runtime exceptions are not defined on the interface.

In certain scenarios, you might want to propagate these runtime exceptions to the client application so that the client application can take the appropriate action.

For example, if a client sends a request (to the SCA component) to create a customer and an authorization error occurs during processing of the request, the component throws a runtime exception. This runtime exception has to be propagated back to the calling client so it can take the appropriate action regarding the authorization. This is achieved by the runtime exception data handler configured on the export binding.

Note: You can configure a runtime exception data handler only on HTTP bindings.

The processing of a runtime exception is similar to the processing of a business fault. If a runtime exception data handler was set up, the following processing occurs:

1. The export binding calls the appropriate data handler with the service runtime exception.
2. The data handler transforms the fault data object to a response message and returns it to the export binding.
3. The export returns the response message to the client.

Fault handling and runtime exception handling are optional. If you do not want to propagate faults or runtime exceptions to the calling client, do not configure the fault data handler or runtime exception data handler.

How faults are handled in import bindings:

A component uses an import to send a request to a service outside the module. When a fault occurs during the processing of the request, the service returns the fault to the import binding. You can configure the import binding to specify how the fault should be processed and returned to the component.

You configure the import binding using IBM Integration Designer. You can specify a fault data handler (or data binding), and you also specify a fault selector.

Fault data handlers

The service that processes the request sends, to the import binding, fault information in the form of an exception or a response message that contains the fault data.

The import binding transforms the service exception or response message into a service business exception or service runtime exception, as shown in the following figure and described in the sections that follow.

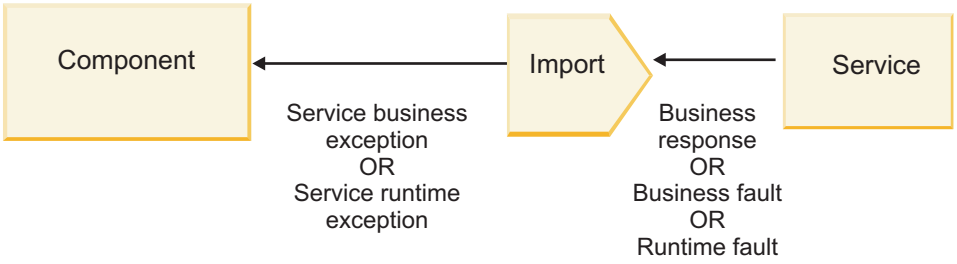


Figure 55. How fault information is sent from the service through the import to the component

You can create a custom data handler or data binding to handle faults.

Fault selectors

When you configure an import binding, you can specify a fault selector. The fault selector determines whether the import response is an actual response, a business exception, or a runtime fault. It also determines, from the response body or header, the native fault name, which is mapped by the binding configuration to the name of a fault in the associated interface.

Two types of prepackaged fault selectors are available for use with JMS, MQ JMS, Generic JMS, WebSphere MQ, and HTTP imports:

Table 51. Prepackaged fault selectors

Fault selector type	Description
Header-based	Determines whether a response message is a business fault, a runtime exception, or a normal message based on the headers in the incoming response message.
SOAP	Determines whether the response SOAP message is a normal response, business fault, or runtime exception.

The following shows examples of header-based fault selectors and the SOAP fault selector.

- Header-based fault selector

  If an application wants to indicate that the incoming message is a business fault, there must be two headers in the incoming message for business faults, which is shown as follows:

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  If an application wants to indicate that the incoming response message is a runtime exception, then there must be one header in the incoming message, which is shown as follows:

  ```
  Header name = FaultType, Header value = Runtime
  ```

- SOAP fault selector

  A business fault can be sent as part of the SOAP message with the following custom SOAP header. "CustomerAlreadyExists" is the name of the fault in this case.

```
<ibmSoap:BusinessFaultName
xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
<ibmSoap:BusinessFaultName>
```

The fault selector is optional. If you do not specify a fault selector, the import binding cannot determine the type of response. The binding therefore treats it as a business response and calls the response data handler or data binding.

You can create a custom fault selector. The steps for creating a custom fault selector are provided in the "Developing a custom fault selector" topic of the IBM Integration Designer information center.

Business faults

A business fault can occur when there is an error in the processing of a request. For example, if you send a request to create a customer and that customer already exists, the service sends a business exception to the import binding.

When a business exception is received by the binding, the processing steps depend on whether a fault selector has been set up for the binding.

- If no fault selector was set up, the binding calls the response data handler or data binding.
- If a fault selector was set up, the following processing occurs:
  1. The import binding calls the fault selector to determine whether the response is business fault, business response, or runtime fault.
  2. If the response is a business fault, the import binding calls the fault selector to provide the native fault name.
  3. The import binding determines the WSDL fault corresponding to the native fault name returned by the fault selector.
  4. The import binding determines the fault data handler that is configured for this WSDL fault.
  5. The import binding calls this fault data handler with the fault data.
  6. The fault data handler transforms the fault data to a data object and returns it to the import binding.
  7. The import binding constructs a service business exception object with the data object and the fault name.
  8. The import returns the service business exception object to the component.

Runtime exceptions

A runtime exception can occur when there is a problem in communicating with the service. The processing of a runtime exception is similar to the processing of a business exception. If a fault selector was set up, the following processing occurs:

1. The import binding calls the appropriate runtime exception data handler with the exception data.
2. The runtime exception data handler transforms the exception data to a service runtime exception object and returns it to the import binding.
3. The import returns the service runtime exception object to the component.

Interoperability between SCA modules and Open SCA services

The IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) provides a simple, yet powerful programming model for constructing applications based on the Open SCA specifications. The SCA modules of IBM Business Process Manager use import and export bindings to interoperate with Open SCA services developed in a Rational Application Developer environment and hosted by the WebSphere Application Server Feature Pack for Service Component Architecture.

An SCA application invokes an Open SCA application by way of an import binding. An SCA application receives a call from an Open SCA application by way of an export binding. A list of supported bindings is shown in “Invoking services over interoperable bindings” on page 71.

Invoking Open SCA services from SCA modules

SCA applications developed with IBM Integration Designer can invoke Open SCA applications developed in a Rational Application Developer environment. This section provides an example of invoking an Open SCA service from an SCA module using an SCA import binding.

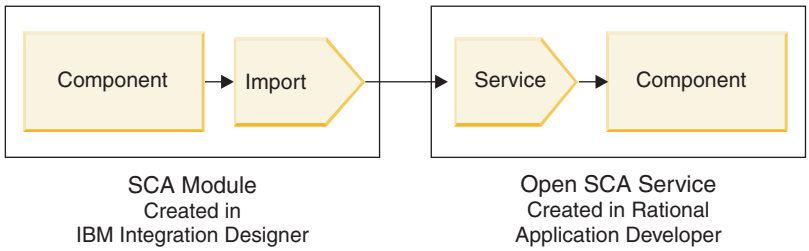


Figure 56. Component in SCA module invoking Open SCA service

No special configuration is required to invoke an Open SCA service.

To connect to an Open SCA service by way of an SCA import binding, you provide the component name and service name of the Open SCA service in the import binding.

1. To obtain the name of the target component and service from the Open SCA composite, perform the following steps:
   a. Ensure that the **Properties** tab is open by clicking **Window** > **Show View** > **Properties**.
   b. Open the composite editor by double-clicking the composite diagram that contains the component and service. For example, for a component named **customer**, the composite diagram is **customer.composite_diagram**.
   c. Click the target component.
   d. In the **Name** field of the **Properties** tab, note the name of the target component.
   e. Click the service icon associated with the component.
   f. In the **Name** field of the **Properties** tab, note the name of the service.
2. To configure the IBM Business Process Manager import to connect it to the Open SCA service, perform the following steps:
   a. In IBM Integration Designer, navigate to the **Properties** tab of the SCA import that you want to connect to the Open SCA service.
   b. In the **Module name** field, enter the component name from step 1d on page 70.
   c. In the **Export name** field, enter the service name from step 1f on page 70.
   d. Save your work by pressing Ctrl+S.

Invoking SCA modules from Open SCA services

Open SCA applications developed in a Rational Application Developer environment can invoke SCA applications developed with IBM Integration Designer. This section provides an example of invoking an SCA module (by way of an SCA export binding) from an Open SCA service.

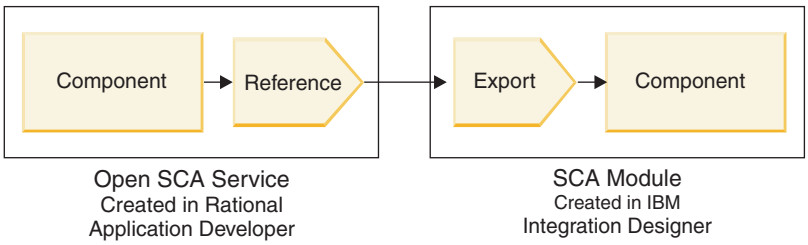


Figure 57. Open SCA service invoking component in SCA module

To connect to an SCA component by way of an Open SCA reference binding, you provide the module name and export name.

1. To obtain the name of the target module and export, perform the following steps:
   a. In IBM Integration Designer, open the module in the assembly editor by double-clicking the module.
   b. Click the export.
   c. In the **Name** field of the **Properties** tab, note the name of the export.
2. Configure the Open SCA reference that you want to connect to the IBM Business Process Manager module and export:
   a. In Rational Application Developer, open the composite editor by double-clicking the composite diagram that contains the component and service.
   b. Click the reference icon of the component reference to display the reference properties in the **Properties** tab.
   c. Click the **Binding** tab on the left side of the page.
   d. Click **Bindings** and then click **Add**.
   e. Select the **SCA** binding.
   f. In the **Uri** field, enter the IBM Business Process Manager module name, followed by a slash ("/"), followed by the export name (which you determined in step 1c on page 71).
   g. Click **OK**.
   h. Save your work by pressing Ctrl+S.

Invoking services over interoperable bindings

The following bindings are supported for interoperability with an Open SCA service.

- SCA binding

  In IBM Business Process Manager, when an SCA module invokes an Open SCA service by way of an SCA import binding, the following invocation styles are supported:

  – Asynchronous (one-way)
  – Synchronous (request/response)

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  Note that the SCA binding supports transaction and security context propagation.
- Web service (JAX-WS) binding with either the SOAP1.1/HTTP or SOAP1.2/HTTP protocol

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  In addition, the following qualities of service are supported:

  – Web Services Atomic Transaction
  – Web Services Security

- EJB binding

  A Java interface is used to define the interaction between an SCA module and an Open SCA service when the EJB binding is used.

  Note that the EJB binding supports transaction and security context propagation.
- JMS bindings

  The SCA import interface and the Open SCA service interface must use a Web services interoperability (WS-I) compliant WSDL interface.

  The following JMS providers are supported:
  - WebSphere Platform Messaging (JMS Binding)
  - WebSphere MQ (MQ JMS Binding)

Note: Business Graphs are not interoperable across any SCA bindings and, therefore, are not supported in interfaces used to interoperate with the WebSphere Application Server Feature Pack for Service Component Architecture.

Binding types

You use protocol-specific *bindings* with imports and exports to specify the means of transporting data into or out of a module.

Selecting appropriate bindings

When you are creating an application, you need to know how to select the binding that is most appropriate to the needs of your application.

The bindings available in IBM Integration Designer provide a range of choices. In this list, you can determine which type of binding might be most suitable for the needs of your application.

Consider a *Service Component Architecture (SCA)* binding when these factors are applicable:
- All services are contained in modules; that is, there are no external services.
- You want to separate function into different SCA modules that interact directly with each other.
- The modules are tightly coupled.

Consider a *Web Service* 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.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.
- The protocol of the external service you are accessing or the service you want to provide is SOAP/HTTP or SOAP/JMS.

Consider an *HTTP* binding when these factors are applicable:
- You need to access an external service over the Internet or provide a service over the Internet, and you are working with other web services such as GET, PUT, and DELETE.
- The services are loosely coupled.
- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise JavaBeans (EJB)* binding when these factors are applicable:
- The binding is for an imported service that is itself an EJB or that needs to be accessed by EJB clients.
- The imported service is loosely coupled.
- Stateful EJB interactions are not required.

- Synchronous communication is preferred; that is, a request from one service can wait for a response from another.

Consider an *Enterprise Information Systems (EIS)* binding when these factors are applicable:
- You need to access a service on an EIS system using a resource adapter.
- Synchronous data transmission is preferred over asynchronous.

Consider a *Java Message Service (JMS)* binding when these factors are applicable:

Important: There are several types of JMS bindings. If you expect to exchange SOAP messages using JMS, consider the Web service binding with the SOAP/JMS protocol. See "Web service bindings" on page 74.
- You need to access a messaging system.
- The services are loosely coupled.
- Asynchronous data transmission is preferred over synchronous.

Consider a *Generic Java Message Service (JMS)* binding when these factors are applicable:
- You need to access a non-IBM vendor messaging system.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *Message Queue (MQ)* binding when these factors are applicable:
- You need to access a WebSphere MQ messaging system and need to use the MQ native functions.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

Consider an *MQ JMS* binding when these factors are applicable:
- You need to access a WebSphere MQ messaging system but can do so within a JMS context; that is, the JMS subset of functions is sufficient for your application.
- The services are loosely coupled.
- Reliability is more important than performance; that is, asynchronous data transmission is preferred over synchronous.

SCA bindings

A Service Component Architecture (SCA) binding lets a service communicate with other services in other modules. An import with an SCA binding lets you access a service in another SCA module. An export with an SCA binding lets you offer a service to other modules.

You use IBM Integration Designer to generate and configure SCA bindings on imports and exports in SCA modules.

If modules are running on the same server or are deployed in the same cluster, an SCA binding is the easiest and fastest binding to use.

After the module that contains the SCA binding is deployed to the server, you can use the administrative console to view information about the binding or, in the case of an import binding, to change selected properties of the binding.

Web service bindings

A Web service binding is the means of transmitting messages from a Service Component Architecture (SCA) component to a Web service (and vice versa).

Web service bindings overview:

A Web service import binding allows you to call an external Web service from your Service Component Architecture (SCA) components. A Web service export binding allows you to expose your SCA components to clients as Web services.

With a Web service binding, you access external services using interoperable SOAP messages and qualities of service (QoS).

You use Integration Designer to generate and configure Web service bindings on imports and exports in SCA modules. The following types of Web service bindings are available:

- SOAP1.2/HTTP and SOAP1.1/HTTP

  These bindings are based on Java API for XML Web Services (JAX-WS), a Java programming API for creating Web services.

  – Use SOAP1.2/HTTP if your Web service conforms to the SOAP 1.2 specification.
  – Use SOAP1.1/HTTP if your Web service conforms to the SOAP 1.1 specification.

  Important: When you deploy an application with a Web service (JAX-WS) binding, the target server must not have the **Start components as needed** option selected. See "Checking the server configuration" on page 82 for details.

  When you select one of these bindings, you can send attachments with your SOAP messages.

  The Web service bindings work with standard SOAP messages. Using one of the Web service JAX-WS bindings, however, you can customize the way that SOAP messages are parsed or written. For example, you can handle nonstandard elements in SOAP messages or apply additional processing to the SOAP message. When you configure the binding, you specify a custom data handler that performs this processing on the SOAP message.

  You can use policy sets with a Web service (JAX-WS) binding. A policy set is a collection of policy types, each of which provides a quality of service (QoS). For example, the WSAddressing policy set provides a transport-neutral way to uniformly address Web services and messages. You use Integration Designer to select the policy set for the binding.

  Note: If you want to use a Security Assertion Markup Language (SAML) policy set, you must perform some additional configuration, as described in "Importing SAML policy sets" on page 80.

- SOAP1.1/HTTP

  Use this binding if you want to create Web services that use a SOAP-encoded message based on Java API for XML-based RPC (JAX-RPC).

- SOAP1.1/JMS

  Use this binding to send or receive SOAP messages using a Java Message Service (JMS) destination.

Regardless of the transport (HTTP or JMS) that is used to convey the SOAP message, Web service bindings always handle request/response interactions synchronously. The thread making the invocation on the service provider is blocked until a response is received from the provider. See "Synchronous invocation" for more information about this invocation style.

Important: The following combinations of Web service bindings cannot be used on exports in the same module. If you need to expose components using more than one of these export bindings, you need to have each in a separate module and then connect those modules to your components using the SCA binding:

- SOAP 1.1/JMS and SOAP 1.1/HTTP using JAX-RPC
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.1/HTTP using JAX-WS
- SOAP 1.1/HTTP using JAX-RPC and SOAP 1.2/HTTP using JAX-WS

After the SCA module that contains the Web service binding is deployed to the server, you can use the administrative console to view information about the binding or to change selected properties of the binding.

Note: Web services allow applications to interoperate by using standard descriptions of services and standard formats for the messages they exchange. For example, the Web service import and export bindings can interoperate with services that are implemented using Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET. When interoperating with such services, you must ensure that:

- The Web Services Description Language (WSDL) file that is used to access a Web service export includes a non-empty SOAP action value for each operation in the interface.
- The Web service client sets either the SOAPAction header or the wsa:Action header when sending messages to a Web service export.

SOAP header propagation:

When handling SOAP messages, you might need to access information from certain SOAP headers in messages that are received, ensure that messages with SOAP headers are sent with specific values, or allow SOAP headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want SOAP headers to be propagated.

- When requests are received at an export or responses received at an import, the SOAP header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When requests are sent from an export or responses sent from an import, SOAP headers can be included in those messages.

The form and presence of the propagated SOAP headers might be affected by policy sets configured on the import or export, as explained in Table 26 on page 76.

To configure the propagation of SOAP headers for an import or export, you select (from the Properties view of Integration Designer) the **Propagate Protocol Header** tab and select the options you require.

WS-Addressing header

The WS-Addressing header can be propagated by the Web service (JAX-WS) binding.

When you propagate the WS-Addressing header, be aware of the following information:

- If you enable propagation for the WS-Addressing header, the header will be propagated into the module in the following circumstances:
  - When requests are received at an export
  - When responses are received at an import
- The WS-Addressing header is not propagated into outbound messages from IBM Business Process Manager (that is, the header is not propagated when requests are sent from an import or when responses are sent from the export).

WS-Security header

The WS-Security header can be propagated by both the Web service (JAX-WS) binding and the Web service (JAX-RPC) binding.

The Web services WS-Security specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

When you propagate the WS-Security header, be aware of the following information:

- If you enable propagation for the WS-Security header, the header will be propagated across the module in the following circumstances:
  - When requests are received at an export
  - When requests are sent from an import
  - When responses are received at an import
- The header will *not*, by default, be propagated when responses are sent from the export. However, if you set the JVM property **WSSECURITY.ECHO.ENABLED** to **true**, the header will be propagated when responses are sent from the export. In this case, if the WS-Security header on the request path is not modified, WS-Security headers might be automatically echoed from requests into responses.
- The exact form of the SOAP message sent from an import for a request or from an export for a response might not exactly match the SOAP message that was originally received. For this reason, any digital signature should be assumed to become invalid. If a digital signature is required in messages that are sent, it must be established using the appropriate security policy set, and WS-Security headers relating to digital signature in received messages should be removed within the module.

To propagate the WS-Security header, you must include the WS-Security schema with the application module. See "Including the WS-Security schema in an application module" on page 77 for the procedure to include the schema.

How headers are propagated

The way that headers are propagated depends on the security policy setting on the import or export binding, as shown in the Table 26 on page 76:

Table 52. How security headers are passed

	Export binding with no security policy	**Export binding with security policy**
Import binding with no security policy	Security headers are passed as-is through the module. They are not decrypted. The headers are sent outbound in the same form in which they were received. The digital signature might become invalid.	Security headers are decrypted and passed through the module with signature verification and authentication. The decrypted headers are sent outbound. The digital signature might become invalid.
Import binding with security policy	Security headers are passed as-is through the module. They are not decrypted. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.	Security headers are decrypted and passed through the module with signature verification and authentication. The headers should not be propagated to the import. Otherwise, an error occurs because of duplication.

Configure appropriate policy sets on the export and import bindings, because this isolates the service requester from changes to the configuration or QoS requirements of the service provider. Having

standard SOAP headers visible in a module can then be used to influence the processing (for example, logging and tracing) in the module. Propagating SOAP headers across a module from a received message to a sent message does mean that the isolation benefits of the module are reduced.

Standard headers, such as WS-Security headers, should not be propagated on a request to an import or response to an export when the import or export has an associated policy set that would normally result in the generation of those headers. Otherwise, an error will occur because of a duplication of the headers. Instead, the headers should be explicitly removed or the import or export binding should be configured to prevent propagation of protocol headers.

Accessing SOAP headers

When a message that contains SOAP headers is received from a Web service import or export, the headers are placed in the headers section of the service message object (SMO). You can access the header information, as described in "Accessing SOAP header information in the SMO".

Including the WS-Security schema in an application module

The following procedure outlines the steps for including the schema in the application module:

- If the computer on which Integration Designer is running has access to the Internet, perform the following steps:
  1. In the Business Integration perspective, select **Dependencies** for your project.
  2. Expand **Predefined Resources** and select either **WS-Security 1.0 schema files** or **WS-Security 1.1 schema files** to import the schema into your module.
  3. Clean and rebuild the project.
- If a computer on which Integration Designer is running does not have Internet access, you can download the schema to a second computer that does have Internet access. You can then copy it to the computer on which Integration Designer is running.
  1. From the computer that has Internet access, download the remote schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Remote WSDL** or **XSD file**.
     c. Import the following schemas:

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. Copy the schemas to the computer that does not have Internet access.
  3. From the computer that has no Internet access, import the schema:
     a. Click **File** > **Import** > **Business Integration** > **WSDL and XSD**.
     b. Select **Local WSDL** or **XSD file**.
  4. Change the schema locations for oasis-wss-wssecurity_secext-1.1.xsd:
     a. Open the schema at *workplace_location/module_name*/StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd.
     b. Change:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
        ```

        to:

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
        ```

     c. Change:

        ```
        <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
        schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
        ```

to:

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
```

5. Change the schema location for oasis-200401-wss-wssecurity-secext-1.0.xsd:
   a. Open the schema at *workplace_location/module_name*/StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd.
   b. Change:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
```

   to:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
```

6. Clean and rebuild the project.

Transport header propagation:

When handling SOAP messages, you might need to access information from certain transport headers in messages that are received, ensure that messages with transport headers are sent with specific values, or allow transport headers to pass across a module.

When you configure a Web service binding in Integration Designer, you can indicate that you want transport headers to be propagated.

- When requests are received at an export or responses are received at an import, the transport header information can be accessed, allowing logic in the module to be based on header values and allowing those headers to be modified.
- When responses are sent from an export or requests are sent from an import, transport headers can be included in those messages.

Specifying propagation of headers

To configure the propagation of transport headers for an import or export, perform the following steps:

1. From the Properties view of Integration Designer, select **Binding** > **Propagation**.
2. Set the transport header propagation option that you require.

Note: Transport header propagation is disabled by default and can be deployed only to a Version 7.0.0.3 (or later) runtime environment. Also note that, for Version 7.0.0.3, transport header propagation is limited to HTTP transport headers only.

If you enable propagation of transport headers, the headers will be propagated across a module from received messages and, if you do not explicitly remove the headers, the headers will be used in subsequent invocations in the same thread.

Note: Transport headers cannot be propagated when you are using the Web service (JAX-RPC) binding.

Accessing the header information

When transport header propagation is enabled for received messages, all transport headers (including customer-defined headers) are visible in the service message object (SMO). You can set the headers to different values or create new ones. Note, however, that there is no checking or validation of the values you set, and any improper or incorrect headers might cause Web service runtime problems.

Consider the following information about the setting of HTTP headers:

- Any changes to the headers that are reserved for the Web service engine will not be honored in the outbound message. For example, the HTTP version or method, Content-Type, Content-Length and SOAPAction headers are reserved for the Web service engine.
- If the header value is a number, the number (rather than the string) should be set directly. For example, use **Max-Forwards = 5** (rather than **Max-Forwards = Max-Forwards: 5**) and **Age = 300** (rather than **Age = Age: 300**).
- If the request message is less than 32 KB in size, the Web service engine removes the Transfer-Encoding header and instead sets the Content-Length header to the fixed size of the message.
- The Content-language is reset by WAS.channel.http on the response path.
- An invalid setting for Upgrade results in a 500 error.
- The following headers append the value reserved by the Web service engine to the customer settings:
  - User-Agent
  - Cache-Control
  - Pragma
  - Accept
  - Connection

You can access the header information in one of the following ways:

- Using a mediation primitive to access the SMO structures

  See the "Related Information" links to find information about using mediation primitives.
- Using the context service SPI

  The following sample code reads the HTTP transport headers from the context service:

