WebSphere. software

IBM

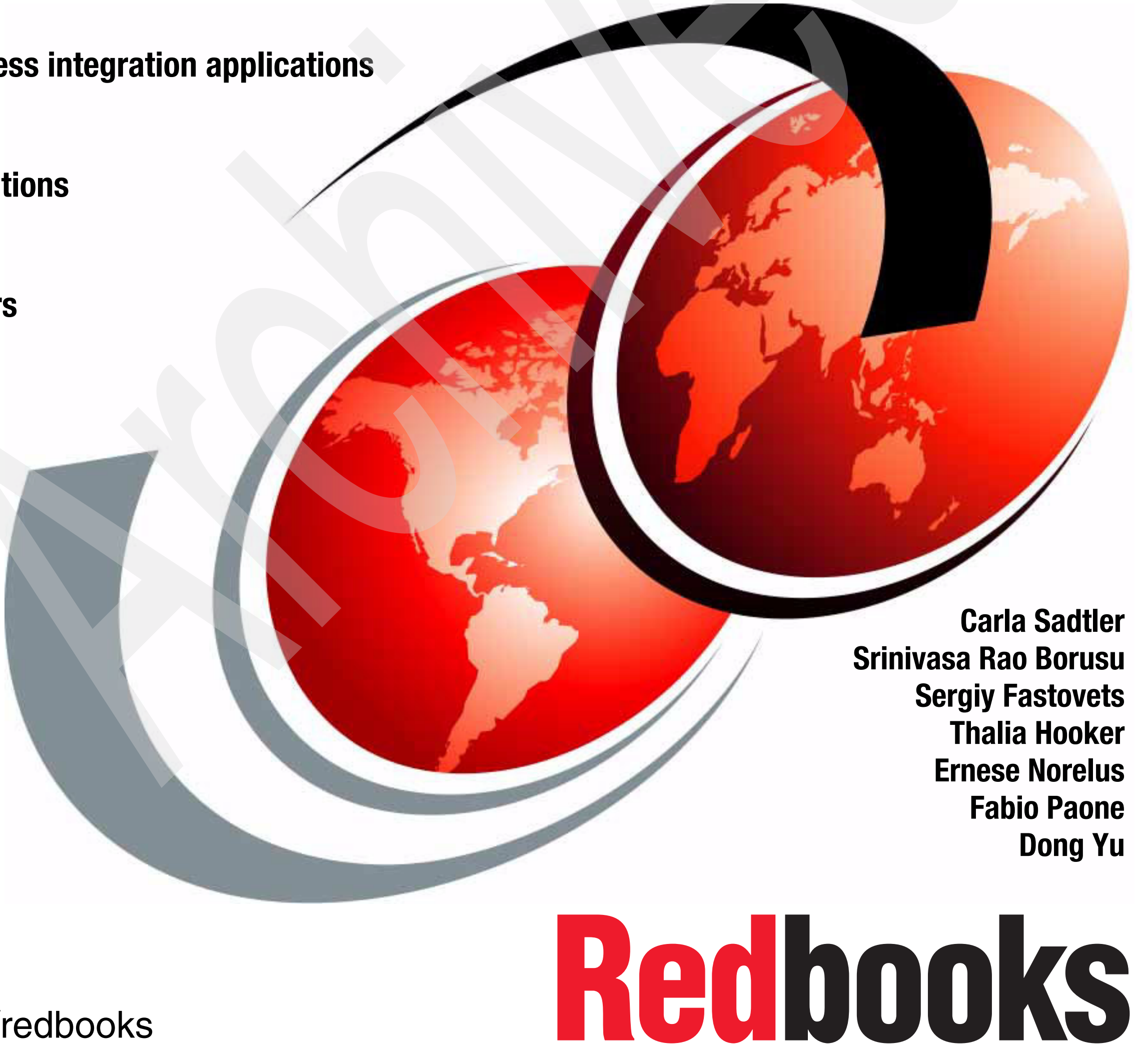
Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus

Part 1: Development

Build business integration applications

Build mediations

Use adapters



Carla Sadtler
Srinivasa Rao Borusu
Sergiy Fastovets
Thalia Hooker
Ernese Norelus
Fabio Paone
Dong Yu

ibm.com/redbooks

Redbooks

International Technical Support Organization

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1: Development

June 2008

SG24-7608-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (June 2008)

This edition applies to IBM WebSphere Process Server V6.1 and IBM WebSphere Enterprise Service Bus V6.1.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
ClearCase®
Cloudscape®
DB2®
developerWorks®
i5/OS®
IBM®
IMS™
Informix®
Lotus®
MVS™
OS/2®
Rational®
Redbooks®
Redbooks (logo) ®
Tivoli®
WebSphere®
Workplace™
z/OS®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

EJB, J2EE, Java, JavaServer, JDBC, JSP, JVM, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, SQL Server, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication provides developers with information about building and testing applications for IBM WebSphere® Process Server and IBM WebSphere Enterprise Service Bus. It helps developers with the tasks of creating business integration applications and mediations. It also includes information about the use of adapters.

This is the first book of a three-part series:

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus:

- *Part 1: Development,* SG24-7608
- *Part 2: Scenario,* SG24-7642
- *Part 3: Runtime,* SG24-7643

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

Carla Sadtler is a Consulting IT Specialist at the ITSO, Raleigh Center. She writes extensively about WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative, supporting MVS™ customers. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Srinivasa Rao Borusu is a Senior IT Specialist at IBM India Software Labs, Bangalore. He is currently working as a WebSphere Consultant with IBM Business Partner Technical Strategy and Enablement (BPTSE) Developer Services team, enabling and supporting worldwide business partners on WebSphere products. Since he joined IBM in 2000, he has played various roles, including Tech Lead for the IBM Java™ Virtual Machine Support during his tenure at OS/2® worldwide support center. His technical portfolio also includes one FILE rated patent, four IBM SOA and WebSphere product certifications, and several published articles.

He holds a masters degree in Computer Applications from the Andhra University, Visakhapatnam, India.

Sergiy Fastovets joined IBM in 1996 in the Research Triangle Park. He has 10 years of experience in design, development, and sales support in the Host Integration area. In 2006, Sergiy transferred to the IBM UK where he is an IT Consultant for WebSphere Software Services. He holds a degree in mathematics from the University of Saint Petersburg, Russia. His native town is Poltava, Ukraine.

Thalia Hooker Ph.D., is a Consulting IT Specialist and member of the WebSphere Americas iPoC Team. She executes proof-of-concepts using the WebSphere platform to show customers how IBM solutions and products can help them meet their integration requirements. These solutions and products include Service Oriented Architecture (SOA) and WebSphere Business Process Management products such as, WebSphere Process Server and WebSphere Enterprise Service Bus.

Ernese Norelus is an IBM Certified Consulting IT Specialist with the ASEAN Software Services team in Singapore. He has been in IT for 10 years, with eight years of experience presenting, teaching, and proposing solution architectures to customers using the WebSphere business integration portfolio. He is also a well-known conference speaker, and he holds certifications in IM/DB2®, Lotus®, Rational®, Tivoli®, and WebSphere. He holds degrees in Biochemistry and Computer Science and Information Technology Management from the Université du Québec à Montréal, Québec, Canada.

Fabio Paone is in WebSphere technical sales supporting the Channel (Business Partners) in Italy. He has four years of experience as a developer in the Rome Tivoli Lab, primarily in J2EE™ development. He is certified as a WebSphere Application Server Administrator (versions 5, 6, and 6.1) and has participated in WebSphere Application Server certification test reviews.

Dong Yu is a Staff Software Engineer at IBM China Development Lab. He has four years of experience in the WebSphere Business Integration Field. His areas of expertise include WebSphere Process Server installation and Configuration. Dong holds a master degree in Software Engineering from Northwestern Polytechnical University.

Thanks to the following people for their contributions to this project:

Stephen Cocks
IBM UK

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Margaret Ticknor
International Technical Support Organization, Raleigh Center

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to the products

This book focuses on the development of business integration solutions for IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus. Development activities are performed using IBM WebSphere Integration Developer. This chapter introduces these products and includes the following topics:

- WebSphere Process Server
- WebSphere Enterprise Service Bus
- WebSphere Adapters
- WebSphere Integration Developer

1.1 WebSphere Process Server

WebSphere Process Server is one of the key products in the IBM WebSphere Business Process Management suite. WebSphere Process Server integrates with WebSphere Portal to deliver business process management through a portal.

In this section, we provide a brief introduction to WebSphere Process Server.

1.1.1 Introduction to the product

A *business process* is a series of tasks executed in a specific order that an organization follows to achieve a larger business goal. WebSphere Process Server is a *process engine* that provides a hosting environment for business processing. It provides support for the Service Component Architecture (SCA) programming model. WebSphere Process Server includes support for both Web Services Business Process Execution Language (WS-BPEL) based process flows and business state machines. It supports the integration of business rules and for the incorporation of tasks that are carried out by users (human tasks) in a business process.

WebSphere Process Server is underpinned by IBM WebSphere Application Server Network Deployment, giving it extensive J2EE capabilities and the qualities of service (QoS) of that product, including the high availability, scalability, and security features. The administrative facilities of WebSphere Application Server have been enhanced to support processes and mediations. In addition, WebSphere Process Server provides several Web-based applications for managing the various aspects of business processes:

- Business Process Choreographer Explorer for managing business process and human tasks.
- Business Process Choreographer Observer that creates reports about events that occur during the execution of business processes and human tasks.
- Business rules manager that assists the business analyst in browsing and modifying business rule values.
- Common Base Event browser to retrieve and view events in the Common Event Infrastructure (CEI) event database.
- Failed event manager to find and manage WebSphere Process Server failed events, which can be a request sent to an application from an external source or an invocation to a Web service.
- Relationship manager to manipulate relationship data manually to correct errors found in automated relationship management or to provide more

complete relationship information. *Relationships* are services that are used to model and maintain associations between business objects and other data.

WebSphere Process Server incorporates the functionality of WebSphere Enterprise Service Bus to execute *mediations*. Mediation service applications intercept and modify messages that are passed between existing services (providers) and clients (requesters) that want to use those services. Both business processes and mediations can be deployed to a WebSphere Process Server application server.

Development of business integration processes for deployment to WebSphere Process Server and mediations for deployment to WebSphere Enterprise Service Bus is done using WebSphere Integration Developer. The tools are designed so that users can compose integrated business solutions easily without programming skills.

Information Center: The WebSphere Process Server V6.1 information center is a critical resource for learning about and using WebSphere Process Server:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wps.610.doc/welcome_top_wps.htm

1.1.2 V6.1 highlights

Note: For a full list of what is new in WebSphere Process Server V6.1, see:

- *What is new in this release (AIX®, HP-UX, Linux®, Solaris™, Windows®, i5/OS®)*

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wps.610.doc/doc/covw_new.html

- *What is new in this release (z/OS®)*

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wps.z.610.doc/doc/covw_new.html

The highlights of this release include:

- New platform support, including support for i5/OS and 64-bit Windows and UNIX® platforms. You can find a full list of supported platforms at:

  http://www-1.ibm.com/support/docview.wss?rs=2307&context=SSQH9M&uid=swg27009829

- Performance enhancements

- New support for export and import bindings:
  - HTTP 1.0 and HTTP 1.1 are now supported as a general transport. Previously, HTTP support was provided only as a transport for SOAP over HTTP Web services.
  - Generic JMS is now supported to provide the ability to use any JMS 1.1 Application Server Facilities (ASF) compliant providers. Previously, support included only WebSphere MQ and the WebSphere Application Server default messaging provider.
  - Support has been added to allow you to use WebSphere Transformation Extender for data bindings
- Business objects can now be validated at run time. Validation can be done implicitly at the interface on all objects (for example, in a test environment) or programmatically.
- New structured activities
  - The *Cyclic Flow* activity allows you to add links that go back to a previous activity. Prior to V6.1, flow could only go forward and While Loops were commonly used to create cycles of activity.
  - The *ForEach* activity allows processing a dynamic number of work branches to be processed in parallel or serially. A completion condition can be specified to terminate the flow before all branches complete.
- Human tasks have been enhanced with extended people directory support, participant substitution, auto-deletion for completed tasks, and batch processing support.
- The Business Flow Manager now supports a generic JMS interface and extensions to the generic Web Services interface.
- The Business Process Choreographer Explorer has been enhanced to allow the substitution of users, suspend human tasks and business processes for a specific time, and to include other usability features.
- Forms that are created using IBM Lotus Forms Designer (integrated into WebSphere Integration Developer) can be used as the user interface for human tasks and processes.
- The WebSphere Portal Server My Task portlet can be extended with portlets that are generated from WebSphere Integration Developer.

1.2 WebSphere Enterprise Service Bus

Information Center: The WebSphere Enterprise Service Bus V6.1 information center is a critical resource for learning about and using WebSphere Enterprise Service Bus:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wesb.61x.root.doc/info/welcome.html

WebSphere Enterprise Service Bus delivers an enterprise service bus (ESB) infrastructure to connect applications that have standards-based interfaces (typically a Web service interface described in a WSDL file). WebSphere Enterprise Service Bus can be obtained as a stand-alone product. It is also included in WebSphere Process Server.

In this section, we provide a brief introduction to WebSphere Enterprise Service Bus.

1.2.1 Introduction to the product

WebSphere Enterprise Service Bus intercepts the requests of service consumers and fulfills additional tasks through mediations in order to support loose coupling. When the mediation completes, the service providers are invoked. WebSphere Enterprise Service Bus provides pre-built mediation primitives and easy-to-use tools to enable rapid construction and implementation mediations.

The mediation tasks include:

- Centralizing the routing logic so that service providers can be exchanged transparently
- Performing tasks like protocol translation and transport mapping
- Acting as a facade in order to provide different interfaces between service consumers and providers
- Adding logic to tasks such as logging and fan-in and fan-out operations

1.2.2 V6.1 highlights

Note: For a full list of what is new in WebSphere Enterprise Service Bus V6.1, see:

- *What is new in this release (AIX, HP-UX, Linux, Solaris, Windows, i5/OS)*

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wesb610.doc/ref/rwesb_releasenotes.html`

- *What is new in this release (z/OS)*

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wesb610.zseries.doc/ref/rwesb_releasenotes.html`

The following new and updated mediation primitives provide some of the new features in WebSphere Enterprise Service Bus V6.1:

- The *Service Invoke* primitive is used to call a service from within a mediation flow, which is similar to a Callout node but the flow continues rather than switching to a response flow. It can retry a failed call, as well as route the call to another service or different endpoints of the service.
- The *Callout* node has been updated so that it also can retry a call to the same service or to a list of endpoints.
- The *Fan Out* and *Fan In* primitives are used to perform aggregation and broadcasting operations. Fan Out allows you to send a message multiple times or to send new messages created using single message as input. Fan In allows you to receive multiple messages and combine them into one message.
- The *Endpoint Lookup* primitive has been updated. The primitive is used to retrieve dynamically a service endpoint from the WebSphere Service Registry Repository. The primitive has been updated so that an alternate target list can be populated during the lookup. This alternate target list is used for service callout retries.
- The *Set Message Type* primitive allows you to overlay message fields with more detailed structures. It lets you do the equivalent of casting a generic data type to a more specific data type.
- The *Business Object Map* primitive allows you to use a business object map for message transformation. This is a new alternative to XSLT transformation.

1.3 WebSphere Adapters

Enterprise information systems (EIS) provide the information infrastructure for an enterprise by providing a set of services to clients, for example:

- Enterprise resource planning (ERP)
- Customer relationship management (CRM)
- Human resource systems (HR)
- Relational database systems

Adapters provide integration with enterprise information systems that do not provide service interfaces. They allow you to take advantage of these existing IT assets in your service-oriented architecture (SOA) without significant programming, thus simplifying integration. Adapters are presented as SCA components.

Adapters can be divided into two classes:

- WebSphere JCA Adapters
- WebSphere Business Integration Adapters

1.3.1 WebSphere JCA Adapters

WebSphere JCA Adapters provide a mechanism that allows for the integration of an existing EIS infrastructure with the WebSphere Process Server and WebSphere Enterprise Server Bus server. They provide a service-oriented approach to EIS integration. Resource adapters offer a consistent framework for access to back-end systems and their applications.

WebSphere JCA Adapters are compliant to the Java Connector Architecture 1.5 specification, the J2EE standard for EIS connectivity. This standard provides a managed framework. That is, quality of service (QoS) is provided by the application server, which offers life cycle management and security to transactions. WebSphere JCA Adapters are also compliant with the Enterprise Metadata Discovery specification with the exception of the IBM CICS® ECI Resource Adapter and the IBM IMS™ Connector for Java.

The following IBM WebSphere Adapters are supported for WebSphere Process Server and WebSphere Enterprise Service Bus:

- IBM CICS ECI Resource Adapter Version 7
- IBM IMS Connector for Java Version 9.1.0.2.4
- IBM WebSphere Adapter for Email Version 6.1
- IBM WebSphere Adapter for FTP Version 6.1
- IBM WebSphere Adapter for Flat Files Version 6.1
- IBM WebSphere Adapter for JDBC™ Version 6.1

- IBM WebSphere Adapter for JD Edwards® EnterpriseOne Version 6.1
- IBM WebSphere Adapter for Oracle® E-Business Suite Version 6.1
- IBM WebSphere Adapter for PeopleSoft® Version 6.1
- IBM WebSphere Adapter for SAP® Software Version 6.1
- IBM WebSphere Adapter for Siebel® Business Applications Version 6.1

WebSphere Adapters are included with WebSphere Integration Developer. CICS, IMS, JD Edwards, Oracle, PeopleSoft, SAP, and Siebel resource adapters are shipped in WebSphere Integration Developer for development purposes only. That is, you can use them to develop and test an application. When deployed to a production server, your application will need a licensed runtime resource adapter. Note that when you build your service, you have the option of embedding the resource adapter with the service. If you use that option, your resource adapter licensing might allow you to use the embedded resource adapter as the licensed runtime resource adapter.

1.3.2 WebSphere Business Integration Adapters

IBM WebSphere Business Integration Adapters do not comply to either the JCA architecture or the Enterprise Metadata Discovery specification. The WebSphere Business Integration Adapter artifacts must first be created using the tool set that is provided with the WebSphere Business Integration Adapter Framework. Unlike the IBM WebSphere Adapters, none of these adapters are supplied for development purposes.

Some additional differences between the IBM WebSphere JCA Adapters and the IBM WebSphere Business Integration Adapters include:

- IBM WebSphere JCA Adapters rely on standard JCA contracts to manage life cycle tasks such as stopping and starting. IBM WebSphere Business Integration Adapters rely on the WebSphere Adapter Framework to manage connectivity.
- With WebSphere Adapters, you use an external service wizard in WebSphere Integration Developer to discover an EIS system and its available information. The external service wizard develops business objects from the discovered information. WebSphere Business Integration Adapters use a separate Object Discovery Agent (ODA) to probe an EIS and generate business object definition schemas.
- An outbound service type in WebSphere Adapters is known as request processing with WebSphere Business Integration Adapters. An inbound service type in WebSphere Adapters is known as event processing with WebSphere Business Integration Adapters.

- WebSphere Business Integration Adapters always reside outside of an application server at run time. The server communicates with this type of adapter through a Java Messaging Service (JMS) transport layer.
- WebSphere Business Integration Adapters use an asynchronous stand-alone runtime architecture (the WBI Framework) with WebSphere MQ or JMS as the underlying transport protocol. Originally designed for WebSphere InterChange Server, they remain available for WebSphere Message Broker, WebSphere Enterprise Service Bus, and WebSphere Process Server users transitioning from WebSphere InterChange Server or WebSphere Business Integration Server.

WebSphere Adapters V6.1 no longer ships the WebSphere Business Integration components. WebSphere Adapters V6.0.2 is available for the users requiring the WebSphere Business Integration components for the following adapters:

- Flat File Adapter
- FTP Adapter
- Email Adapter
- JDBC Adapter
- PeopleSoft Adapter
- JD Edwards EnterpriseOne Adapter
- SAP Software Adapter
- Siebel Business Applications Adapter
- Oracle eBusiness Suite Adapter

1.4 WebSphere Integration Developer

> **Information Center:** The WebSphere Integration Developer V6.1 information center is a critical resource for learning about and using WebSphere Integration Developer:
>
> http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.wbit.610.help.nav.doc/topics/welcome.html

WebSphere Integration Developer is the common tool for building SOA-based integration solutions across WebSphere Process Server, WebSphere Enterprise Service Bus, and WebSphere Adapters. It simplifies integration with rich features that accelerate the adoption of SOA by rendering existing IT assets as service components, encouraging reuse and efficiency.

In this section, we provide a brief introduction to WebSphere Integration Developer.

1.4.1 Introduction to the product

WebSphere Integration Developer enables integration developers to assemble complex business solutions that require minimal skills, whether they involve processes, mediations, adapters, or code components. Users can construct process and integration solutions using drag-and-drop technology without having a working knowledge of Java.

In addition to providing the tools that are necessary to build and assemble these artifacts, the product includes a full test framework that allows you to execute results in a seamless fashion in an environment that is identical to that found in production but without having to perform the steps to administer and configure such an environment.

1.4.2 V6.1 highlights

WebSphere Integration Developer has been enhanced to support the new features and functions in WebSphere Enterprise Service Bus V6.1 and WebSphere Process Server V6.1.

2

Concepts for development

This chapter provides information about key concepts that are required to develop applications for WebSphere Enterprise Service Bus and WebSphere Process Server.

The chapter includes the following topics:

- SCA and WebSphere Process Server
- WebSphere Process Server components

2.1 SCA and WebSphere Process Server

Service Component Architecture (SCA) is a set of specifications that describe a model for building applications and systems using service-oriented architecture (SOA). SCA encourages an organization of business application code based on components that implement business logic. These business applications offer their capabilities as services through interfaces and consume services that are offered by other components through references.

SCA separates business logic from implementation. The implementation of business processes is contained in *service components* (also referred to as *components*), which can be assembled graphically in WebSphere Integration Developer. The service components can be implemented later.

There are four basic tasks that are required to build service-oriented applications for WebSphere Process Server and WebSphere Enterprise Service Bus using SCA:

- The implementation of service components that provide services and consume other services.
- The assembly of service components to build business applications, through the wiring of service references to services.
- The generation of bindings to define the transport and protocol that are used to connect to external clients and services.
- The assignment of quality of service attributes.

2.1.1 Service components

A service component consists of an implementation, one or more interfaces that defines its inputs, outputs and faults, and zero or more references. A reference identifies the interface of another service or component that this component requires or consumes.

Figure 2-1 on page 13 shows an SCA module with its service components, interfaces and references. The implementation types for the service component are specific to a business integration module in WebSphere Process Server.

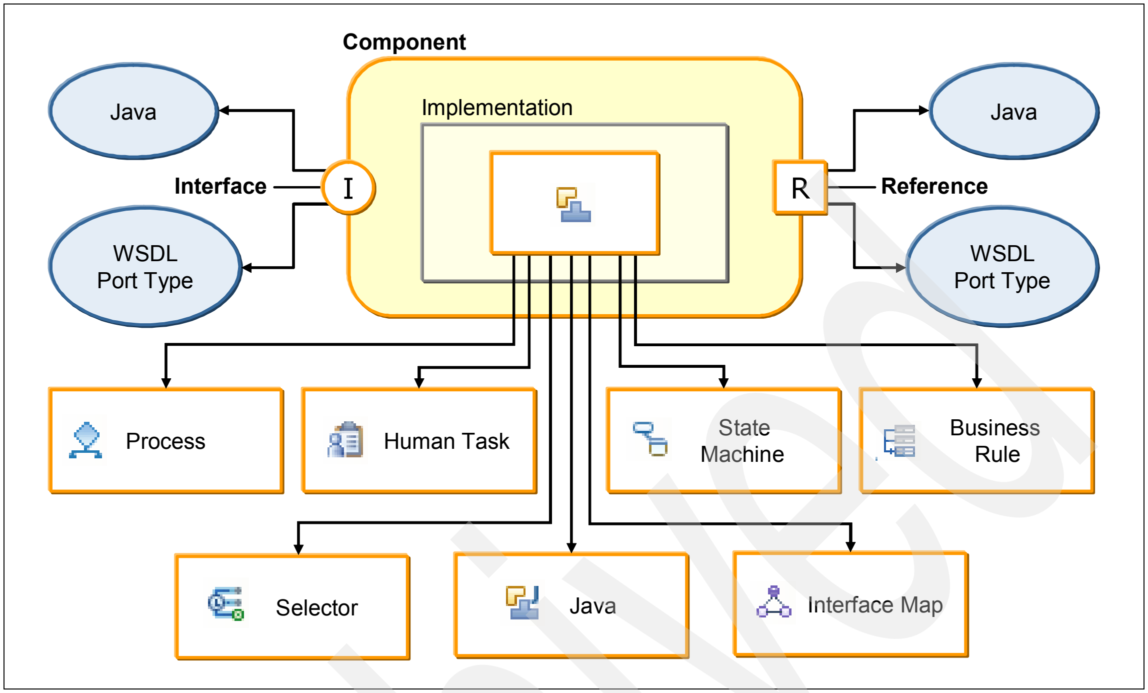


Figure 2-1 Block diagram of a component

A service component has an implementation that is associated with it that performs the logic of the module. You can implement services using a variety of programming paradigms, from process-flow style BPEL processes, to state machine-style event management, to declarative business rules style. The style of implementation that you select is determined both by your comfort level with a given paradigm and the nature of the problem.

The following implementation types are available for service components in a business integration module:

- Interface maps
- Business state machines
- Java objects
- Processes
- Human tasks
- Selectors
- Rule groups (business rules)

A service component in a mediation module is implemented as a *mediation flow*.

2.1.2 Service component assembly

A service component is presented in a standard block diagram referred to as a *module assembly*, or in WebSphere Integration Developer, as an *assembly diagram*. Figure 2-2 shows an example of an assembly diagram. It contains components that are wired together. The figure shows the implementation for the components, but you can also add untyped components to an assembly diagram to implement later.

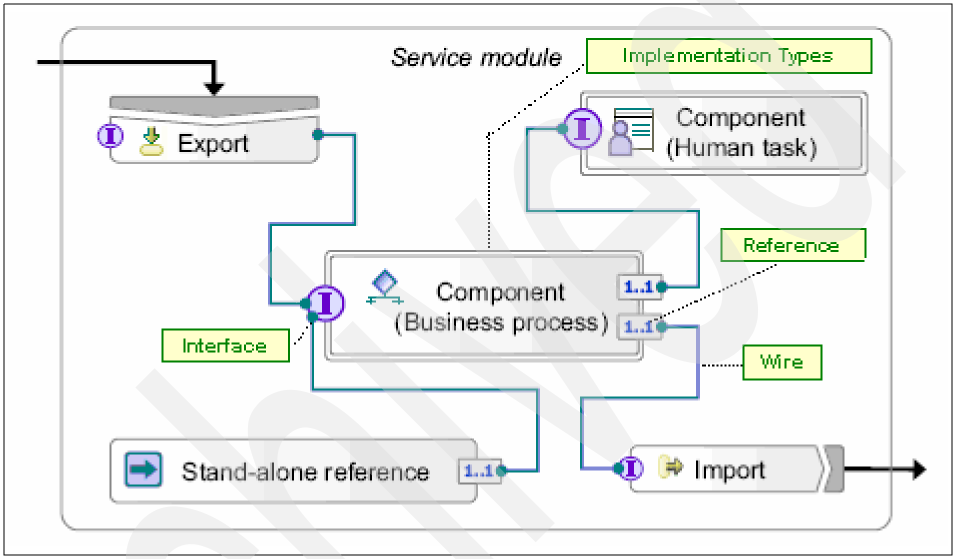


Figure 2-2 Service component module

Interfaces

An *interface* provides the input and output of a component and is independent of the internal implementation of the component. The interface specifies the operations that can be called and the data that is passed, such as input, output parameters, and exceptions.

All components have WSDL type interfaces. Only Java components support Java type interfaces in addition to WSDL type interfaces. If a component, import, or export has more than one interface, all the interfaces must be of the same type.

An interface supports synchronous and asynchronous interaction styles.

References

A *reference* (sometimes referred to as a *partner reference*) is required when one component uses another component. It is defined on the component that wants to use another component. A reference specifies the interface that is used in the invocation of the other component.

The implementation type of the component determines the type of interface that its partner references can have. All components support WSDL interfaces in their partner references.

A service component can include zero or more references to other service components or imports included in the current module.

You can define references in two ways:

- You can in-line the reference in the service component definition. In this approach, the references are available only to the service component in which the references are included.
- You can include reference definitions within the stand-alone references file. In this approach, the references can be used by a non-SCA client or by another component within the module. An example of a non-SCA component that might use a reference in the stand-alone references file is a user interface component such as a JSP™ that needs the ability to invoke a particular service. In order to invoke the service, the client needs a reference so that it can use the SCA run time to lookup the appropriate target.

By simply naming a reference and specifying its interface, the component implementation author can defer binding that reference to an actual service until later. At that later time, the integration specialist will do so by wiring the reference to the interface of another component or import. This *loose coupling*, which allows for deferred binding and the re-use of implementations, is one of the key reasons for using SCA.

Stand-alone references

Stand-alone references allow services that are not defined as SCA components (for example, JavaServer™ Pages) to invoke SCA components. Stand-alone references contain partner references that identify the components to call. On their own, stand-alone references do not have any implementation or interface.

A module assembly can contain one stand-alone reference artifact. You can add partner references to the stand-alone references and wire them to target components or target imports.

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

Exports

An *export* component exposes an interface for use by external service callers or other SCA modules. An export has an associated binding that describes the physical communication mechanism to be used. The interface for the export describes how a caller must interact with the module. It shields the implementation of the module from the caller. The export component is wired to the first component in the module, for example, a business process or mediation flow component.

Imports

Import components identify services outside of a module, so they can be called from within the module. An import component has an associated binding that specifies the means of transporting the data to and from the external service.

Every import has an associated interface. Thus, the import can be wired to any other component that has a matching Interface type reference. The caller of the import has no knowledge that it is calling an import, only that it is calling another SCA component that has a specific interface.

Wires

Wires are used to assemble these nodes in a module assembly. There are two types of wires:

- The first type of wire comes from a partner reference (the source) that is defined for a component or stand-alone references and goes to a component or import (the target). In this case, the wire identifies the component or import (target) that is accessed when the source component uses that partner reference. By default, a partner reference allows only one wire leading from it unless the partner reference's multiplicity property is changed to `0...n`.

  The target of a wire should support the interface or interfaces that the source specifies. If the partner reference on a source node cannot find a matching interface on the target node, you will have the option to create an interface map or to add a new interface on the target. Also, a WSDL partner reference cannot be wired directly to a Java interface.

- The second type of wire comes from an export (the source) and goes to a component or import (the target), as illustrated in Figure 2-3. In this case, the wire identifies the (target) component that provides the service. An export can only have one wire leading out of it.

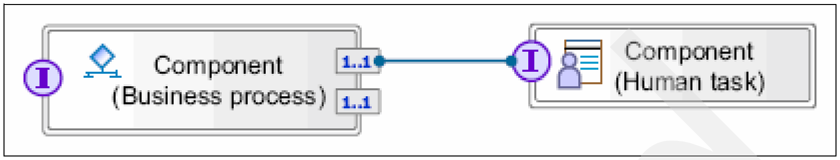


Figure 2-3 Wire the business process component to a human task component

Data is passed between the components in the form a business objects (in business integration modules) or service message objects (in mediation modules).

2.1.3 Import and export bindings

Bindings determine specifically how the import and export components interact with clients outside the module. Bindings specify the message format, protocol, and invocation style details for a particular interface.

In order to convert data external to WebSphere Process Server to and from data objects, some transport bindings require a data binding to be specified. The data binding translates between the data that is provided by the transport binding as part of a message and a data object. For information about the data bindings for each transport binding type, refer to the WebSphere Process Server V6.1.0 information center, which is available at:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.websphere.wps.610.doc/welcome_top_wps.htm`

Messaging bindings

Using a resource adapter or using the JMS API directly are two ways to connect to a messaging provider. Another way is to use native calls to a messaging provider such as IBM WebSphere MQ. These calls are made using the Message Queue Interface (MQI). MQ JMS uses the JMS API (as opposed to MQI) to access WebSphere MQ.

The following messaging bindings are available for use on import and export components:

- MQ binding

  WebSphere MQ bindings allow interoperation with native WebSphere MQ or WebSphere Message Broker applications using WebSphere MQ. Conversion for the data to and from an MQ message is accomplished through the MQ header and body data bindings.

  The MQ binding does not currently support the publish-subscribe method of distributing messages. MQ client connections are the only type of connections supported.

  The binding settings specify the information that is required for connection to MQ, including the destination names, queue manager name, and the host name for the queue manager.

- MQ JMS binding

  MQ JMS is a set of Java classes that enables communication with JMS applications using WebSphere MQ as the messaging provider. If you are using an MQ JMS binding, you can define the destination using one of the following methods:

  - Specify the JNDI name for a pre-configured messaging provider resource, that is the JNDI name for the queue or topic and the JNDI name for the connection factory
  - Specify properties that are required to configure a new messaging provider resource, meaning, the WebSphere queue manager and WebSphere queue name.

  If the import or export component has an interface with a one-way operation, then both queues (point-to-point) and topics (publish/subscribe) are supported by the MQ JMS binding. For request-response operations, only point-to-point is supported.

  In WebSphere Message Broker, you can use either MQ or JMS input and output nodes as the corresponding message point. No special configuration is necessary on the MQ JMS binding that would make it necessary that you use one or the other.

- JMS binding

  The JMS bindings provide support for JMS communication using “plain” JMS messages (that is not SOAP). You have the following options for JMS bindings:

  - The default messaging provider that is included in WebSphere Application Server. The transport mechanism for this provider is the service integration bus within WebSphere Application Server.
  - The WebSphere MQ messaging provider. You can use an installed WebSphere MQ as the messaging provider.

  Destinations are defined to the JNDI namespace and the bindings designate the JNDI name for the destination.

- Generic JMS binding

  The generic JMS binding application provides integration with non-JCA 1.5-compliant JMS providers that support JMS 1.1 and implement the optional JMS Application Server Facility, including Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic, and WebSphere MQ.