```
HeadersType headerType = ContextService.INSTANCE.getHeaders();
HTTPHeaderType httpHeaderType = headerType.getHTTPHeader();
List HTTPHeader  httpHeaders = httpHeaderType.getHeader();
if(httpHeaders!=null){
 for(HTTPHeader httpHeader: httpHeaders){
   String httpHeadername = httpHeader.getName();
   String httpHeaderValue = httpHeader.getValue();
     }
}
List PropertyType  properties = headerType.getProperties();
if(properties!=null){
  for(PropertyType property: properties){
   String propertyName = property.getName();
   String propertyValue = property.getValue().toString();
     }
}
```

Troubleshooting

If you encounter problems when sending the revised headers, you can intercept the TCP/IP message by using tools such as the TCP/IP Monitor in Integration Designer. You access the TCP/IP Monitor by selecting **Run/Debug** > **TCP/IP Monitor** from the Preferences page.

You can also view the header values using the JAX-WS engine trace: **org.apache.axis2.*=all: com.ibm.ws.websvcs.*=all:**

Working with Web service (JAX-WS) bindings:

When you use Web service (JAX-WS) bindings with your applications, you can add a Security Assertion Markup Language (SAML) quality of service (QOS) to the binding. You must first use the administrative console to import the policy set. You can also use the administrative console to make sure that the server is properly configured for use with the Web service (JAX-WS) binding.

Importing SAML policy sets:

Security Assertion Markup Language (SAML) is an XML-based OASIS standard for exchanging user identity and security attributes information. When you configure a web service (JAX-WS) binding in Integration Designer, you can specify an SAML policy set. You first use the administrative console of IBM Business Process Manager to make the SAML policy sets available so that they can be imported into Integration Designer.

The SAML policy sets are typically located in the profile configuration directory:

profile_root/config/templates/PolicySets

Before you begin this procedure, verify that the following directories (which contain the policy sets) are located in the profile configuration directory:

- SAML11 Bearer WSHTTPS default
- SAML20 Bearer WSHTTPS default
- SAML11 Bearer WSSecurity default
- SAML20 Bearer WSSecurity default
- SAML11 HoK Public WSSecurity default
- SAML20 HoK Public WSSecurity default
- SAML11 HoK Symmetric WSSecurity default
- SAML20 HoK Symmetric WSSecurity default
- Username WSHTTPS default

If the directories are not in the profile configuration directory, copy them to that directory from the following location:

app_server_root/profileTemplates/default/documents/config/templates/PolicySets

You import the policy sets into the administrative console, select the ones you want to make available to Integration Designer, and then save a .zip file for each of those policy sets to a location that is accessible by Integration Designer.

1. Import the policy sets by following these steps:
   a. From the administrative console, click **Services** > **Policy Sets** > **Application policy sets**.
   b. Click **Import** > **From Default Repository**.
   c. Select the SAML default policy sets, and click **OK**.
2. Export the policy sets so that they can be used by Integration Designer:
   a. From the Application policy sets page, select the SAML policy set you want to export, and click **Export**.

      Note: If the Application policy sets page is not currently displayed, click **Services** > **Policy Sets** > **Application Policy Sets** from the administrative console.
   b. On the next page, click the .zip file link for the policy set.
   c. In the File Download window, click **Save** and indicate a location that is accessible by Integration Designer.
   d. Click **Back**.
   e. Complete steps 2a on page 81 through 2d on page 81 for each policy set you want to export.

The SAML policy sets are saved in .zip files and are ready to be imported into Integration Designer.

Import the policy sets into Integration Designer, as described in the topic “Policy sets”.

Invoking web services that require HTTP basic authentication:

HTTP basic authentication employs a user name and password to authenticate a service client to a secure endpoint. You can set up HTTP basic authentication when sending or receiving web service requests.

You set up HTTP basic authentication for receiving web service requests by configuring the Java API for XML Web Services (JAX-WS) export binding, as described in the Creating and assigning security roles to web service exports.

HTTP basic authentication can be enabled for web service requests that are sent by a JAX-WS import binding in one of two ways:

- When configuring the import binding in an SCA module, you can select the supplied HTTP authentication policy set named BPMHTTPBasicAuthentication (which is provided with the web service (JAX-WS) import binding) or any other policy set that includes the HTTPTransport policy.
- When constructing the SCA module, you can use mediation flow capabilities to dynamically create a new HTTP authentication header and specify the user name and password information in the header.

Note: The policy set has precedence over the value specified in the header. If you want to use the value set in the HTTP authentication header at run time, do not attach a policy set that includes the HTTPTransport policy. Specifically, do not use the default BPMHTTPBasicAuthentication policy set, and, if you have defined a policy set, make sure it excludes the HTTPTransport policy.

For more information about web service policy sets and policy bindings and how they are used, see Web services policy sets of the WebSphere Application Server Information Center.

- To use the supplied policy set, perform the following steps:
  1. Optional: In the administrative console, create a client general policy binding or edit an existing one that includes the HTTPTransport policy with the required user ID and password values.
  2. In IBM Integration Designer, generate a web service (JAX-WS) import binding and attach the BPMHTTPBasicAuthentication policy set.
  3. Perform *one* of the following steps:
     - In IBM Integration Designer, in the web service (JAX-WS) import binding properties, specify the name of an existing client general policy binding that includes the HTTPTransport policy.
     - After deploying the SCA module, use the administrative console to either select an existing client policy binding, or create a new client policy binding and then associate it with the import binding.
  4. Optional: In the administrative console of the process server, edit the selected policy set binding to specify the required ID and password.
- To specify the user name and password in the HTTP authentication header, perform one of the following sets of steps:
  - Use the HTTP Header Setter mediation primitive in IBM Integration Designer to create the HTTP authentication header, and specify the user name and password.
  - If additional logic is required, use Java code in a custom mediation primitive (as shown in the following example) to:
    1. Create an HTTP authentication header.
    2. Specify the user name and password information.
    3. Add the new HTTP authentication header to HTTPControl.
    4. Set the updated HTTPControl back in the Context service.

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
```

```
}
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

Checking the server configuration:

When you deploy an application with a web service (JAX-WS) binding, you must make sure that the server on which the application is deployed does not have the **Start components as needed** option selected.

You can check to see whether this option is selected by performing the following steps from the administrative console:

1. Click **Servers** > **Server types** > **WebSphere application servers**.
2. Click the server name.
3. From the Configuration tab, determine whether **Start components as needed** is selected.
4. Perform one of the following steps:
   - If **Start components as needed** is selected, remove the check, and then click **Apply**.
   - If **Start components as needed** is not selected, click **Cancel**.

Attachments in SOAP messages:

You can send and receive SOAP messages that include binary data (such as PDF files or JPEG images) as attachments. Attachments can be *referenced* (that is, represented explicitly as message parts in the service interface) or *unreferenced* (in which arbitrary numbers and types of attachments can be included).

A referenced attachment can be represented in one of the following ways:

- MTOM attachments use the SOAP Message Transmission Optimization Mechanism (http://www.w3.org/TR/soap12-mtom/) specified encoding. MTOM attachments are enabled through a configuration option in the import and export bindings and are the recommended way to encode attachments for new applications.
- As a wsi:swaRef-typed element in the message schema

  Attachments defined using the wsi:swaRef type conform to the Web Services Interoperability Organization (WS-I) *Attachments Profile Version 1.0* (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.
- As a top-level message part, using a binary schema type

  Attachments represented as top-level message parts conform to the *SOAP Messages with Attachments* (http://www.w3.org/TR/SOAP-attachments) specification.

  Attachments represented as top-level message parts can also be configured to ensure that the WSDL document and messages produced by the binding conform to the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html).

An unreferenced attachment is carried in a SOAP message without any representation in the message schema.

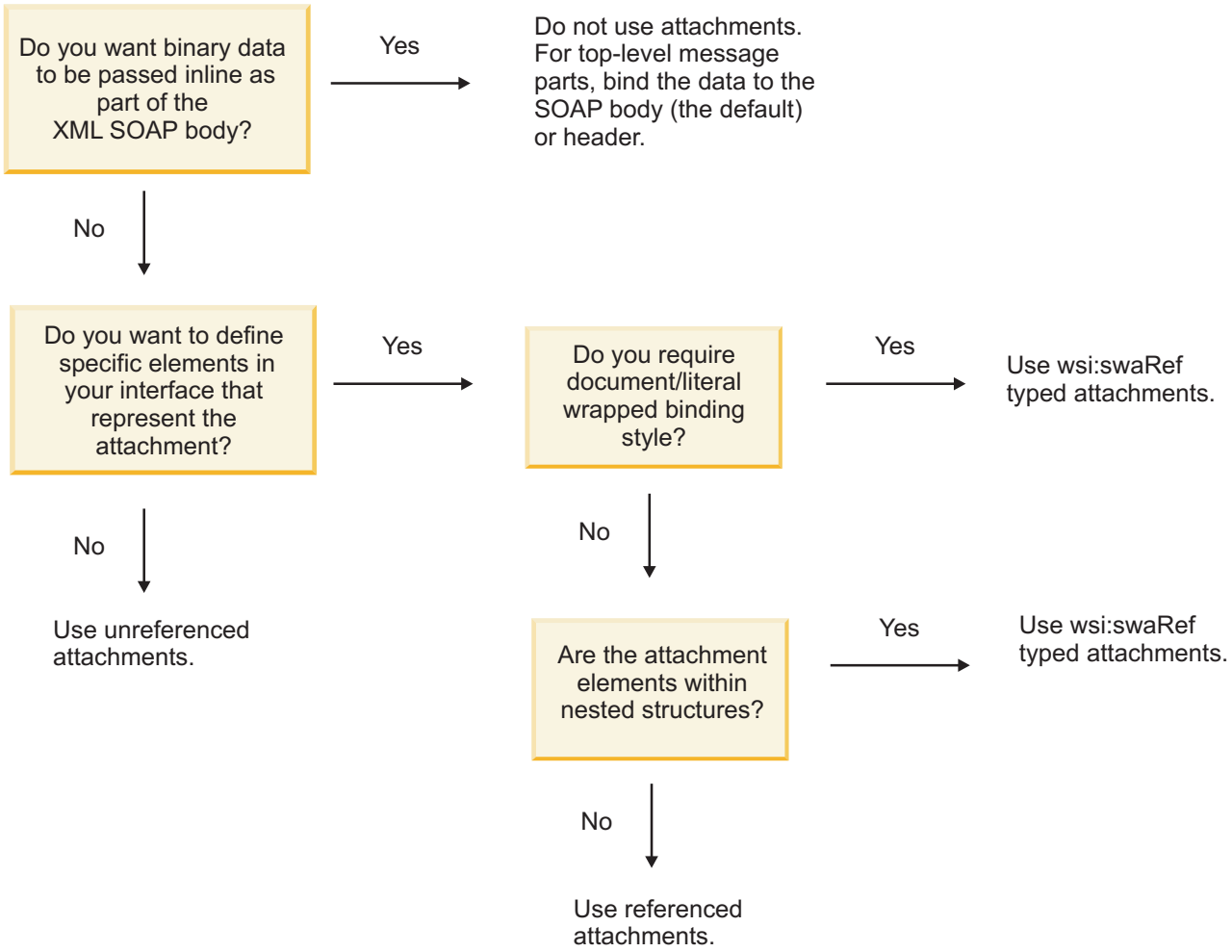
In all cases, except MTOM attachments, the WSDL SOAP binding should include a MIME binding for attachments to be used, and the maximum size of the attachments should not exceed 20 MB.

Note: To send or receive SOAP messages with attachments, you must use one of the Web service bindings based on the Java API for XML Web Services (JAX-WS).

How to choose the appropriate attachment style:

When designing a new service interface that includes binary data, consider how that binary data is carried in the SOAP messages that are sent and received by the service.

Message Transmission Optimization Mechanism (MTOM) should be used for attachments if the connected web service application supports it. If not, the following diagram shows how other attachment styles are chosen. Use the following set of questions to determine the appropriate attachment style:



MTOM attachments: top-level message parts:

You can send and receive web service messages which include SOAP Message Transmission Optimization Mechanism (MTOM) attachments. In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

By sending or receiving a referenced attachment in a SOAP message, the binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

The following is a sample of an MTOM SOAP message:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;
type="application/xop+xml"; start="
<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>"; start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header/>
      <soapenv:Body>
         <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
            <input>
            <imageData><xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
            href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/></imageData>
            </input>
         </sendImage>
      </soapenv:Body>
   </soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

Note that in the MTOM sample, the content-type for the SOAP envelope is **application/xop+xml** and the binary data is replaced by an **xop:Include** element like below:

```
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
```

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

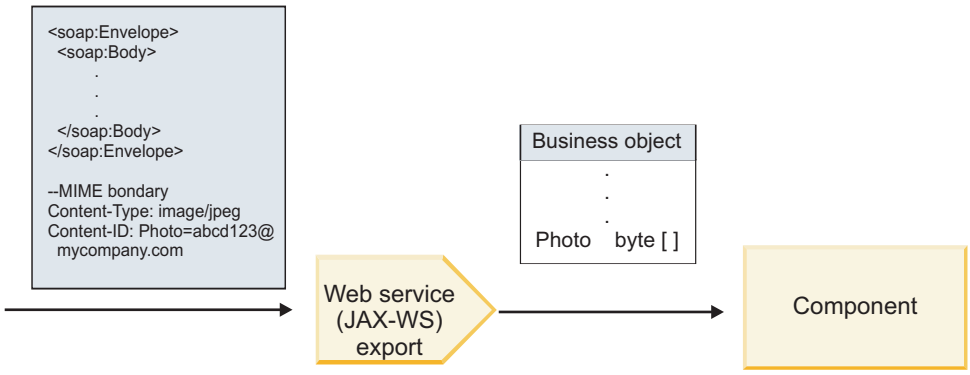


Figure 58. How the Web service (JAX-WS) export binding processes a SOAP message with a referenced attachment

MTOM attachment attributes

- MTOM can support attachment elements within nested structures.
- MTOM is only available for the base64Binary type.
- MTOM can support attachment elements within nested structures meaning that the **bodyPath** for MTOM attachments are the **xpath** location for the element where the MTOM attachment is held. The computing logic for **bodyPath** is strictly following the schema to generate the **xpath** location as shown in the examples below:
  - For a non-array type (**maxOccurs** is 1): /sendImage/input/imageData
  - For a array type (**maxOccurs** > 1): /sendImage/input/imageData[1]
- Mixed attachment types are not supported, meaning that if MTOM is enabled on the import binding, the MTOM attachment will be generated. If MTOM is disabled or if the MTOM configuration value is left as the default value on the export binding, the incoming MTOM message is not supported.

Referenced attachments: swaRef-typed elements:

You can send and receive SOAP messages that include attachments represented in the service interface as swaRef-typed elements.

An swaRef-typed element is defined in the Web Services Interoperability Organization (WS-I) *Attachments Profile* Version 1.0 (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), which defines how message elements are related to MIME parts.

In the SOAP message, the SOAP body contains an swaRef-typed element that identifies the content ID of the attachment.

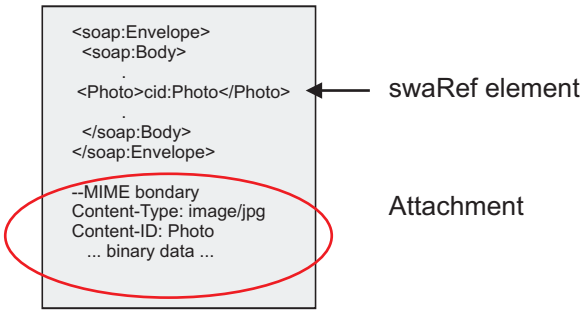


Figure 59. A SOAP message with an swaRef element

The WSDL for this SOAP message contains an swaRef-typed element within a message part that identifies the attachment.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

The WSDL should also contain a MIME binding that indicates MIME multipart messages are to be used.

Note: The WSDL does *not* include a MIME binding for the specific swaRef-typed message element, because MIME bindings apply only to top-level message parts.

Attachments represented as swaRef-typed elements can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Inbound processing of attachments

You use Integration Designer to configure an export binding to receive the attachment. You create a module and its associated interface and operations, including an element of type swaRef. You then create a Web service (JAX-WS) binding.

Note: See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

When a client passes a SOAP message with an swaRef attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the content ID of the attachment in the business object.

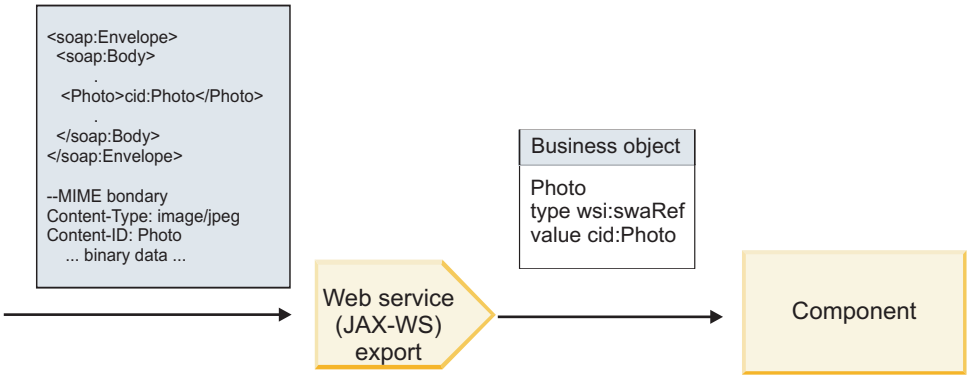


Figure 60. How the Web service (JAX-WS) export binding processes a SOAP message with an swaRef attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 16 on page 87, when swaRef attachments are accessed by components, the attachment content identifier appears as an element of type swaRef.

Each attachment of a SOAP message also has a corresponding **attachments** element in the SMO. When using the WS-I swaRef type, the **attachments** element includes the attachment content type and content ID as well as the actual binary data of the attachment.

To obtain the value of an swaRef attachment, it is therefore necessary to obtain the value of the swaRef-typed element, and then locate the **attachments** element with the corresponding **contentID** value.

Note that the **contentID** value typically has the **cid:** prefix removed from the swaRef value.

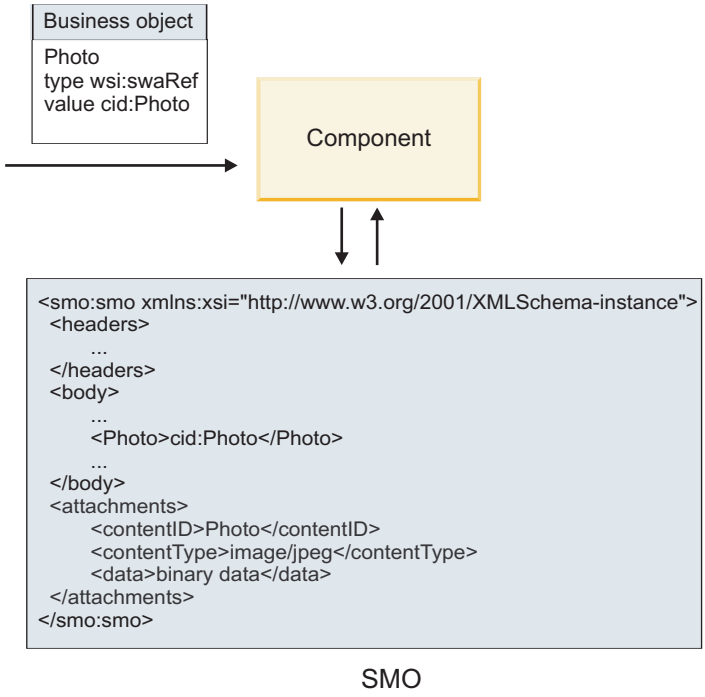


Figure 61. How swaRef attachments appear in the SMO

Outbound processing

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines the attachment that will be passed to the Web service.

When an SCA message is received by a Web service (JAX-WS) import binding, swaRef-typed elements are sent as attachments if the import is wired to a mediation flow component and the swaRef-typed element has a corresponding **attachments** element.

For outbound processing, swaRef-typed elements are always sent with their content ID values; however the mediation module must ensure that there is a corresponding **attachments** element with a matching **contentID** value.

Note: To conform to the WS-I Attachments Profile, the **content ID** value should follow the "content-id part encoding," as described in section 3.8 of the WS-I *Attachments Profile* 1.0.

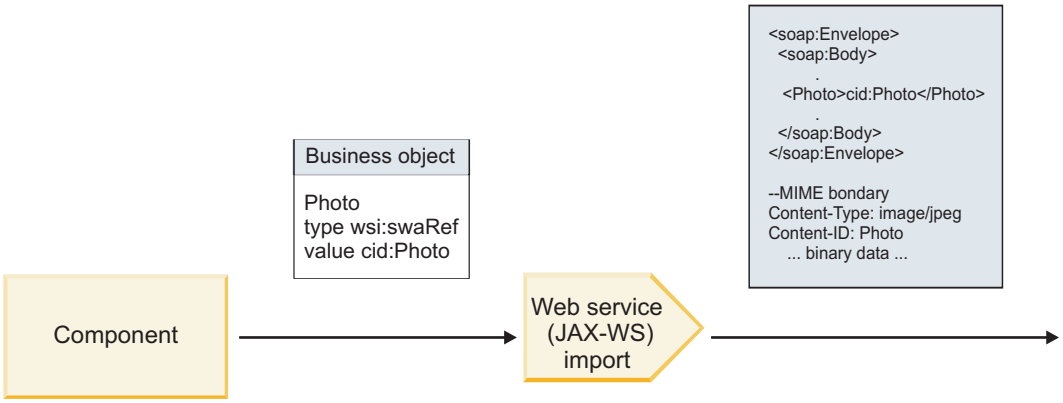


Figure 62. How the Web service (JAX-WS) import binding generates a SOAP message with an swaRef attachment

Setting attachment metadata in a mediation flow component

If, in the SMO, there is an swaRef-typed element value and an **attachments** element, the binding prepares the SOAP message (with the attachment) and sends it to a recipient.

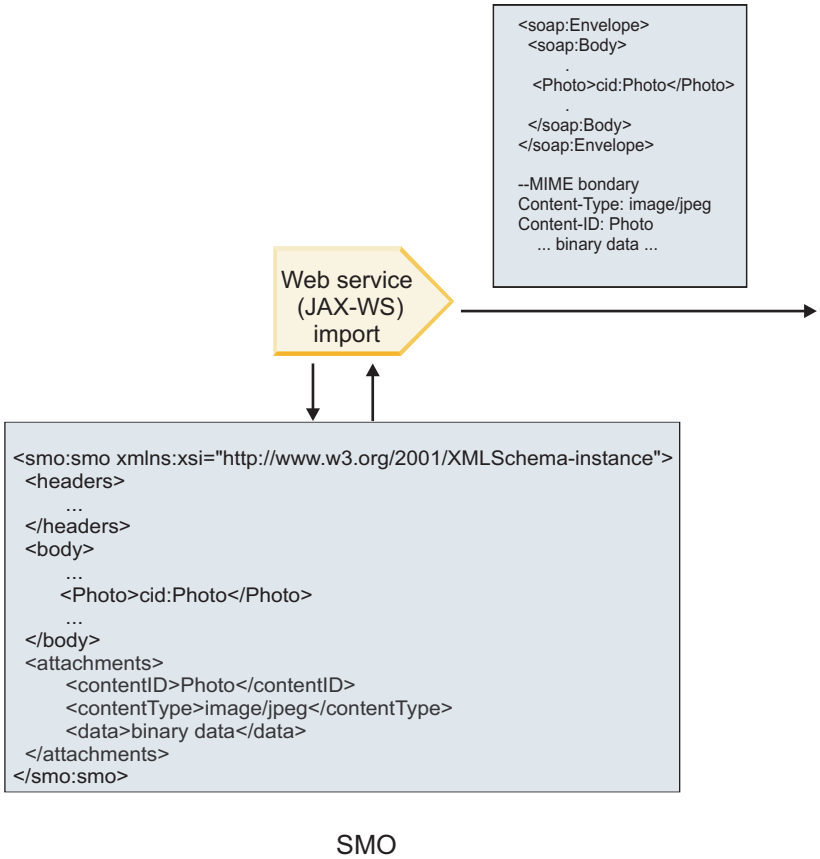


Figure 63. How an swaRef attachment in the SMO is accessed to create the SOAP message

The **attachments** element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Referenced attachments: top-level message parts:

You can send and receive SOAP messages that include binary attachments that are declared as parts in your service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachment or attachments are in subsequent parts.

What is the advantage of sending or receiving a referenced attachment in a SOAP message? The binary data that makes up the attachment (which is often quite large) is held separately from the SOAP message body so that it does not need to be parsed as XML. This results in more efficient processing than if the binary data were held within an XML element.

Types of SOAP messages with referenced attachments

Beginning with Version 7.0.0.3 of IBM Business Process Manager, you have a choice about how the SOAP message is generated:

- **WS-I-compliant messages**

  The runtime can generate SOAP messages that comply with the WS-I *Attachments Profile Version 1.0* and the WS-I *Basic Profile Version 1.1*. In a SOAP message that is compliant with these profiles, only one message part is bound to the SOAP body; for those that are bound as attachments, the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is used to relate the attachment to the message part.
- **Non-WS-I-compliant messages**

  The runtime can generate SOAP messages that do not comply with the WS-I profiles but that are compatible with the messages generated in Version 7.0 or 7.0.0.2 of IBM Business Process Manager. The SOAP messages use top-level elements named after the message part with an **href** attribute that holds the attachment **content-id**, but the content-id part encoding (as described in the WS-I *Attachments Profile Version 1.0*) is not used.

Selecting WS-I compliance for Web service exports

You use Integration Designer to configure an export binding. You create a module and its associated interface and operations. You then create a Web service (JAX-WS) binding. The Referenced attachments page displays all the binary parts from the created operation, and you select which parts will be attachments. You then specify, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body.

  Note: This option can be used only when the corresponding WSDL file is also WS-I compliant.

A WSDL file that is generated by Integration Designer Version 7.0.0.3 is compliant with WS-I. However, if you import a WSDL file that is not compliant with WS-I, you cannot select this option.

- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body.

Note: Only top-level message parts (that is, elements defined in the WSDL portType as parts within the input or output message) that have a binary type (either base64Binary or hexBinary) can be sent or received as referenced attachments.
See the "Working with attachments" topic in the Integration Designer information center for more detailed information.

For WS-I-compliant messages, the content-ID that is generated in the SOAP message is a concatenation of the following elements:

- The value of the **name** attribute of the **wsdl:part** element referenced by the **mime:content**
- The character **=**
- A globally unique value, such as a UUID
- The character **@**
- A valid domain name

Inbound processing of referenced attachments

When a client passes a SOAP message with an attachment to a Service Component Architecture (SCA) component, the Web service (JAX-WS) export binding first removes the attachment. It then parses the SOAP part of the message and creates a business object. Finally, the binding sets the attachment binary in the business object.

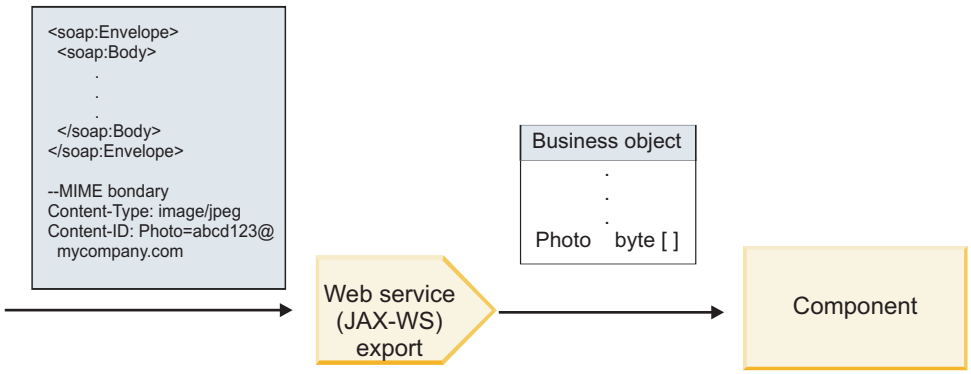


Figure 64. How the Web service (JAX-WS) export binding processes a WS-I compliant SOAP message with a referenced attachment

Accessing attachment metadata in a mediation flow component

As shown in Figure 19 on page 91, when referenced attachments are accessed by components, the attachment data appears as a byte array.

Each referenced attachment of a SOAP message also has a corresponding **attachments** element in the SMO. The **attachments** element includes the attachment content type and the path to the message body element where the attachment is held.

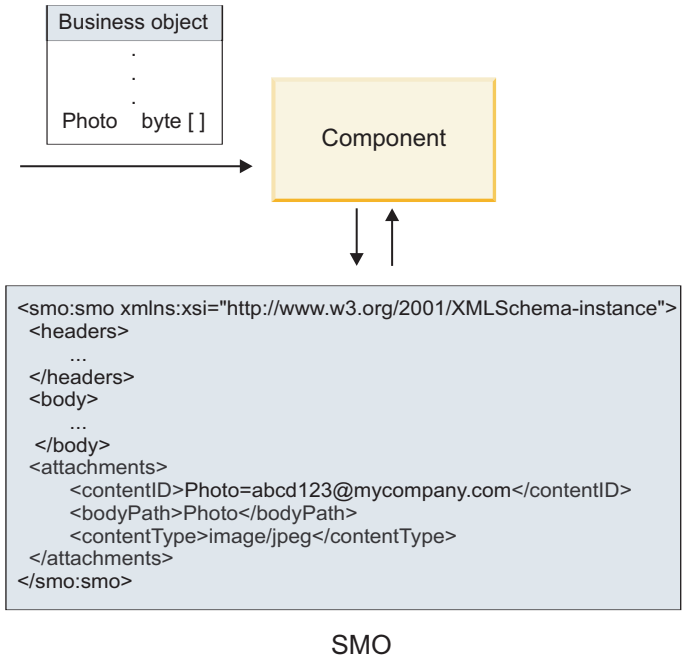


Figure 65. How referenced attachments appear in the SMO

Important: The path to the message body element is not automatically updated if the message is transformed and the attachment moved. You can use a mediation flow to update the **attachments** element with the new path (for example, as part of the transform or using a separate message element setter).

How outbound SOAP messages are constructed

You use Integration Designer to configure a Web service (JAX-WS) import binding to invoke an external Web service. The import binding is configured with a WSDL document that describes the Web service to be invoked and defines which message parts should be passed as attachments. You can also indicate, on the Specify the WS-I AP 1.0 compliance page of Integration Designer, one of the following choices:

- **Use WS-I AP 1.0 compliant SOAP message**

  If you select this option, you also specify which message part should be bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding do not include elements in the SOAP body that refer to the attachments; the relationship is expressed by way of the attachment content ID including the message part name.
- **Use non WS-I AP 1.0 compliant SOAP message**

  If you select this option, which is the default, the first message part is bound to the SOAP body; all others are bound to attachments or headers. Messages sent by the binding include one or more elements in the SOAP body that refer to the attachments by way of an **href** attribute.

Note: The part that represents an attachment, as defined in the WSDL, must be a simple type (either base64Binary or hexBinary). If a part is defined by a complexType, that part cannot be bound as an attachment.

Outbound processing of referenced attachments

The import binding uses information in the SMO to determine how the binary top-level message parts are sent as attachments.

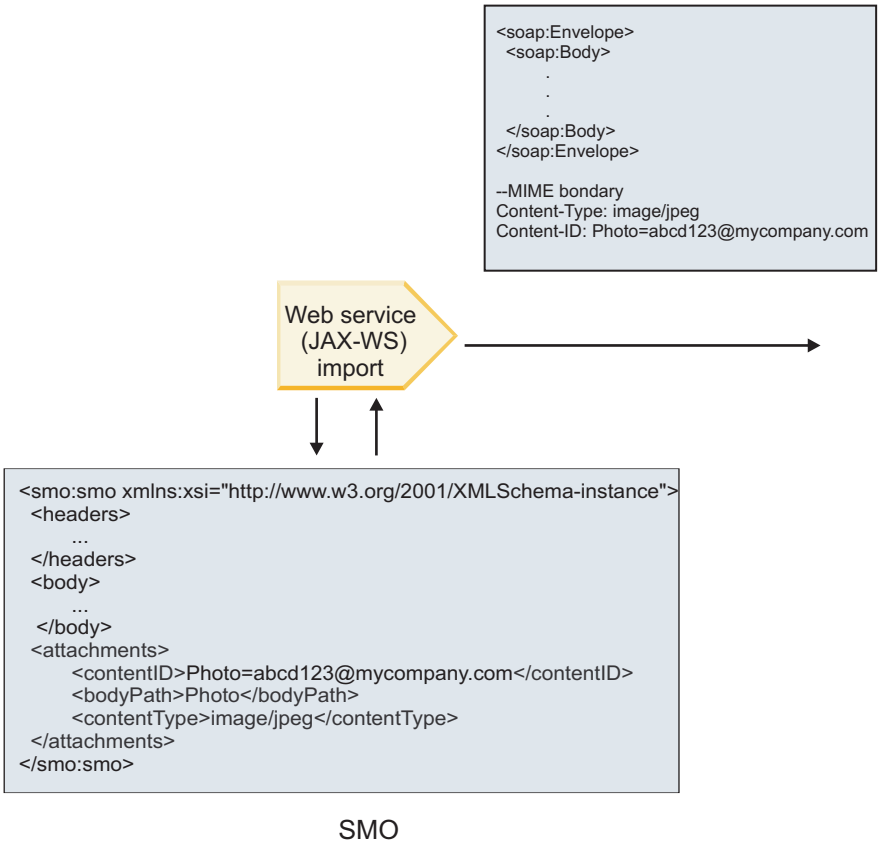


Figure 66. How the referenced attachment in the SMO is accessed to create the SOAP message

The **attachments** element is present in the SMO only if a mediation flow component is connected directly to the import or export; it does not get passed across other component types. If the values are needed in a module that contains other component types, a mediation flow component should be used to copy the values into a location where they can then be accessed in the module, and another mediation flow component used to set the correct values before an outbound invocation by way of a Web service import.

The binding uses a combination of the following conditions to determine how (or whether) the message is sent:

- Whether there is a WSDL MIME binding for the top-level binary message part and, if so, how the content type is defined
- Whether there is an **attachments** element in the SMO whose **bodyPath** value references a top-level binary part

How attachments are created when an attachment element exists in the SMO

The following table shows how an attachment is created and sent if the SMO contains an **attachment** element with a **bodyPath** that matches a message name part:

Table 53. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	**How message is created and sent**
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to **application/octet-stream**.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to the type defined in the WSDL MIME content element.
Not present	Message part is sent as an attachment. **Content-Id** is set to the value in the attachments element if present; otherwise, one is generated. **Content-Type** is set to the value in the attachments element if present; otherwise, it is set to **application/octet-stream**. **Note:** Sending message parts as attachments when not defined as such in the WSDL may break compliance with the WS-I Attachments Profile 1.0 and so should be avoided if possible.

How attachments are created when no attachment element exists in the SMO

The following table shows how an attachment is created and sent if the SMO does not contain an **attachment** element with a **bodyPath** that matches a message name part:

Table 54. How the attachment is generated

Status of WSDL MIME binding for top-level binary message part	**How message is created and sent**
Present with one of the following: • No defined content type for the message part • Multiple content types defined • Wildcard content type defined	Message part is sent as an attachment. **Content-Id** is generated. **Content-Type** is set to **application/octet-stream**.
Present with single, non-wildcard content for the message part	Message part is sent as an attachment. **Content-Id** is generated. **Content-Type** is set to the type defined in the WSDL MIME content element.
Not present	Message part is not sent as an attachment.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Unreferenced attachments:

You can send and receive *unreferenced* attachments that are not declared in the service interface.

In a MIME multipart SOAP message, the SOAP body is the first part of the message, and the attachments are in subsequent parts. No reference to the attachment is included in the SOAP body.

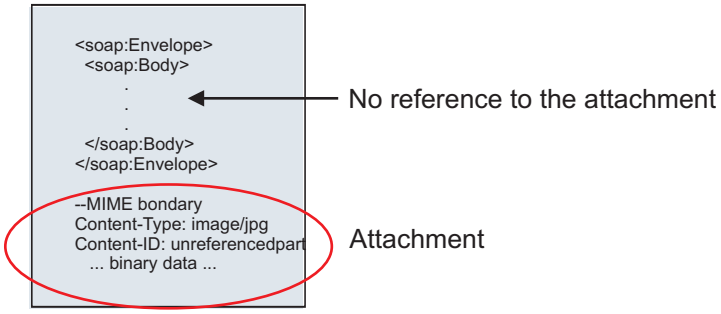


Figure 67. A SOAP message with an unreferenced attachment

You can send a SOAP message with an unreferenced attachment through a Web service export to a Web service import. The output message, which is sent to the target Web service, contains the attachment.

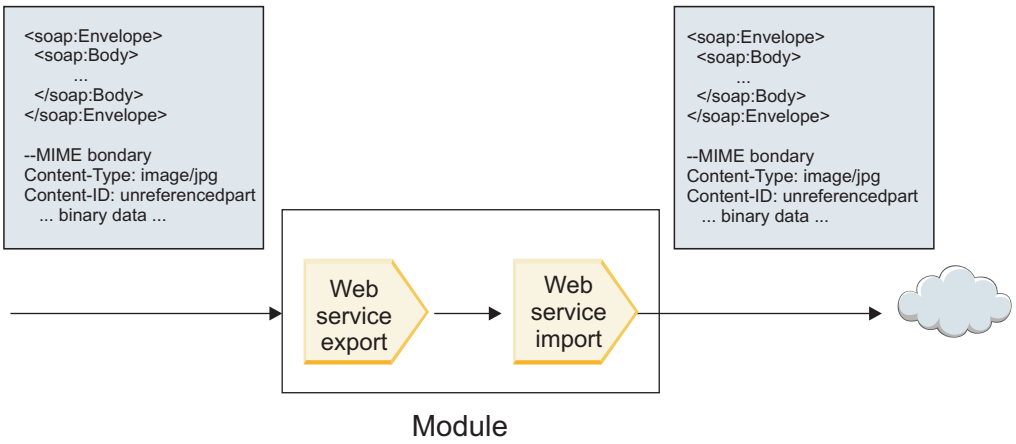


Figure 68. An attachment passing through an SCA module

In Figure 23 on page 95, the SOAP message, with the attachment, passes through without modification.

You can also modify the SOAP message by using a mediation flow component. For example, you can use the mediation flow component to extract data from the SOAP message (binary data in the body of the message, in this case) and create a SOAP with attachments message. The data is processed as part of the

attachments element of a service message object (SMO).

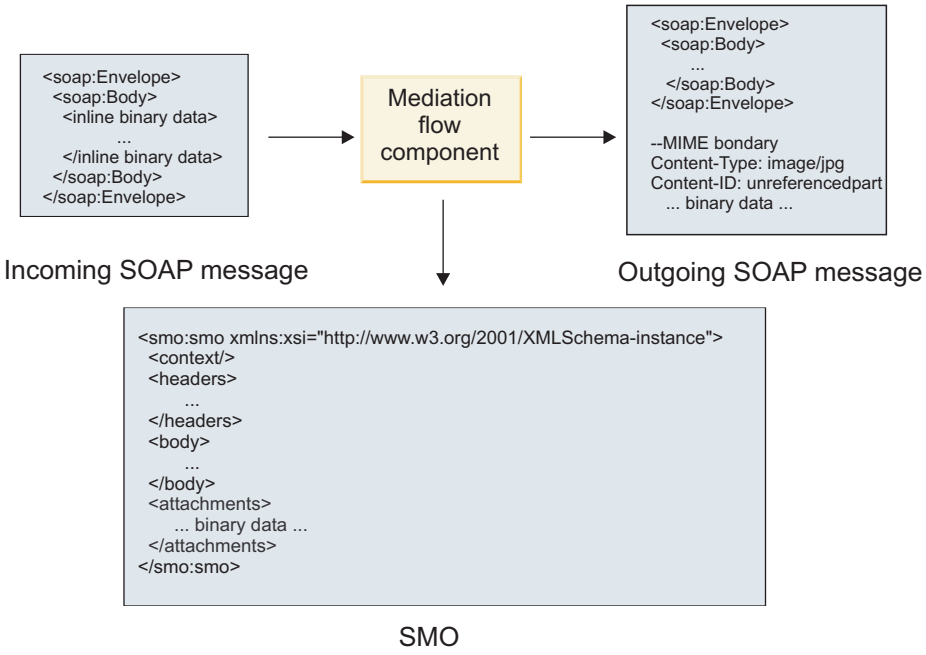


Figure 69. A message processed by a mediation flow component

Conversely, the mediation flow component can transform the incoming message by extracting and encoding the attachment and then transmitting the message with no attachments.

Instead of extracting data from an incoming SOAP message to form a SOAP with attachments message, you can obtain the attachment data from an external source, such as a database.

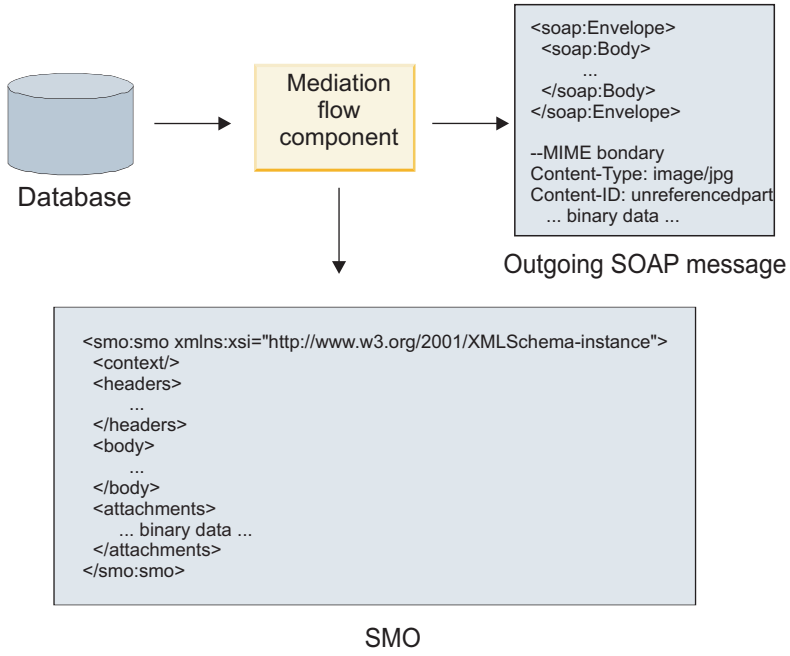


Figure 70. An attachment obtained from a database and added to the SOAP message

Conversely, the mediation flow component can extract the attachment from an incoming SOAP message and process the message (for example, store the attachment in a database).

Unreferenced attachments can be propagated only across mediation flow components. If an attachment must be accessed by or propagated to another component type, use a mediation flow component to move the attachment to a location that is accessible by that component.

Important: As described in "XML representation of SMO," the Mapping mediation primitive transforms messages using an XSLT 1.0 transformation. The transformation operates on an XML serialization of the SMO. The Mapping mediation primitive allows the root of the serialization to be specified, and the root element of the XML document reflects this root.

When you are sending SOAP messages with attachments, the root element you choose determines how attachments are propagated.

- If you use "/body" as the root of the XML map, all attachments are propagated across the map by default.
- If you use "/" as the root of the map, you can control the propagation of attachments.

Use of WSDL document style binding with multipart messages:

The Web Services Interoperability Organization (WS-I) organization has defined a set of rules regarding how Web services should be described by way of a WSDL and how the corresponding SOAP messages should be formed, in order to ensure interoperability.

These rules are specified in the WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html). In particular, the WS-I Basic Profile 1.1 R2712 states: "A document-literal binding MUST be serialized as an ENVELOPE with a soap:Body whose child element is an instance of the global element declaration referenced by the corresponding wsdl:message part."

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, only one of those parts should be bound to the SOAP body in order to be compliant with the WS-I Basic Profile 1.1.

Further, the WS-I Attachments Profile 1.0 R2941 states: "A wsdl:binding in a DESCRIPTION SHOULD bind every wsdl:part of a wsdl:message in the wsdl:portType to which it refers to one of soapbind:body, soapbind:header, soapbind:fault , soapbind:headerfault, or mime:content.".

This means that, when using a document style SOAP binding for an operation with messages (input, output, or fault) that are defined with multiple parts, all parts other than the one selected to be bound to the SOAP body must be bound as attachments or headers.

The following approach is used when WSDL descriptions are generated for exports with Web service (JAX-WS and JAX-RPC) bindings in this case:

- You can choose which message part is bound to the SOAP body if there is more than one non-binary-typed element. If there is a single non-binary typed element, that element is automatically bound to the SOAP body.
- For the JAX-WS binding, all other message parts of type "hexBinary" or "base64Binary" are bound as referenced attachments. See "Referenced attachments: top-level message parts" on page 89.
- All other message parts are bound as SOAP headers.

The JAX-RPC and JAX-WS import bindings honor the SOAP binding in an existing WSDL document with multipart document style messages even if it does bind multiple parts to the SOAP body; however, you are not able to generate Web service clients for such WSDL documents in Rational Application Developer.

Note: The JAX-RPC binding does not support attachments.

The recommended pattern when using multipart messages with an operation that has document style SOAP binding is therefore:

1. Use the document/literal wrapped style. In this case, messages always have a single part; however, attachments have to be unreferenced (as described in “Unreferenced attachments” on page 95) or swaRef-typed (as described in “Referenced attachments: swaRef-typed elements” on page 86) in this case.
2. Use the RPC/literal style. In this case, there are no restrictions on the WSDL binding in terms of number of parts bound to the SOAP body; the SOAP message that results always has a single child that represents the operation being invoked, with the message parts being children of that element.
3. For the JAX-WS binding, have at most one message part that is not of type "hexBinary" or "base64Binary", unless it is acceptable to bind the other non-binary parts to SOAP headers.
4. Any other cases are subject to the behavior described.

Note: Additional restrictions exist when you use are using SOAP messages that do not conform to the WS-I *Basic Profile Version 1.1*.

- The first message part should be non-binary.
- When receiving multipart document-style SOAP messages with referenced attachments, the JAX-WS binding expects each referenced attachment to be represented by a SOAP body child element with an href attribute value that identifies the attachment by its content ID. The JAX-WS binding sends referenced attachments for such messages in the same way. This behavior is not compliant with the WS-I Basic Profile.

  To ensure that your messages comply with the Basic Profile, follow approach 1 on page 98 or 2 on page 98 in the previous list or avoid the use of referenced attachments for such messages and use unreferenced or swaRef-typed attachments instead.

HTTP bindings

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

Hypertext Transfer Protocol (HTTP) is a widely-used protocol for transferring information on the Web. When you are working with an external application that uses the HTTP protocol, an HTTP binding is necessary. The HTTP binding transforms the data that is passed in as a message in native format to a business object in an SCA application. The HTTP binding also can transform the data that is passed out as a business object to the native format expected by the external application.

Note: If you want to interact with clients and services that use the Web services SOAP/HTTP protocol, consider using one of the Web service bindings, which provide additional functionality with respect to handling Web services standard qualities of service.

Some common scenarios for using the HTTP binding are described in the following list:

- SCA-hosted services can invoke HTTP applications using an HTTP import.
- SCA-hosted services can expose themselves as HTTP-enabled applications, so they can be used by HTTP clients, using an HTTP export.
- IBM Business Process Manager and Process Server can communicate between themselves across an HTTP infrastructure, consequently users can manage their communications according to corporate standards.
- IBM Business Process Manager and Process Server can act as mediators of HTTP communications, transforming and routing messages, which improves the integration of applications using a HTTP network.
- IBM Business Process Manager and Process Server can be used to bridge between HTTP and other protocols, such as SOAP/HTTP Web services, Java Connector Architecture (JCA)-based resource adapters, JMS, and so on.

Detailed information about creating HTTP import and export bindings can be found in the Integration Designer information center. See the **Developing integration applications** > **Accessing external services with HTTP**> topics.

HTTP bindings overview:

The HTTP binding provides connectivity to HTTP-hosted applications. It mediates communication between HTTP applications and allows existing HTTP-based applications to be called from a module.

HTTP import bindings

The HTTP import binding provides outbound connectivity from Service Component Architecture (SCA) applications to an HTTP server or applications.

The import invokes an HTTP endpoint URL. The URL can be specified in one of three ways:

- The URL can be set dynamically in the HTTP headers by way of the dynamic override URL.
- The URL can be set dynamically in the SMO target address element.
- The URL can be specified as a configuration property on the import.

This invocation is always synchronous in nature.

Although HTTP invocations are always request-reply, the HTTP import supports both one-way and two-way operations and ignores the response in the case of a one-way operation.

HTTP export bindings

The HTTP export binding provides inbound connectivity from HTTP applications to an SCA application.

A URL is defined on the HTTP export. HTTP applications that want to send request messages to the export use this URL to invoke the export.

The HTTP export also supports pings.

HTTP bindings at runtime

An import with an HTTP binding at runtime sends a request with or without data in the body of the message from the SCA application to the external Web service. The request is made from the SCA application to the external Web service, as shown in Figure 26 on page 99.

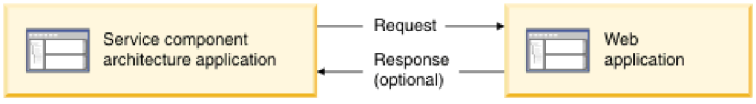


Figure 71. Flow of a request from the SCA application to the Web application

Optionally, the import with the HTTP binding can receive data back from the Web application in a response to the request.

With an export, the request is made by a client application to a Web service, as shown in Figure 27 on page 100.

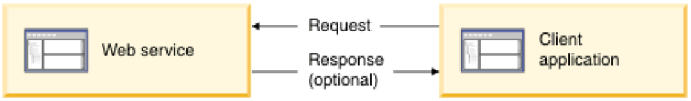


Figure 72. Flow of a request from the Web service to the client application.

The Web service is a Web application running on the server. The export is implemented in that Web application as a servlet so the client sends its request to a URL address. The servlet passes the request to the SCA application in the runtime.

Optionally, the export may send data to the client application in response to the request.

HTTP headers:

HTTP import and export bindings allow configuration of HTTP headers and their values to be used for outbound messages. The HTTP import uses these headers for requests, and the HTTP export uses them for responses.

Statically configured headers and control information take precedence over values dynamically set at runtime. However, the dynamic override URL, Version, and Method control values override the static values, which are otherwise considered defaults.

The binding supports the dynamic nature of the HTTP import URL by determining the value of HTTP target URL, Version, and Method at run time. These values are determined by extracting the value of Endpoint Reference, Dynamic Override URL, Version, and Method.

- For Endpoint Reference, use com.ibm.websphere.sca.addressing.EndpointReference APIs or set the /headers/SMOHeader/Target/address field in the SMO header.
- For Dynamic Override URL, Version, and Method, use the HTTP control parameters section of the Service Component Architecture (SCA) message. Note that the Dynamic Override URL takes precedence over the target Endpoint Reference; however, the Endpoint Reference applies across bindings, so it is the preferred approach and should be used where possible.

The control and header information for outbound messages under HTTP export and import bindings is processed in the following order:

1. Header and control information excluding HTTP dynamic override URL, Version, and Method from the SCA Message (lowest priority)
2. Changes from the administrative console on the export/import level
3. Changes from the administrative console on the method level of the export or import
4. Target address specified by way of the Endpoint Reference or the SMO header
5. Dynamic Override URL, Version, and Method from the SCA message
6. Headers and control information from the data handler or data binding (highest priority)

The HTTP export and import will populate inbound direction headers and control parameters with data from the incoming message (HTTPExportRequest and HTTPImportResponse) only if protocol header propagation is set to **True**. Inversely, the HTTP export and import will read and process outbound headers and control parameters (HTTPExportResponse and HTTPImportRequest) only if protocol header propagation is set to **True**.

Note: Data handler or data binding changes to headers or control parameters in the import response or export request will not alter the processing instructions of the message inside the import or export binding and should be used only to propagate modified values to downstream SCA components.

The context service is responsible for propagating the context (including the protocol headers, such as the HTTP header, and the user context, such as account ID) along an SCA invocation path. During

development in IBM Integration Designer, you can control the propagation of context by way of import and export properties. For more details, see the import and export bindings information in the IBM Integration Designer information center.

Supplied HTTP header structures and support

Table 29 on page 101 itemizes the request/response parameters for HTTP Import and HTTP Export requests and responses.

Table 55. Supplied HTTP header information

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
URL	Ignored	Not set	Read from the request message. **Note:** Query string is also part of the URL control parameter.	Ignored
Version (possible values: 1.0, 1.1; default is 1.1)	Ignored	Not set	Read from the request message	Ignored
Method	Ignored	Not set	Read from the request message	Ignored
Dynamic Override URL	If set in the data handler or data binding, overrides the HTTP Import URL. Written to the message in the request line. **Note:** Query string is also part of the URL control parameter.	Not set	Not set	Ignored
Dynamic Override Version	If set, overrides the HTTP Import Version. Written to the message in the request line.	Not set	Not set	Ignored
Dynamic Override Method	If set, overrides the HTTP Import Method. Written to the message in the request line.	Not set	Not set	Ignored
Media Type (This control parameter carries part of the value of the Content-Type HTTP header.)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	If present, written to the message as part of Content-Type header. **Note:** This control element value should be provided by the data handler or data binding.

Table 55. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
Character set (default: UTF-8)	If present, written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.	Read from the response message, Content-Type header	Read from the request message, Content-Type header	Supported; written to the message as part of the Content-Type header. **Note:** This control element value should be provided by the data binding.
Transfer Encoding (Possible values: chunked, identity; default is identity)	If present, written to the message as a header and controls how the message transformation is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the message transformation is encoded.
Content Encoding (Possible values: gzip, x-gzip, deflate, identity; default is identity)	If present, written to the message as a header and controls how the payload is encoded.	Read from the response message	Read from the request message	If present, written to the message as a header and controls how the payload is encoded.
Content-Length	Ignored	Read from the response message	Read from the request message	Ignored
StatusCode (default: 200)	Not supported	Read from the response message	Not supported	If present, written to the message in the response line
ReasonPhrase (default: OK)	Not supported	Read from the response message	Not supported	Control value ignored. The message response line value is generated from the StatusCode.
Authentication (contains multiple properties)	If present, used to construct the Basic Authentication header. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable	Read from the request message Basic Authentication header. The presence of this header does not indicate the user has been authenticated. Authentication should be controlled in the servlet configuration. **Note:** The value for this header will be encoded only on the HTTP protocol. In the SCA, it will be decoded and passed as clear text.	Not applicable
Proxy (contains multiple properties: Host, Port, Authentication)	If present, used to establish connection through proxy.	Not applicable	Not applicable	Not applicable

Table 55. Supplied HTTP header information (continued)

Control name	HTTP Import request	HTTP Import response	HTTP Export request	HTTP Export response
SSL (contains multiple properties: Keystore, Keystore Password, Trustore, Trustore Password, ClientAuth)	If populated and the destination url is HTTPS, it is used to establish a connection through SSL.	Not applicable	Not applicable	Not applicable

HTTP data bindings:

For each different mapping of data between a Service Component Architecture (SCA) message and an HTTP protocol message, a data handler or an HTTP data binding must be configured. Data handlers provide a binding-neutral interface that allows reuse across transport bindings and represent the recommended approach; data bindings are specific to a particular transport binding. HTTP-specific data binding classes are supplied; you can also write custom data handlers or data bindings.

Note: The three HTTP data binding classes described in this topic (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML, and HTTPServiceGatewayDataBinding) are deprecated as of IBM Business Process Manager Version 7.0. Instead of using the data bindings described in this topic, consider the following data handlers:

- Use SOAPDataHandler instead of HTTPStreamDataBindingSOAP.
- Use UTF8XMLDataHandler instead of HTTPStreamDataBindingXML
- Use GatewayTextDataHandler instead of HTTPServiceGatewayDataBinding

Data bindings are provided for use with HTTP imports and HTTP exports: binary data binding, XML data binding, and SOAP data binding. A response data binding is not required for one-way operations. A data binding is represented by the name of a Java class whose instances can convert both from HTTP to ServiceDataObject and vice-versa. A function selector must be used on an export which, in conjunction with method bindings, can determine which data binding is used and which operation is invoked. The supplied data bindings are:

- Binary data bindings, which treat the body as unstructured binary data. The implementation of the binary data binding XSD schema is as follows:

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>

  <xsd:complexType name="HTTPBytesBody">
    <xsd:complexContent>
      <xsd:extension base="tns:HTTPBaseBody">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:hexBinary"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

- XML data bindings, which support the body as XML data. The implementation of the XML data binding is similar to the JMS XML data binding and has no restrictions on the interface schema.
- SOAP data bindings, which support the body as SOAP data. The implementation of the SOAP data binding has no restrictions on the interface schema.

Implementing custom HTTP data bindings

This section describes how to implement a custom HTTP data binding.

Note: The recommended approach is to implement a custom data handler because it can be reused across transport bindings.

HTTPStreamDataBinding is the principal interface for handling custom HTTP messages. The interface is designed to allow handling of large payloads. However, in order for such implementation to work, this data binding must return the control information and headers before writing the message into the stream.

The methods and their order of execution, listed below, must be implemented by the custom data binding.

To customize a data binding, write a class that implements HTTPStreamDataBinding. The data binding should have four private properties:

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

The HTTP binding will invoke the customized data binding in the following order:

- Outbound processing (DataObject to Native format):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- Inbound processing (Native format to DataObject):
  1. setControlParameters(...)
  2. setHeaders(...)
  3. convertFromNativeData(...)
  4. isBusinessException()
  5. getDataObject()
  6. getControlParameters()
  7. getHeaders()

You need to invoke setDataObject(...) in convertFromNativeData(...) to set the value of dataObject, which is converted from native data to the private property "pDataObject".

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
```

```
  this.pHeaders = arg0;
}
/*
* Add http header "IsBusinessException" in pHeaders.
* Two steps:
* 1.Remove all the header with name IsBusinessException (case-insensitive) first.
*   This is to make sure only one header is present.
* 2.Add the new header "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //remove all the header with name IsBusinessException (case-insensitive) first.
  //This is to make sure only one header is present.
  //add the new header "IsBusinessException", code example:
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Get header "IsBusinessException" from pHeaders, return its boolean value
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Customer-developed method to
  //Read data from HTTPInputStream
  //Convert it to DataObject
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

EJB bindings

Enterprise JavaBeans (EJB) import bindings enable Service Component Architecture (SCA) components to invoke services provided by Java EE business logic running on a Java EE server. EJB export bindings allow SCA components to be exposed as Enterprise JavaBeans so that Java EE business logic can invoke SCA components otherwise unavailable to them.

EJB import bindings:

EJB import bindings allow an SCA module to call EJB implementations by specifying the way that the consuming module is bound to the external EJB. Importing services from an external EJB implementation allows users to plug their business logic into the IBM Business Process Manager environment and participate in a business process.

You use Integration Designer to create EJB import bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB import using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB import based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and import files based on the services discovered.
- Creating EJB import using the assembly editor

  You can create an EJB import within an assembly diagram using the Integration Designer assembly editor. From the palette, you can use either an Import or use a Java class to create the EJB binding.

The generated import has data bindings that make the Java-WSDL connection instead of requiring a Java bridge component. You can directly wire a component with a Web Services Description Language (WSDL) reference to the EJB import that communicates to an EJB-based service using a Java interface.

The EJB import can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation to the Java EE business logic can be local (for EJB 3.0 only) or remote.

- Local invocation is used when you want to call Java EE business logic that resides on the same server as the import.

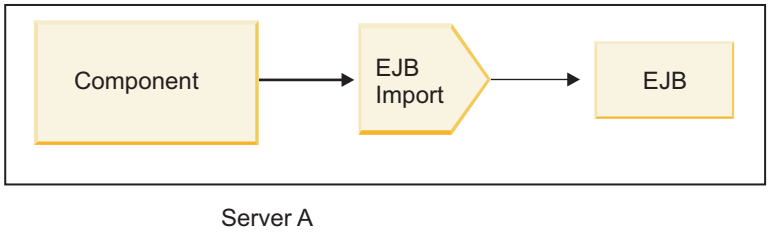


Figure 73. Local invocation of an EJB (EJB 3.0 only)

- Remote invocation is used when you want to call Java EE business logic that does not reside on the same server as the import.

  For example, in the following figure, an EJB import uses the Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) to invoke an EJB method on another server.

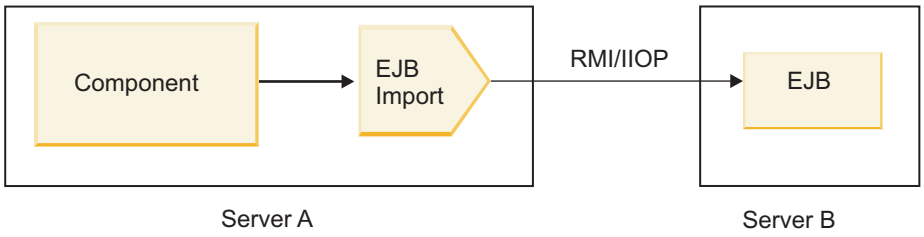


Figure 74. Remote invocation of an EJB

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB import bindings contain the following major components:

- JAX-WS data handler
- EJB fault selector

- EJB import function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler, function selector, and fault selector to perform the tasks otherwise completed by the components that are part of the EJB import bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB export bindings:

External Java EE applications can invoke an SCA component by way of an EJB export binding. Using an EJB export lets you expose SCA components so that external Java EE applications can invoke those components using the EJB programming model.

Note: The EJB export is a stateless bean.

You use Integration Designer to create EJB bindings. You can use either of the following procedures to generate the bindings:

- Creating EJB export bindings using the external service wizard

  You can use the external service wizard in Integration Designer to build an EJB export service based on an existing implementation. The external service wizard creates services based on criteria that you provide. It then generates business objects, interfaces, and export files based on the services discovered.
- Creating EJB export bindings using the assembly editor

  You can create an EJB export using the Integration Designer assembly editor.

Important: A Java 2 Platform, Standard Edition (J2SE) client cannot invoke the EJB export client that is generated in Integration Designer.

You can generate the binding from an existing SCA component, or you can generate an export with an EJB binding for a Java interface.

- When you generate an export for an existing SCA component that has an existing WSDL interface, the export is assigned a Java interface.
- When you generate an export for a Java interface, you can select either a WSDL or a Java interface for the export.

  Note: A Java interface used to create an EJB export has the following limitations with regard to the objects (input and output parameters and exceptions) passed as parameters on a remote call:
  - They must be of concrete type (instead of an interface or abstract type).
  - They must conform to the Enterprise JavaBeans specification. They must be serializable and have the default no-argument constructor, and all properties must be accessible through getter and setter methods.

    Refer to the Sun Microsystems, Inc., Web site at http://java.sun.com for information about the Enterprise JavaBeans specification.

  In addition, the exception must be a checked exception, inherited from java.lang.Exception, and it must be singular (that is, it does not support throwing multiple checked exception types).

  Note also that the business interface of a Java EnterpriseBean is a plain Java interface and must not extend javax.ejb.EJBObject or javax.ejb.EJBLocalObject. The methods of the business interface should not throw java.rmi.Remote.Exception.

The EJB export bindings can interact with Java EE business logic using either the EJB 2.1 programming model or the EJB 3.0 programming model.

The invocation can be local (for EJB 3.0 only) or remote.

- Local invocation is used when the Java EE business logic calls an SCA component that resides on the same server as the export.
- Remote invocation is used when the Java EE business logic does not reside on the same server as the export.

  For example, in the following figure, an EJB uses RMI/IIOP to call an SCA component on a different server.

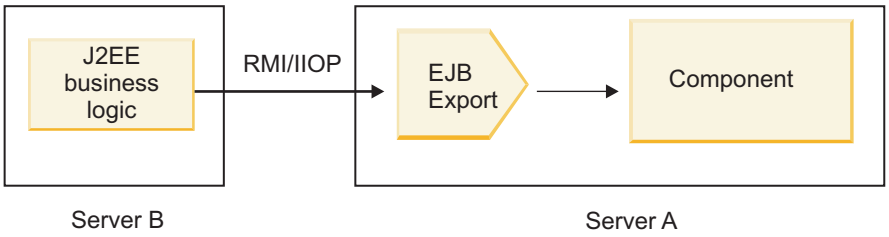


Figure 75. Remote call from a client to an SCA component by way of an EJB export

When it configures the EJB binding, Integration Designer uses the JNDI name to determine the EJB programming model level and the type of invocation (local or remote).

EJB export bindings contain the following major components:

- JAX-WS data handler
- EJB export function selector

If your user scenario is not based on the JAX-WS mapping, you might need a custom data handler and function selector to perform the tasks otherwise completed by the components that are part of the EJB export bindings. This includes the mapping normally completed by the custom mapping algorithm.

EJB binding properties:

EJB import bindings use their configured JNDI names to determine the EJB programming model level and type of invocation (local or remote). EJB import and export bindings use the JAX-WS data handler for data transformation. The EJB import binding uses an EJB import function selector and an EJB fault selector, and the EJB export binding uses an EJB export function selector.

JNDI names and EJB import bindings:

When it configures the EJB binding on an import, Integration Designer uses the JNDI name to determine the EJB programming model level and type of invocation (local or remote).

If no JNDI name is specified, the default EJB interface binding is used. The default names that are created depend on whether you are invoking EJB 2.1 JavaBeans or EJB 3.0 JavaBeans.

Note: Refer to the "EJB 3.0 application bindings overview" topic in theWebSphere Application Server information center for more detailed information about naming conventions.

- EJB 2.1 JavaBeans

  The default JNDI name preselected by Integration Designer is the default EJB 2.1 binding, which takes the form **ejb/** plus the home interface, separated by slashes.

  For example, for the home interface of EJB 2.1 JavaBeans for com.mycompany.myremotebusinesshome, the default binding is:

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  For EJB 2.1, only remote EJB invocation is supported.
- EJB 3.0 JavaBeans

The default JNDI name preselected by Integration Designer for the local JNDI is the fully qualified class name of the local interface preceded by **ejblocal:**. For example, for the fully qualified interface of the local interface com.mycompany.mylocalbusiness, the preselected EJB 3.0 JNDI is:

```
ejblocal:com.mycompany.mylocalbusiness
```

For the remote interface com.mycompany.myremotebusiness, the preselected EJB 3.0 JNDI is the fully qualified interface:

```
com.mycompany.myremotebusiness
```

The EJB 3.0 default application bindings are described at the following location: EJB 3.0 application bindings overview.

Integration Designer will use the "short" name as the default JNDI location for EJBs using the version 3.0 programming model.

Note: If the deployed JNDI reference of the target EJB is different from the default JNDI binding location because a custom mapping was used or configured, the target JNDI name must be properly specified. You can specify the name in Integration Designer before deployment, or, for the import binding, you can change the name in the administrative console (after deployment) to match the JNDI name of the target EJB.

For more information on creating EJB bindings, see the section devoted to Working with EJB bindings in the Integration Designer information center.

JAX-WS data handler:

The Enterprise JavaBeans (EJB) import binding uses the JAX-WS data handler to turn request business objects into Java object parameters and to turn the Java object return value into the response business object. The EJB export binding uses the JAX-WS data handler to turn request EJBs into request business objects and to turn the response business object into a return value.

This data handler maps data from the SCA-specified WSDL interface to the target EJB Java interface (and vice versa) using the Java API for XML Web Services (JAX-WS) specification and the Java Architecture for XML Binding (JAXB) specification.

Note: Current support is restricted to the JAX-WS 2.1.1 and JAXB 2.1.3 specifications.

The data handler specified at the EJB binding level is used to perform request, response, fault, and runtime exception processing.

Note: For faults, a specific data handler can be specified for each fault by specifying the faultBindingType configuration property. This overrides the value specified at the EJB binding level.

The JAX-WS data handler is used by default when the EJB binding has a WSDL interface. This data handler cannot be used to transform a SOAP message representing a JAX-WS invocation to a data object.

The EJB import binding uses a data handler to transform a data object into a Java Object array (Object[]). During outbound communications, the following processing takes place:

1. The EJB binding sets the expected type, expected element, and targeted method name in the BindingContext to match those specified in the WSDL.
2. The EJB binding invokes the transform method for the data object requiring data transformation.
3. The data handler returns an Object[] representing the parameters of the method (in the order of their definition within the method).
4. The EJB binding uses the Object[] to invoke the method on the target EJB interface.

The binding also prepares an Object[] to process the response from the EJB invocation.

- The first element in the Object[] is the return value from the Java method invocation.

- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For parameters of type Out, the values must be returned in the response data object.

The data handler processes and transforms values found in the Object[] and then returns a response to the data object.

The data handler supports xs:AnyType, xs:AnySimpleType, and xs:Any along with other XSD data types. To enable support for xs:Any, use the **@XmlAnyElement (lax=true)** for the JavaBeans property in the Java code, as shown in the following example:

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

This makes the property object in TestType an xs:any field. The Java class value used in the xs:any field should have the **@XmlAnyElement** annotation. For example, if Address is the Java class being used to populate the object array, the Address class should have the annotation **@XmlRootElement**.

Note: To customize the mapping from the XSD type to Java types defined by the JAX-WS specification, change the JAXB annotations to fit your business need. The JAX-WS data handler supports xs:any, xs:anyType, and xs:anySimpleType.

The following restrictions are applicable for the JAX-WS data handler:

- The data handler does not include support for the header attribute **@WebParam** annotation.
- The namespace for business object schema files (XSD files) does not include default mapping from the Java package name. The annotation **@XMLSchema** in package-info.java also does not work. The only way to create an XSD with a namespace is to use the **@XmlType** and **@XmlRootElement** annotations. **@XmlRootElement** defines the target namespace for the global element in JavaBeans types.
- The EJB import wizard does not create XSD files for unrelated classes. Version 2.0 does not support the **@XmlSeeAlso** annotation, so if the child class is not referenced directly from the parent class, an XSD is not created. The solution to this problem is to run SchemaGen for such child classes.

  SchemaGen is a command line utility (located in the *WPS_Install_Home*/bin directory) provided to create XSD files for a given bean. These XSDs must be manually copied to the module for the solution to work.

EJB fault selector:

The EJB fault selector determines if an EJB invocation has resulted in a fault, a runtime exception, or a successful response.

If a fault is detected, the EJB fault selector returns the native fault name to the binding runtime so the JAX-WS data handler can convert the exception object into a fault business object.

On a successful (non-fault) response, the EJB import binding assembles a Java object array (Object[]) to return the values.

- The first element in the Object[] is the return value from the Java method invocation.
- The subsequent values represent the input parameters for the method.

This is required to support the In/Out and Out types of parameters.

For exception scenarios, the binding assembles an Object[] and the first element represents the exception thrown by the method.

The fault selector can return any of the following values:

Table 56. Return values

Type	Return value	Description
Fault	**ResponseType.FAULT**	Returned when the passed Object[] contains an exception object.
Runtime exception	**ResponseType.RUNTIME**	Returned if the exception object does not match any of the declared exception types on the method.
Normal response	**ResponseType.RESPONSE**	Returned in all other cases.

If the fault selector returns a value of **ResponseType.FAULT**, the native fault name is returned. This native fault name is used by the binding to determine the corresponding WSDL fault name from the model and to invoke the correct fault data handler.

EJB function selector:

The EJB bindings use an import function selector (for outbound processing) or an export function selector (for inbound processing) to determine the EJB method to call.

Import function selector

For outbound processing, the import function selector derives the EJB method type based on the name of the operation invoked by the SCA component that is wired to the EJB import. The function selector looks for the @WebMethod annotation on the Integration Designer-generated JAX-WS annotated Java class to determine the associated target operation name.

- If the @WebMethod annotation is present, the function selector uses the @WebMethod annotation to determine the correct Java method mapping for the WSDL method.
- If the @WebMethod annotation is missing, the function selector assumes that the Java method name is the same as the invoked operation name.

Note: This function selector is valid only for a WSDL-typed interface on an EJB import, not for a Java-typed interface on an EJB import.

The function selector returns a java.lang.reflect.Method object that represents the method of the EJB interface.

The function selector uses a Java Object array (Object[]) to contain the response from the target method. The first element in the Object[] is a Java method with the name of the WSDL, and the second element in the Object[] is the input business object.

Export function selector

For inbound processing, the export function selector derives the target method to be invoked from the Java method.

The export function selector maps the Java operation name invoked by the EJB client into the name of the operation in the interface of the target component. The method name is returned as a String and is resolved by the SCA runtime depending on the interface type of the target component.

EIS bindings

Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved using EIS exports and EIS imports that support JCA 1.5 resource adapters and Websphere Adapters.

Your SCA components might require that data be transferred to or from an external EIS. When you create an SCA module requiring such connectivity, you will include (in addition to your SCA component) an import or export with an EIS binding for communication with a specific external EIS.

Resource adapters in IBM Integration Designer are used within the context of an import or an export. You develop an import or an export with the external service wizard and, in developing it, include the resource adapter. An EIS import, which lets your application invoke a service on an EIS system, or an EIS export, which lets an application on an EIS system invoke a service developed in IBM Integration Designer, are created with a resource adapter. For example, you would create an import with the JD Edwards adapter to invoke a service on the JD Edwards system.

When you use the external service wizard, the EIS binding information is created for you. You can also use another tool, the assembly editor, to add or modify binding information. See the Accessing external services with adapters for more information.

After the SCA module containing the EIS binding is deployed to the server, you can use the administrative console to view information about the binding or to configure the binding.

EIS bindings overview:

The EIS (enterprise information system) binding, when used with a JCA resource adapter, lets you access services on an enterprise information system or make your services available to the EIS.

The following example shows how an SCA module named ContactSyncModule synchronizes contact information between a Siebel system and an SAP system.

1. The SCA component named ContactSync listens (by way of an EIS application export named Siebel Contact) for changes to Siebel contacts.
2. The ContactSync SCA component itself makes use of an SAP application (through an EIS application import) in order to update the SAP contact information accordingly.

Because the data structures used for storing contacts are different in Siebel and SAP systems, the ContactSync SCA component must provide mapping.

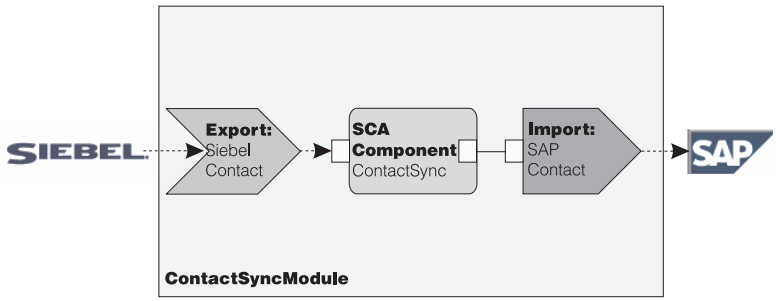


Figure 76. Flow from a Siebel system to an SAP system

The Siebel Contact export and the SAP Contact import have the appropriate resource adapters configured.

Key features of EIS bindings:

An EIS import is a Service Component Architecture (SCA) import that allows components in the SCA module to use EIS applications defined outside the SCA module. An EIS import is used to transfer data from the SCA component to an external EIS; an EIS export is used to transfer data from an external EIS into the SCA module.

Imports

The role of the EIS import is to bridge the gap between SCA components and external EIS systems. External applications can be treated as an EIS import. In this case, the EIS import sends data to the external EIS and optionally receives data in response.

The EIS import provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an external EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

On the client side of the import, there is an interface, exposed by the EIS import application, with one or more methods, each taking data objects as arguments and return values. On the implementation side, there is a Common Client Interface (CCI) implemented by a resource adapter.

The runtime implementation of an EIS import connects the client-side interface and the CCI. The import maps the invocation of the method on the interface to the invocation on the CCI.

Bindings are created at three levels: the interface binding, which then uses the contained method bindings, which in turn use data bindings.

The interface binding relates the interface of the import to the connection to the EIS system providing the application. This reflects the fact that the set of applications, represented by the interface, is provided by the specific instance of the EIS and the connection provides access to this instance. The binding element contains properties with enough information to create the connection (these properties are part of the javax.resource.spi.ManagedConnectionFactory instance).

The method binding associates the method with the specific interaction with the EIS system. For JCA, the interaction is characterized by the set of properties of the javax.resource.cci.InteractionSpec interface implementation. The interaction element of the method binding contains these properties, along with the name of the class, thus providing enough information to perform the interaction. The method binding uses data bindings describing the mapping of the argument and result of the interface method to EIS representation.

The runtime scenario for an EIS import is as follows:

1. The method on the import interface is invoked using the SCA programming model.
2. The request, reaching the EIS import, contains the name of the method and its arguments.
3. The import first creates an interface binding implementation; then, using data from the import binding, it creates a ConnectionFactory and associates the two. That is, the import calls setConnectionFactory on the interface binding.
4. The method binding implementation matching the invoked method is created.
5. The javax.resource.cci.InteractionSpec instance is created and populated; then, data bindings are used to bind the method arguments to a format understood by the resource adapter.
6. The CCI interface is used to perform the interaction.
7. When the call returns, the data binding is used to create the result of the invocation, and the result is returned to the caller.

Exports

The role of the EIS export is to bridge the gap between an SCA component and an external EIS. External applications can be treated as an EIS export. In this case, the external application sends its data in the form of periodic notifications. An EIS export can be thought of as a subscription application listening to an external request from an EIS. The SCA component that uses the EIS export views it as a local application.

The EIS export provides SCA components with a uniform view of the applications external to the module. This allows components to communicate with an EIS, such as SAP, Siebel, or PeopleSoft, using a consistent SCA model.

The export features a listener implementation receiving requests from the EIS. The listener implements a resource adapter-specific listener interface. The export also contains a component implementing interface, exposed to the EIS through the export.

The runtime implementation of an EIS export connects the listener with the component implementing interface. The export maps the EIS request to the invocation of the appropriate operation on the component. Bindings are created at three levels: a listener binding, which then uses a contained native method binding, which in turn uses a data binding .

The listener binding relates the listener receiving requests with the component exposed through the export. The export definition contains the name of the component; the runtime locates it and forwards requests to it.

The native method binding associates the native method or the event type received by the listener to the operation implemented by the component exposed by way of the export. There is no relationship between the method invoked on the listener and the event type; all the events arrive through one or more methods of the listener. The native method binding uses the function selector defined in the export to extract the native method name from the inbound data and data bindings to bind the data format of the EIS to a format understood by the component.

The runtime scenario for an EIS export is as follows:

1. The EIS request triggers invocation of the method on the listener implementation.
2. The listener locates and invokes the export, passing to it all the invocation arguments.
3. The export creates the listener binding implementation.
4. The export instantiates the function selector and sets it on the listener binding.
5. The export initializes native method bindings and adds them to the listener binding. For each native method binding, the data bindings are also initialized.
6. The export invokes the listener binding.
7. The listener binding locates exported components and uses the function selector to retrieve the native method name.
8. This name is used to locate the native method binding, which then invokes the target component.

The adapter interaction style allows for the EIS export binding to invoke the target component either asynchronously (the default) or synchronously.

Resource adapters

You develop an import or an export with the external service wizard and, in developing it, include a resource adapter. The adapters that come with IBM Integration Designer used to access CICS, IMS, JD Edwards, PeopleSoft, SAP and Siebel systems are intended for development and test purposes only. This means you use them to develop and test your applications.

Once you deploy your application, you will need licensed runtime adapters to run your application. However, when you build your service you can embed the adapter with your service. Your adapter licensing might allow you to use the embedded adapter as the licensed runtime adapter. These adapters are compliant with the Java EE Connector Architecture (JCA 1.5). JCA, an open standard, is the Java EE standard for EIS connectivity. JCA provides a managed framework; that is, Quality of Service (QoS) is provided by the application server, which offers lifecycle management and security to transactions. They are also compliant with the Enterprise Metadata Discovery specification with the exception of IBM CICS ECI Resource Adapter and IBM IMS Connector for Java.

The WebSphere Business Integration Adapters, an older set of adapters, are also supported by the wizard.

Java EE resources

The EIS module, an SCA module that follows the EIS module pattern, can be deployed to the Java EE platform.

The deployment of the EIS module to the Java EE platform results in an application that is ready to execute, packaged as an EAR file and deployed to the server. All Java EE artifacts and resources are created; the application is configured and ready to be run.

JCA Interaction Spec and Connection Spec dynamic properties:

The EIS binding can accept input for the InteractionSpec and ConnectionSpec specified by using a well-defined child data object that accompanies the payload. This allows for dynamic request-response interactions with a resource adapter through the InteractionSpec and component authentication through the ConnectionSpec.

The javax.cci.InteractionSpec carries information on how the interaction request with the resource adapter should be handled. It can also carry information on how the interaction was achieved after the request. These two-way communications through the interactions are sometimes referred to as *conversations*.

The EIS binding expects the payload that will be the argument to the resource adapter to contain a child data object called **properties**. This property data object will contain name/value pairs, with the name of the Interaction Spec properties in a specific format. The formatting rules are:

- Names must begin with the prefix **IS**, followed by the property name. For example, an interaction spec with a JavaBeans property called **InteractionId** would specify the property name as **ISInteractionId**.
- The name/value pair represents the name and the value of the simple type of the Interaction Spec property.

In this example, an interface specifies that the input of an operation is an **Account** data object. This interface invokes an EIS import binding application with the intention to send and receive a dynamic InteractionSpec property called **workingSet** with the value **xyz**.

The business graph or business objects in the server contain an underlying **properties** business object that permits the sending of protocol-specific data with the payload. This **properties** business object is built-in and does not need to be specified in the XML schema when constructing a business object. It only needs to be created and used. If you have your own data types defined based on an XML schema, you need to specify a **properties** element that contains your expected name/value pairs.

```
BOFactory dataFactory = (BOFactory) \
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
 //skip to payload for non doc-lit
 DataObject docLitWrapper = dataFactory.createByElement /
 ("http://mytest/eis/Account", "AccountWrapper");
```

Create the payload.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload


  //Construct properties data for dynamic interaction

  DataObject properties = account.createDataObject("properties");
```

For name workingSet, set the value expected (**xyz**).

```
properties.setString("ISworkingSet", "xyz");


  //Invoke the service with argument

  Service accountImport = (Service) \
  serviceManager.locateService("AccountOutbound");
  DataObject result = accountImport.invoke("createAccount", docLitWrapper);

  //Get returned property
  DataObject retProperties = result.getDataObject("properties");

  String workingset = retProperties.getString("ISworkingSet");
```

You can use ConnectionSpec properties for dynamic component authentication. The same rules apply as above, except that the property name prefix needs to be **CS** (instead of **IS**). ConnectionSpec properties are not two-way. The same **properties** data object can contain both IS and CS properties.

To use ConnectionSpec properties, set the **resAuth** specified on the import binding to **Application**. Also, make sure the resource adapter supports component authorization. See chapter 8 of the J2EE Connector Architecture Specification for more details.

External clients with EIS bindings:

The server can send messages to, or receive messages from, external clients using EIS bindings.

An external client, for example a Web portal or an EIS, needs to send a message to an SCA module in the server or needs to be invoked by a component from within the server.

The client invokes the EIS import as with any other application, using either the Dynamic Invocation Interface (DII) or Java interface.

1. The external client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a service interface implementation.
2. The client creates an input argument, a generic data object, created dynamically using the data object schema. This step is done using the Service Data Object DataFactory interface implementation.
3. The external client invokes the EIS and obtains the required results.

Alternatively, the client can invoke the EIS import using the Java interface.

1. The client creates an instance of the ServiceManager and looks up the EIS import using its reference name. The result of the lookup is a Java interface of the EIS import.
2. The client creates an input argument and a typed data object.
3. The client invokes EIS and obtains the required results.

The EIS export interface defines the interface of the exported SCA component that is available to the external EIS applications. This interface can be thought of as the interface that an external application (such as SAP or PeopleSoft) will invoke through the implementation of the EIS export application runtime.

The export uses EISExportBinding to bind exported services to the external EIS application. It allows you to subscribe an application contained in your SCA module to listen for EIS service requests. The EIS export binding specifies the mapping between the definition of inbound events as it is understood by the resource adapter (using Java EE Connector Architecture interfaces) and the invocation of SCA operations.

The EISExportBinding requires external EIS services to be based on Java EE Connector Architecture 1.5 inbound contracts. The EISExportBinding requires that a data handler or data binding be specified either at the binding level or the method level.

JMS bindings

A Java Message Service (JMS) provider enables messaging based on the Java Messaging Service API and programming model. It provides JMS connection factories to create connections for JMS destinations and to send and receive messages.

JMS bindings can be used when you interact with the Service Integration Bus (SIB) provider binding, and are compliant with JMS and JCA 1.5.

You can use the JMS export and import bindings a Service Component Architecture (SCA) module to make calls to, and receive messages from, external JMS systems.

The JMS import and export bindings provide integration with JMS applications using the JCA 1.5-based SIB JMS provider that is part of WebSphere Application Server. Other JCA 1.5–based JMS resource adapters are not supported

JMS bindings overview:

JMS bindings provide connectivity between the Service Component Architecture (SCA) environment and JMS systems.

JMS bindings

The major components of both JMS import and JMS export bindings are:

- Resource adapter: enables managed, bidirectional connectivity between an SCA module and external JMS systems
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Key features of JMS bindings

Special headers

Special header properties are used in JMS imports and exports to tell the target how to handle the message.

For example, TargetFunctionName maps from the native method to the operation method.

Java EE resources

A number of Java EE resources is created when JMS imports and exports are deployed to a Java EE environment.

`ConnectionFactory`
Used by clients to create a connection to the JMS provider.

ActivationSpec
Imports use this for receiving the response to a request; exports use it when configuring the message endpoints that represent message listeners in their interactions with the messaging system.

Destinations
- Send destination: on an import, this is where the request or outgoing message is sent; on an export, this is the destination where the response message will be sent, if not superseded by the JMSReplyTo header field in the incoming message.
- Receive destination: where the incoming message should be placed; with imports, this is a response; with exports, this is a request.
- Callback destination: SCA JMS system destination used to store correlation information. Do not read or write to this destination.

The installation task creates the ConnectionFactory and three destinations. It also creates the ActivationSpec to enable the runtime message listener to listen for replies on the receive destination. The properties of these resources are specified in the import or export file.

JMS integration and resource adapters:

The Java Message Service (JMS) provides integration through an available JMS JCA 1.5-based resource adapter. Complete support for JMS integration is provided for the Service Integration Bus (SIB) JMS resource adapter.

Use a JMS provider for JCA 1.5 resource adapter when you want to integrate with an external JCA 1.5-compliant JMS system. External services compliant with JCA 1.5 can receive messages and send messages to integrate with your service component architecture (SCA) components using the SIB JMS resource adapter.

The use of other provider-specific JCA 1.5 resource adapters is not supported.

JMS import and export bindings:

You can make SCA modules interact with services provided by external JMS applications using JMS import and export bindings.

JMS import bindings

Connections to the associated JMS provider of JMS destinations are created by using a JMS connection factory. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the JMS import binding are supported, depending on the type of operation being invoked:
- One-way: The JMS import puts a message on the send destination configured in the import binding. Nothing is set in the replyTo field of the JMS header.
- Two-way (request-response): The JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default), or from the request message correlation ID. The import binding can also be configured

to use a temporary dynamic response destination to correlate responses with requests. A temporary destination is created for each request and the import uses this destination to receive the response.

The receive destination is set in the replyTo header property of the outbound message. A message listener is deployed to listen on the receive destination, and when a reply is received, the message listener passes the reply back to the component.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that JMS is an asynchronous binding. If a calling component invokes a JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 32 on page 119 illustrates how the import is linked to the external service.

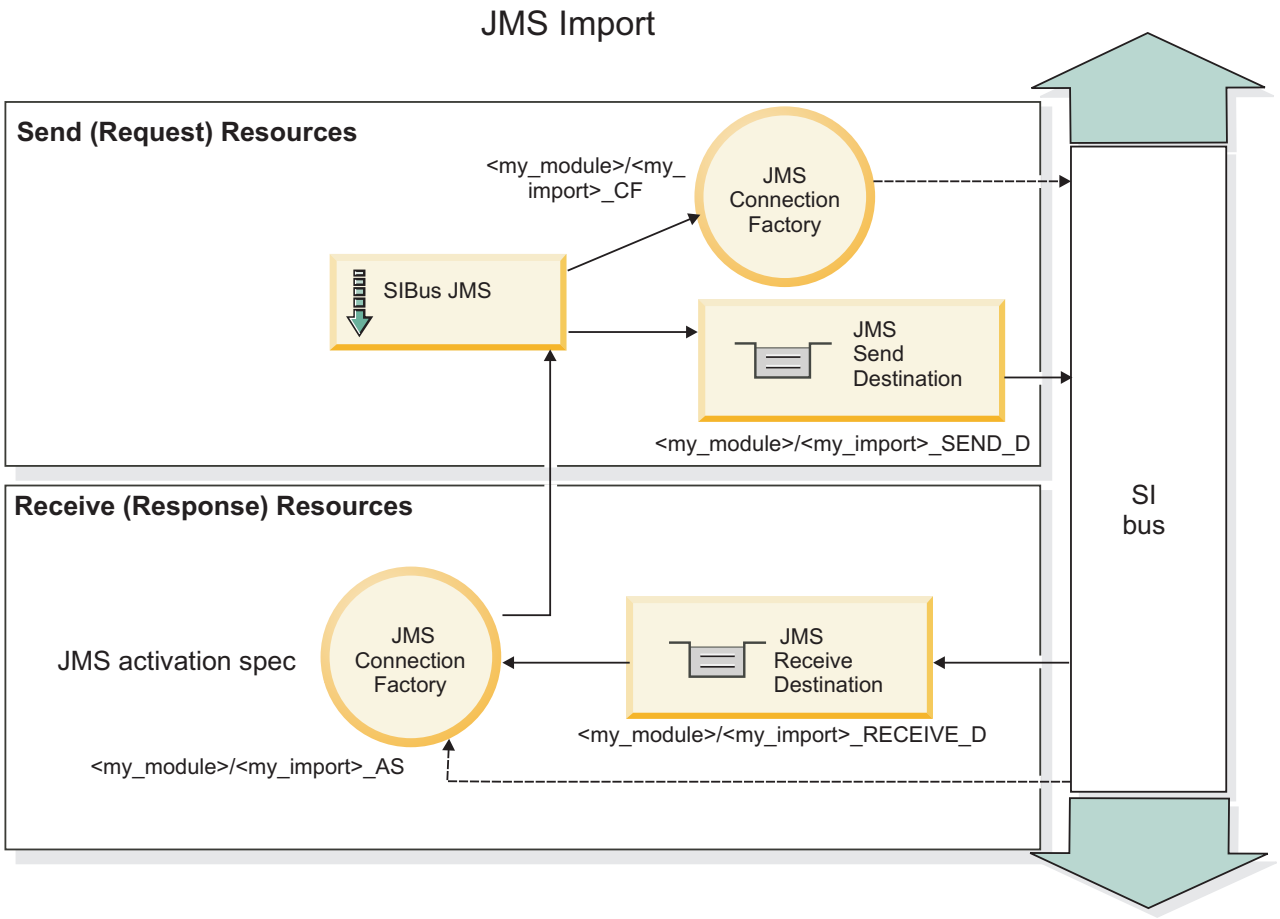


Figure 77. JMS import binding resources

JMS export bindings

JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection that is part of a JMS export is a configurable activation specification.

A JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

A message listener is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the incoming message overrides the destination specified in the send.

Figure 33 on page 120 illustrates how the external requester is linked to the export.

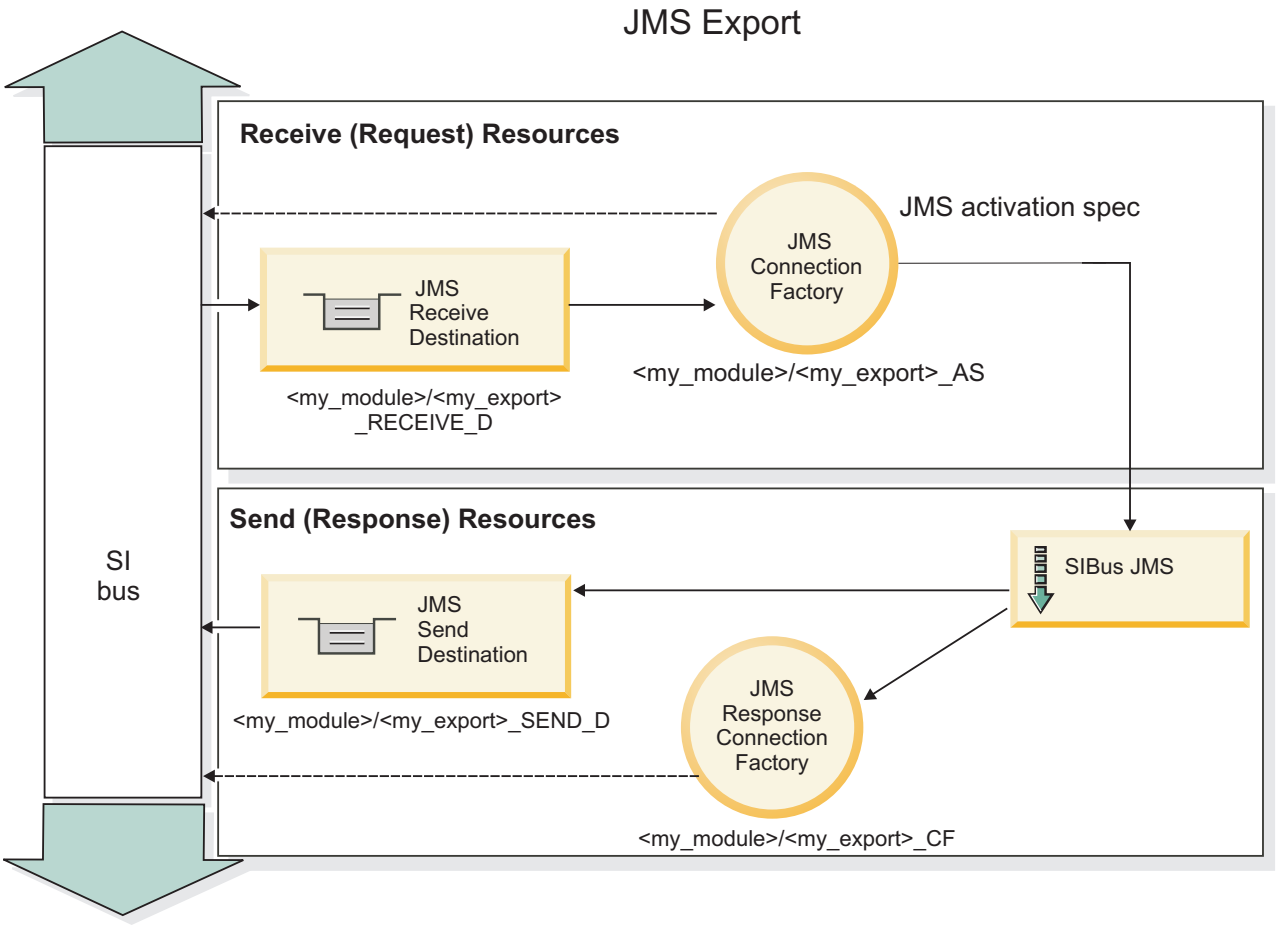


Figure 78. JMS export binding resources

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties
- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

JMS temporary dynamic response destination correlation scheme:

The temporary dynamic response destination correlation scheme causes a unique dynamic queue or topic to be created for each request sent.