  Destinations are defined to the JNDI namespace and the bindings designate the JNDI name for the destination.

Guidelines for messaging bindings

A key decision by developers is choosing the best strategy for a particular application. Some guidelines are:

- If you are working with native JMS applications, then use JMS bindings.
- If you are working with JMS applications that are using the MQ backbone, then use MQ JMS bindings.
- If you are working with native MQ applications, then use the MQ binding.

The portability versus performance trade-off is another factor to consider.

- If portability is important, choose JMS bindings.
- If performance is important, using WebSphere MQ with native calls might be the appropriate choice.

MQ JMS bindings provide some portability with some performance benefits over JMS bindings. The choice of strategy decides what type of binding to select.

Web service binding

Web services bindings allow the call-out of SOAP-based Web services (through imports), as well as exposing SOAP-based Web services interfaces (through exports). Both SOAP/HTTP and SOAP/JMS are supported.

Both the HTTP and JMS transport types are presented as one binding type in WebSphere Integration Developer, but when you generate a Web service binding, you are asked which transport type you want to use. SOAP/HTTP is more common than SOAP/JMS.

As SOAP is well-defined and maps cleanly into the SCA and SDO model, there is no requirement (or facility) to provide any form of data binding, either for imports or exports. The Web services bindings support the well-known encoding styles: document/literal, doc-lit-wrapped, RPC-literal, and RPC-encoded.

When you generate a Web service binding for an export, a WSDL file for the service is created. When you generate a Web service binding for an import, you can specify an existing WSDL file for the service or launch a wizard to help you define the service.

A Web services binding that uses SOAP/JMS, supports JMS using the default WebSphere Application Server default messaging provider in a point-to-point configuration. The SOAP/JMS binding does not support: generic JMS, MQ JMS, or the JMS broadcast mode.

SCA binding

An SCA binding on an import component specifies an SCA-bound export component in another module. That export can be connected to either a component or import. As a result, the SCA import binding requires only two pieces of information:

- The name of the export that it is calling
- The module containing that export

The export must also have an SCA export binding. If the export has any export binding other than SCA, you cannot import it in another module with the SCA import binding. You must import it with the corresponding import binding type.

HTTP binding

HTTP is a widely-used protocol for transferring information on the Web. The HTTP binding supports the HTTP 1.0, HTTP 1.1, and SSL protocols.

Messages are presented to components in a manner that preserves HTTP format and message header information. An existing data binding framework is extended for HTTP conventions and provides mapping between SCA messages and HTTP message headers and bodies.

When you install an SCA module containing HTTP imports or exports, the runtime environment is automatically configured appropriately to allow connectivity to HTTP.

An HTTP binding on an import specifies the endpoint URL for the target application. An HTTP binding on an export specifies the context path for your module.

WebSphere Adapters

WebSphere Adapters enable interaction with enterprise information systems. The Enterprise Service Discovery tool can be used to create import and export components representing applications on the systems. An EIS binding is associated with the adapter.

2.1.4 Quality of service

Qualifiers in SCA allow developers to place quality of service (QoS) requirements on the SCA run time. There are several different categories of qualifiers available in SCA:

- Security
- Transactions (with ActivitySessions as a special type)
- Reliable messaging

Each qualifier has a particular scope within the Service Component Definition Language (SCDL) specification for an SCA component where the qualifier can be added (interface, implementation, and partner reference).

2.2 WebSphere Process Server components

WebSphere Integration Developer provides an assembly editor where the developer groups service components into modules and specifies which service interfaces are exposed by the module to outside consumers. Services that are available include imported components such as Java beans or Web services and service components that WebSphere Process Server and WebSphere Enterprise Service Bus provide. Modules are then connected to form complete integration solutions.

Figure 2-4 shows the components of WebSphere Process Server and indicates where they are developed in WebSphere Integration Developer.

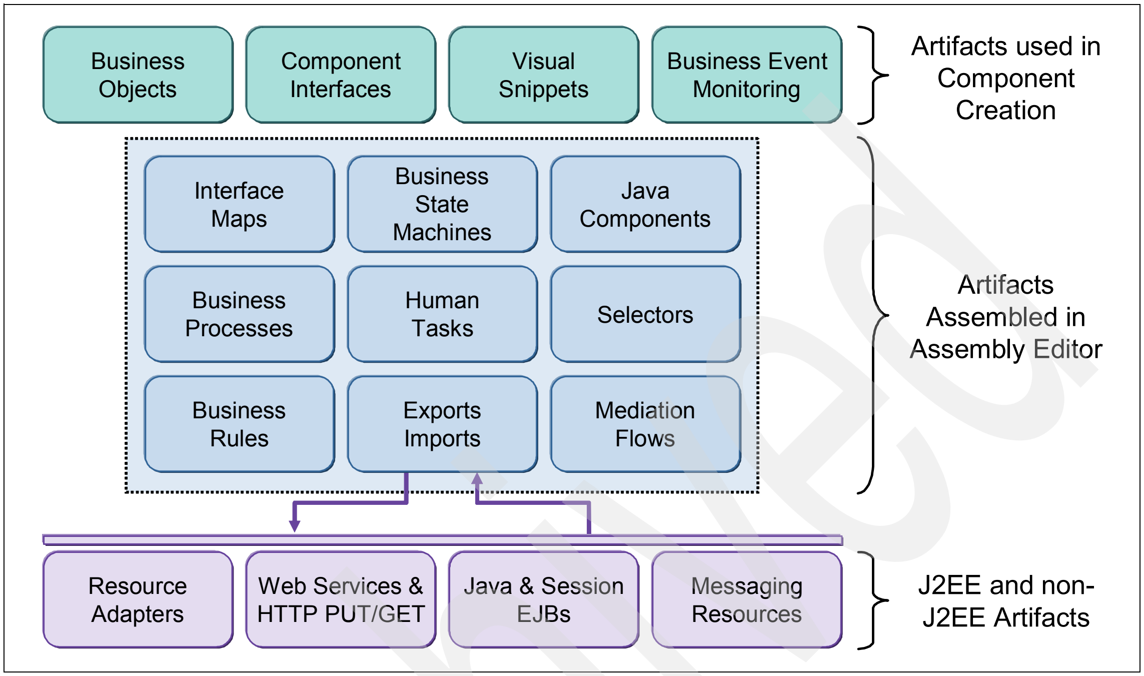


Figure 2-4 Key features of WebSphere Integration Developer

2.2.1 WebSphere Process Server specific components

WebSphere Process Server provides a hosting environment for business processing. WebSphere Process Server includes ESB functionality that is provided by WebSphere Enterprise Service Bus. Both business processes and mediations can be deployed to a WebSphere Process Server application server.

This section focuses on the components of WebSphere Process Server that support business processing applications. This type of application is developed in a module, also referred to as a *business integration module*.

Business state machines

A *business state machine* provides another way of modeling a business process, enabling a business process to be represented based on states and events.

It is an event-driven business application in which external operations trigger changes that guide the state machine from one discrete mode to another. Each mode is an individual state, and this mode determines what activities and operations can occur.

State machines are useful modeling aids for developing real-time or event-driven systems because they show dynamic behavior. You can develop state machines during all phases of a software project and for business modeling. You can use state machines in the following situations:

- During business modeling, you can create state machines to model a use case scenario.
- During analysis and design, you can model event-driven objects that react to events outside an object's context.
- During analysis and design, you can use several state machine diagrams to show different aspects of the same state machine and its behavior.

Business processes

The *business process* component in WebSphere Process Server implements a WS-BPEL compliant process engine.

Business processes are an important concept in today's integrated environments and play a key role in business-to-business and enterprise application integration scenarios by exposing the appropriate invocation and interaction patterns. Processes are the building blocks for developing consistent distributed applications in heterogeneous environments.

Process-based applications are composed of two distinct pieces:

- A process model that describes the logical order in which the different activities of the process model are being carried out
- The individual services and components that implement the various activities

Therefore, a business process is a set of business-related activities, rules, and conditions that are invoked in a defined sequence to achieve a business goal. Those activities can be a variety of different resources that are represented as one or more SCA components. All activities are logically connected with implicit or explicit connectors (control links), which define the logical order in which the business process is executed. Process activities can be executed in sequence or in parallel, depending on the business requirement and the internal dependency of the process steps.

The purpose of the business process container (BPC) is to interpret the process templates, manage the life cycle of business processes, navigate through the associated process model, and integrate the appropriate business functions.

A business process runs under one of two execution modes:

- *Microflows* run under a single transaction in a short period of time.
- *Long-running* business processes run in a series of chained transactions over days, months, or even years. Long-running processes can be optimized with respect to transaction boundaries.

Business process developers can declare transaction behavior for invoke, human task, and snippet activities:

- *Commit before* guarantees that the activity runs in a new transaction.
- *Commit after* guarantees that the current transaction is committed after executing the activity.
- *Requires own* guarantees that the activity runs in a new transaction and the current transaction is committed after executing the activity.
- *Participates* behavior states that the activity runs within an existing transaction, if one is available.

Human tasks

Human tasks in WebSphere Process Server are stand-alone components that can be used to assign work to employees or to invoke any other service.

Human tasks are used to allow components to “invoke humans as services.” The *human* is actually a person that is authorized to select that task from a group of authorized users. Because the human is considered simply a service, the replacement of one service for another is just a simple matter of component wiring or assembly. As a result, an operation performed by a person in the first place could be replaced by an operation represented as a business process, a Web service invocation, an operation on an enterprise bean, a CICS transaction, or other component interfaces.

Human tasks provide a common interface for humans to deal with human centric and automatic tasks in a uniform way. Client applications using the HTM API provide users with a single view to start, for example, a BPEL process, a document management system, or a Workplace™ application. The API allows users to create and start tasks independently of their implementation and, as a result, to provide a uniform administration scheme throughout applications.

A task activity contains the data that is required to complete the task. Some of this information comes from the specific business scenario (as an example, the creation of purchase requisition). The properties of a task type represent the meta information specific for that task. Meta information can be specified when creating a task or by defining a task type called *task template*.

The human task manager provides the human task capabilities for WebSphere Process Server to:

- Start a process or other service components
- Implement staff activities
- Allow for process administration
- Create tasks dynamically that involves humans or services

The human task manager addresses the following basic scenarios:

- The *machine-to-human* (M2H) scenario is the classical staff service component scenario where a process engine (a *machine*) creates tasks for people who participate in the execution of a business process.
- The *human-to-machine* (H2M) scenario is different in the sense that it allows a person to create a task that is executed by an automatic service. An example of an automatic service is a BPEL business process or an arbitrary Web service that can do a variety of functions, such as invoking a SOAP service, calling a CICS transaction, invoking a method on an enterprise bean, or retrieving a document from a document management store.
- In the *human-to-human* (H2H) scenario a task is created by a person (a human) for another person. An example is a travel approval request that is created by an employee for his manager. Especially for ad-hoc scenarios, this scenario plays a major role.

Selectors

A *selector's* main job is to determine dynamically which implementation to invoke based on some defined set of criteria (currently, based on a date and time). Think of a selector as a router that routes inputs and outputs to the appropriate target component based on a certain date (past, current, or future).

Business rules

Business rules are a means of implementing and enforcing business policy by externalizing business function, which enables dynamic changes of a business process.

Relationships

A *relationship* establishes associations between semantically equivalent business entities. For example, when customer business objects exist in different EIS applications with different keys, correlating key values allows synchronization of record creation and updates across systems. WebSphere Process Server manages the relationship data, not the application, and simplifies the task of the integration developer.

Relationship is one of the transformation rules that you can set from a business object map. Therefore, the participants are at the business object level. Relationships are an SCA artifact that you can define in a shared library, and those modules that contain participating business objects need to include the library as a dependency.

Business objects

Business objects are enhanced Service Data Objects (SDOs). SDOs provide an abstraction that can be used over various types of data, providing a common mechanism for accessing data. Business objects include extensions that are important for integration solutions and are used to further describe the data that is being exchanged between SCA services.

Special business object types

Application-specific business objects (ASBOs) are business objects with a data structure that corresponds to a particular application. Adapters create an instance of an ASBO at run time to hold application data, information about the data, and actions to be performed on the data.

Generic business objects (GBOs) are business objects with no application-specific Information and not tied to any application. It is a canonical representation of the information that is held in an ASBO. The generic business object serves as a common application- and implementation-independent data set.

Business graph

A *business graph* is the container around the top level of a business object. The business graph adds change summary, event summary, and verb capability to the business object. For example, a business graph can represent a synchronization scenario between two EIS systems.

2.2.2 WebSphere Enterprise Service Bus specific components

Development of mediations for WebSphere Enterprise Service Bus development is similar to that of developing business processes for WebSphere Process Server. However, a mediation is developed within a mediation module that has a mediation flow component as its implementation. The mediation module is a specific type of SCA component that can process or mediate service interactions.

Mediation flow components

A *mediation module* can have one mediation flow component. The implementation of the mediation module is a *mediation flow*.

Mediation flows contain mediation logic and the processing steps of a request declared in a graphical way using mediation primitives. Mediation flows are distinguished between a request flow and a response flow. In both directions, logic can be added or modifications can be applied to process a message.

Service message objects

Messages can come from a variety of sources, so the payload has to be able to carry a number of different types of messages. Mediation primitives need to be able to operate on these messages, and service message objects (SMOs) represents the common representation that is needed. The SMO model is a pattern for using SDO DataObjects to represent messages. SMO extends SDO with additional information to support the needs of a messaging subsystem.

2.2.3 Common event infrastructure

Events occur all the time within a software system. An *event* can be defined as something interesting that happens within the environment. This might be as simple as the completion of a step in a process or as sophisticated as the arrival of a new customer order. The Common Event Infrastructure (CEI) is a powerful technology that is implemented within WebSphere Process Server and that provides for the creation, consumption, and warehousing of events. The events are generated by a standard API and are available to be consumed by applications that might also use this API. The content of an event is configurable by the component that produces it but always contains a base set of attributes that are common to all events.

CEI provides the ability to audit, monitor, and track the execution of processes throughout the entire life cycle. The application that creates the event object is called the event source. The event source passes the event object to the event infrastructure. The event infrastructure's role is to extract information from the WebSphere runtime environment and add it to the event object. The event object is then passed onto any applications that have expressed an interest in receiving it. These applications are called event consumers. The event infrastructure also stores the event object in a database for later retrieval.

3

Basics of development

WebSphere Integration Developer is the common tool for building SOA-based integration solutions across WebSphere Process Server, WebSphere Enterprise Service Bus, and WebSphere Adapters. This chapter discusses the development concepts and methods that are common across these products using WebSphere Integration Developer. It includes the following topics:

- WebSphere Integration Developer key features
- Working with WebSphere Integration Developer
- Project types
- Business objects
- Interfaces
- Module assembly
- Business object maps
- Using a stand-alone reference
- Working with databases in the workspace
- Derby ij tool
- Deploying modules
- Test tools
- Team development

3.1 WebSphere Integration Developer key features

WebSphere Integration Developer is a development environment for building integrated applications. It enables integration developers to create, manage, and test services for IBM WebSphere Process Server and WebSphere Enterprise Service Bus. The features in WebSphere Integration Developer separate business logic from implementation details.

3.1.1 Base tools and run times

WebSphere Integration Developer is built on the Rational Software Development V7.0.0.5 Platform, which is based on Eclipse 3.2 technology. Each IBM product that is built on the Rational Software Development Platform coexists and shares plug-ins and features with other products that are based upon this platform. The Rational Software Development Platform is installed once per system with the first product that is installed. As other products that are built on this platform are installed on the system, only the necessary plug-ins are installed.

Note: For more information about Eclipse and tutorials, visit the following Web site and explore the Getting Started pages:

`http://www.eclipse.org`

3.1.2 Developer roles

There are two primary user roles that are associated with WebSphere Integration Developer:

- integration developer
- Application developer

Integration developer

The *integration developer* integrates existing and new services and users into the business process definition the service composition components. The specialist typically uses visual composition tools and service-bus configuration tools to wire abstract service components that comprise the business processes. The *integration specialist*, along with the *enterprise architect*, is involved in establishing an approach that satisfies the security and quality of service requirements of the enterprise when composing services from business partners and other service providers outside of the enterprise operational environment.

The integration developer is the primary role. This role focuses on building service-oriented solutions. This user role expects the tooling to simplify and abstract advanced IT implementation details. The integration developer is familiar with basic programming constructs such as loops, conditions, and string manipulation.

Application developer

The *application developer* implements the design for services that are provided by the software architect. This includes using an appropriate language and technology in which to implement the services, and following the design for those components provided by the software architect. It might also include working with existing application functions, languages and technologies. The developer assists the software architect in identifying potential re-use of these functions and in helping determine how best to extend or re-factor those functions to enable a better fit with the business design.

Unlike the integration developer, the application developer is typically knowledgeable in one or more programming languages or application development platforms. The application developer also has a basic understanding of one or more technologies that are associated with building integrated business applications such as SOA, process choreography, workflow, WSDL, or BPEL. Because the application developer has more extensive knowledge of the implementation details that are associated with building a business application, it is this user who typically implements the application specific business logic for the actual business application being built. In addition, with the introduction of the SCA programming model, it is also the application developer who exposes specific application logic as an SCA service component.

3.1.3 Workspace and perspectives

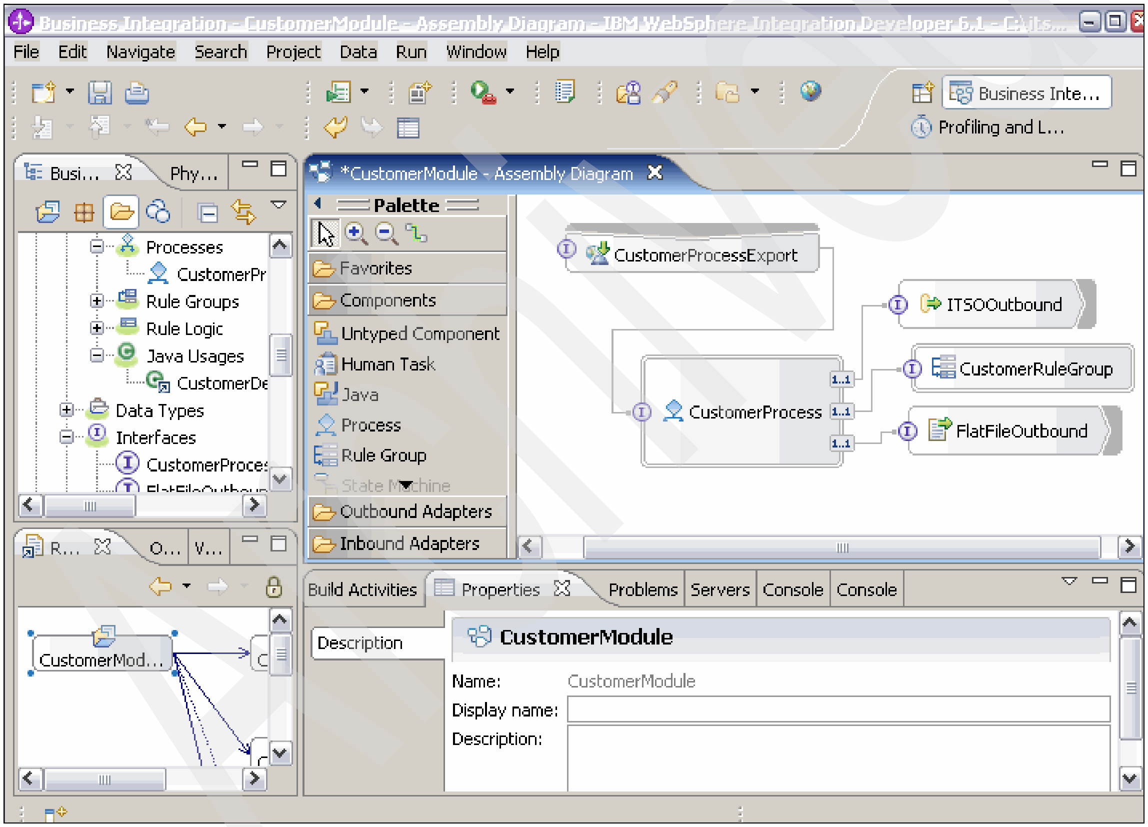
When you open WebSphere Integration Developer, you select a workspace to contain projects and test environment configuration. A *workspace* is a collection of projects and other physical resources that you are currently developing in the workbench. You can have multiple workspaces to segregate project development, switching between workspaces when necessary.

WebSphere Integration Developer provides perspectives for development, testing, and team repositories. A *perspective* is a grouping of views that are geared toward specific tasks. The primary perspective for working with business modules is the Business Integration perspective.

3.1.4 Assembly diagram

When you create a module (for business services) or mediation module, an assembly diagram for the module is created. As you add components to the module, you populate the assembly diagram with these components.

Figure 3-1 shows the Business Integration perspective with one module defined. The assembly diagram for the module is in the editor area on the upper right. Selecting a component displays information about it in the Properties view (bottom right). From the assembly diagram, you define the interfaces to each component, generate binding information for access to the module, and generate the implementation for service components.



Name: CustomerModule

Display name:

Description:

Figure 3-1 Business Integration perspective - assembly diagram

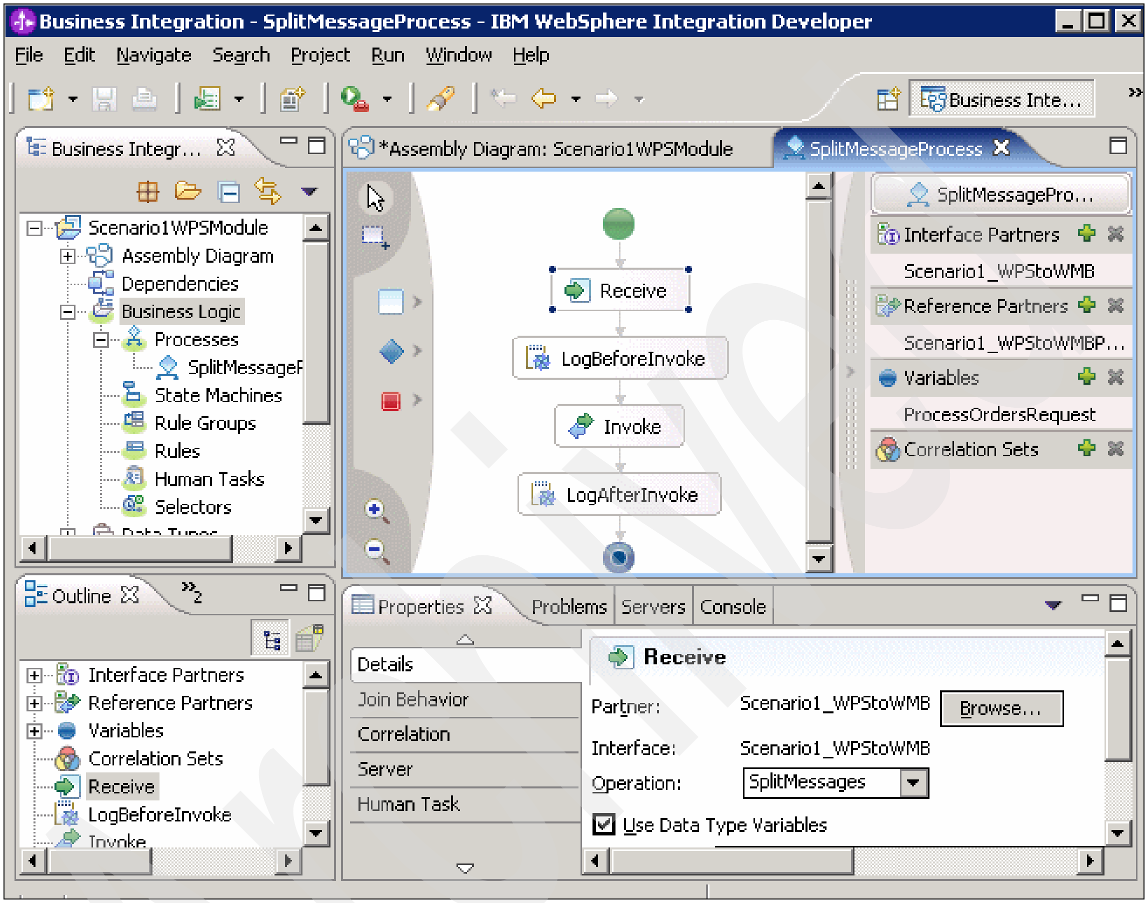
Icons in the module assembly diagram indicate the type of component or artifact. For example, Figure 3-1 shows:

- An export for the module. A Web service binding was generated for this component, denoted with the icon.
- An interface is defined on each component, denoted with the icon.
- References denoted by the icon.
- An import component with a JMS binding, denoted with the icon.
- A rule group, denoted by the icon.
- An import component for an adapter interface. It has an EIS binding, denoted by the icon.
- The service component in the middle was implemented as a business process, denoted by the icon.

3.1.5 Editors

Each artifact has an associated *editor* that provides features suited for editing that artifact (for example, WSDL files, XML, components, and so forth). The two primary editors that you use are the *business process editor* and the *mediation flow editor.*

Double-clicking a component in the assembly diagram opens the component for development with the appropriate editor. For example, Figure 3-2 shows a business process open with the editor.



Partner: Scenario1_WPStoWMB

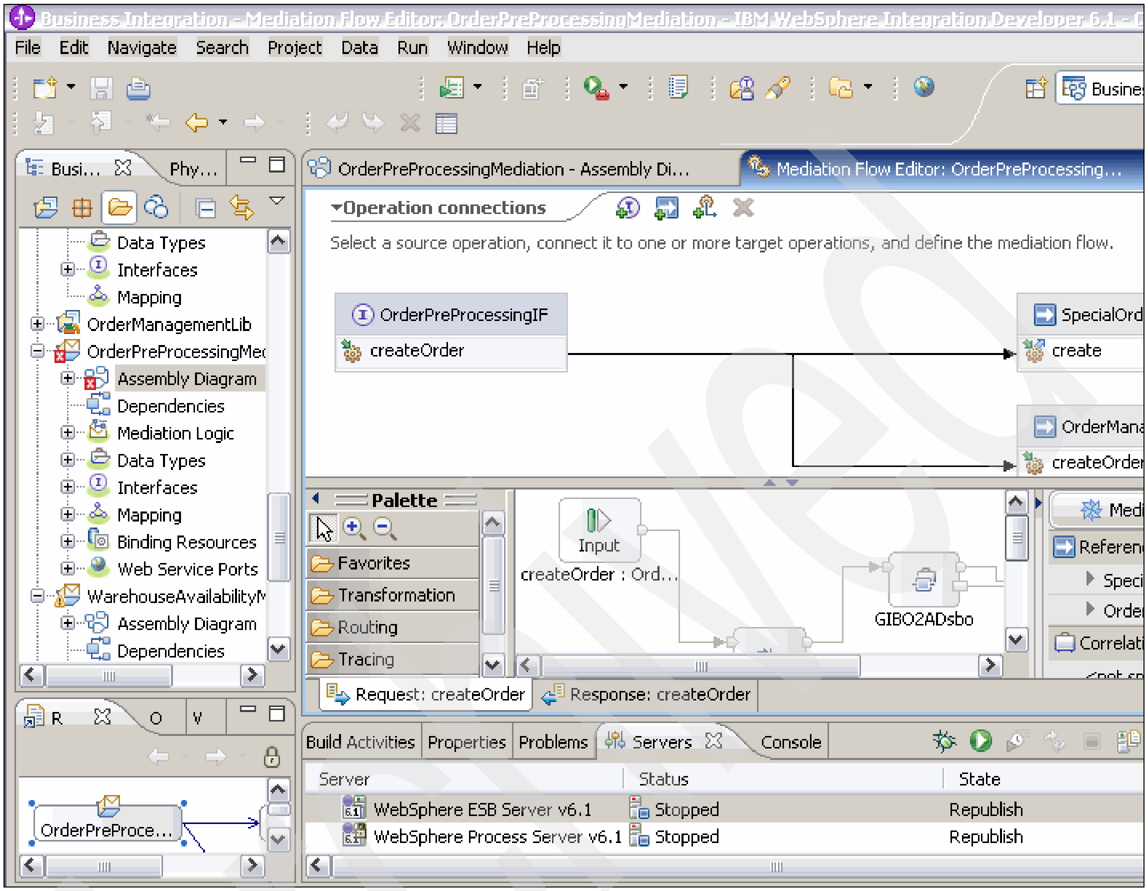
Interface: Scenario1_WPStoWMB

Operation: SplitMessages

☑ Use Data Type Variables

Figure 3-2 Business process editor

Figure 3-3 shows a mediation flow open in the editor.



Select a source operation, connect it to one or more target operations, and define the mediation flow.

Server	Status	State
WebSphere ESB Server v6.1	Stopped	Republish
WebSphere Process Server v6.1	Stopped	Republish

Figure 3-3 Mediation flow editor

3.1.6 Business Integration view

The Business Integration perspective has a Business Integration view that provides a logical view of the key resources in each module, mediation module, and library (as shown in Figure 3-4 on page 36). Use this view to locate artifacts with which you want to work. Double-click the artifact to open it in an editor.

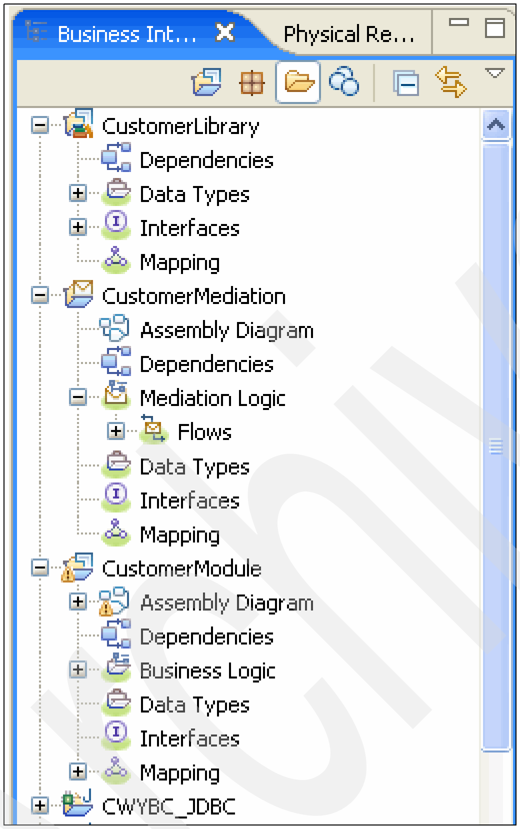
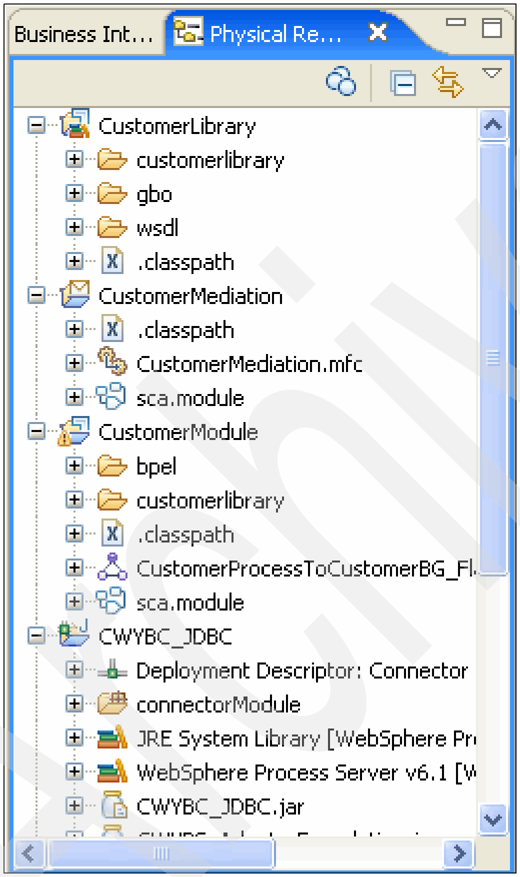


Figure 3-4 Business Integration view

3.1.7 Physical Resources view

The Physical Resources view shows all of the file-level resources from the modules and libraries in their natural form (as shown in Figure 3-5). For example, business objects are stored as XSD files, and interfaces are represented in WSDL format. You can also see generated artifacts that are not shown in the business Integration view, such as classpath files, the sca.module files, and the file resources for the resource adapters.



- CustomerLibrary
  - customerlibrary
  - gbo
  - wsdl
  - .classpath
- CustomerMediation
  - .classpath
  - CustomerMediation.mfc
  - sca.module
- CustomerModule
  - bpel
  - customerlibrary
  - .classpath
  - CustomerProcessToCustomerBG_Fl
  - sca.module
- CWYBC_JDBC
  - Deployment Descriptor: Connector
  - connectorModule
  - JRE System Library [WebSphere Pr
  - WebSphere Process Server v6.1 [W
  - CWYBC_JDBC.jar

Figure 3-5 Physical Resources view

3.1.8 Other Business Integration views

The Business Integration perspective has additional views that you might find useful as you develop your modules. You can add views that are not in the perspective by default by selecting **Window** → **Show View**.

References view

The References view shows all of the artifacts that reference the selected item in the Business Integration view, and it shows all the artifacts that it references (as shown in Figure 3-6).