The static response destination specified in the import is used to derive the nature of the temporary dynamic destination queue or topic. This is set in the **ReplyTo** field of the request, and the JMS import listens for responses on that destination. When the response is received it is requeued to the static response destination for asynchronous processing. The **CorrelationID** field of the response is not used, and does not need to be set.

Transactional issues

When a temporary dynamic destination is being used, the response must be consumed in the same thread as the sent response. The request must be sent outside the global transaction, and must be committed before it is received by the backend service, and a response returned.

Persistence

Temporary dynamic queues are short-lived entities and do not guarantee the same level of persistence associated with a static queue or topic. A temporary dynamic queue or topic will not survive a server restart and neither will messages. After the message has been requeued to the static response destination it retains the persistence defined in the message.

Timeout

The import waits to receive the response on the temporary dynamic response destination for a fixed amount of time. This time interval will be taken from the SCA Response Expiration time qualifier, if it is set, otherwise the time defaults to 60 seconds. If the wait time is exceeded the import throws a `ServiceTimeoutRuntimeException`.

External clients:

The server can send messages to, or receive messages from, external clients using JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA module in the server, or it can be invoked by a component from within the server.

The JMS export components deploy message listeners to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications with the export binding.

JMS imports interact with external clients by sending messages to, and receiving messages from, JMS queues.

Working with external clients:

An external client (that is, outside the server) might need to interact with an application installed in the server.

Consider a very simple scenario in which an external client wants to interact with an application on the server. The figure depicts a typical simple scenario.

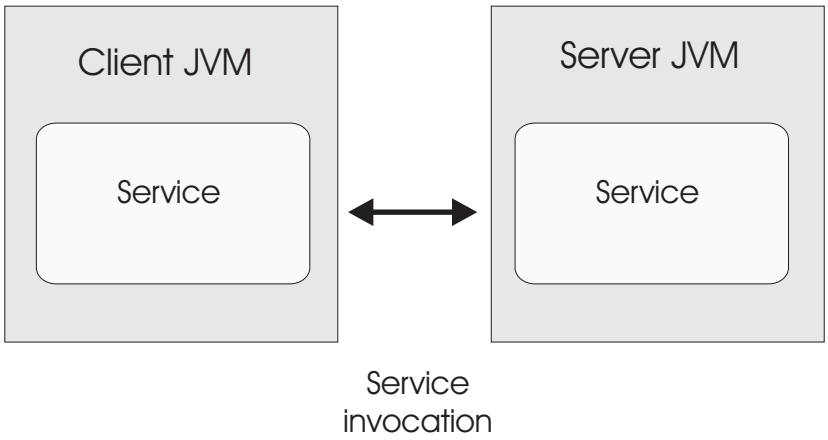


Figure 79. Simple use-case scenario: external client interacts with server application

The SCA application includes an export with a JMS binding; this makes the application available to external clients.

When you have an external client in a Java virtual machine (JVM) separate from your server, there are several steps you must take in order to make a connection and interact with a JMS export. The client obtains an InitialContext with the correct values and then looks up the resources through JNDI. The client then uses the JMS 1.1 specification client to access the destinations and the send and receive messages on the destinations.

The default JNDI names of the resources created automatically by the runtime are listed in the configuration topic of this section. However, if you have pre-created resources, use those JNDI names.

1. Configure JMS destinations and the connection factory to send the message.
2. Make sure that the JNDI context, the port for the SIB resource adapter, and the messaging bootstrapping port are correct.

   The server uses some default ports, but if there are more servers installed on that system, alternate ports are created at installation time to avoid conflicts with other server instances. You can use the administrative console to determine which ports your server is employing . Go to **Servers** > **Application Servers** > ***your_server_ name*** > **Configuration** and click **Ports** under **Communication**. You can then edit the port being used.
3. The client obtains an initial context with the correct values and then looks up the resources through JNDI.
4. Using JMS 1.1 specifications, the client accesses the destinations and the send and receive messages on the destinations.

Troubleshooting JMS bindings:

You can diagnose and fix problems with JMS bindings.

Implementation exceptions

In response to various error conditions, the JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Primary failure conditions

The primary failure conditions of JMS bindings are determined by transactional semantics, by JMS provider configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the JMS provider or destination.

  A failure to connect to the JMS provider to receive messages will result in the message listener failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the destination until they are successfully retrieved (or expired).

  A failure to connect to the JMS provider to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding or data handler causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected late response messages.

  The import expects only one response message for each request message. Also the valid time period in which a response can be received is determined by the SCA Response Expiration qualifier on the request. When a response arrives or the expiration time is exceeded, the correlation record is deleted. If response messages arrive unexpectedly or arrive late, a Service Runtime Exception is thrown.
- Service timeout runtime exception caused by late response when using the temporary dynamic response destination correlation scheme.

  The JMS import will timeout after a period of time determined by the SCA response expiration qualifier, or if this is not set it will default to 60 seconds.

JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the underlying SIB destination of the JMS destination has a maximum failed deliveries value greater than **1**. Setting this value to **2** or more enables interaction with the failed event manager during SCA invocations for the JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

When any runtime exception, including a **DataBindingException**, is thrown:

- If the mediation flow is configured to be transactional, the JMS message , by default, is stored in the Failed Event Manager for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the Failed Event Manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

Generic JMS bindings

The Generic JMS binding provides connectivity to third-party JMS 1.1 compliant providers. The operation of the Generic JMS bindings is similar to that of JMS bindings.

The service provided through a JMS binding allows a Service Component Architecture (SCA) module to make calls or receive messages from external systems. The system can be an external JMS system.

The Generic JMS binding provides integration with non-JCA 1.5-compliant JMS providers that support JMS 1.1 and implement the optional JMS Application Server Facility. The Generic JMS binding supports those JMS providers (including Oracle AQ, TIBCO, SonicMQ, WebMethods, and BEA WebLogic) that do not support JCA 1.5 but do support the Application Server Facility of the JMS 1.1 specification. The WebSphere embedded JMS provider (SIBJMS), which is a JCA 1.5 JMS provider, is not supported by this binding; when using that provider, use the "JMS bindings" on page 117.

Use this Generic binding when integrating with a non-JCA 1.5-compliant JMS-based system within an SCA environment. The target external applications can then receive messages and send messages to integrate with an SCA component.

Generic JMS bindings overview:

Generic JMS bindings are non-JCA JMS bindings that provide connectivity between the Service Component Architecture (SCA) environment and JMS systems that are compliant with JMS 1.1 and that implement the optional JMS Application Server Facility.

Generic JMS bindings

The major aspects of Generic JMS import and export bindings include the following:
- Listener port: enables non-JCA-based JMS providers to receive messages and dispatch them to a Message Driven Bean (MDB)
- Connections: encapsulate a virtual connection between a client and a provider application
- Destinations: used by a client to specify the target of messages it produces or the source of messages it consumes
- Authentication data: used to secure access to the binding

Generic JMS import bindings

Generic JMS import bindings allow components within your SCA module to communicate with services provided by external non-JCA 1.5-compliant JMS providers.

The connection part of a JMS import is a connection factory. A connection factory, the object a client uses to create a connection to a provider, encapsulates a set of connection configuration parameters defined by an administrator. Each connection factory is an instance of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface.

Interaction with external JMS systems includes the use of destinations for sending requests and receiving replies.

Two types of usage scenarios for the Generic JMS import binding are supported, depending on the type of operation being invoked:

- One-way: The Generic JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The Generic JMS import puts a message on the send destination and then persists the reply it receives from the SCA component.

  The receive destination is set in the replyTo header property of the outbound message. A message driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified. Static properties can be set from the Generic JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that Generic JMS is an asynchronous binding. If a calling component invokes a Generic JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 35 on page 126 illustrates how the import is linked to the external service.

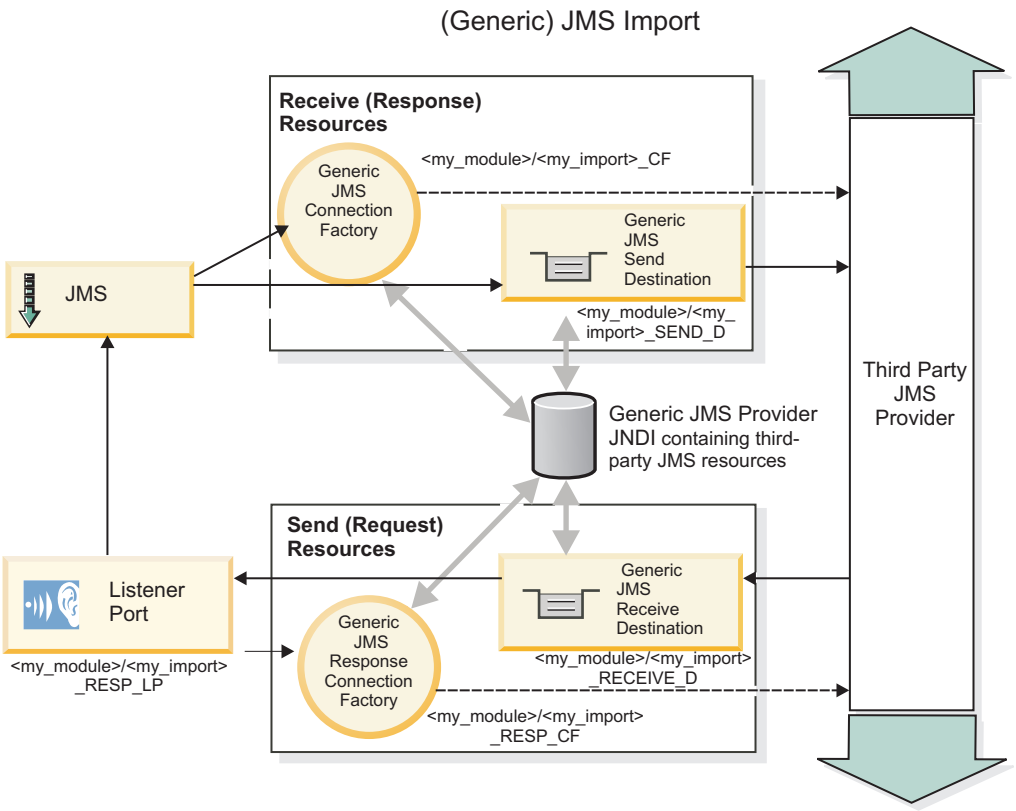


Figure 80. Generic JMS import binding resources

Generic JMS export bindings

Generic JMS export bindings provide the means for SCA modules to provide services to external JMS applications.

The connection part of a JMS export is composed of a ConnectionFactory and a ListenerPort.

A Generic JMS export has send and receive destinations.

- The receive destination is where the incoming message for the target component should be placed.
- The send destination is where the reply will be sent, unless the incoming message has overridden this using the replyTo header property.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding.

- The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply.
- The destination specified in the replyTo field of the incoming message overrides the destination specified in the send field.
- For request/response scenarios, the import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response to copy the request message ID to the correlation ID field of the response message (default), or the response can copy the request correlation ID to the correlation ID field of the response message.

Figure 36 on page 127 illustrates how the external requester is linked to the export.

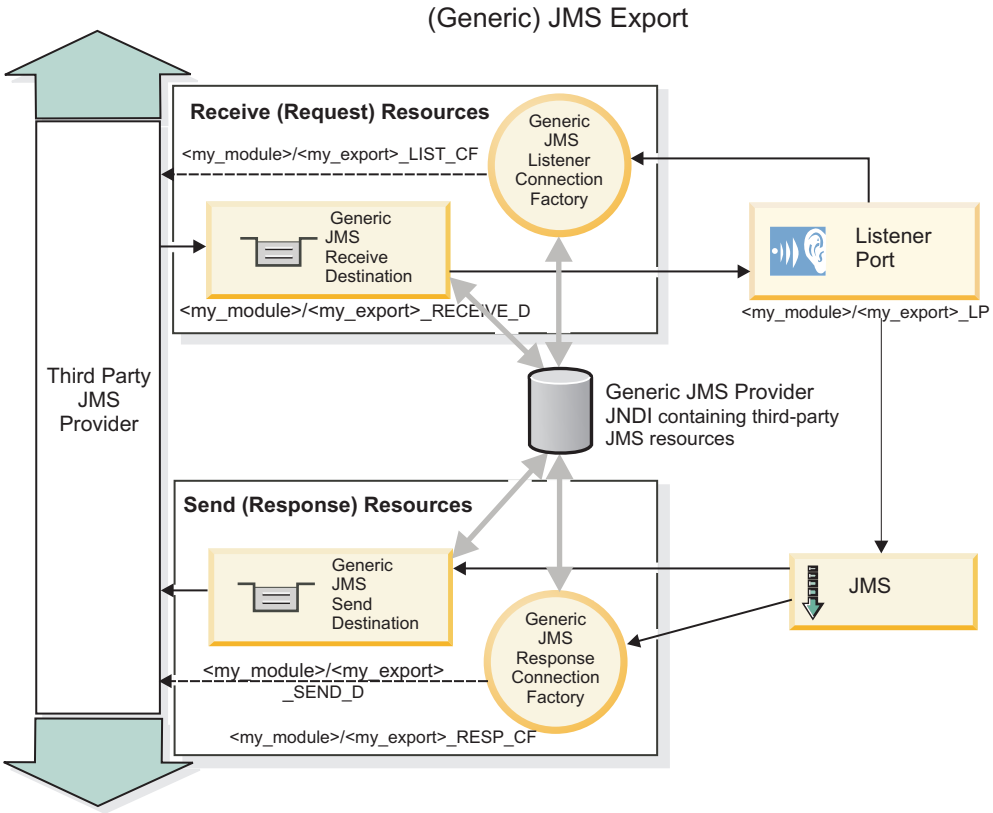


Figure 81. Generic JMS export binding resources

Key features of Generic JMS bindings:

The features of the Generic JMS import and export binding are consistent with those of the WebSphere embedded JMS and MQ JMS import bindings. Key features include header definitions and access to existing Java EE resources. However, because of its generic nature, there are no JMS provider-specific connectivity options, and this binding has limited capability to generate resources at deployment and installation.

Generic imports

Like the MQ JMS import application, the Generic JMS implementation is asynchronous and supports three invocations: one-way, two-way (also known as request-response), and callback.

When the JMS import is deployed, a message driven bean (MDB) provided by the runtime environment is deployed. The MDB listens for replies to the request message. The MDB is associated with (listens on) the destination sent with the request in the replyTo header field of the JMS message.

Generic exports

Generic JMS export bindings differ from EIS export bindings in their handling of the return of the result. A Generic JMS export explicitly sends the response to the replyTo destination specified on the incoming message. If none is specified, the send destination is used.

When the Generic JMS export is deployed, a message driven bean (a different MDB than the one used for Generic JMS imports) is deployed. It listens for the incoming requests on the receive destination and then dispatches the requests to be processed by the SCA runtime.

Special headers

Special header properties are used in Generic JMS imports and exports to tell the target binding how to handle the message.

For example, the TargetFunctionName property is used by the default function selector to identify the name of the operation in the export interface that is being invoked.

Note: The import binding can be configured to set the TargetFunctionName header to the operation name for each operation.

Java EE resources

A number of Java EE resources are created when a JMS binding is deployed to a Java EE environment.

- Listener port for listening on the receive (response) destination (two-way only) for imports and on the receive (request) destination for exports
- Generic JMS connection factory for the outboundConnection (import) and inboundConnection (export)
- Generic JMS destination for the send (import) and receive (export; two-way only) destinations
- Generic JMS connection factory for the responseConnection (two-way only and optional; otherwise, outboundConnection is used for imports, and inboundConnection is used for exports)
- Generic JMS destination for the receive (import) and send (export) destination (two-way only)
- Default messaging provider callback JMS destination used to access the SIB callback queue destination (two-way only)
- Default messaging provider callback JMS connection factory used to access the callback JMS destination (two-way only)
- SIB callback queue destination used to store information about the request message for use during response processing (two-way only)

The installation task creates the ConnectionFactory, the three destinations, and the ActivationSpec from the information in the import and export files.

Generic JMS headers:

Generic JMS headers are Service Data Objects (SDO) that contain all the properties of the Generic JMS message properties. These properties can be from the inbound message or they can be the properties that will be applied to the outbound message.

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

The following properties are set statically on the methodBinding:

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

The Generic JMS binding also supports dynamic modification of JMS headers and properties in the same manner as the JMS and MQ JMS bindings.

Some Generic JMS providers place restrictions on which properties can be set by the application and in what combinations. You must consult your third-party product documentation for more information. However, an additional property has been added to the methodBinding, ignoreInvalidOutboundJMSProperties, which allows any exceptions to be propagated.

The Generic JMS headers and message properties are used only when the base service component architecture SCDL binding switch is turned on. When the switch is turned on, context information is propagated. By default, this switch is on. To prevent context information propagation, change the value to **false**.

When context propagation is enabled, header information is allowed to flow to the message or to the target component. To turn on and off context propagation, specify **true** or **false** for the contextPropagationEnabled attribute of the import and export bindings. For example:

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

The default is **true**.

Troubleshooting Generic JMS bindings:

You can diagnose and fix problems with Generic JMS binding.

Implementation exceptions

In response to various error conditions, the Generic JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

Troubleshooting Generic JMS message expiry

A request message by the JMS provider is subject to expiration.

Request expiry refers to the expiration of a request message by the JMS provider when the JMSExpiration time on the request message is reached. As with other JMS bindings, the Generic JMS binding handles the request expiry by setting expiration on the callback message placed by the import to be the same as for the outgoing request. Notification of the expiration of the callback message will indicate that the request message has expired and the client should be notified by means of a business exception.

If the callback destination is moved to the third-party provider, however, this type of request expiry is not supported.

Response expiry refers to the expiration of a response message by the JMS provider when the JMSExpiration time on the response message is reached.

Response expiry for the generic JMS binding is not supported, because the exact expiry behavior of a third-party JMS provider is not defined. You can, however, check that the response is not expired if and when it is received.

For outbound request messages, the JMSExpiration value will be calculated from the time waited and from the requestExpiration values carried in the asyncHeader, if set.

Troubleshooting Generic JMS connection factory errors

When you define certain types of connection factories in your Generic JMS provider, you might receive an error message when you try to start an application. You can modify the external connection factory to avoid this problem.

When launching an application, you might receive the following error message:

```
MDB Listener Port JMSConnectionFactory type does not match
JMSDestination type
```

This problem can arise when you are defining external connection factories. Specifically, the exception can be thrown when you create a JMS 1.0.2 Topic Connection Factory, instead of a JMS 1.1 (unified) Connection Factory (that is, one that is able to support both point-to-point and publish/subscribe communication).

To resolve this issue, take the following steps:

1. Access the Generic JMS provider that you are using.
2. Replace the JMS 1.0.2 Topic Connection Factory that you defined with a JMS 1.1 (unified) Connection Factory.

When you launch the application with the newly defined JMS 1.1 Connection Factory, you should no longer receive an error message.

Generic JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a generic JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than 1. Setting this value to 1 or more enables interaction with the failed event manager during SCA invocations for the generic JMS bindings.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

WebSphere MQ JMS bindings

The WebSphere MQ JMS binding provides integration with external applications that use a WebSphere MQ JMS-based provider.

Use the WebSphere MQ JMS export and import bindings to integrate directly with external JMS or MQ JMS systems from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ JMS-based service by way of an import, the WebSphere MQ JMS import binding utilizes a destination to which data will be sent and a destination where the reply can be received. Conversion of the data to and from a JMS message is accomplished through the JMS data handler or data binding edge component.

When an SCA module provides a service to WebSphere MQ JMS clients, the WebSphere MQ JMS export binding utilizes a destination where the request can be received and the response can be sent. The conversion of the data to and from a JMS message is done through the JMS data handler or data binding.

The function selector provides a mapping to the operation on the target component to be invoked.

WebSphere MQ JMS bindings overview:

The WebSphere MQ JMS binding provides integration with external applications that use the WebSphere MQ JMS provider.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ JMS bindings will use, before running an application containing these bindings.

WebSphere MQ JMS import bindings

The WebSphere MQ JMS import allows components within your SCA module to communicate with services provided by WebSphere MQ JMS-based providers. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Two types of usage scenarios for WebSphere MQ JMS import bindings are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ JMS import puts a message on the send destination configured in the import binding. Nothing is sent to the replyTo field of the JMS header.
- Two-way (request-response): The WebSphere MQ JMS import puts a message on the send destination.

  The receive destination is set in the replyTo header field. A message-driven bean (MDB) is deployed to listen on the receive destination, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field in Integration Designer) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

For both one-way and two-way usage scenarios, dynamic and static header properties can be specified.Static properties can be set from the JMS import method binding. Some of these properties have special meanings to the SCA JMS runtime.

It is important to note that WebSphere MQ JMS is an asynchronous binding. If a calling component invokes a WebSphere MQ JMS import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the JMS service.

Figure 37 on page 133 illustrates how the import is linked to the external service.

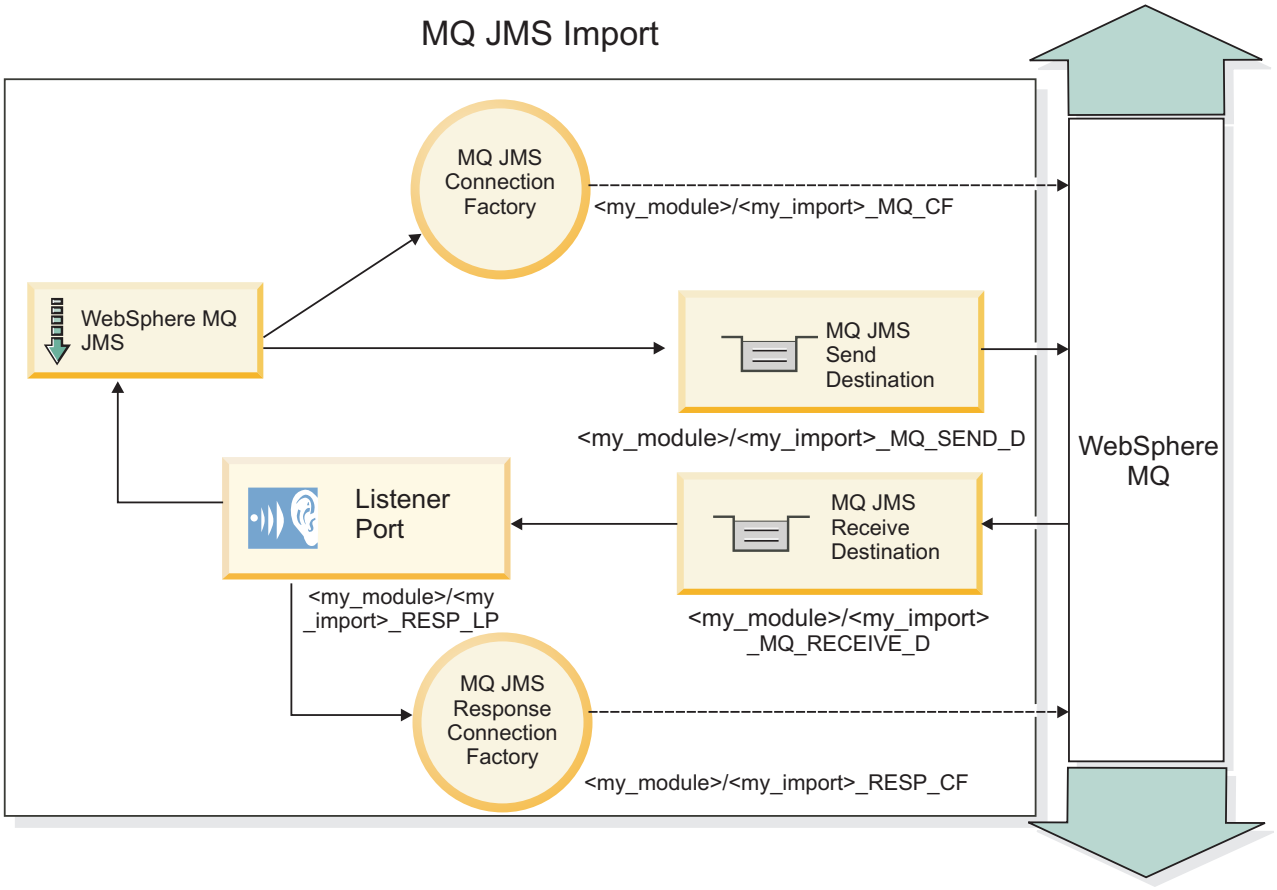


Figure 82. WebSphere MQ JMS import binding resources

WebSphere MQ JMS export bindings

The WebSphere MQ JMS export binding provides the means for SCA modules to provide services to external JMS applications on the WebSphere MQ-based JMS provider.

An MDB is deployed to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked component provides a reply. The destination specified in the replyTo field of the response message overrides the destination specified in the send field.

Figure 38 on page 134 illustrates how the external requester is linked to the export.

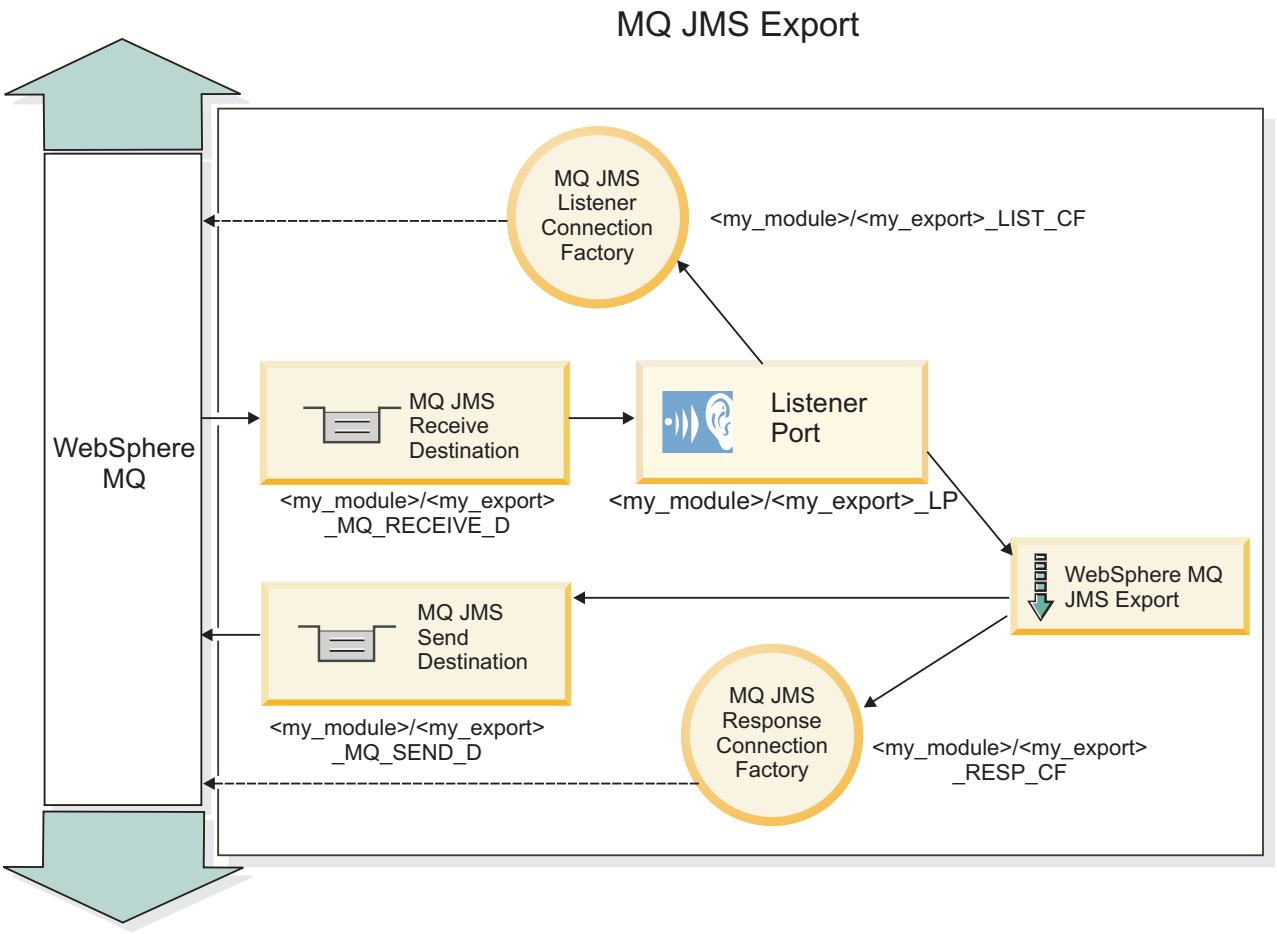


Figure 83. WebSphere MQ JMS export binding resources

Note: Figure 37 on page 133 and Figure 38 on page 134 illustrate how an application from a previous version of IBM Business Process Manager is linked to an external service. For applications developed for IBM Business Process Manager Version 7.0, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of WebSphere MQ JMS bindings:

Key features of WebSphere MQ JMS bindings include headers, Java EE artifacts, and created Java EE resources.