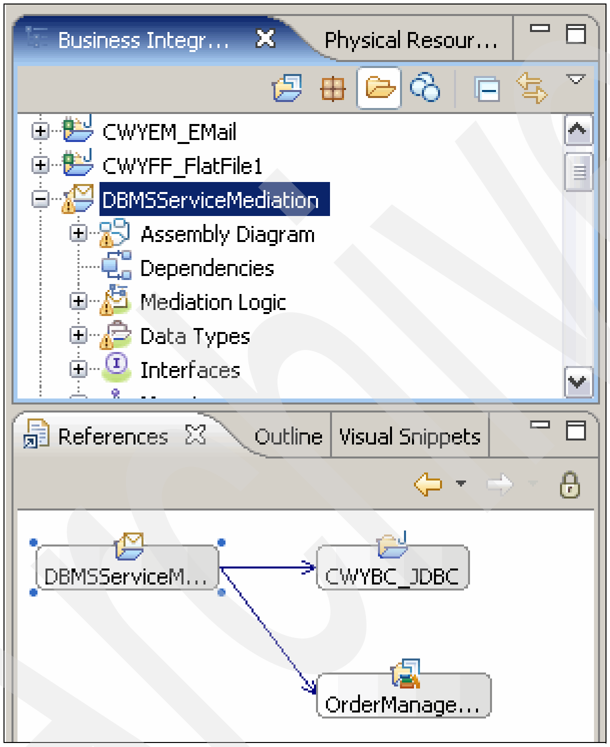
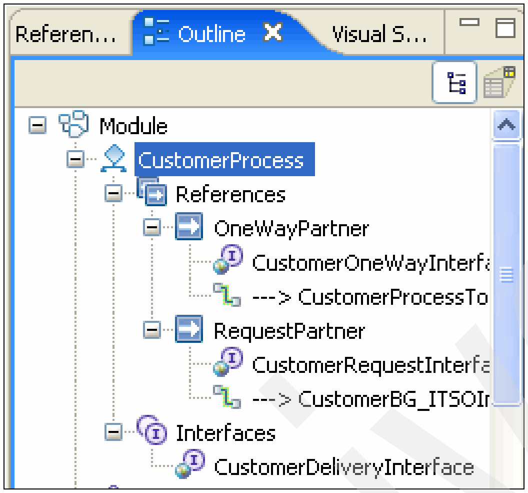


Figure 3-6 References view

Outline view in Show Outline mode

The Outline view in the Show Outline mode displays all the elements in the assembly diagram that is open in the editor pane. You enable the Show Outline mode by clicking the icon.



Referen... | Outline | Visual S...

- Module
  - CustomerProcess
    - References
      - OneWayPartner
        - CustomerOneWayInterfa
        - ---> CustomerProcessTo
      - RequestPartner
        - CustomerRequestInterfa
        - ---> CustomerBG_ITSOIr
    - Interfaces
      - CustomerDeliveryInterface

Figure 3-7 Outline view

3.2 Working with WebSphere Integration Developer

The basic steps that you follow for development in WebSphere Integration Developer are very similar, regardless of whether you are creating modules or mediation modules. This section takes you through these basic development steps. We discuss more details for developing the component implementations in Chapter 4, “Building business integration applications” on page 127 and Chapter 5, “Building mediations” on page 285.

3.2.1 Typical development flow

This section provides an overview of the development process for business integration and mediation modules.

A typical development flow is as follows:

1. Start WebSphere Integration Developer and open a workspace (described in 3.2.2, “Start WebSphere Integration Developer” on page 42)
2. Switch to the Business Integration perspective for development (described in 3.2.3, “Using the Business Integration perspective” on page 43).

3. Create a library to store artifacts, such as business objects and interfaces, that are shared among multiple modules (described in 3.3.1, “Libraries” on page 44).

   Best Practice: Artifacts such as business objects (data types) and interfaces need to be created within a library and referenced by the module. A library, such as a module, contains resources and code for applications. Libraries allow you to share these resources and code between modules. However, unlike modules, libraries cannot be deployed by themselves.

4. Create a new module or mediation module (described in 3.3.2, “Modules and mediation modules” on page 46).

   A module has an assembly diagram that shows the components of the module and how the interfaces and references are wired together. The module is the basic unit of deployment for WebSphere Process Server.

5. Create the business objects to contain the application data, for example, customer or order data (described in 3.4, “Business objects” on page 53).

6. Create the interface and define the interface operations for each component. The interface determines what data can be passed from one component to another (described in 3.5, “Interfaces” on page 63).

7. Create and implement the service components.

   A service component for a module can be implemented as one of the following components:

   – Java objects
   – Business processes
   – Human tasks
   – State machines
   – Rule groups (Business rules)
   – Selectors
   – Interface maps

   We discuss how to implement the components in a business integration module in Chapter 4, “Building business integration applications” on page 127.

   A service component for a mediation module will be implemented always as a mediation flow.

   We discuss how to implement the components in a business integration module in Chapter 5, “Building mediations” on page 285.

8. Build the module assembly by adding the service components, imports, and exports to the assembly diagram. Bind the imports and exports to a protocol (described in 3.6, “Module assembly” on page 69).

9. Test the module in the integrated test environment (described in 3.12, “Test tools” on page 105).

10. Deploy the module to WebSphere Process Server.

    When you deploy a module to the test environment or to WebSphere Process Server, WebSphere Integration Developer packages the module as a J2EE EAR file.

    For any given module project, there can be up to three J2EE staging projects that are generated with naming conventions based off of the module project’s name:

    – An Enterprise Application project.

    – An EJB™ project (which includes the generated EJBs that represent the runtime artifacts that make components into reality).

    – A Dynamic Web project (which includes artifacts that represent Web components, for example servlets and JSPs). A dynamic Web project is generated when needed.

    You cannot see these projects in the Business Integration view. To view these projects, you need to change perspectives, for example, the Web perspective.

    > **Note:** J2EE staging projects (EJB projects and Web projects) are generated artifacts from the deployment process. When you issue a **Project** → **Clean** command, the system discards the former J2EE staging project and regenerates it.

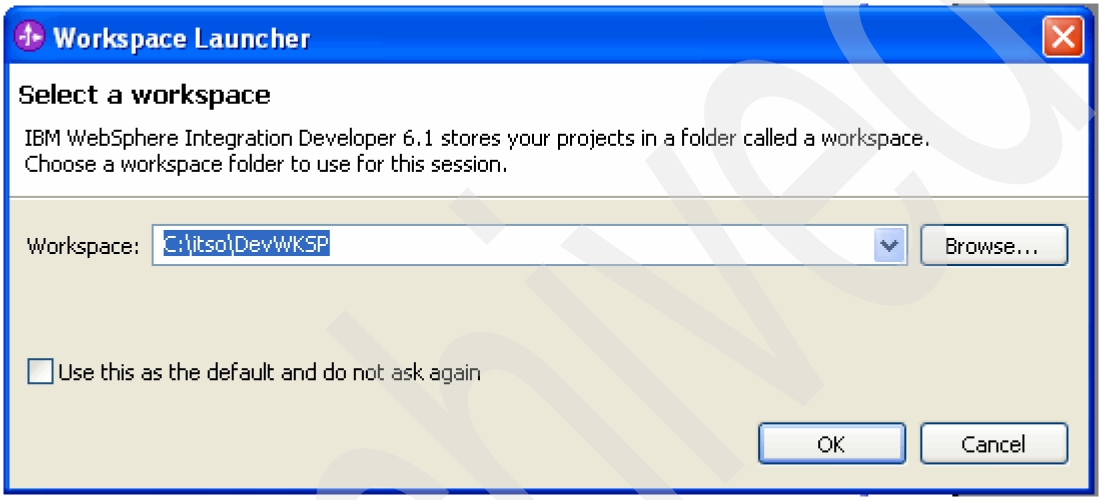
    This step is discussed in *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 3: Run time*, SG24-7643.

11. Share the tested module with others on the team by putting it in a repository (described in 3.13, “Team development” on page 115).

3.2.2 Start WebSphere Integration Developer

To start the WebSphere Integration Developer 6.1 with a new workspace, follow these steps:

1. From the start menu select **Start** → **Programs** → **IBM WebSphere Integration Developer** → **IBM WebSphere Integration Developer 6.1** → **WebSphere Integration Developer 6.1**.
2. Start WebSphere Integration Developer and create a new workspace in C:\itso\DevWKSP, using upper case characters where specified.



Workspace Launcher

Select a workspace

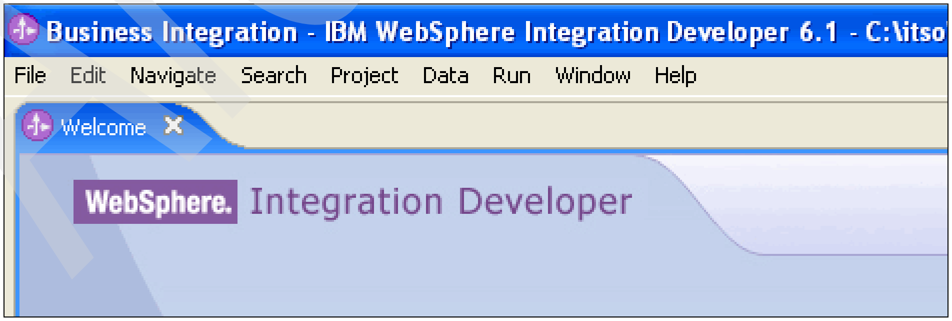
IBM WebSphere Integration Developer 6.1 stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\itso\DevWKSP

□ Use this as the default and do not ask again

Figure 3-8 Workspace launcher

3. When the WebSphere Integration Developer 6.1 opens, close the Welcome page.

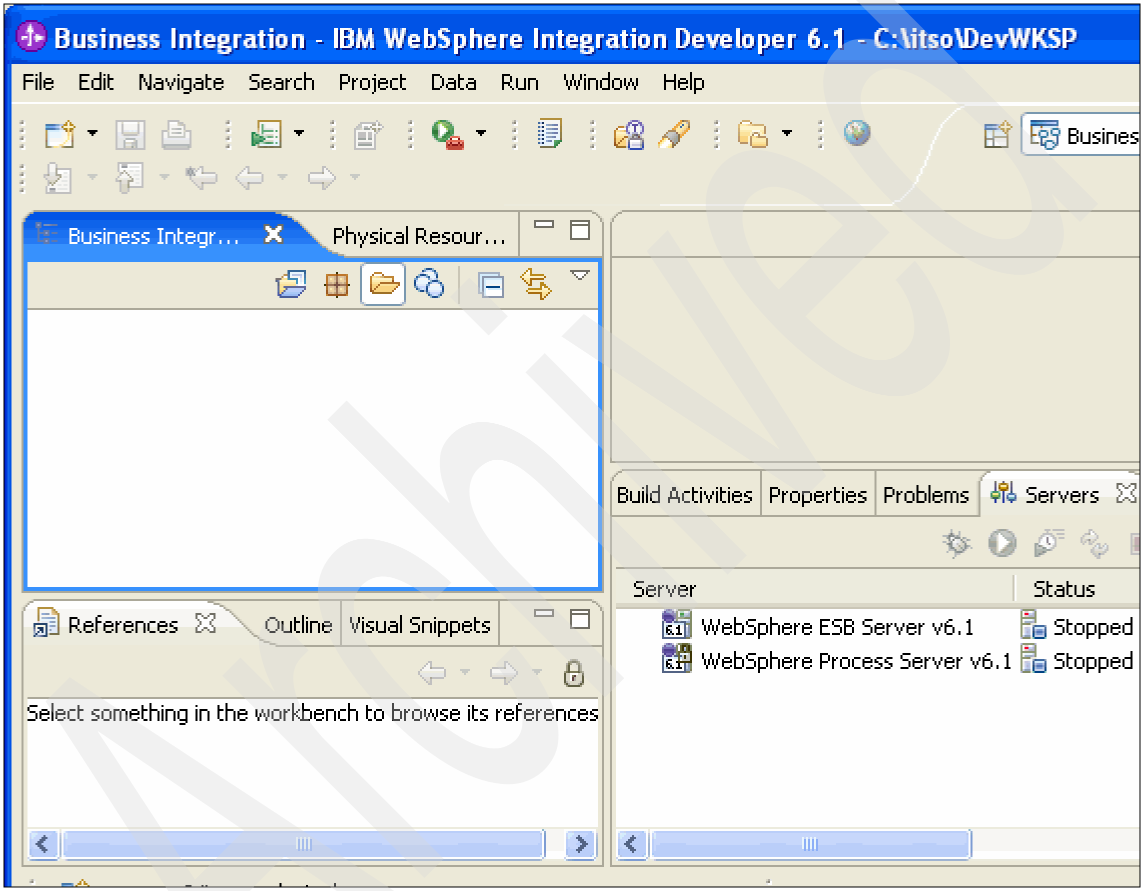


Business Integration - IBM WebSphere Integration Developer 6.1 - C:\itso
File
Edit
Navigate
Search
Project
Data
Run
Window
Help
Welcome
WebSphere. Integration Developer

Figure 3-9 Business Integration Welcome page launcher

3.2.3 Using the Business Integration perspective

The Business Integration view provides a logical view of the key resources in each module, mediation module, and library. Within each project, the resources are categorized by type. You can also use the Business Integration view to navigate through Java and J2EE resources and to open the various Java and J2EE editors. See Figure 3-10.



Select something in the workbench to browse its references

Server	Status
WebSphere ESB Server v6.1	Stopped
WebSphere Process Server v6.1	Stopped

Figure 3-10 Business Integration perspective

By default, when you launch the product, the Business Integration perspective is opened. This perspective includes views that are designed for working with SCA modules and their implementation. If the Business Integration perspective is not open, you can open it as follows:

1. Select **Window** → **Open Perspective** on the menu bar.
2. Select **Other** from the drop-down menu.
3. Select the **Business Integration perspective**. If you do not see the perspective, select **Show all**.

The Business Integration view is the primary navigation tool for the perspective. Modules and their artifacts are organized in a logical manner. Double-clicking an item in this view opens it with the appropriate editor in the editor area (upper right section of the perspective).

The Physical Resources view provides a comparable view of the modules and their artifacts, but it is organized to reflect the actual file structure on the system.

3.3 Project types

A *project* is an organized collection of folders or packages that are related to a single work effort. The project types that you work with most in WebSphere Integration Developer are *modules*, *libraries*, and *mediation modules*.

3.3.1 Libraries

Often, artifacts need to be shared so that resources in several modules can use them. A library project is used to store these resources. Libraries contain the following artifact types:

- Data types
- Interfaces
- Mappings

In order for a module or mediation module to use the resources from a library, the library has to be added as a *dependency* to the module. A library cannot be deployed by itself. However, you can add a library to the module and select to deploy it with the module.

Also, you can add library dependencies to a library. For example, if one library uses resources in another library, then you need to add the library dependency.

You can define dependencies when you create the project, if the library exists, or later using the dependency editor.

Creating a library

To create a new library, follow these steps in the Business Integration perspective:

1. Right-click the Business Integration view and select **New** → **Library** from the context menu (Figure 3-11). If Library is not an option, select **Other** → **Library**.

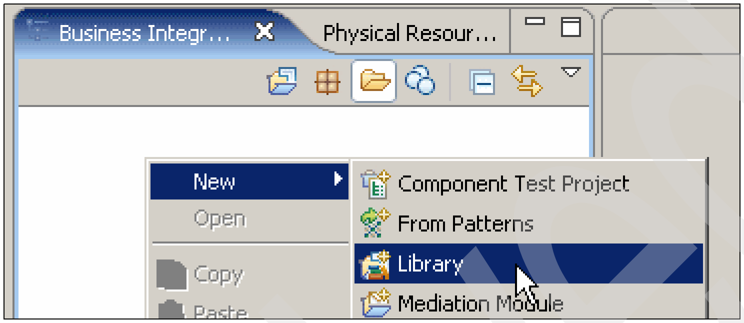
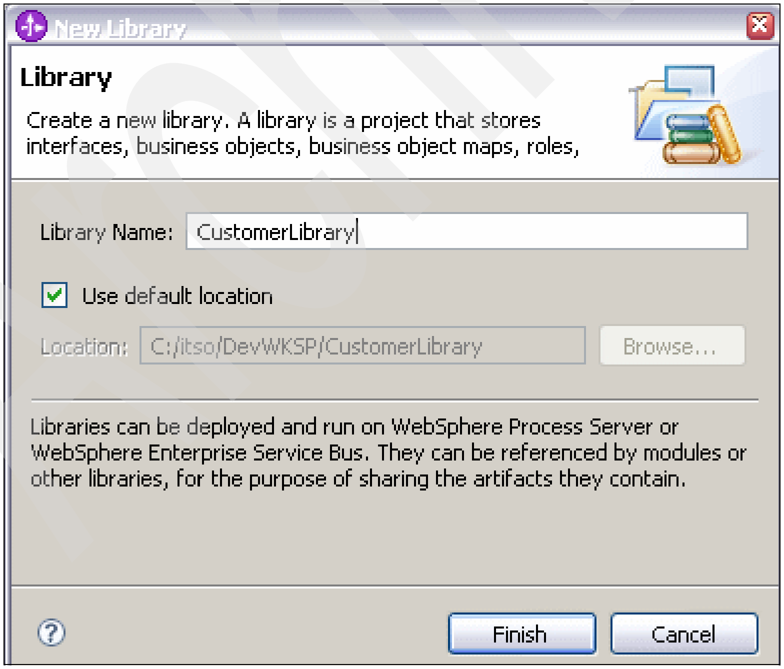


Figure 3-11 Create a new library

2. Enter a name for the library and ensure that the “Use default location” option is selected (Figure 3-12). Click **Finish**.


New Library

Library

Create a new library. A library is a project that stores interfaces, business objects, business object maps, roles,

Library Name: CustomerLibrary

☑ Use default location

Location: C:/itso/DevWKSP/CustomerLibrary

Libraries can be deployed and run on WebSphere Process Server or WebSphere Enterprise Service Bus. They can be referenced by modules or other libraries, for the purpose of sharing the artifacts they contain.