Headers

A JMS message header contains a number of predefined fields containing values used by both clients and providers to identify and to route messages. You can use binding properties to configure these headers with fixed values, or the headers can be specified dynamically at runtime.

`JMSCorrelationID`
Links to a related message. Typically, this field is set to the message identifier string of the message that is being replied to.

`TargetFunctionName`
This header is used by one of the supplied function selectors to identify the operation being invoked. Setting the TargetFunctionName JMS header property in messages sent to a JMS export allows this function selector to be used. The property can be set directly in JMS client applications or when

connecting an import with a JMS binding to such an export. In this case, the JMS import binding should be configured to set the TargetFunctionName header for each operation in the interface to the name of the operation.

Correlation schemes

The WebSphere MQ JMS bindings provide various correlation schemes that are used to determine how to correlate request messages with response messages.

RequestMsgIDToCorrelID
The JMSMessageID is copied to the JMSCorrelationID field. This is the default setting.

RequestCorrelIDToCorrelID
The JMSCorrelationID is copied to the JMSCorrelationID field.

Java EE resources

A number of Java EE resources are created when an MQ JMS import is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the MQ JMS provider.

Response Connection Factory
Used by the SCA MQ JMS runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations

- Send destination:
  - Imports: Where the request or outgoing message is sent.
  - Exports: Where the response message will be sent if it is not superseded by the JMSReplyTo header field of the incoming message.
- Receive destination:
  - Imports: Where the response or incoming message should be placed.
  - Exports: Where the incoming or request message should be placed.

JMS headers:

A JMS message contains two types of headers–the JMS system header and multiple JMS properties. Both types of headers can be accessed either in a mediation module in the Service Message Object (SMO) or by using the ContextService API.

JMS system header

The JMS system header is represented in the SMO by the JMSHeader element, which contains all the fields typically found in a JMS header. Although these can be modified in the mediation (or ContextService), some JMS system header fields set in the SMO will not be propagated in the outbound JMS message as they are overridden by system or static values.

The key fields in the JMS system header that can be updated in a mediation (or ContextService) are:

- **JMSType** and **JMSCorrelationID** – values of the specific predefined message header properties

- **JMSDeliveryMode** – values for delivery mode (persistent or nonpersistent; default is persistent)
- **JMSPriority** – priority value (0 to 9; default is JMS_Default_Priority)

JMS properties

JMS properties are represented in the SMO as entries in the Properties list. The properties can be added, updated, or deleted in a mediation or by using the ContextService API.

Properties can also be set statically in the JMS binding. Properties that are set statically override settings (with the same name) that are set dynamically.

User properties propagated from other bindings (for example, an HTTP binding) will be output in the JMS binding as JMS properties.

Header propagation settings

The propagation of the JMS system header and properties either from the inbound JMS message to downstream components or from upstream components to the outbound JMS message can be controlled by the Propagate Protocol Header flag on the binding.

When Propagate Protocol Header is set, header information is allowed to flow to the message or to the target component, as described in the following list:

- JMS export request

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.
- JMS export response

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS export binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS export binding.
- JMS import request

  Any JMS header fields set in the context service will be used in the outbound message, if not overridden by static properties set on the JMS import binding. Any properties set in the context service will be used in the outbound message if not overridden by static properties set on the JMS import binding.
- JMS import response

  The JMS header received in the message will be propagated to target components by way of the context service. JMS properties received in the message will be propagated to target components by way of the context service.

External clients:

The server can send messages to, or receive messages from, external clients using WebSphere MQ JMS bindings.

An external client (such as a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ JMS export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications via the export binding.

WebSphere MQ JMS imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ JMS bindings:

You can diagnose and fix problems with WebSphere MQ JMS bindings.

Implementation exceptions

In response to various error conditions, the MQ JMS import and export implementation can return one of two types of exceptions:

- Service Business Exception: this exception is returned if the fault specified on the service business interface (WSDL port type) occurred.
- Service Runtime Exception: raised in all other cases. In most cases, the cause exception will contain the original exception (JMSException).

  For example, an import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

WebSphere MQ JMS-based SCA messages not appearing in the failed event manager

If SCA messages originated through a WebSphere MQ JMS interaction fail, you would expect to find these messages in the failed event manager. If such messages are not appearing in the failed event manager, ensure that the value of the maximum retries property on the underlying listener port is equal to or greater than **1**. Setting this value to **1** or more enables interaction with the failed event manager during SCA invocations for the MQ JMS bindings.

Misusage scenarios: comparison with WebSphere MQ bindings

The WebSphere MQ JMS binding is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model. The WebSphere MQ import and export, however, are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data handler or data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

WebSphere MQ bindings

The WebSphere MQ binding provides Service Component Architecture (SCA) connectivity with WebSphere MQ applications.

Use the WebSphere MQ export and import bindings to integrate directly with a WebSphere MQ-based system from your server environment. This eliminates the need to use MQ Link or Client Link features of the Service Integration Bus.

When a component interacts with a WebSphere MQ service by way of an import, the WebSphere MQ import binding uses a queue to which data will be sent and a queue where the reply can be received.

When an SCA module provides a service to WebSphere MQ clients, the WebSphere MQ export binding uses a queue where the request can be received and the response can be sent. The function selector provides a mapping to the operation on the target component to be invoked.

Conversion of the payload data to and from an MQ message is done through the MQ body data handler or data binding. Conversion of the header data to and from an MQ message is done through the MQ header data binding.

For information about the WebSphere MQ versions supported, see the detailed system requirements web page.

WebSphere MQ bindings overview:

The WebSphere MQ binding provides integration with native MQ-based applications.

WebSphere MQ administrative tasks

The WebSphere MQ system administrator is expected to create the underlying WebSphere MQ Queue Manager, which the WebSphere MQ bindings will use, before running an application containing these bindings.

WebSphere administrative tasks

You must set the **Native library path** property of the MQ resource adapter in Websphere to the WebSphere MQ version supported by the server, and restart the server. This ensures that the libraries of a supported version of WebSphere MQ are being used. Detailed hardware and software requirements can be found on the IBM support pages.

WebSphere MQ import bindings

The WebSphere MQ import binding allows components within your SCA module to communicate with services provided by external WebSphere MQ-based applications. You must be using a supported version of WebSphere MQ. Detailed hardware and software requirements can be found on the IBM support pages.

Interaction with external WebSphere MQ systems includes the use of queues for sending requests and receiving replies.

Two types of usage scenarios for the WebSphere MQ import binding are supported, depending on the type of operation being invoked:

- One-way: The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field of the import binding. Nothing is sent to the replyTo field of the MQMD header.
- Two-way (request-response): The WebSphere MQ import puts a message on the queue configured in the **Send destination queue** field

  The receive queue is set in the replyTo MQMD header field. A message driven bean (MDB) is deployed to listen on the receive queue, and when a reply is received, the MDB passes the reply back to the component.

  The import binding can be configured (using the **Response correlation scheme** field) to expect the response message correlation ID to have been copied from the request message ID (the default) or from the request message correlation ID.

It is important to note that WebSphere MQ is an asynchronous binding. If a calling component invokes a WebSphere MQ import synchronously (for a two-way operation), the calling component is blocked until the response is returned by the WebSphere MQ service.

Figure 39 on page 140 illustrates how the import is linked to the external service.

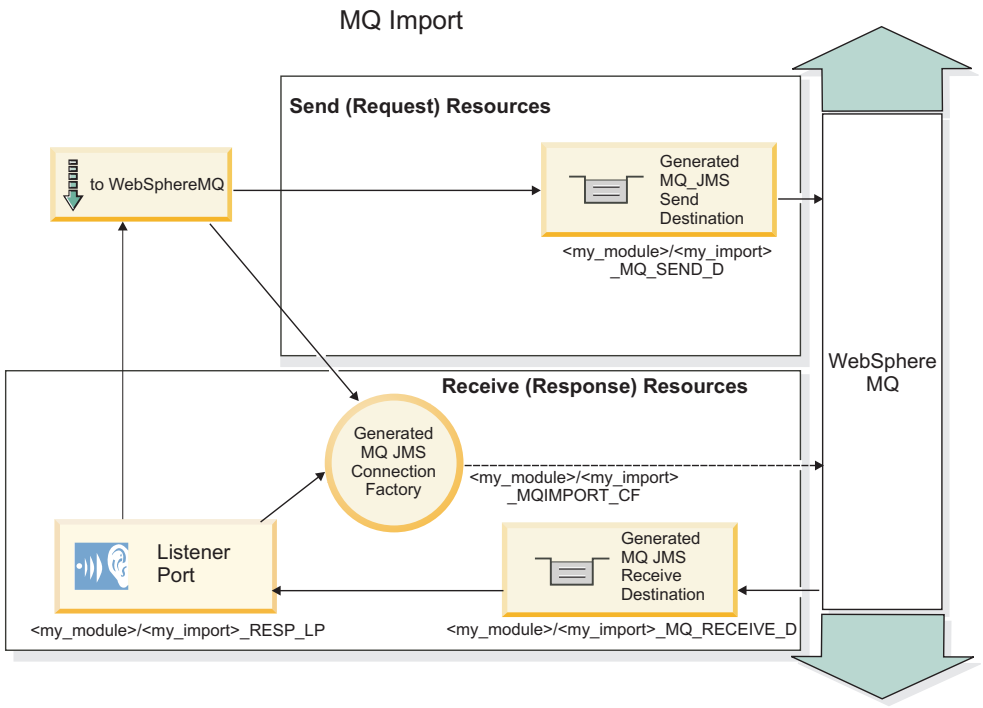


Figure 84. WebSphere MQ import binding resources

WebSphere MQ export bindings

The WebSphere MQ export binding provides the means for SCA modules to provide services to external WebSphere MQ-based applications.

An MDB is deployed to listen to requests incoming to the **Receive destination queue** specified in the export binding. The queue specified in the **Send destination queue** field is used to send the reply to the inbound request if the invoked component provides a reply. The queue specified in the replyTo field of the response message overrides the queue specified in the **Send destination queue** field.

Figure 40 on page 141 illustrates how the external requester is linked to the export.

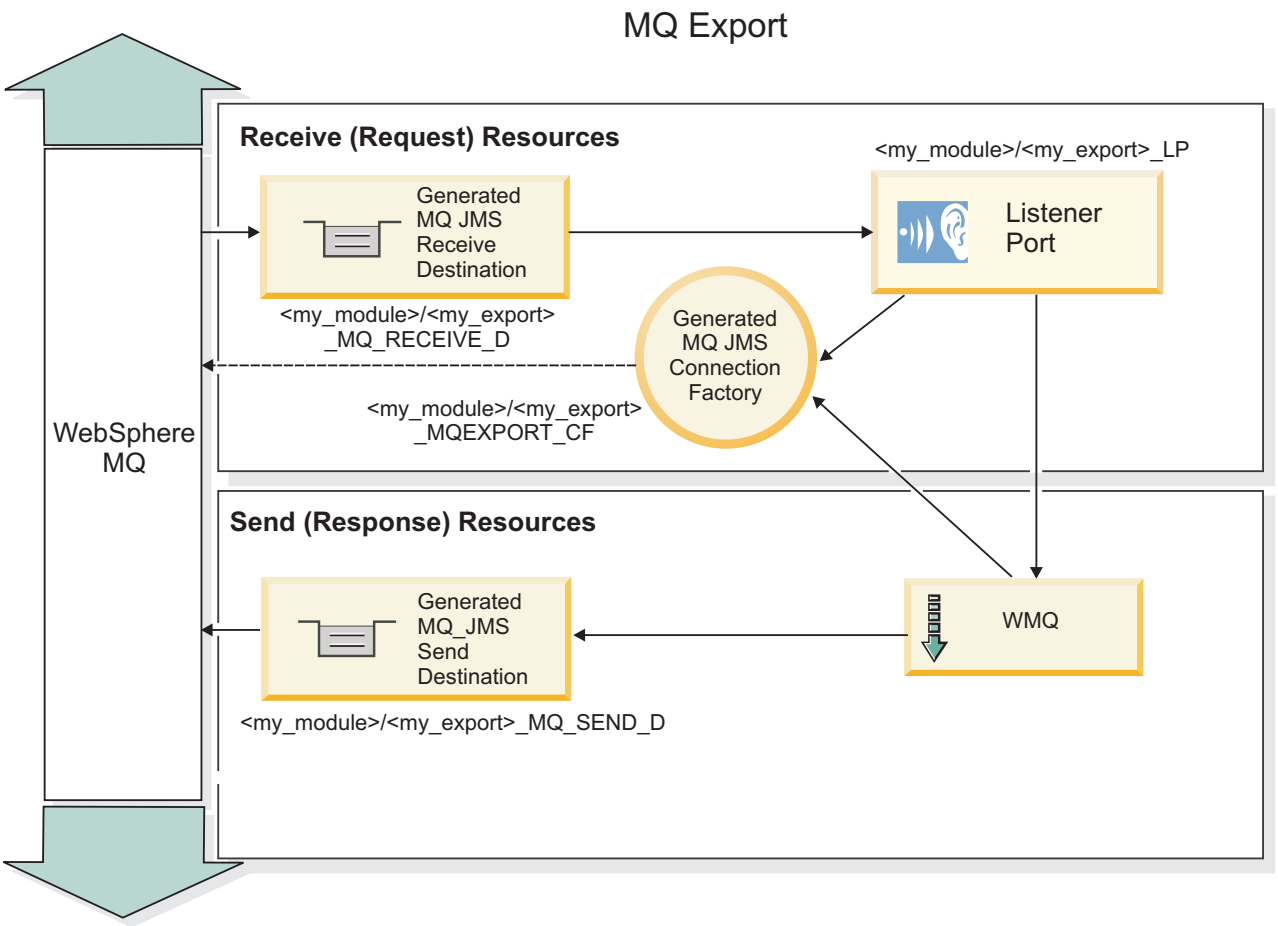


Figure 85. WebSphere MQ export binding resources

Note: Figure 39 on page 140 and Figure 40 on page 141 illustrate how an application from a previous version of IBM Business Process Manager is linked to an external service. For applications developed for IBM Business Process Manager Version 7.x or later, the Activation specification is used instead of the Listener Port and Connection Factory.

Key features of a WebSphere MQ binding:

Key features of a WebSphere MQ binding include headers, Java EE artifacts, and created Java EE resources.

Correlation schemes

A WebSphere MQ request/reply application can use one of a number of techniques to correlate response messages with requests, built around the MQMD MessageID and CorrelID fields. In the vast majority of cases, the requester lets the queue manager select a MessageID and expects the responding application to copy this into the CorrelID of the response. In most cases, the requester and responding application implicitly know which correlation technique is in use. Occasionally the responding application will honor various flags in the Report field of the request that describe how to handle these fields.

Export bindings for WebSphere MQ messages can be configured with the following options:

Response MsgId options:

`New MsgID`
Allows the queue manager to select a unique MsgId for the response (default).

Copy from Request MsgID
Copies the MsgId field from the MsgId field in the request.

Copy from SCA message
Sets the MsgId to be that carried in WebSphere MQ headers in the SCA response message, or lets the queue manager define a new Id if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the MsgId. The MQRO_NEW_MSG_ID and MQRO_PASS_MSG_ID options are supported and behave like New MsgId and Copy from Request MsgID, respectively.

Response CorrelId options:

Copy from Request MsgID
Copies the CorrelId field from the MsgId field in the request (default).

Copy from Request CorrelID
Copies the CorrelId field from the CorrelId field in the request.

Copy from SCA message
Sets the CorrelId to be carried in WebSphere MQ headers in the SCA response message or leaves it blank if the value does not exist.

As Report Options
Inspects the Report field of the MQMD in the request for a hint as to how to handle the CorrelId. The MQRO_COPY_MSG_ID_TO_CORREL_ID and MQRO_PASS_CORREL_ID options are supported and behave like Copy from Request MsgID and Copy from Request CorrelID, respectively.

Import bindings for WebSphere MQ messages can be configured with the following options:

Request MsgId options:

New MsgID
Allows the queue manager to select a unique MsgId for the request (default).

Copy from SCA message
Sets the MsgId to be carried in WebSphere MQ headers in the SCA request message or lets the queue manager define a new Id if the value does not exist.

Response correlation options:

Response has CorrelID copied from MsgId
Expects the response message to have a CorrelId field set, per the MsgId of the request (default).

Response has MsgID copied from MsgId
Expects the response message to have a MsgId field set, per the MsgId of the request.

Response has CorrelID copied from CorrelId
Expects the response message to have a CorrelId field set, per the CorrelId of the request.

Java EE resources

A number of Java EE resources are created when a WebSphere MQ binding is deployed to a Java EE environment.

Parameters

MQ Connection Factory
Used by clients to create a connection to the WebSphere MQ provider.

Response Connection Factory
Used by the SCA MQ runtime when the send destination is on a different Queue Manager than the receive destination.

Activation specification
An MQ JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

Destinations
- Send destination: where the request or outgoing message is sent (import); where the response message will be sent (export), if not superseded by the MQMD ReplyTo header field in the incoming message.
- Receive destination: where the response/request or incoming message should be placed.

WebSphere MQ headers:

WebSphere MQ headers incorporate certain conventions for conversion to the service component architecture (SCA) messages.

WebSphere MQ messages consist of a system header (the MQMD), zero or more other MQ headers (system or custom), and a message body. If multiple message headers exist in the message, the order of the headers is significant.

Each header contains information describing the structure of the following header. The MQMD describes the first header.

How MQ headers are parsed

An MQ Header data binding is used to parse MQ headers. The following headers are supported automatically:
- MQRFH
- MQRFH2
- MQCIH
- MQIIH

Headers that start with **MQH** are handled differently. Specific fields of the header are not parsed; they remain as unparsed bytes.

For other MQ headers, you can write custom MQ header data bindings to parse those headers.

How MQ headers are accessed

MQ headers can be accessed in the product in one of two ways:
- Through the service message object (SMO) in a mediation
- Through the ContextService API

MQ headers are represented internally with the SMO MQHeader element. MQHeader is a container of header data that extends MQControl but contains a value element of anyType. It contains the MQMD, MQControl (MQ message body control information), and a list of other MQ headers.
- MQMD represents the contents of the WebSphere MQ message description, except for information determining the structure and encoding of the body.
- MQControl contains information determining the structure and encoding of a message body.
- MQHeaders contain a list of MQHeader objects.

The MQ header chain is unwound so that, inside the SMO, each MQ header carries its own control information (CCSID, Encoding, and Format). Headers can be added or deleted easily, without altering other header data.

Setting fields in the MQMD

You can update the MQMD using the Context API or through the service message object (SMO) in a mediation. The following fields are automatically propagated to the outbound MQ message:

- Encoding
- CodedCharacterSet
- Format
- Report
- Expiry
- Feedback
- Priority
- Persistence
- CorrelId
- MsgFlags

Configure the MQ binding on an Import or Export to propagate the following properties to the outbound MQ message:

MsgID
Set **Request Message ID** to `copy from SCA message`.

MsgType
Clear the **Set message type to MQMT_DATAGRAM or MQMT_REQUEST for request-response operation** check box.

ReplyToQ
Clear the **Override reply to queue of request message** check box.

ReplyToQMgr
Clear the **Override reply to queue of request message** check box.

From version 7.0 onwards, context fields can be overridden using a custom property on the JNDI destination definition. Set the custom property MDCTX with value `SET_IDENTITY_CONTEXT` on the send destination to propagate the following fields to the outbound MQ message:

- UserIdentifier
- AppIdentityData

Set the custom property MDCTX with value `SET_ALL_CONTEXT` on the send destination to propagate the following properties to the outbound MQ message:

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Some fields are not propagated to the outbound MQ message. The following fields are overridden during the send of the message:

- BackoutCount
- AccountingToken

- PutDate
- PutTime
- Offset
- OriginalLength

Adding MQCIH statically in a WebSphere MQ binding:

IBM Business Process Manager supports adding MQCIH header information statically without using a mediation module.

There are various ways to add MQCIH header information to a message (for example, by using the Header Setter mediation primitive). It might be useful to add this header information statically, without the use of an additional mediation module. Static header information, including the CICS program name, the transaction ID, and other data format header details, can be defined and added as part of the WebSphere MQ binding.

WebSphere MQ, the MQ CICS Bridge, and CICS must be configured for MQCIH header information to be added statically.

You can use Integration Designer to configure the WebSphere MQ import with the static values that are required for the MQCIH header information.

When a message arrives and is processed by the WebSphere MQ import, a check is made to see if MQCIH header information is already present in the message. If the MQCIH is present, the static values defined in the WebSphere MQ import are used to override the corresponding dynamic values in the message. If the MQCIH is not present, one is created in the message and the static values defined in the WebSphere MQ import are added.

The static values defined in the WebSphere MQ import are specific to a method. You can specify different static MQCIH values for different methods within the same WebSphere MQ import.

This facility is not used to provide default values if the MQCIH does not contain specific header information because a static value defined in the WebSphere MQ import will override a corresponding value provided in the incoming message.

External clients:

IBM Business Process Manager can send messages to, or receive messages from, external clients using WebSphere MQ bindings.

An external client (for example, a Web portal or an enterprise information system) can send a message to an SCA component in the application by way of an export or it can be invoked by an SCA component in the application by way of an import.

The WebSphere MQ export binding deploys message driven beans (MDBs) to listen to requests incoming to the receive destination specified in the export binding. The destination specified in the send field is used to send the reply to the inbound request if the invoked application provides a reply. Thus, an external client is able to invoke applications by way of the export binding.

WebSphere MQ imports bind to, and can deliver message to, external clients. This message might or might not demand a response from the external client.

More information on how to interact with external clients using WebSphere MQ can be found at the WebSphere MQ information center.

Troubleshooting WebSphere MQ bindings:

You can diagnose and fix faults and failure conditions that occur with WebSphere MQ bindings.

Primary failure conditions

The primary failure conditions of WebSphere MQ bindings are determined by transactional semantics, by WebSphere MQ configuration, or by reference to existing behavior in other components. The primary failure conditions include:

- Failure to connect to the WebSphere MQ queue manager or queue.

  A failure to connect to WebSphere MQ to receive messages will result in the MDB Listener Port failing to start. This condition will be logged in the WebSphere Application Server log. Persistent messages will remain on the WebSphere MQ queue until they are successfully retrieved (or expired by WebSphere MQ).

  A failure to connect to WebSphere MQ to send outbound messages will cause rollback of the transaction controlling the send.
- Failure to parse an inbound message or to construct an outbound message.

  A failure in the data binding causes rollback of the transaction controlling the work.
- Failure to send the outbound message.

  A failure to send a message causes rollback of the relevant transaction.
- Multiple or unexpected response messages.

  The import expects only one response message for each request message. If more than one response arrives, or if a late response (one for which the SCA response expiration has expired) arrives, a Service Runtime Exception is thrown. The transaction is rolled back, and the response message is backed out of the queue or handled by the failed event manager.

Misusage scenarios: comparison with WebSphere MQ JMS bindings

The WebSphere MQ import and export are principally designed to interoperate with native WebSphere MQ applications and expose the full content of the WebSphere MQ message body to mediations. The WebSphere MQ JMS binding, however, is designed to interoperate with JMS applications deployed against WebSphere MQ, which exposes messages according to the JMS message model.

The following scenarios should be built using the WebSphere MQ JMS binding, not the WebSphere MQ binding:

- Invoking a JMS message-driven bean (MDB) from an SCA module, where the MDB is deployed against the WebSphere MQ JMS provider. Use a WebSphere MQ JMS import.
- Allowing the SCA module to be called from a Java EE component servlet or EJB by way of JMS. Use a WebSphere MQ JMS export.
- Mediating the contents of a JMS MapMessage, in transit across WebSphere MQ. Use a WebSphere MQ JMS export and import in conjunction with the appropriate data binding.

There are situations in which the WebSphere MQ binding and WebSphere MQ JMS binding might be expected to interoperate. In particular, when you are bridging between Java EE and non-Java EE WebSphere MQ applications, use a WebSphere MQ export and WebSphere MQ JMS import (or vice versa) in conjunction with appropriate data bindings or mediation modules (or both).

Undelivered messages

If WebSphere MQ cannot deliver a message to its intended destination (because of configuration errors, for example), it sends the messages instead to a nominated dead-letter queue.

In doing so, it adds a dead-letter header to the start of the message body. This header contains the failure reasons, the original destination, and other information.

MQ-based SCA messages not appearing in the failed event manager

If SCA messages originated because of a WebSphere MQ interaction failure, you would expect to find these messages in the failed event manager. If these messages are not showing in the failed event manager, check that the underlying WebSphere MQ destination has a maximum failed deliveries value greater than 1. Setting this value to 2 or more allows interaction with the failed event manager during SCA invocations for the WebSphere MQ bindings.

MQ failed events are replayed to the wrong queue manager

When a predefined connection factory is to be used for outbound connections, the connection properties must match those defined in the activation specification used for inbound connections.

The predefined connection factory is used to create a connection when replaying a failed event and must therefore be configured to use the same queue manager from which the message was originally received.

Handling exceptions:

The way in which the binding is configured determines how exceptions that are raised by data handlers or data bindings are handled. Additionally, the nature of the mediation flow dictates the behavior of the system when such an exception is thrown.

A variety of problems can occur when a data handler or data binding is called by your binding. For example, a data handler might receive a message that has a corrupt payload, or it might try to read a message that has an incorrect format.

The way your binding handles such an exception is determined by how you implement the data handler or data binding. The recommended behavior is that you design your data binding to throw a **DataBindingException**.

The situation is similar for a data handler. Since the data handler is invoked by the data binding, any data handler exception is wrapped into a data binding exception. Therefore a **DataHandlerException** is reported to you as a **DataBindingException**.

When any runtime exception, including a **DataBindingException** exception, is thrown:

- If the mediation flow is configured to be transactional, the JMS message is stored in the Failed Event Manager by default for manual replay or deletion.

  Note: You can change the recovery mode on the binding so that the message is rolled back instead of being stored in the failed event manager.
- If the mediation flow is not transactional, the exception is logged and the message is lost.

The situation is similar for a data handler. Because the data handler is called by the data binding, a data handler exception is produced inside a data binding exception. Therefore, a **DataHandlerException** is reported to you as a **DataBindingException**.

Limitations of bindings

The bindings have some limitations in their use that are listed here.

Limitations of the MQ binding:

The MQ binding has some limitations in its use that are listed here.

No publish-subscribe message distribution

The publish-subscribe method of distributing messages is not currently supported by the MQ binding though WMQ itself supports publish-subscribe. However, the MQ JMS binding does support this method of distribution.

Shared receive queues

Multiple WebSphere MQ export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each MQ import must have a different receive queue because the MQ import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each MQ export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- MQ imports and exports can point to the same send queue.

Limitations of the JMS, MQ JMS, and generic JMS bindings:

The JMS and MQ JMS bindings have some limitations.

Implications of generating default bindings

The limitations of using the JMS, MQ JMS, and generic JMS bindings are discussed in the following sections:

- Implications of generating default bindings
- Response correlation scheme
- Bidirectional support

When you generate a binding, several fields will be filled in for you as defaults, if you do not choose to enter the values yourself. For example, a connection factory name will be created for you. If you know that you will be putting your application on a server and accessing it remotely with a client, you should at binding creation time enter JNDI names rather than take the defaults since you will likely want to control these values through the administrative console at run time.

However, if you did accept the defaults and then find later that you cannot access your application from a remote client, you can use the administrative console to explicitly set the connection factory value. Locate the provider endpoints field in the connection factory settings and add a value such as <server_hostname>:7276 (if using the default port number).

Response correlation scheme

If you use the CorrelationId To CorrelationId response correlation scheme, used to correlate messages in a request-response operation, you must have a dynamic correlation ID in the message.

To create a dynamic correlation ID in a mediation module using the mediation flow editor, add a Mapping mediation primitive before the import with the JMS binding. Open the mapping editor. The known service component architecture headers will be available in the target message. Drag a field containing a unique ID in the source message onto the correlation ID in the JMS header in the target message.

Bidirectional support

Only ASCII characters are supported for Java Naming and Directory Interface (JNDI) names at runtime.

Shared receive queues

Multiple export and import bindings expect that any messages present on their configured receive queue are intended for that export or import. Import and export bindings should be configured with the following considerations:

- Each import binding must have a different receive queue because the import binding assumes all messages on the receive queue are responses to requests that it sent. If the receive queue is shared by more than one import, responses could be received by the wrong import and will fail to be correlated with the original request message.
- Each export should have a different receive queue, because otherwise you cannot predict which export will get any particular request message.
- Imports and exports can point to the same send queue.

Business objects

The computer software industry has developed several programming models and frameworks in which *business objects* provide a natural representation of the business data for application processing.

In general, these business objects:

- Are defined using industry standards
- Transparently map data to database tables or enterprise information systems
- Support remote invocation protocols
- Provide the data programming model foundation for application programming

Process Designer and Integration Designer provide developers with one such common business object model for representing different kinds of business entities from different domains. At development time, this model enables developers to define business objects as XML schema definitions.

At run time, the business data defined by the XML schema definitions is represented as Java Business Objects. In this model, business objects are loosely based on early drafts of the Service Data Object (SDO) specification and provide the complete set of programming model application interfaces required to manipulate business data.

Defining business objects

You define business objects using the business object editor in Integration Designer. The business object editor stores the business objects as XML schema definitions.

Using XML schema to define business objects provides several advantages:

- XML schema provide a standards-based data definition model and a foundation for interoperability between disparate heterogeneous systems and applications. XML schema are used in conjunction with the Web Services Description Language (WSDL) to provide standards-based interface contracts among components, applications, and systems.
- XML schema define a rich data definition model for representing business data. This model includes complex types, simple types, user-defined types, type inheritance, and cardinality, among other features.
- Business objects can be defined by business interfaces and data defined in the Web Services Description Language, as well as by XML schema from industry standards organizations or from other systems and applications. Integration Designer can import these business objects directly.

Integration Designer also provides support for discovering business data in databases and enterprise information systems and then generating the standards-based XML schema business object definition of that business data. Business objects generated in this fashion are often referred to as *application specific business objects* because they mimic the structure of the business data defined in the enterprise information system.

When a process is manipulating data from many different information systems, it can be valuable to transform the disparate representation of business data (for example, CustomerEIS1 and CustomerEIS2 or OrderEIS1 and OrderEIS2) into a single canonical representation (for example, Customer or Order). The canonical representation is often referred to as the *generic business object*.

Business object definitions, particularly for generic business objects, are frequently used by more than one application. To support this reuse, Integration Designer allows business objects to be created in libraries that can then be associated with multiple application modules.

The Web Services Description Language (WSDL) defines the he contracts for the services provided and consumed by a Service Component Architecture (SCA) application module, as well as the contracts used to create the components within an application module. In a contract, a WSDL can represent both the operations and business objects (which are defined by XML schema to represent the business data).

Working with business objects

Service Component Architecture (SCA) provides the framework for defining an application module, the services it provides, the services it consumes, and the composition of components that provide the business logic of the application module. Business objects play an important role in the application, defining the business data that is used to describe the service and component contracts and the business data that the components manipulate.

The following diagram depicts an SCA application module and illustrates many of the places in which the developer works with business objects.

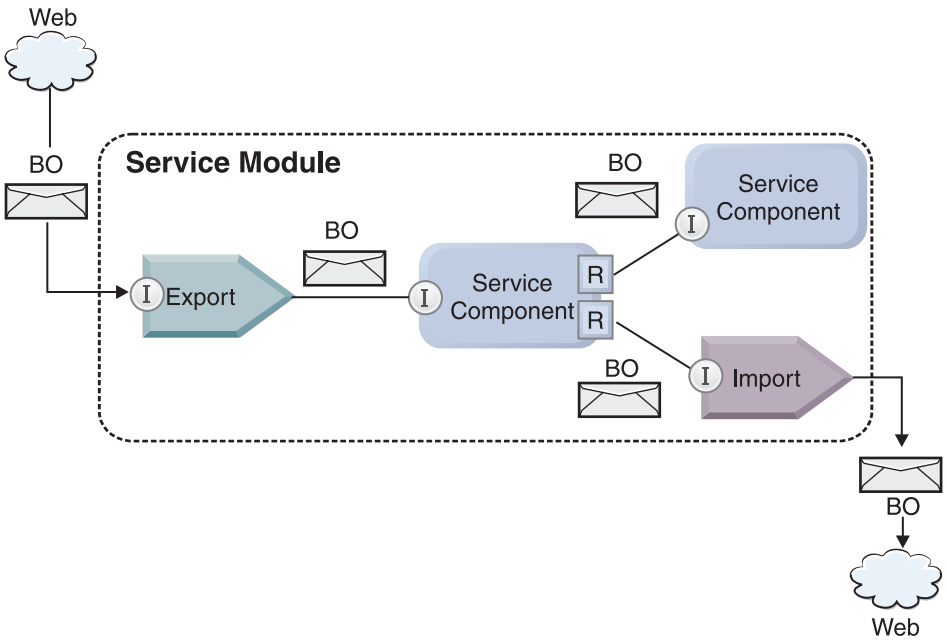


Figure 86. Business objects represent the data that flows between services in an application

Note: This topic describes how business objects are used by SCA application modules. If you are using Java interfaces, the SCA application modules can also process Java objects.

Business object programming model

The business object programming model consists of a set of Java interfaces that represent:

- The business object definition and instance data
- A set of services that support the operations on the business objects

Business object type definitions are represented by the commonj.sdo.Type and commonj.sdo.Property interfaces. The business object programming model provides a set of rules for mapping the XML schema complex type information to the Type interface and each of the elements in the complex type definition to the Property interface.

Business object instances are represented by the commonj.sdo.DataObject interface. The business object programming model is untyped, which means that the same commonj.sdo.DataObject interface can be used to represent different business object definitions, such as Customer and Order. The definition of which properties can be set and retrieved from each business object is determined by type information defined in the XML schema associated with each business object.

The business object programming model behavior is based on the Service Data Object 2.1 specification.

Business object services support various lifecycle operations (such as creation, equality, parsing, and serialization) on business objects.

For specifics on the business object programming model, see Programming using business object services and the Generated API and SPI documentation on business objects.

Bindings, data bindings, and data handlers

As shown in Figure 41 on page 150, business data that is used to invoke services provided by SCA application modules is transformed into business objects so that the SCA components can manipulate the business data. Similarly, the business objects manipulated by SCA components are converted into the data format required by the external services.

In some cases, such as the web service binding, the binding used to export and import services automatically transforms the data into the appropriate format. In other cases, such as the JMS binding, developers can provide a data binding or data handler that converts non-native formats into business objects represented by the DataObject interface.

For more information on developing data bindings and data handlers, refer to “Data handlers” on page 60 and “Data bindings” on page 61.

Components

SCA components define their provision and consumption service contracts using a combination of the Web Services Description Language and XML schema. The business data that SCA passes between components is represented as business objects using the DataObject interface. SCA verifies that these business object types are compatible with the interface contract defined by the component to be invoked.

The programming model abstractions for manipulating business objects vary from component to component. The POJO component and the mediation flow component Custom primitive provide direct manipulating of the business objects by enabling Java programming directly using the business object programming interfaces and services. Most components provide higher level abstractions for manipulating business objects, but also provide snippets of Java code for defining custom behavior in the business object interfaces and services.

Business objects can be transformed using either the combination of the Interface Flow Mediation and Business Object Map component or the mediation flow component and its XML Map primitive. These business object transformation capabilities are useful for converting application specific business objects to and from generic business objects.

Special business objects

Service message objects and business graphs are two specialized types of business objects that are used for specific application purposes.

Service message object

A service message object (SMO) is a specialized business object that is used by mediation flow components to represent the collection of data associated with a service invocation.

A SMO has a fixed top-level structure consisting of headers, context, body, and attachments (if present).

- Headers carry information related to the service invocation over a particular protocol or binding. Examples are SOAP headers and JMS headers.
- Context data carries additional logical information associated with the invocation while it is being processed by the mediation flow component. This information is typically not part of the application data sent or received by clients.
- The body of the SMO carries the payload business data, which represents the core application message or invocation data in the form of a standard business object.

The SMO can also carry attachment data for Web service invocations using SOAP with attachments.

Mediation flows perform such tasks as request routing and data transformation, and the SMO provides the combined view of header and payload contents in a single unified structure.

Business graph

A business graph is a special business object used to provide support for data synchronization in integration scenarios.

Consider an example in which two enterprise information systems have a representation of a specific order. When the order changes in one system, a message can be sent to the other system to synchronize the order data. Business graphs support the notion of sending just the portion of the order that changed to the other system and annotating it with change-summary information to define the type of change.

In this example, an Order business graph would convey to the other system that one of the line items in the order was deleted and that the projected ship date property of the order was updated.

Business graphs can easily be added to existing business objects in Integration Designer. They are most frequently found in scenarios in which WebSphere adapters are being used and to support the migration of WebSphere InterChange Server applications.

Business object parsing mode

Integration Designer provides a property on modules and libraries you can use to configure XML parsing mode for business objects to either eager or lazy.

- If the option is set to *eager*, XML byte streams are eagerly parsed to create the business object.
- If the option is set to *lazy*, the business object is created normally, but the actual parsing of the XML byte stream is deferred and partially parsed only when the business object properties are accessed.

In either XML parsing mode, non-XML data is always eagerly parsed to create the business object.

Considerations when choosing the business object parsing mode

The business object parsing mode determines how XML data is parsed at runtime. A business object parsing mode is defined on a module or library when it is created. You can change the parsing mode of the module or library, however you should be aware of the implications.

The business object parsing mode is set at the module and library level. Modules that were created in a version of IBM Integration Designer prior to version 7 will run in the eager parsing mode without any changes required. By default, modules and libraries that are created in IBM Integration Designer version 7 and later versions will be given the most suitable parsing mode depending on a number of factors such as the parsing mode of existing projects in your workspace, or the parsing mode of dependent projects or other projects in the same solution and so on. You can change the business object parsing mode of a module or library to suit your implementation, however you should be aware of the following considerations.

Considerations

- The lazy business object parsing mode processes XML data faster; however there are compatibility differences between the eager mode and the lazy mode that you need to be aware of before changing the configuration of a module or library. These differences will affect the runtime behavior of the modules. For information on which parsing mode is optimal for your application, see "Benefits of using lazy versus eager parsing mode" in the related links.
- A module can only be configured to run in one parsing mode. Libraries can be configured to support either parsing modes or both parsing modes. A library that is configured to support both parsing modes may be referenced by both a module using the eager parsing mode and a module using the lazy parsing mode. The parsing mode of a library at run time is determined by the modules that reference the library. At runtime, a module declares its parsing mode, and that parsing mode is used by the module and any libraries that the module uses.
- Modules and libraries that are configured for different parsing modes are compatible in the following cases:
  - Modules and libraries configured with the lazy parsing mode are compatible with libraries that use either the lazy parsing mode, or both the eager and the lazy parsing modes.
  - Modules and libraries configured with the eager parsing mode are compatible with libraries that use either the eager parsing mode, or both the eager and the lazy parsing modes.
  - Libraries configured with the lazy and eager parsing modes are compatible only with libraries that use both lazy and eager parsing modes.
- Use the same parsing mode for interacting modules that communicate using the SCA binding. If modules communicate using different parsing modes, performance problems may result.

Related concepts:

“Benefits of using lazy versus eager parsing mode” on page 153
Some applications benefit from lazy XML parsing mode, while others see improved performance with eager parsing mode. It is recommended that you benchmark your application in both parsing modes to determine which mode best suites the specific characteristics of your application.

Benefits of using lazy versus eager parsing mode

Some applications benefit from lazy XML parsing mode, while others see improved performance with eager parsing mode. It is recommended that you benchmark your application in both parsing modes to determine which mode best suites the specific characteristics of your application.

Applications that parse large XML data streams are likely to see performance improvements when the lazy XML parsing mode is used. The performance benefits increase as the size of the XML byte stream increases, and the amount of data from the byte stream that is accessed by the application decreases.

Note: The business object lazy parsing mode is supported in WebSphere Process Server version 7.0.0.3, and later versions. It is also supported in IBM Process Server. Modules and mediation modules that include mediation flow components are not supported.

The following applications are likely to perform better using an eager parsing mode:

- Applications that parse non-XML data streams
- Applications that use messages that are created using the BOFactory service
- Applications that parse very small XML messages

Related reference:

"Considerations when choosing the business object parsing mode" on page 153
The business object parsing mode determines how XML data is parsed at runtime. A business object parsing mode is defined on a module or library when it is created. You can change the parsing mode of the module or library, however you should be aware of the implications.

Application migration and development considerations

If you are configuring an application that was originally developed using eager parsing mode to now use lazy parsing mode, or if you are planning to switch an application between lazy and eager parsing mode, be aware of the differences between modes and the considerations when switching modes.

Error handling

If the XML byte stream being parsed is ill-formed, parsing exceptions occur.

- In eager XML parsing mode, those exceptions occur as soon as the business object is parsed from the inbound XML stream.
- If lazy XML parsing mode is configured, the parsing exceptions occur latently when the business object properties are accessed and the portion of the XML that is ill-formed is parsed.

To deal with ill-formed XML, select one of the following options:

- Deploy an enterprise service bus on the edges to validate inbound XML
- Author lazy error-detection logic at the point where business object properties are accessed

Exception stacks and messages

Because the eager and lazy XML parsing modes have different underlying implementations, stack traces thrown by the business object programming interfaces and services have the same exception class name, but they might not contain the same exception message or wrapped set of implementation-specific exception classes.

XML serialization format

The lazy XML parsing mode provides a performance optimization that attempts to copy unmodified XML from the inbound byte stream to the outbound byte stream upon serialization. The result is increased performance, but the serialization format of the outbound XML byte stream might be different if the entire business object was updated in lazy XML parsing mode or if it was running in eager XML parsing mode.

Although the XML serialization format might not be precisely syntactically equivalent, the semantic value provided by the business object is equivalent independent of the parsing modes, and XML can be safely passed between applications running in different parsing modes with semantic equivalence.

Business object instance validator

The lazy XML parsing business object mode instance validator provides a higher fidelity validation of business objects, particularly facet validation of property values. Because of these improvements, the lazy parsing mode instance validator catches additional issues that are not caught in eager parsing mode and provides more detailed error messages.

Version 602 XML Maps

Mediation flows originally developed before WebSphere Integration Developer Version 6.1 might contain Mapping primitives that use a map or stylesheet that cannot execute directly in lazy XML parsing mode. When an application is migrated for use in lazy XML parsing mode, map files associated with Mapping primitives can be automatically updated by the migration wizard to run in the new mode. However, if a Mapping primitive refers directly to a stylesheet that has been edited manually, the stylesheet is not migrated and cannot execute in lazy XML parsing mode.

Private unpublished APIs

If an application is taking advantage of unpublished, private, implementation-specific business object programming interfaces, the application is likely to fail compilation when the parsing mode is switched. In eager parsing mode, these private interfaces are typically business object implementation classes defined by the Eclipse Modeling Framework (EMF).

In all cases, it is strongly recommend that private APIs be removed from the application.

Service Message Object EMF APIs

A mediation component in IBM Integration Designer provides the ability to manipulate message content using the Java classes and interfaces provided in the com.ibm.websphere.sibx.smobo package. In lazy XML parsing mode, the Java interfaces in the com.ibm.websphere.sibx.smobo package can still be used, but methods that refer directly to Eclipse Modeling Framework (EMF) classes and interfaces or that are inherited from EMF interfaces are likely to fail.

The ServiceMessageObject and its contents cannot be cast to EMF objects in lazy XML parsing mode.

BOMode service

The BOMode service is used to determine whether the currently executing XML parsing mode is eager or lazy.

Migration

All applications before version 7.0.0.0 are running in eager XML parsing mode. When they are runtime migrated using the BPM runtime migration tools, they continue to run in eager XML parsing mode.

To enable an application earlier than version 7.0.0.0 to be configured to use the lazy XML parsing mode, you first use Integration Designer to migrate the artifacts of the application. After migration, you then configure the application to use lazy XML parsing.

See Migrating source artifacts for information on migrating artifacts in Integration Designer, and see Configuring the business object parsing mode of modules and libraries for information on setting the parsing mode.

Relationships

A relationship is an association between two or more data entities, typically business objects. In IBM Business Process Manager Advanced, relationships can be used to transform data that is equivalent across business objects and other data but that is represented differently, or they can be used to draw associations across different objects found in different applications. They can be shared across applications, across solutions, and even across products.

The relationship service in IBM Business Process Manager Advanced provides the infrastructure and operations for managing relationships. Because it enables you to deal with business objects regardless of

where they reside, it can provide a unified holistic view across all applications in an enterprise, and serve as a building block for BPM solutions. Because relationships are extensible and manageable, they can be used in complex integration solutions.

What are relationships?

A relationship is an association between business objects. Each business object in a relationship is called a *participant* in the relationship. Each participant in the relationship is distinguished from other participants based on the function, or *role*, it serves in that relationship. A relationship contains a list of roles.

The relationship *definition* describes each role and specifies how the roles are related. It also describes the overall "shape" of the relationship. For example, this role can have only one participant, but this other role can have as many participants as necessary. You might define a *car-owner* relationship, for instance, where one owner might own multiple cars. For example, one instance could have the following participants for each of these roles:

- Car (Ferrari)
- Owner (John)

The relationship definition is a template for the relationship *instance*. The instance is the run-time instantiation of the relationship. In the *car-owner* example, an instance might describe any of the following associations:

- John owns Ferrari
- Sara owns Mazda
- Bob owns Ferrari

Using relationships frees you from the need to custom build relationship tracking persistence within your business logic. For certain scenarios, the relationship service does all the work for you. See the example described in the section on Identity relationships.

Scenarios

Here is a typical example of a situation in which an integration solution might use relationships. A large corporation buys multiple companies, or business units. Each business unit uses different software to monitor personnel and laptops. The company needs a way to monitor its employees and their laptops. It wants a solution that enables them to:

- View all the employees in the various business units as if they were in one database
- Have a single view of all their laptops
- Allow employees to log on to the system and buy a laptop
- Accommodate the different enterprise application systems in the various business units

To accomplish this, the company needs a way to ensure, for example, that John Smith and John A. Smith in different applications are seen as the same employee. For Example, they need a way to consolidate a single entity across multiple application spaces.

More complex relationship scenarios involve building BPEL processes that draw relationships across different objects found in multiple applications. With complex relationship scenarios, the business objects reside in the integration solution, and not in the applications. The relationship service provides a platform for managing relationships persistently. Before the relationship service, you would have to build your own object persistence service. Two examples of complex relationship scenarios are:

- You have a **car** business object with a VIN number in an SAP application, and you want to track the fact that this car is owned by someone else. However, the ownership relationship is with someone in a PeopleSoft application. In this pattern of relationships, you have two solutions and you need to build a cross-bridge between them.

- A large retail company wants to be able to monitor merchandise returned for cash back or credit. There are two different applications involved: an order management system (OMS) for purchases, and a returns management system (RMS) for returns. The business objects reside in more than one application, and you need a way to show the relationships that exist between them.

Common usage patterns

The most common relationship patterns are *equivalence* patterns. These are based on cross-referencing, or correlation. There are two types of relationships that fit this pattern: *non-identity* and *identity*.

- **Non-identity relationships** establish associations between business objects or other data on a one-to-many or many-to-many basis. For each relationship instance, there can be one or more instances of each participant. One type of non-identity relationship is a static lookup relationship. An example of this is a relationship in which **CA** in an SAP application is related to **California** in a Siebel application.
- **Identity relationships** establish associations between business objects or other data on a one-to-one basis. For each relationship instance, there can be only one instance of each participant. Identity relationships capture cross-references between business objects that are semantically equivalent, but that are identified differently within different applications. Each participant in the relationship is associated with a business object that has a value (or combination of values) that uniquely identifies the object. Identity relationships typically transform the key attributes of business objects, such as ID numbers and product codes.

  For example, if you have **car** business objects in SAP, PeopleSoft, and Siebel applications, and you want to build a solution that synchronizes them, you would normally need to introduce hand-built relationship synchronization logic in six maps:

  SAP -> generic

  generic -> SAP

  PeopleSoft-> generic

  generic-> PeopleSoft

  Siebel-> generic

  generic-> Siebel

  However, if you use relationships in your solution, the relationship service provides prebuilt pattern implementations that maintains all these relationship instances for you.

Tools for working with relationships

The *relationship editor* in Integration Designer is the tool you use to model and design business integration relationships and roles. For detailed background and task information about creating relationships and using the relationship editor, see Creating relationships.

The *relationship service* is an infrastructure service in IBM Business Process Manager that maintains relationships and roles in the system and provides operations for relationship and role management.

The *relationship manager* is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

Relationships can be invoked programmatically through the relationship service APIs.

Relationship service

The relationship service stores relationship data in relationship tables, where it keeps track of application-specific values across applications and across solutions. The relationship service provides operations for relationship and role management.

How relationships work

Relationships and roles are defined using the graphical interface of the relationship editor tool in Integration Designer. The relationship service stores the correlation data in tables in the relationship database in the default data source that you specify when you configure the relationship service. A separate table (sometimes called a participant table) stores information for each participant in the relationship. The relationship service uses these relationship tables to keep track of the related application-specific values and propagate updated information across all the solutions.

Relationships, which are business artifacts, are deployed within a project or in a shared library. At the first deployment, the relationship service populates the data.

At run time, when maps or other IBM Business Process Manager components need a relationship instance, the instances of the relationship are either updated or retrieved, depending on the scenario.

Relationship and role instance data can be manipulated through three means:

- IBM Business Process Manager component Java snippet invocations of the relationship service APIs
- Relationship transformations in the IBM Business Process Manager business object mapping service
- The relationship manager tool

For detailed background and task information on creating relationships, identifying relationship types, and using the relationship editor, see Creating relationships topic.

Relationship manager

The relationship manager is the administrative interface for managing relationships. It is accessed through the Relationship Manager pages of the administrative console.

The relationship manager provides a graphical user interface for creating and manipulating relationship and role data at run time. You can manage relationship entities at all levels: relationship instance, role instance, and attribute data and property data levels. With the relationship manager, you can:

- View a list of the relationships in the system and detailed information for individual relationships
- Manage relationship instances:
  - Query relationship data to view subsets of instance data
  - Query relationship data to view subsets of instance data using database views
  - View a list of relationship instances that match a relationship query and detailed information about an instance
  - Edit the property values for a relationship instance
  - Create and delete relationship instances
- Manage roles and role instances:
  - View details about a role or a role instance
  - Edit role instance properties
  - Create and delete role instances for a relationship
  - Roll back relationship instance data to a point in time when you know the data is reliable
- Import data from an existing static relationship into your system, or export data from an existing static relationship to an RI or CSV file
- Remove relationship schema and data from the repository when the application that uses it is uninstalled

Relationships in Network Deployment environments

Relationships can be used in Network Deployment (ND) environments without any extra configuration.

In Network Deployment (ND) environments, relationships are installed in an application cluster. Relationships are then visible within the cluster, and all servers in the cluster have access to the instance data stored in the relationship database. The ability to run the relationship service in an ND environment makes it scalable and highly available.

The relationship manager allows relationships to be managed across different clusters through a centralized administrative interface. You connect the relationship manager to a server in a cluster by selecting its relationship MBean.

Relationship service APIs

Relationships can be invoked programmatically through the relationship service APIs, within or outside of business object maps.

Three API types are available:

- Relationship instance manipulation APIs (including create, update, delete instance data directly)
- Relationship pattern support APIs (including correlate(), correlateforeignKeyLookup)
- Relationship lookup patterns (lookup APIs)

The enterprise service bus in IBM Business Process Manager

IBM Business Process Manager supports the integration of application services, including the same capabilities as WebSphere Enterprise Service Bus.

Connecting services through an enterprise service bus

With an enterprise service bus (ESB), you can maximize the flexibility of an SOA. Participants in a service interaction are connected to the ESB, rather than directly to one another.

When the service requester connects to the ESB, the ESB takes responsibility for delivering its requests, using messages, to a service provider offering the required function and quality of service. The ESB facilitates requester-provider interactions and addresses mismatched protocols, interaction patterns, or service capabilities. An ESB can also enable or enhance monitoring and management. The ESB provides virtualization and management features that implement and extend the core capabilities of SOA.

The ESB abstracts the following features:

Location and identity
Participants do not have to know the location or identity of other participants. For example, requesters do not have to be aware that a request could be serviced by any of several providers; service providers can be added or removed without disruption.

Interaction protocol
Participants do not have to share the same communication protocol or interaction style. For example, a request expressed as SOAP over HTTP can be serviced by a provider that only understands SOAP over Java Message Service (JMS).

Interface
Requesters and providers do not have to agree on a common interface. An ESB reconciles differences by transforming request and response messages into a form expected by the provider.

Qualities of (interaction) service
Participants, or systems administrators, declare their quality-of-service requirements, including authorization of requests, encryption and decryption of message contents, automatic auditing of service interactions, and how their requests should be routed (for example, optimizing for speed or cost).

Interposing the ESB between participants enables you to modulate their interaction through a logical construct called a *mediation*. Mediations operate on messages in-flight between requesters and providers. For example, mediations can be used to find services with specific characteristics that a requester is asking for, or to resolve interface differences between requesters and providers. For complex interactions, mediations can be chained sequentially.

Using mediations, an enterprise service bus carries out the following actions between requester and service:

- *Routing* messages between services. An enterprise service bus offers a common communication infrastructure that can be used to connect services, and thereby the business functions they represent, without the need for programmers to write and maintain complex connectivity logic.
- *Converting* transport protocols between requester and service. An enterprise service bus provides a consistent, standards-based way to integrate business functions that use different IT standards. This enables integration of business functions that could not normally communicate, such as to connect applications in departmental silos or to enable applications in different companies to participate in service interactions.
- *Transforming* message formats between requester and service. An enterprise service bus enables business functions to exchange information in different formats, with the bus ensuring that the information delivered to a business function is in the format required by that application.
- *Handling* business events from disparate sources. An enterprise service bus supports event-based interactions in addition to the message exchanges to handle service requests.

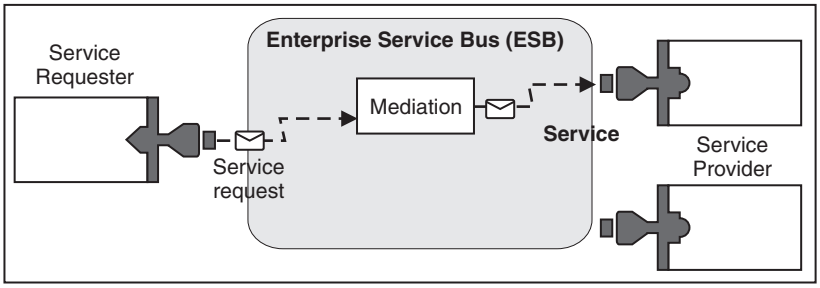


Figure 87. An enterprise service bus. The enterprise service bus is routing messages between applications, which are requesters or providers of services. The bus is converting transport protocols and transforming message formats between requesters and providers. In this figure, each application uses a different protocol (represented by the different geometric shapes of their connectors) and uses different message formats.

By using the enterprise service bus you can focus on your core business rather than your computer systems. You can change or add to the services if required; for example, to respond to changes in the business requirement, to add extra service capacity, or to add new capabilities. You can make the required changes by reconfiguring the bus, with little or no impact to existing services and applications that use the bus.

Enterprise service bus messaging infrastructure

IBM Business Process Manager includes enterprise service bus capabilities. IBM Business Process Manager supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based, messaging infrastructure in an integrated enterprise service bus.

The enterprise service capabilities that you can use for your enterprise applications provide not only a transport layer but mediation support to facilitate service interactions. The enterprise service bus is built around open standards and service-oriented architecture (SOA). It is based on the robust Java EE infrastructure and associated platform services provided by IBM WebSphere Application Server Network Deployment.

IBM Business Process Manager is powered by the same technology available with IBM WebSphere Enterprise Service Bus. This capability is part of the underlying functionality of IBM Business Process Manager, and no additional license for WebSphere Enterprise Service Bus is required to take advantage of these capabilities.

However, you can deploy additional stand-alone licenses of WebSphere Enterprise Service Bus around your enterprise to extend the connectivity reach of the process integration solutions powered by IBM Business Process Manager. For example, WebSphere Enterprise Service Bus can be installed closer to an SAP application to host an IBM WebSphere Adapter for SAP and to transform SAP messages before sending that information across the network to a business process choreographed by IBM Business Process Manager.

Messaging or queue destination hosts

A messaging or queue destination host provides the messaging function within a server. A server becomes the messaging destination host when you configure it as the messaging target.

A messaging engine runs within a server. The messaging engine provides messaging functions and a connection point for applications to connect to the bus. Service Component Architecture (SCA) asynchronous communication, JMS imports and exports, asynchronous internal processing use message queues on the messaging engine.

The deployment environment connects the message source to the message target through the bus when the application modules are deployed. Knowing the message source and message target helps you determine what type of deployment environment you need.

Applications can store persistent data in a data store, which is a set of tables in a database or schema, or in a file store. The messaging engine uses an instance of a JDBC data source to interact with that database.

Configure the messaging destination host when you define your deployment environment by using **Server** from the administrative console or designate the server as the destination host during software installation.

Data stores:

Every messaging engine can use a data store, which is a set of tables in a database or schema that store persistent data.

All of the tables in the data store are held in the same database schema. You can create each data store in a separate database. Alternatively, you can create multiple data stores in the same database, with each data store using a different schema.

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

JDBC providers

You can use JDBC providers to interact applications with relational databases.

Applications use JDBC providers to interact with relational databases. The JDBC provider supplies the specific JDBC driver implementation class for access to a specific type of database. To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the Java EE Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

Refer to the examples of both Typical stand-alone environment setup and Typical deployment environment setup in the previous topic.

For more information on JDBC providers, see “JDBC providers” in the WebSphere Application Server information center.

Service integration buses for IBM Business Process Manager

A service integration bus is a managed communication mechanism that supports service integration through synchronous and asynchronous messaging. A bus consists of interconnecting messaging engines that manage bus resources. It is one of the WebSphere Application Server technologies on which IBM Business Process Manager is based.

Some buses are automatically created for use by the system, the Service Component Architecture (SCA) applications that you deploy, and by other components. You can also create buses to support service integration logic or other applications, for example, to support applications that act as service requesters and providers within IBM Business Process Manager, or to link to WebSphere MQ.

A bus destination is a logical address to which applications can attach as a producer, consumer, or both. A queue destination is a bus destination that is used for point-to-point messaging.

Each bus can have one or more bus members, each of which is either a server or a cluster.

The *bus topology* is the physical arrangement of application servers, messaging engines, and WebSphere MQ queue managers, and the pattern of bus connections and links between them, that makes up your enterprise service bus.

Some service integration buses are created automatically to support IBM Business Process Manager. Up to six buses are created when you create your deployment environment or configure a server or cluster to support SCA applications. These buses each have five authentication aliases that you must configure.

SCA system bus:

The *SCA system bus* is a service integration bus that is used to host queue destinations for Service Component Architecture (SCA) modules. The SCA run time, which supports mediation modules, uses queue destinations on the system bus as an infrastructure to support asynchronous interactions between components and modules.