Figure 3-12 Name the new library

The new library is created, and you can view the structure of the library in the Business Integration view, as shown in Figure 3-13.

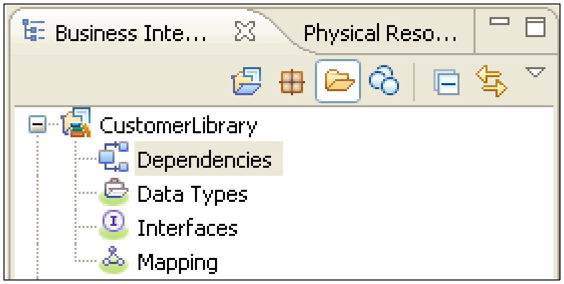


Figure 3-13 Library structure

3.3.2 Modules and mediation modules

The fundamental project type for building an SCA application in WebSphere Integration Developer is a *module*. A module is a project that is used for development, version management, and organizing business integration resources.

A module that includes business integration logic is referred to simply as a module or sometimes as a *business integration module.* A business integration module can contain various types of components, such as business processes, human tasks, rules, state machines, and so forth, that are assembled together to achieve an intended business goal. A module must be deployed to a WebSphere Process Server runtime environment.

Modules that provide mediation services are referred to as *mediation modules*. Mediation modules contain mediation flows and other components that are required to provide the mediation. Mediation modules can be deployed to WebSphere Enterprise Service Bus or WebSphere Process Server runtime environments.

Modules and mediation modules, packaged as enterprise archive files (EAR files) are the basic unit of deployment into WebSphere Process Server and WebSphere Enterprise Service Bus runtime environments.

Creating a module

The following steps illustrate how to create a module and the resulting structure of the module:

1. Right-click in the Business Integration view and select **New** → **Module** (Figure 3-14). If Module is not an option, select **Other** → **Module**. The New Module wizard opens.

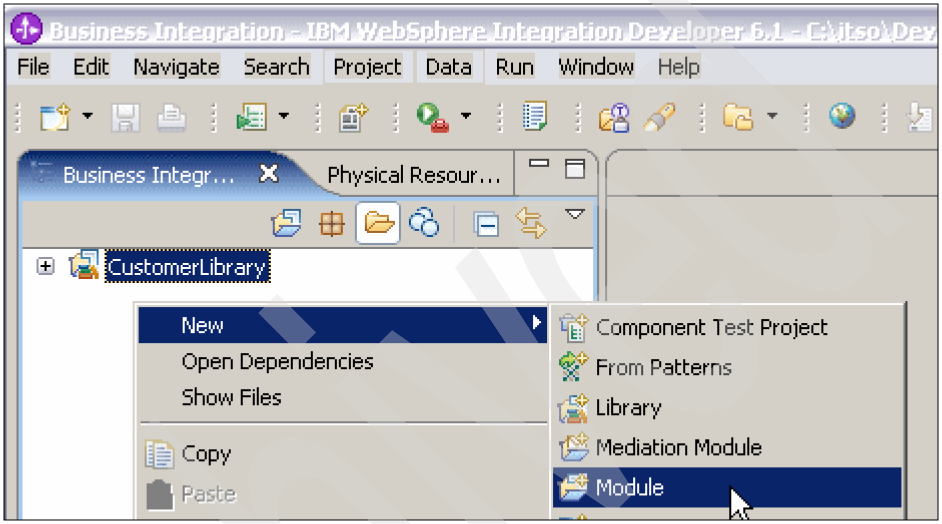
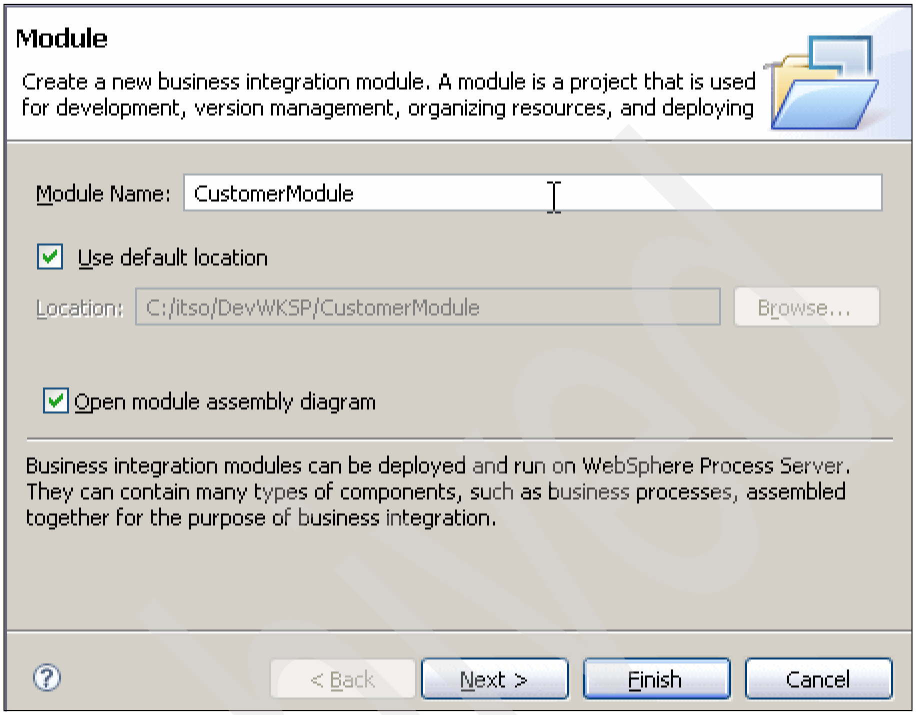


Figure 3-14 Create a new module

2. In the **Module Name** field, type the module name, as shown in Figure 3-15.



Module

Create a new business integration module. A module is a project that is used for development, version management, organizing resources, and deploying

Module Name: CustomerModule

☑ Use default location

Location: C:/itso/DevWKSP/CustomerModule

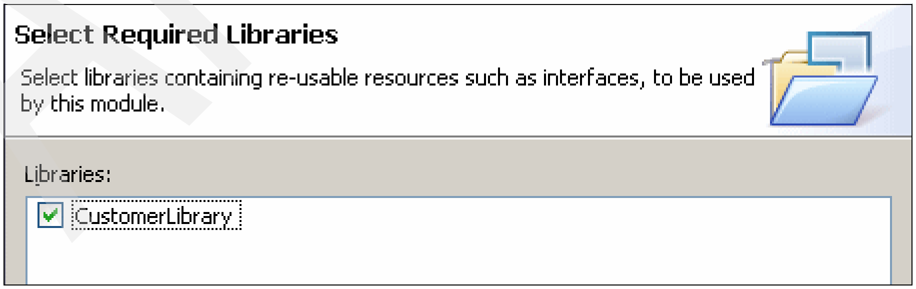
☑ Open module assembly diagram

Business integration modules can be deployed and run on WebSphere Process Server. They can contain many types of components, such as business processes, assembled together for the purpose of business integration.

Figure 3-15 Create a new module

If you have a library defined in your workspace, click **Next** to select the required libraries for this module. If not, click **Finish**. In this example, we have a library, and click **Next**.

3. Select any libraries that you want to add as a dependency as shown in Figure 3-16 and click **Finish**.



Select Required Libraries

Select libraries containing re-usable resources such as interfaces, to be used by this module.

Libraries:

☑ CustomerLibrary

Figure 3-16 Add dependencies

4. The new module and artifacts display in the Business Integration view, as shown in Figure 3-17.

   Folders exist for the artifact types that can be placed in the module, and the assembly diagram opens.

   > **Note:** Dependencies, data types, interfaces, and mappings can be shared across modules only when they are stored in libraries. You need to create in the module only artifacts of these types that you do not intend to use beyond the module.

   The assembly diagram opens but is initially empty.

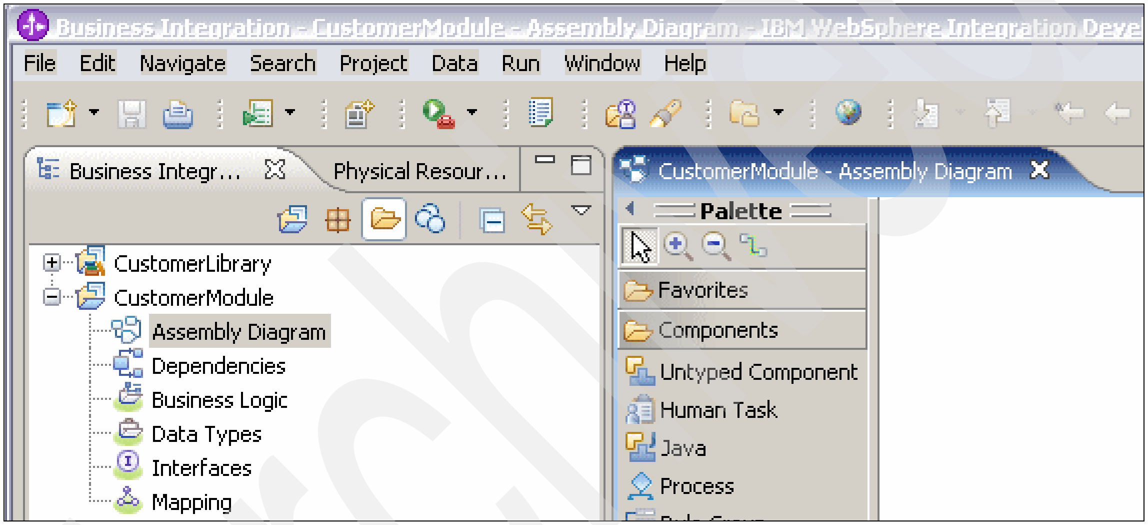


Figure 3-17 New module

5. The next step is to create the artifacts (for example, business objects, and interfaces), to implement the logic (Figure 3-18 shows the possible artifacts that can be created to implement the business logic), and to populate the assembly diagram with the components for the module.

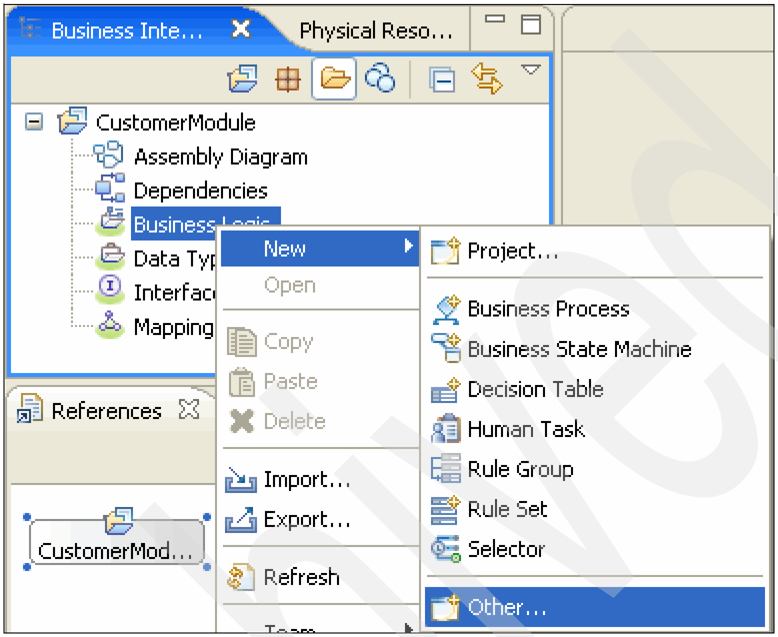
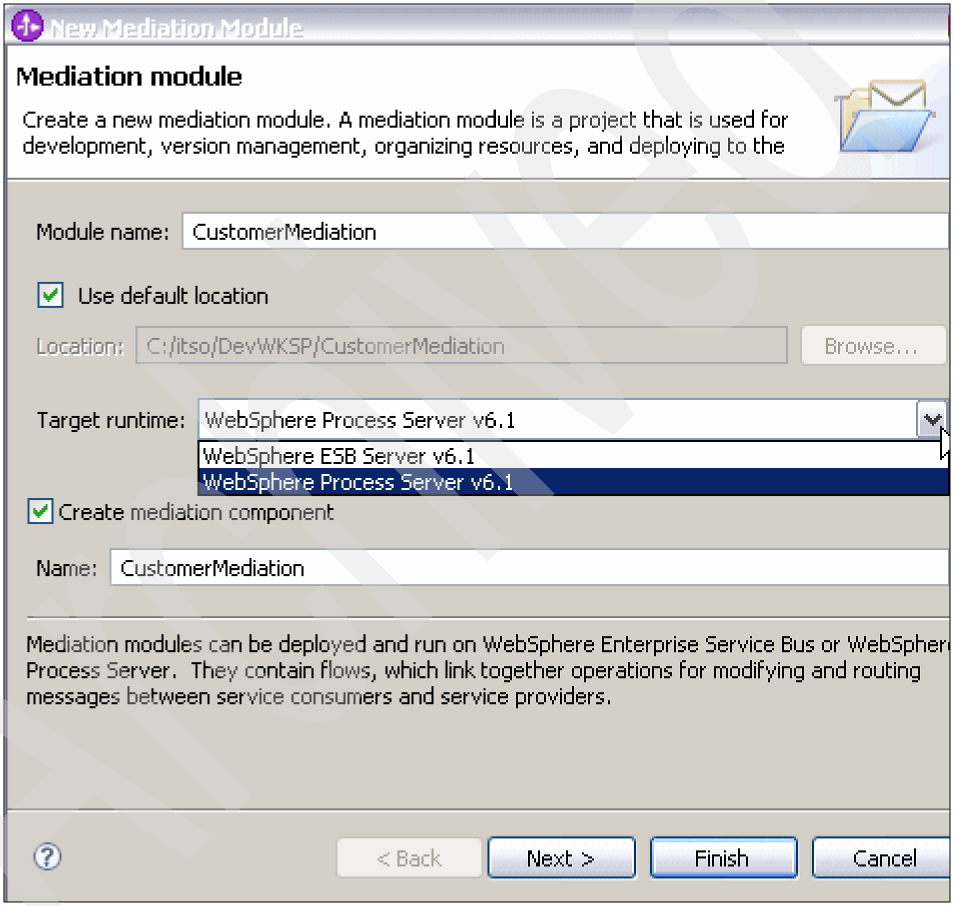


Figure 3-18 Create new business logic artifacts

Creating a mediation module

To create a mediation module, open the Business Integration perspective and do the following steps:

1. Right-click in the white space of the Business Integration view and select **New** → **Mediation Module** from the context menu.
2. Enter a name for the mediation. Select the target run time (mediations can run in either WebSphere Process Server or WebSphere Enterprise Service Bus run times). Then, click **Next**. See Figure 3-19.



New Mediation Module

Mediation module

Create a new mediation module. A mediation module is a project that is used for development, version management, organizing resources, and deploying to the

Module name: CustomerMediation

☑ Use default location

Location: C:/itso/DevWKSP/CustomerMediation

Target runtime: WebSphere Process Server v6.1

WebSphere ESB Server v6.1
WebSphere Process Server v6.1

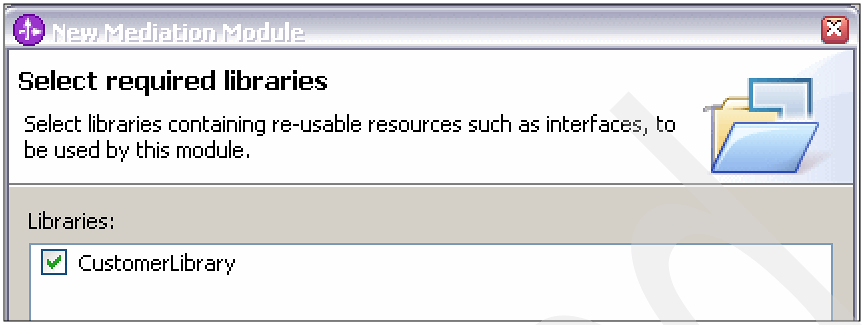
☑ Create mediation component

Name: CustomerMediation

Mediation modules can be deployed and run on WebSphere Enterprise Service Bus or WebSpher
Process Server. They contain flows, which link together operations for modifying and routing messages between service consumers and service providers.

Figure 3-19 Create the module

3. To add dependencies, select the required libraries as illustrated in Figure 3-20.



New Mediation Module

Select required libraries

Select libraries containing re-usable resources such as interfaces, to be used by this module.

Libraries:

☑ CustomerLibrary

Figure 3-20 Add libraries as dependencies

4. Click **Finish**. The new mediation module displays in the Business Integration view, and the assembly diagram for the module opens, as shown in Figure 3-21. Note that the assembly module has one component by default that represents the mediation flow because we elected to create the mediation component in step 2.