The system bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The system bus provides a scope within which resources, such as queue destinations, are configured for mediation modules and interaction endpoints. The bus enables message routing between endpoints. You can specify the quality of service for the bus, including priority and reliability.

The bus name is SCA.SYSTEM.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

SCA application bus:

The application bus destinations support the asynchronous communication of WebSphere Business Integration Adapters and other System Component Architecture components.

The application bus is automatically created when you create a deployment environment or when you configure a server or cluster to support SCA applications. The application bus is similar to service integration buses you might create to support service integration logic or other applications.

The bus name is SCA.APPLICATION.busID.Bus. The authentication alias used for securing this bus is SCA_Auth_Alias.

The Common Event Infrastructure bus:

The Common Event Infrastructure bus is used for transmitting common base events, asynchronously, to the configured Common Event Infrastructure server.

The bus name is CommonEventInfrastructure_Bus. The authentication alias used for securing this bus is CommonEventInfrastructureJMSAuthAlias

The Business Process Choreographer bus:

Use the Business Process Choreographer bus name and authentication for internal message transmission.

The Business Process Choreographer bus is used for transmitting messages internally and for business flow manager's Java Messaging Service (JMS) API.

The bus name is BPC.cellName.Bus. The authentication alias is BPC_Auth_Alias

Performance Data Warehouse bus:

The Performance Data Warehouse bus is used for transmitting messages internally by the infrastructure and for communicating with IBM Business Process Manager clients.

The Performance Data Warehouse bus is automatically created when you create a deployment environment.

The bus name is PERFDW.busID.Bus. The authentication alias used for securing this bus is PERFDWME_Auth_Alias.

Process Server bus:

The Process Server bus is used for transmitting messages internally by the infrastructure and for communicating with IBM Business Process Manager clients.

The Process Server bus is automatically created when you create a deployment environment.

The bus name is PROCSVR.busID.Bus. The authentication alias used for securing this bus is PROCSVRME_Auth_Alias.

Service applications and service modules

A service module is a Service Component Architecture (SCA) module that provides services in the run time. When you deploy a service module to IBM Business Process Manager, you build an associated service application that is packaged as an enterprise archive (EAR) file.

Service modules are the basic units of deployment and can contain components, libraries, and staging modules used by the associated service application. Service modules have exports and, optionally, imports to define the relationships between modules and service requesters and providers. WebSphere Process Server supports modules for business services and mediation modules. Both modules and mediation modules are types of SCA modules. 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.

Deploying a service application

The process of deploying an EAR file containing a service application is the same as the process of deploying any EAR file. You can modify values for mediation parameters at deployment time. After you have deployed an EAR file containing an SCA module, you can view details about the service application and its associated module. You can see how a service module is connected to service requesters (through exports) and service providers (through imports).

Viewing SCA module details

The service module details that you can view depend upon the SCA module. They include the following attributes.

- SCA module name
- SCA module description
- Associated application name
- SCA module version information, if the module is versioned
- SCA module imports:
  - Import interfaces are abstract definitions that describe how an SCA module accesses a service.
  - Import bindings are concrete definitions that specify the physical mechanism by which an SCA module accesses a service. For example, using SOAP/HTTP.
- SCA module exports:
  - Export interfaces are abstract definitions that describe how service requesters access an SCA module.
  - Export bindings are concrete definitions that specify the physical mechanism by which a service requester accesses an SCA module, and indirectly, a service.
- SCA module properties

Imports and import bindings

Imports define interactions between SCA modules and service providers. SCA modules use imports to permit components to access external services (services that are outside the SCA module) using a local representation. Import bindings define the specific way that an external service is accessed.

If SCA modules do not need to access external services, they are not required to have imports. Mediation modules usually have one or more imports that are used to pass messages or requests on to their intended targets.

Interfaces and bindings

An SCA module import needs at least one interface, and an SCA module import has a single binding.

- Import interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many import interfaces.
- Import bindings are concrete definitions that specify the physical mechanism that SCA modules use to access an external service.

Supported import bindings

IBM Business Process Manager supports the following import bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit components to invoke Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.

- HTTP bindings permit you to access applications using the HTTP protocol.
- Enterprise JavaBeans (EJB) import bindings enable SCA components to invoke services provided by Java EE business logic running on a Java EE server.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.
- WebSphere MQ bindings permit interoperability with WebSphere MQ. You can use WebSphere MQ bindings only with remote queue managers by way of a WebSphere MQ client connection; you cannot use them with local queue managers. Use WebSphere MQ bindings if you want to communicate with native WebSphere MQ applications.

Dynamic invocation of services

Services can be invoked through any supported import binding. A service is normally found at an endpoint specified in the import. This endpoint is called a static endpoint. It is possible to invoke a different service by overriding the static endpoint. Dynamic override of static endpoints lets you invoke a service at another endpoint, through any supported import binding. Dynamic invocation of services also permits you to invoke a service where the supported import binding does not have a static endpoint.

An import with an associated binding is used to specify the protocol and its configuration for dynamic invocation. The import used for the dynamic invocation can be wired to the calling component, or it can be dynamically selected at runtime.

For Web service and SCA invocations, it is also possible to make a dynamic invocation without an import, with the protocol and configuration deduced from the endpoint URL. The invocation target type is identified from the endpoint URL. If an import is used, the URL must be compatible with the protocol of the import binding.

- An SCA URL indicates invocation of another SCA module.
- An HTTP or a JMS URL by default indicates invocation of a Web service; for these URLs, it is possible to provide an additional binding type value that indicates that the URL represents an invocation by way of an HTTP or JMS binding.
- For a Web service HTTP URL, the default is to use SOAP 1.1, and a binding type value can be specified that indicates the use of SOAP 1.2.

Exports and export bindings

Exports define interactions between SCA modules and service requesters. SCA modules use exports to offer services to others. Export bindings define the specific way that an SCA module is accessed by service requesters.

Interfaces and bindings

An SCA module export needs at least one interface.

- Export interfaces are abstract definitions that define a set of operations using Web Services Description Language (WSDL), an XML language for describing Web services. An SCA module can have many export interfaces.
- Export bindings are concrete definitions that specify the physical mechanism that service requesters use to access a service. Usually, an SCA module export has one binding specified. An export with no binding specified is interpreted by the run time as an export with an SCA binding.

Supported export bindings

IBM Business Process Manager supports the following export bindings:

- SCA bindings connect SCA modules to other SCA modules. SCA bindings are also referred to as default bindings.
- Web Service bindings permit exports to be invoked as Web services. The supported protocols are SOAP1.1/HTTP, SOAP1.2/HTTP, and SOAP1.1/JMS.

  You can use a SOAP1.1/HTTP or SOAP1.2/HTTP binding based on the Java API for XML Web Services (JAX-WS), which allows interaction with services using document or RPC literal bindings and which uses JAX-WS handlers to customize invocations. A separate SOAP1.1/HTTP binding is provided to allow interaction with services that use an RPC-encoded binding or where there is a requirement to use JAX-RPC handlers to customize invocations.
- HTTP bindings permit exports to be accessed using the HTTP protocol.
- Enterprise JavaBeans (EJB) export bindings allow SCA components to be exposed as EJBs so that Java EE business logic can invoke SCA components otherwise unavailable to them.
- Enterprise information system (EIS) bindings provide connectivity between SCA components and an external EIS. This communication is achieved through the use of resource adapters.
- Java Message Service (JMS) 1.1 bindings permit interoperability with the WebSphere Application Server default messaging provider. JMS can exploit various transport types, including TCP/IP and HTTP or HTTPS. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported.
- Generic JMS bindings permit interoperability with third-party JMS providers that integrate with the WebSphere Application Server using the JMS Application Server Facility (ASF).
- WebSphere MQ JMS bindings permit interoperability with WebSphere MQ-based JMS providers. The JMS Message class and its five subtypes (Text, Bytes, Object, Stream, and Map) are automatically supported. If you want to use WebSphere MQ as a JMS provider, use WebSphere MQ JMS bindings.
- WebSphere MQ bindings permit interoperability with WebSphere MQ. You use a remote (or client) connection to connect to an MQ queue manager on a remote machine. A local (or bindings) connection is a direct connection to WebSphere MQ. This can be used only for a connection to an MQ queue manager on the same machine. WebSphere MQ will permit both types of connection, but MQ bindings only support the "remote" (or "client") connection.

Mediation modules

Mediation modules are Service Component Architecture (SCA) modules that can change the format, content, or target of service requests.

Mediation modules operate on messages that are in-flight between service requesters and service providers. You can route messages to different service providers and to amend message content or form. Mediation modules can provide functions such as message logging, and error processing that is tailored to your requirements.

You can change certain aspects of mediation modules, from the administrative console, without having to redeploy the module.

Components of mediation modules

Mediation modules contain the following items:

- Imports, which define interactions between SCA modules and service providers. They allow SCA modules to call external services as if they were local. You can view mediation module imports and modify the binding.
- Exports, which define interactions between SCA modules and service requesters. They allow an SCA module to offer a service and define the external interfaces (access points) of an SCA module. You can view mediation module exports.
- SCA components, which are building blocks for SCA modules such as mediation modules. You can create and customize SCA modules and components graphically, using Integration Designer. After you deploy a mediation module you can customize certain aspects of it from the administrative console, without having to redeploy the module.

  Usually, mediation modules contain a specific type of SCA component called a *mediation flow component*. Mediation flow components define mediation flows.

  A mediation flow component can contain none, one, or a number of mediation primitives. IBM Business Process Manager supports a supplied set of mediation primitives that provide functionality for message routing and transformation. For additional mediation primitive flexibility, use the Custom Mediation primitive to call custom logic.

  The purpose of a mediation module that does not contain a mediation flow component is to transform service requests from one protocol to another. For example, a service request might be made using SOAP/JMS but might need transforming to SOAP/HTTP before sending on.

  Note: You can view and make certain changes to mediation modules from IBM Business Process Manager. However, you cannot view or change the SCA components inside a module from IBM Business Process Manager. Use Integration Designer to customize SCA components.

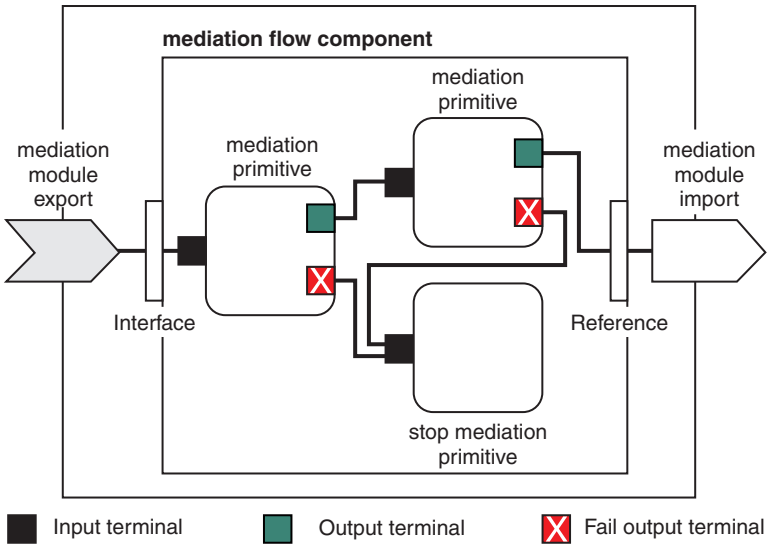


Figure 88. Simplified example of a mediation module. The mediation module contains one mediation flow component, which contains mediation primitives.

- Properties

  Mediation primitives have properties, some of which can be displayed in the administrative console as additional properties of an SCA module.

  For mediation primitive properties to be visible from the IBM Business Process Manager administrative console, the integration developer must promote the properties. Certain properties lend themselves to

being administratively configured and Integration Designer describes these as promotable properties, because they can be promoted from the integration cycle to the administrative cycle. Other properties are not suitable for administrative configuration, because modifying them can affect the mediation flow in such a way that the mediation module needs to be redeployed. Integration Designer lists the properties that you can choose to promote under the promoted properties of a mediation primitive.

You can use the IBM Business Process Manager administrative console to change the value of promoted properties without having to redeploy a mediation module, or restart the server or module.

Generally, mediation flows use property changes immediately. However, if property changes occur in a deployment manager cell, they take effect on each node as that node is synchronized. Also, mediation flows that are in-flight continue to use previous values.

Note: From the administrative console, you can only change property values, not property groups, names or types. If you want to change property groups, names or types, you must use Integration Designer.

- A mediation module or dependent library may also define subflows. A subflow encapsulates a set of mediation primitives wired together as a reusable piece of integration logic. A primitive can be added to a mediation flow to invoke a subflow.

Deploying mediation modules

Mediation modules are created using Integration Designer, and are generally deployed to IBM Business Process Manager inside an enterprise archive (EAR) file.

You can change the value of promoted properties at deployment time.

You can export a mediation module from Integration Designer, and cause Integration Designer to package the mediation module inside a Java archive (JAR) file, and the JAR file inside an EAR file. You can then deploy the EAR file, by installing a new application from the administrative console.

Mediation modules can be thought of as one entity. However, SCA modules are defined by a number of XML files stored in a JAR file.

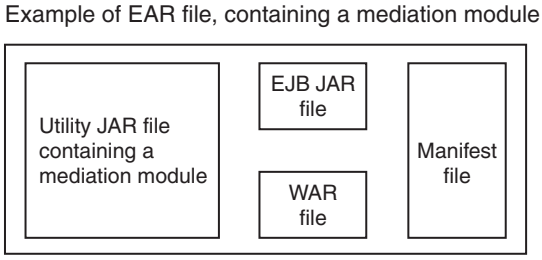


Figure 89. Simplified example of an EAR file containing a mediation module. The EAR file contains JARs. The utility JAR file contains a mediation module.

Mediation primitives

Mediation flow components operate on message flows between service components. The capabilities of a mediation component are implemented by *mediation primitives*, which implement standard service implementation types.

A mediation flow component has one or more flows. For example, one for request and one for reply.

IBM Business Process Manager supports a supplied set of mediation primitives, which implement standard mediation capabilities for mediation modules or modules deployed into IBM Business Process Manager. If you need special mediation capabilities, you can develop your own custom mediation primitives.

A mediation primitive defines an "in" operation that processes or handles messages that are represented by service message objects (SMOs). A mediation primitive can also define "out" operations that send messages to another component or module.

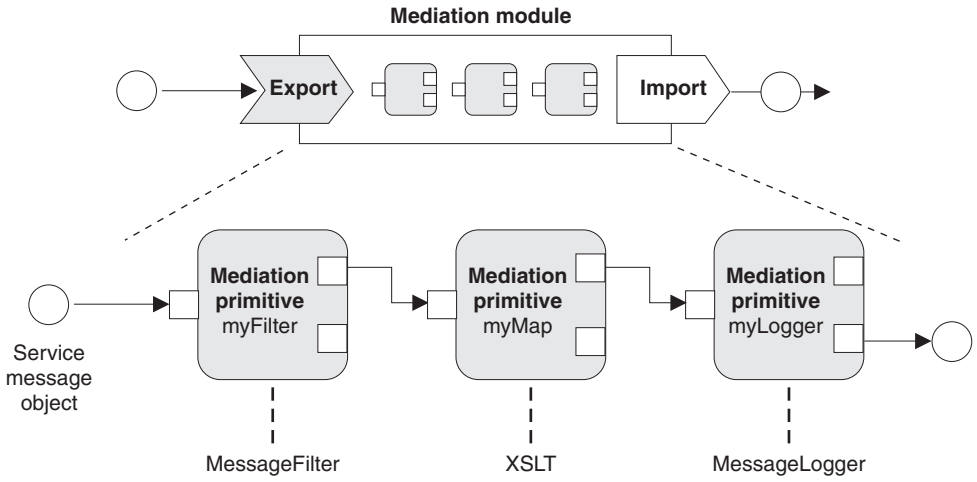


Figure 90. Mediation module containing three mediation primitives

You can use Integration Designer to configure mediation primitives and set their properties. Some of these properties can be made visible to the runtime administrator by promoting them. Any mediation primitive property that can be promoted can also be a dynamic property. A dynamic property can be overridden, at run time, using a policy file.

Integration Designer also allows you to graphically model and assemble mediation flow components from mediation primitives, and assemble mediation modules or modules from mediation flow components. The administrative console refers to mediation modules and modules as SCA modules.

Integration Designer also allows the definition of subflows in modules or their dependent libraries. A subflow can contain any mediation primitive except for the Policy Resolution mediation primitive. A subflow is invoked from a request or response flow, or from another subflow using the Subflow mediation primitive. Properties promoted from mediation primitives in a subflow are exposed as properties on the Subflow mediation primitives. These may then be promoted again until they reach the module level at which point they can then be modified by the runtime administrator.

Supported mediation primitives

The following set of mediation primitives are supported by IBM Business Process Manager:

Business Object Map

Transforms messages.

- Defines message transformations using a business object map, which can be reused.
- Allows you to define message transformations graphically, using the business object map editor.
- Can alter the content of a message.
- Can transform an input message type to a different output message type.

Custom Mediation

Allows you to implement your own mediation logic in Java code. The Custom Mediation primitive combines the flexibility of a user-defined mediation primitive, with the simplicity of a pre-defined mediation primitive. You can create complex transformations and routing patterns by:

- Creating Java code.
- Creating your own properties.
- Adding new terminals.

You can call a service from a Custom Mediation primitive, but the Service Invoke mediation primitive is designed to call services and provides additional functionality, such as retry.

Data Handler

Allows you to transform a part of a message. It is used to convert an element of a message from a physical format to a logical structure or a logical structure to a physical format. The primary usage of the primitive is to convert a physical format, such as a Text string within a JMS Text Message object, into a logical Business Object structure and back again. This mediation is commonly used to:

- Transform a section of the input message from a defined structure to another - an example of this would be where the SMO includes a string value that is comma delimited and you want to parse this into a specific Business Object.
- Alter the message type – an example would be when a JMS export has been configured to use a JMS basic typed data binding and within the mediation module the integration developer decides that the content should be inflated to a specific BO structure.

Database Lookup

Modifies messages, using information from a user-supplied database.

- You must set up a database, data source, and any server authentication settings for the Database Lookup mediation primitive to use. Use the administrative console to help you do this.
- The Database Lookup mediation primitive can read from only one table.
- The specified key column must contain a unique value.
- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Endpoint Lookup

Allows for the dynamic routing of requests, by searching for service endpoints in a repository.

- Service endpoint information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- IBM Business Process Manager needs to know which registry to use, therefore, you must create WSRR access definitions using the IBM Business Process Manager administrative console.

Event Emitter

Enhances monitoring by letting you send events from inside a mediation flow component.

- You can suspend the mediate action by deselecting the check box.
- You can view Event Emitter events using the Common Base Events browser on IBM Business Process Manager.
- You should only send events at a significant point in a mediation flow, for performance reasons.
- You can define the parts of the message that the event contains.
- The events are sent in the form of Common Base Events and are sent to a Common Event Infrastructure server.
- To fully use the Event Emitter information, event consumers need to understand the structure of the Common Base Events. The Common Base Events has an overall schema, but this does

not model the application specific data, which is contained in the extended data elements. To model the extended data elements, the Integration Designer tools generate a Common Event Infrastructure event catalog definition file for each of the configured Event Emitter mediation primitives. Event catalog definition files are export artifacts that are provided to help you; they are not used by Integration Designer or by the IBM Business Process Manager runtime. You should refer to the event catalog definition files when you create applications to consume Event Emitter events.

- You can specify other monitoring from IBM Business Process Manager. For example, you can monitor events to be emitted from imports and exports.

Fail Stops a particular path in the flow, and generates an exception.

Fan In Helps aggregate (combine) messages.

- Can only be used in combination with the Fan Out mediation primitive.
- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- The Fan In mediation primitive receives messages until a decision point is reached, then one message is output.
- The shared context should be used to hold aggregation data.

Fan Out

Helps split and aggregate (combine) messages.

- Together, the Fan Out and Fan In mediation primitives allow aggregation of data into one output message.
- In iterate mode, the Fan Out mediation primitive lets you iterate through a single input message that contains a repeating element. For each occurrence of the repeating element, a message is sent.
- The shared context should be used to hold aggregation data.

HTTP Header Setter

Provides a mechanism for managing headers in HTTP messages.

- Can create, set, copy, or delete HTTP message headers.
- Can set multiple actions to change multiple HTTP headers.

Mapping

Transforms messages.

- Allows you to perform Extensible Stylesheet Language (XSL) transformations or Business Object Map transformations.
- You transform messages using an XSLT 1.0 or an XSLT 2.0 transformation or a Business Object Map transformation. The XSL transformations operate on an XML serialization of the message, whereas the Business Object Map transformation operates on the Service Data Objects (SDO).

Message Element Setter

Provides a simple mechanism for setting the content of messages.

- Can change, add or delete message elements.
- Does not change the type of the message.
- The data in the value columns must be either a simple XML schema type, or an XML schema type that extends a simple XML schema type.

Message Filter

Routes messages down different paths, based on the message content.

- You can suspend the mediate action by deselecting the check box.

Message Logger

Logs messages in a relational database or through your own custom logger. The messages are stored as XML, therefore, data can be post-processed by XML-aware applications.

- You can suspend the mediate action by deselecting the check box.
- The rational database schema (table structure) is defined by IBM.
- By default, the Message Logger mediation primitive uses the Common database. The runtime maps the data source at **jdbc/mediation/messageLog** to the Common database.
- You can set Handler implementation classes to customize the behavior of the custom logger. Optionally, you can provide Formatter implementation classes, Filter implementation classes, or both to customize the behavior of the custom logger.

MQ Header Setter
Provides a mechanism for managing headers in MQ messages.
- Can create, set, copy, or delete MQ message headers.
- Can set multiple actions to change multiple MQ headers.

Policy Resolution
Allows for the dynamic configuration of requests, by searching for service endpoints, and associated policy files, in a repository.
- You can use a policy file to dynamically override the promoted properties of other mediation primitives.
- Service endpoint information and policy information is retrieved from a WebSphere Service Registry and Repository (WSRR). The WSRR registry can be local or remote.
- You make registry changes from the WSRR administrative console.
- IBM Business Process Manager needs to know which registry to use, therefore, you must create WSRR access definitions using the IBM Business Process Manager administrative console.

Service Invoke
Calls a service from inside a mediation flow, rather than waiting until the end of the mediation flow and using the callout mechanism.
- If the service returns a fault, you can retry the same service or call another service.
- The Service Invoke mediation primitive is a powerful mediation primitive that can be used on its own for simple service calls, or in combination with other mediation primitives for complex mediations.

Set Message Type
During integration development, lets you treat weakly-typed message fields as though they are strongly-typed. A field is weakly-typed if it can contain more than one type of data. A field is strongly-typed if its type and internal structure are known.
- At runtime, the Set Message Type mediation primitive lets you check that the content of a message matches the data types you expect.

SOAP Header Setter
Provides a mechanism for managing headers in SOAP messages.
- Can create, set, copy, or delete SOAP message headers.
- Can set multiple actions to change multiple SOAP headers.

Stop Stops a particular path in the flow, without generating an exception.

Type Filter
Allows you to direct messages down a different path of a flow, based on their type.

WebSphere eXtreme Scale Retrieve
You can retrieve information from an eXtreme Scale server cache environment.
- You can look up values in the cache, and store them as elements in the message using a key.
- Combining the eXtreme Scale Store and Retrieve mediation primitives, you can cache the response from a back-end system. Future requests will not require access to that back-end system.

- You must create eXtreme Scale definitions using the WebSphere ESB administrative console, so you can specify which eXtreme Scale server to use.

WebSphere eXtreme Scale Store

You can store information into an eXtreme Scale server cache environment.

- You can store information in a eXtreme Scale cache using a key and an object.
- Combining the eXtreme Scale Store and Retrieve mediation primitives, you can use the Store mediation primitive to store data within the cache, and use the Retrieve mediation primitive to retrieve data previously stored within the cache.
- You must create eXtreme Scale definitions using the WebSphere ESB administrative console, so you can specify which eXtreme Scale server to use.

Dynamic routing

You can route messages in various ways using endpoints defined at integration time or endpoints determined, dynamically, at run time.

Dynamic routing covers two message routing cases:

- Message routing where the flow is dynamic, but all possible endpoints are predefined in a Service Component Architecture (SCA) module.
- Message routing where the flow is dynamic, and the endpoint selection is also dynamic. The service endpoints are selected from an external source at run time

Dynamic endpoint selection

The run time has the capability to route request and response messages to an endpoint address identified by a message header element. This message header element can be updated by mediation primitives, in a mediation flow. The endpoint address could be updated with information from a registry, a database, or with information from the message itself. Routing of response messages applies only when the response is being sent by a Web service JAX-WS export.

In order for the run time to implement dynamic routing on a request or response, the SCA module must have the `Use dynamic endpoint if set in the message header` property set. Integration developers can set the `Use dynamic endpoint if set in the message header` property or they can promote it (make it visible at run time), so that the runtime administrator can set it. You can view module properties in the Module Properties window. To see the window, click **Applications** > **SCA Modules** > **Module Properties**. The integration developer gives promoted properties alias names, and these are the names displayed on the administrative console.

Registry

You can use IBM WebSphere Service Registry and Repository (WSRR) to store service endpoint information, and then create SCA modules to retrieve endpoints from the WSRR registry.

When you develop SCA modules, you use the Endpoint Lookup mediation primitive to allow a mediation flow to query a WSRR registry for a service endpoint, or a set of service endpoints. If an SCA module retrieves a set of endpoints then it must use another mediation primitive to select the preferred one.

Mediation policy control of service requests

You can use mediation policies to control mediation flows between service requesters and service providers.

You can control mediation flows using mediation policies stored in IBM WebSphere Service Registry and Repository (WSRR). The implementation of service policy management in WSRR is based on the Web Services Policy Framework (WS-Policy).

In order to control service requests using mediation policies, you must have suitable Service Component Architecture (SCA) modules and mediation policy documents in your WSRR registry.

How to attach a mediation policy to a service request

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry. Therefore, an SCA module must contain a mediation flow component in order to support mediation policy control of service requests.

In the registry, you can attach one or more mediation policies to an SCA module, or to a target service used by the SCA module. Attached mediation policies could be used (are in scope) for all service messages processed by that SCA module. The mediation policies can have policy attachments that define conditions. Mediation policy conditions allow different mediation policies to apply in different contexts. In addition, mediation policies can have classifications, which can be used to specify a governance state.

WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository (WSRR) product allows you to store, access, and manage information about service endpoints and mediation policies. You can use WSRR to make your service applications more dynamic, and more adaptable to changing business conditions.

Introduction

Mediation flows can use WSRR as a dynamic lookup mechanism, providing information about service endpoints or mediation policies.

To configure access to WSRR, you create WSRR definition documents using the administrative console. Alternatively, you can use the WSRR administration commands from the wsadmin scripting client. WSRR definitions and their connection properties are the mechanism used to connect to a registry instance, and retrieve a service endpoint or mediation policy.

Service endpoints

You can use WSRR to store information about services that you already use, that you plan to use, or that you want to be aware of. These services might be in your systems, or in other systems. For example, an application could use WSRR to locate the most appropriate service to satisfy its functional and performance needs.

When you develop an SCA module that needs to access service endpoints from WSRR, you must include an Endpoint Lookup mediation primitive in the mediation flow. At run time, the Endpoint Lookup mediation primitive obtains service endpoints from the registry.

Mediation policies

You can also use WSRR to store mediation policy information. Mediation policies can help you to control service requests, by dynamically overriding module properties. If WSRR contains mediation policies that are attached to an object representing either your SCA module or your target service, then the mediation policies could override the module properties. If you want different mediation policies to apply in different contexts, you can create mediation policy conditions.

Note: Mediation policies are concerned with the control of mediation flows, and not with security.

When you develop an SCA module that needs to make use of a mediation policy, you must include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry.

WebSphere eXtreme Scale

By using the WebSphere eXtreme Scale (eXtreme Scale) product you can provide a caching system that you can integrate with a IBM Business Process Manager application. Using eXtreme Scale with IBM Business Process Manager can improve service response times and reliability, and provide additional integration functionality.

eXtreme Scale acts as an elastic, scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. With eXtreme Scale, you can also get qualities of service such as transactional integrity, high availability, and predictable response times.

You can use mediation flows to access the eXtreme Scale caching function by including the WebSphere eXtreme Scale mediation primitives within your flow. When you develop a Service Component Architecture (SCA) module that needs to store information in an eXtreme Scale cache, you must include the WebSphere eXtreme Scale Store mediation primitive in the mediation flow. If you want to retrieve information from an eXtreme Scale cache, you must include the WebSphere eXtreme Scale Retrieve mediation primitive. By combining the two mediation primitives in a mediation flow you can cache the response from a back-end system, so that future requests can retrieve the response from the cache.

To configure access to eXtreme Scale , you must create a WebSphere eXtreme Scale definition using the administrative console. Alternatively, you can use the WebSphere eXtreme Scale administration commands from the wsadmin scripting client. An eXtreme Scale definition is the mechanism used by the WebSphere eXtreme Scale Retrieve and Store mediation primitives to connect to an eXtreme Scale server.

Message Service clients

Message Service clients is available for C/C++ and .NET to enable non-Java applications to connect to the enterprise service bus.

Message Service Clients for C/C++ and .NET provides an API called XMS that has the same set of interfaces as the Java Message Service (JMS) API. Message Service Client for C/C++ contains two implementations of XMS, one for use by C applications and another for use by C++ applications. Message Service Client for .NET contains a fully-managed implementation of XMS, which can be used by any .NET compliant language.

You can obtain Message Service Clients for .NET from http://www.ibm.com/support/docview.wss?uid=swg24011756.

You can obtain Message Service Clients for C/C++ from http://www.ibm.com/support/docview.wss?uid=swg24007092.

You can also install and use the Java EE client support from WebSphere Application Server Network Deployment, including Web services Client, EJB Client, and JMS Client.