   A mediation module can have the following implementation types:

   – Mediation flow
   – Java

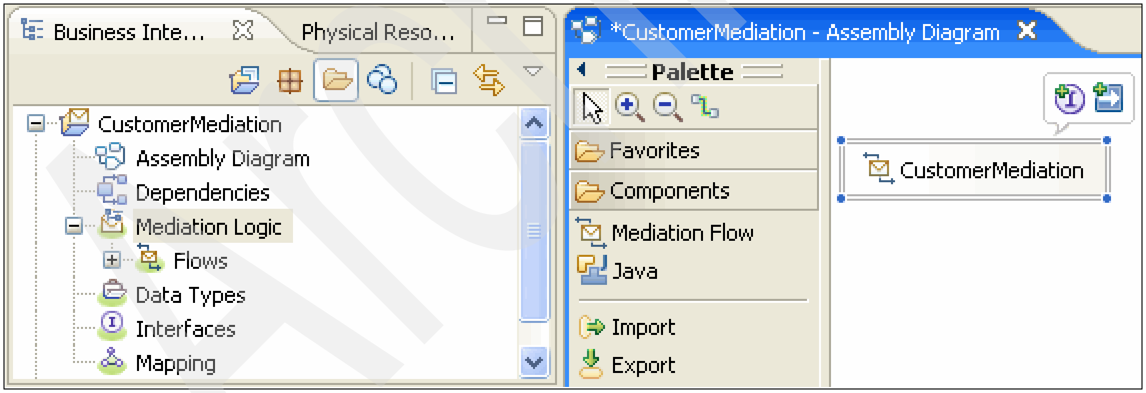


Figure 3-21 CustomerMediation assembly diagram

3.4 Business objects

Business objects are one of the primary building blocks of any business integration solution. They are containers for application data that represent business functions or elements, such as a customer or an invoice. WebSphere Integration Developer enables the simple creation of business objects using the business object editor.

3.4.1 Business object fields

A business object contains fields that define the content of the business object. Business objects can extend (define a super-set of fields) other business objects through parent/child relationships; however, a business object can only inherit from a single parent. These objects can also be used in conjunction with each other to perform a desired task. You can create and edit business objects using the business object editor.

Essentially, business object fields are the mechanism through which you define what information a business object can hold and how that information should be accessible. Each field has a name, type, cardinality, and other optional properties. A business object is simply a container for the data that is specified in its fields. An empty business object without fields is essentially useless because it does not have the means to actually hold any data.

Business objects are the primary data structure for:

- Business data
- Data types that are defined in WSDL (interface) definitions

WebSphere Integration Developer's *business object editor* is used to build and edit business objects and their fields through a graphical interface.

You can add fields to a business object after you create it. In addition, you have the following options:

- Create a new business object that inherits data from an exiting business object
- Create a new business object from entries in an existing business object
- A business object can contain scalars or arrays of other business objects

3.4.2 Business graph

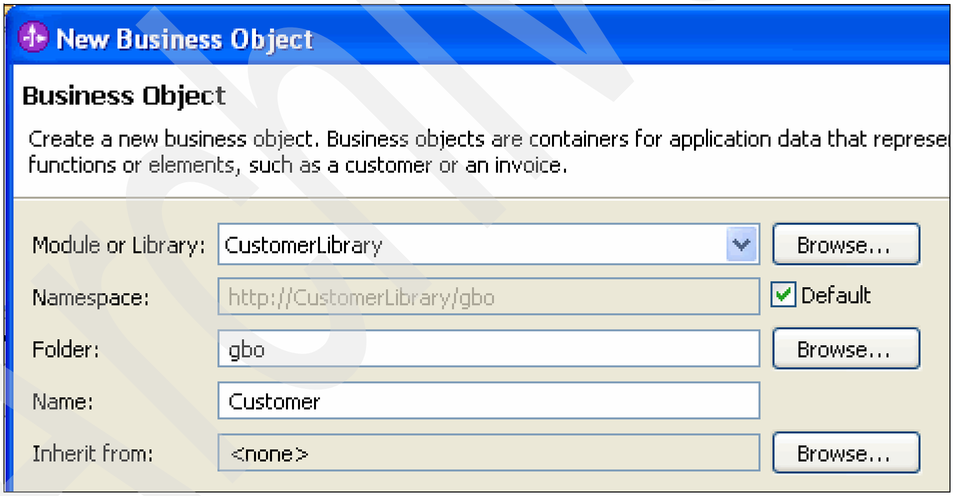
Business graphs are related to business objects. A business graph definition is the wrapper added around a simple business object or a hierarchy of business objects to provide additional capabilities, such as carrying change summary and event summary information related to the business objects in the business graph.

3.4.3 Creating a business object

Business objects are created in the Data Types folder in a library, module, or mediation module. To share a business object among modules, create it in a library.

To create a new Customer business object:

1. In the Business Integration view, right-click the Data Types folder and select **New** → **Business Object** from the context menu.
2. Enter the name for the new business object and, optionally, a folder name (Figure 3-22). If the folder does not exist, it is created.



New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represe
functions or elements, such as a customer or an invoice.

Module or Library: CustomerLibrary

Namespace: http://CustomerLibrary/gbo ☑ Default

Folder: gbo

Name: Customer

Inherit from: <none>

Figure 3-22 New Business Object wizard

3. Click **Finish**.

The new business object (Customer in this example) is created in the location that you specified and opens in the business object editor. See Figure 3-23.

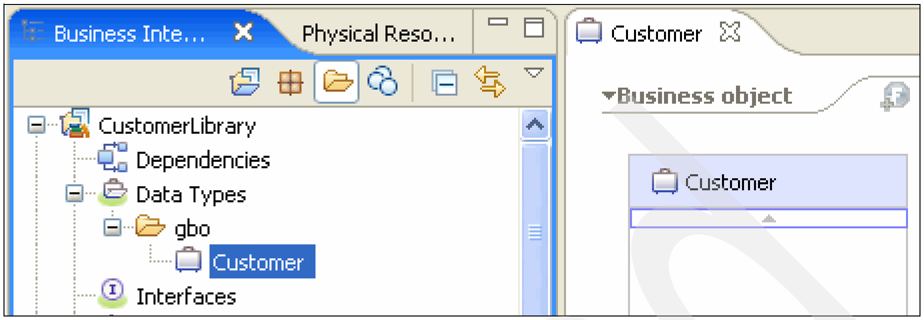


Figure 3-23 New business object

3.4.4 Adding fields to a business object

With the business object editor, you can build and edit business objects and business graphs (Figure 3-24). Use this editor to add, delete, and reorder fields and to change the type of an field. To add a simple field to the business object:

1. Click the Add a field to a business object icon.
2. Replace the default name, field1, with the name of the field (that is, CustomerId), as shown in Figure 3-24.

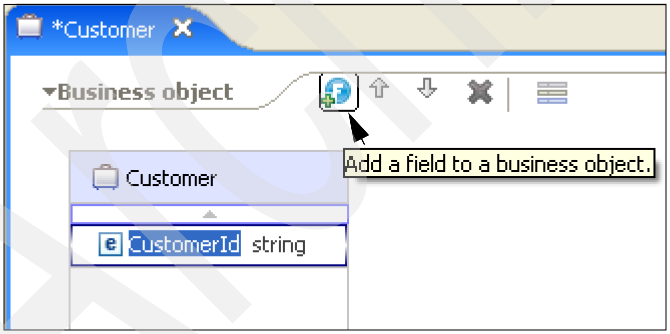


Figure 3-24 Business Object editor

3. You can take the default type, String, or select a new type by clicking String and selecting a type, existing business object, or **New** (to create a new object) from the drop-down menu as shown in Figure 3-25.

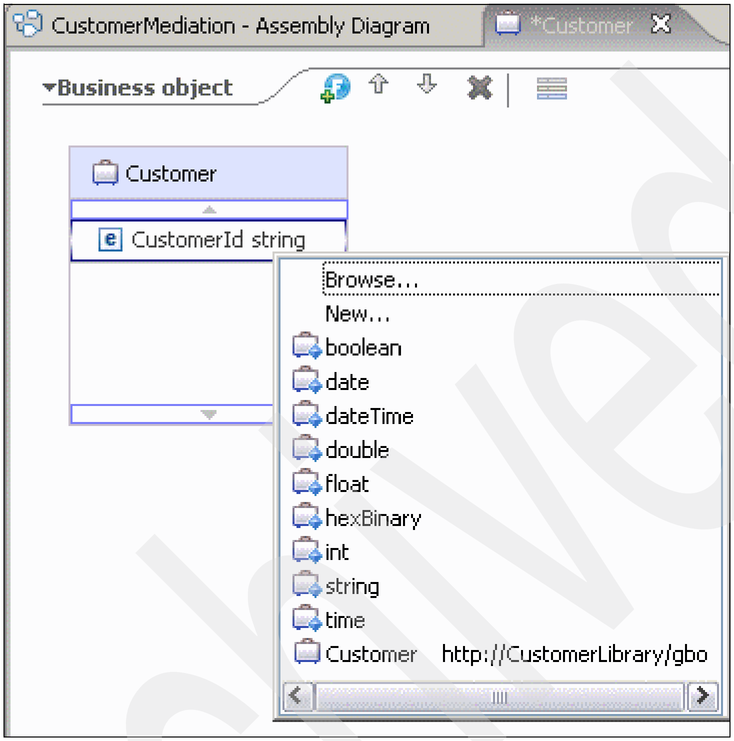


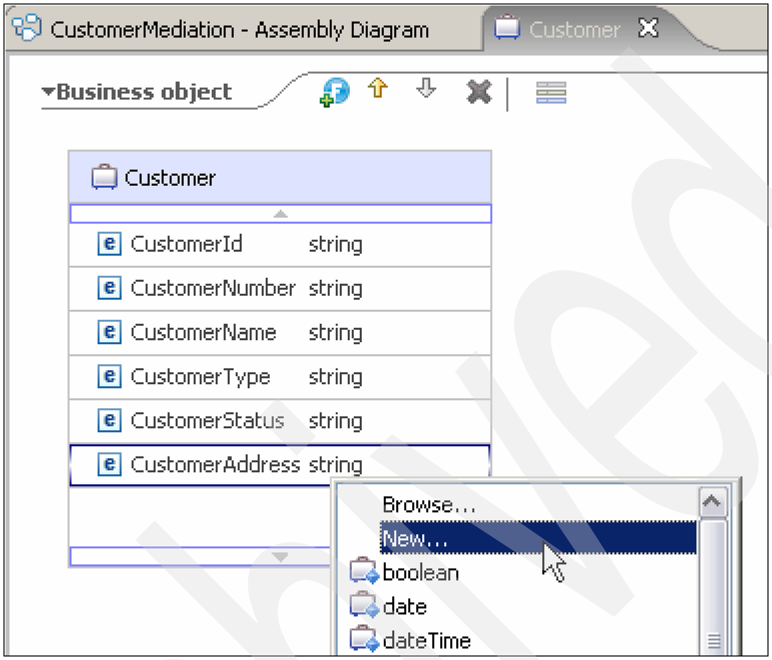
Figure 3-25 Select the field type

4. Repeat the previous step to add each field. If you create the fields in the wrong order accidentally, you can use the up and down arrows in the toolbar to change the order.

A *field type* can be a simple type or a business object type. You can select from the list of existing business objects in the drop-down menu, or you can create a new one. In the next example, we create a new business object called *Address* as the data type for the CustomerAddress field in the Customer business object. This example also illustrates the use of an array for a field.

To create a new business object type for CustomerAddress, follow these steps:

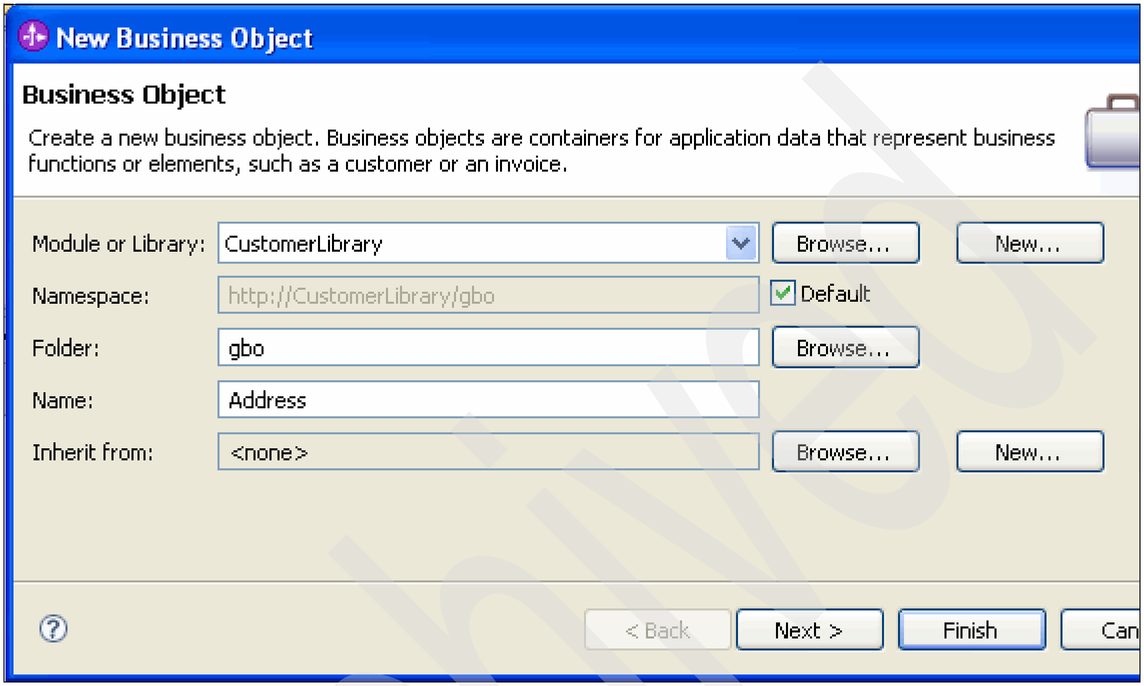
1. Click **string** and click **New** to start the New Business Object wizard (Figure 3-26).



Customer	
CustomerId	string
CustomerNumber	string
CustomerName	string
CustomerType	string
CustomerStatus	string
CustomerAddress	string

Figure 3-26 Building a complex business object

2. Enter the name and location values for the new business object and click **Finish** (Figure 3-27). In this example, the new business object, Address, is placed in a library.



New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or Library: CustomerLibrary

Namespace: http://CustomerLibrary/gbo ☑ Default

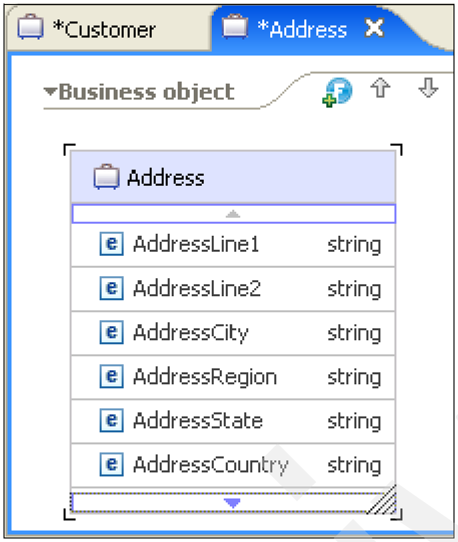
Folder: gbo

Name: Address

Inherit from: <none>

Figure 3-27 Creating a super-set business object

3. The new business object opens, and you can populate the fields as shown in Figure 3-28.

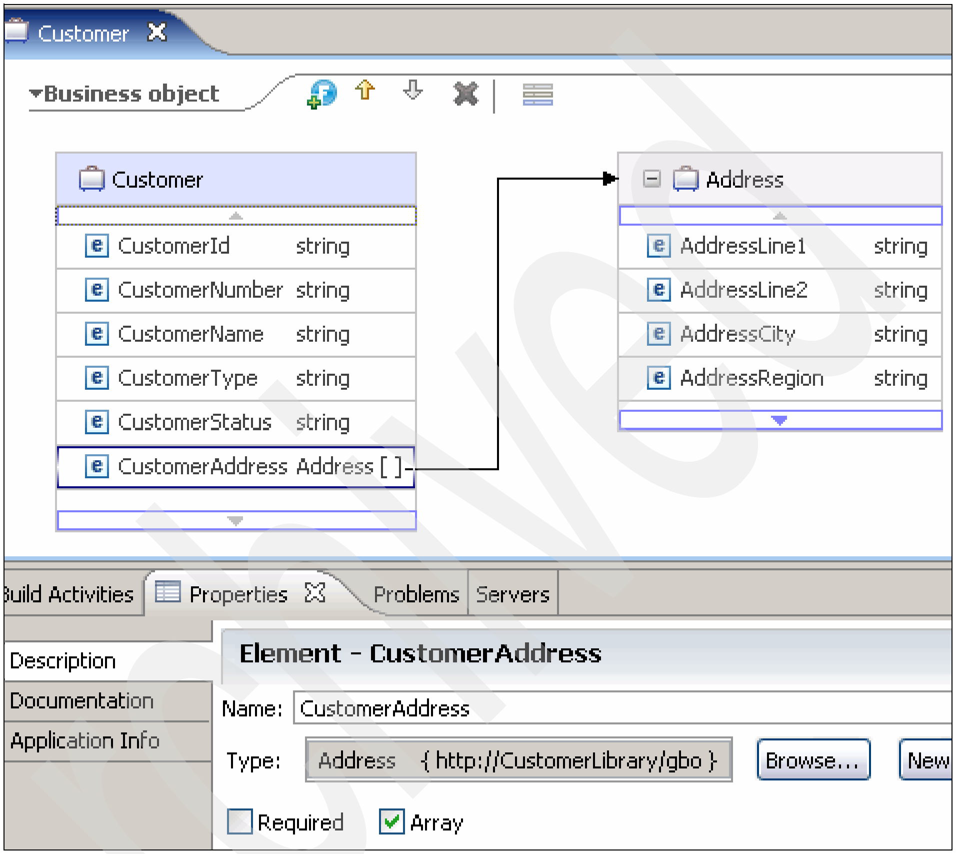


AddressLine1	string
AddressLine2	string
AddressCity	string
AddressRegion	string
AddressState	string
AddressCountry	string

Figure 3-28 Derived business object

4. Save and close the new business object to go back to the original business object, Customer. The new Address business object shows as the type for the CustomerAddress field.

5. Next, define CustomerAddress as an array of Address objects. Select **CustomerAddress** and in the Properties view (lower right pane), and then select the Description tab. Select **Array**. Note the brackets ([ ]) display in the field data type to indicate it is an array. See Figure 3-29.



Customer	
CustomerId	string
CustomerNumber	string
CustomerName	string
CustomerType	string
CustomerStatus	string
CustomerAddress	Address [ ]

Address	
AddressLine1	string
AddressLine2	string
AddressCity	string
AddressRegion	string

Element - CustomerAddress

Name: CustomerAddress

Type: Address { http://CustomerLibrary/gbo }

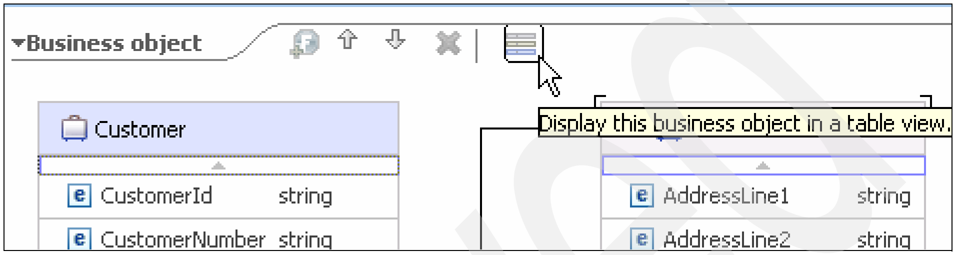
□ Required ☑ Array

Figure 3-29 List of addresses as a complex element type

6. Save the business object.

You can view and edit business objects in table format to make it easier to examine the fields and their properties in a single view. To open the Customer business object definition as a table:

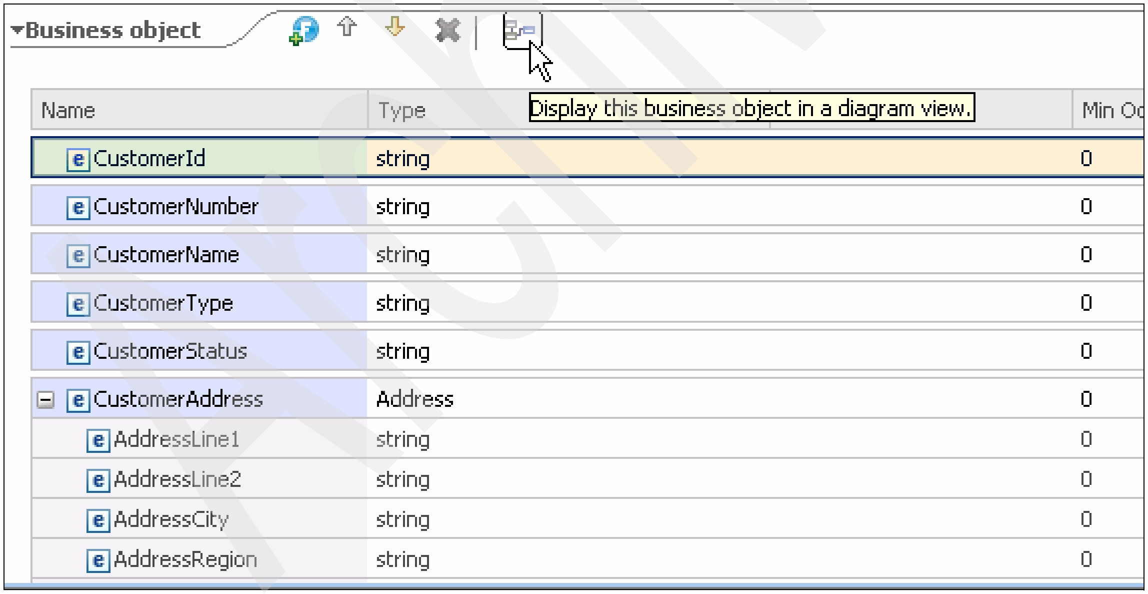
1. Double-click the Customer business object in the Business Integration view to open it.
2. In the business object editor, click the Display this business object in a table view icon as shown in Figure 3-30.



Display this business object in a table view.

Figure 3-30 Display business object as a table

Figure 3-31 shows the table view. You can toggle back and forth between these views.



Display this business object in a diagram view.

Name	Type	Min Oc
CustomerId	string	0
CustomerNumber	string	0
CustomerName	string	0
CustomerType	string	0
CustomerStatus	string	0
⊟ CustomerAddress	Address	0
AddressLine1	string	0
AddressLine2	string	0
AddressCity	string	0
AddressRegion	string	0

Figure 3-31 Table view of a business object

3.4.5 Creating a business graph

You can also create a business graph of your business object. In the Business Integration view, right-click the business object and select **Create a Business Graph** from the context menu.

The new business graph definition opens in the editor as shown in Figure 3-32.

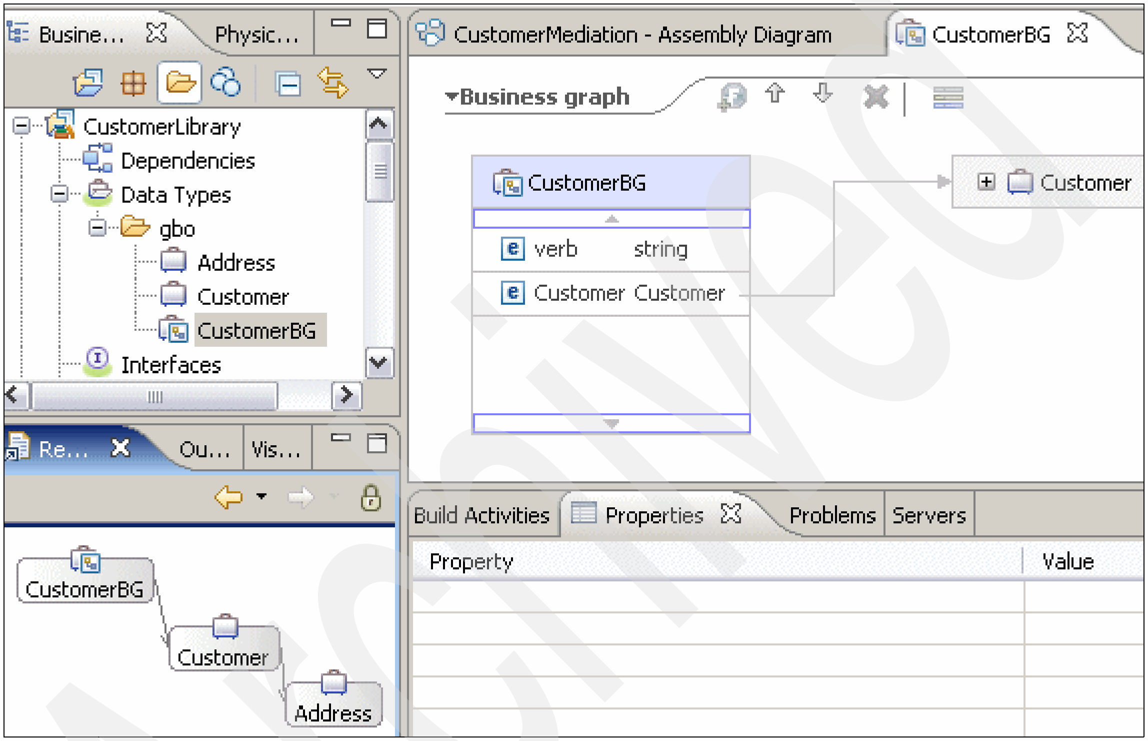
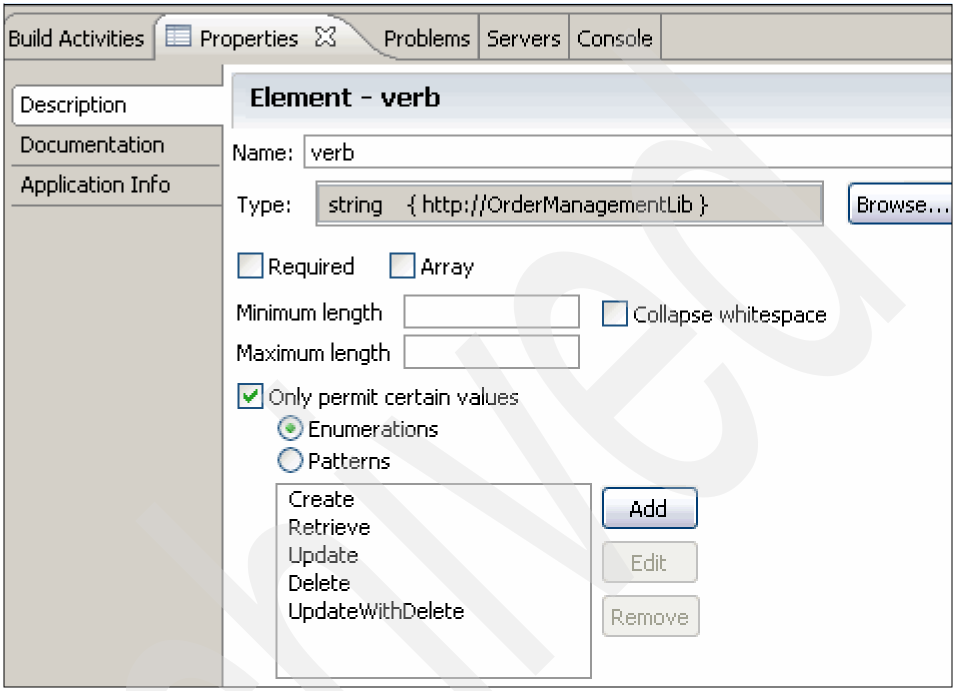


Figure 3-32 Business graph created from a business object

The business graph provides a verb field. Figure 3-33 shows the initial properties of this field. The verbs that are allowed for this business graph are enumerated in the properties. The back-end system that receives this container uses the verb to determine what to do with the container.



Element - verb

Name: verb

Type: string { http://OrderManagementLib }

☐ Required ☐ Array

Minimum length

Maximum length

☐ Collapse whitespace

☑ Only permit certain values

● Enumerations

○ Patterns

Create
Retrieve
Update
Delete
UpdateWithDelete

Figure 3-33 Business graph verb properties

3.5 Interfaces

Interfaces link the components in a module. The inputs and outputs of each component, specified by the interface, determine which data can be passed from one component to another. An interface is created independently from the implementation of the component. An interface can also be created for a component that has no implementation; that is, the implementation is done later.

The following approaches can be taken in developing an interface. Which approach you choose depends on your circumstance.

- *Top-down development*: Use this methodology when there is no existing interface to work with. You launch the interface editor, provide a name for the interface, and add one or more operations to it. Inputs, outputs, and faults are added to each operation.

- *Bottom-up development*: Use this approach when you already have an interface that was created as a WSDL file. In this case, you import the interface into a module and then start the interface editor. The editor displays the interface as operations, inputs, outputs, and faults. You can then use the interface editor to modify operations, inputs, outputs, and faults.
- *Meet-in-the-middle development*: Use this approach when you have an interface that either exactly matches the interface needs for a component or is close to what you need. Drag the interface onto the component, and it either is a perfect fit, or you might need to make a few modifications to it with the interface editor to make the interface work. Meet-in-the-middle development saves you development time.

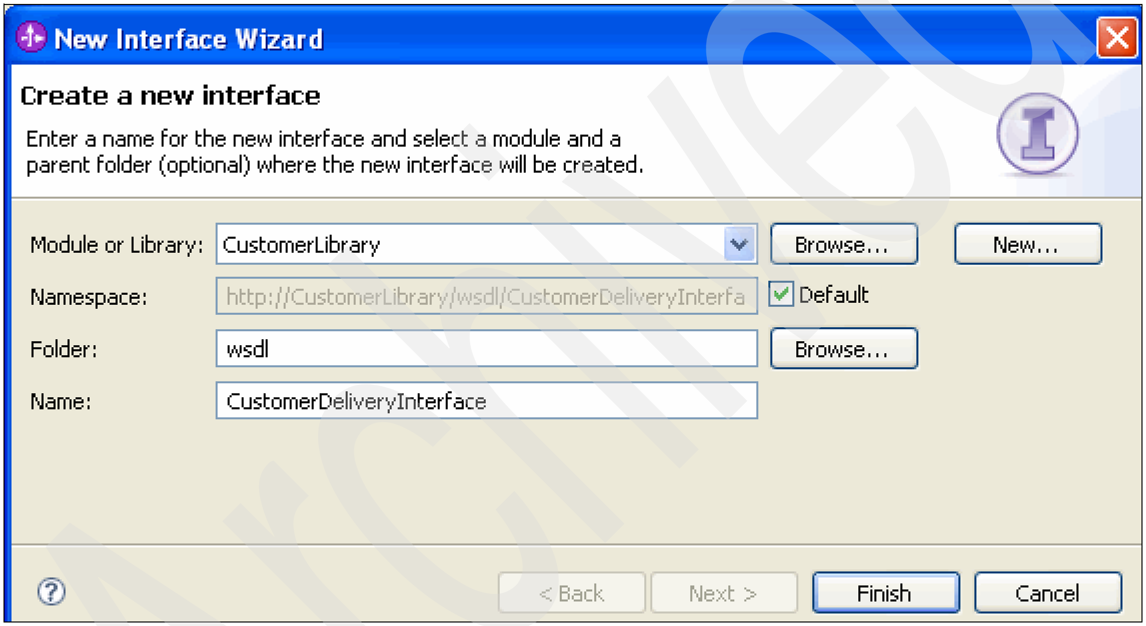
An interface map enables you to connect components, despite disparities in their respective interfaces. For example, when two components that have interfaces with different method names need to be wired together, an interface map enables these two components to interact. The interface acts as a mediator between the two components, mapping the operations and parameters of both the source and target interfaces.

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

3.5.1 Creating a new interface

This section describes the process of creating a new interface. The example that we use here creates a new interface called *CustomerDeliveryInterface* with a one-way operation that accepts a CustomerBG business graph input. To create a new interface:

1. In the Business Integration view, expand the library or project folder where you will create the interface. Right-click **Interfaces** and select **New** → **Interface** from the context menu.
2. Enter the name for the interface and the location in the first panel of the New Interface wizard. Click **Finish**. See Figure 3-34.



New Interface Wizard

Create a new interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module or Library: CustomerLibrary

Namespace: http://CustomerLibrary/wsdl/CustomerDeliveryInterfa ☑ Default

Folder: wsdl

Name: CustomerDeliveryInterface

Figure 3-34 New interface wizard

The new interface is added to the folder in the Business Integration view, and the interface opens for editing, as shown in Figure 3-35.

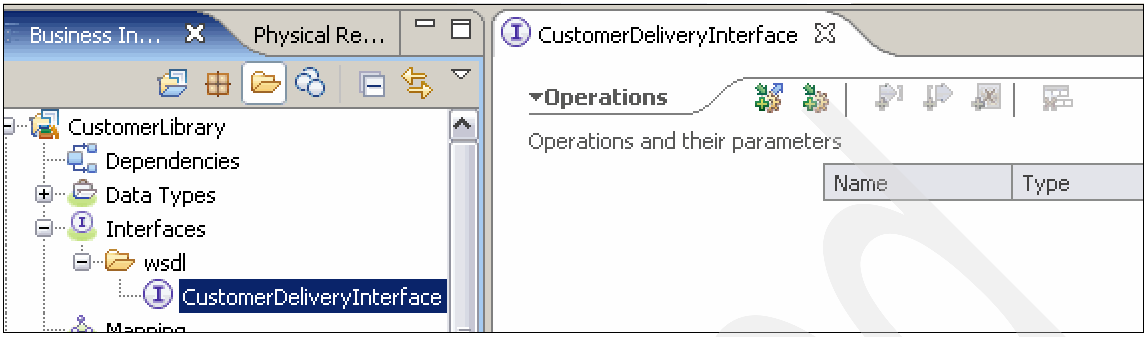


Figure 3-35 CustomerDeliveryInterface

3.5.2 Adding operations

You can use the interface editor to build WSDL port type interfaces that are used to define some SCA components (Figure 3-36). You use this editor to add and to remove operations and specify an operation's inputs and outputs.

Two operation types are supported:

- *Request-response operations*: A request is sent, and a response is returned to the interface. The operation has an input and an output that is defined by default. You can add additional input and outputs for the operation, and you can add faults.
- *One-way operations*: A request is sent, and no response is needed. The operation has only an input defined. You can add additional inputs.

To add a new operation and input parameter to the interface:

1. In the interface editor, click the Add One Way Operation icon as shown in Figure 3-36.

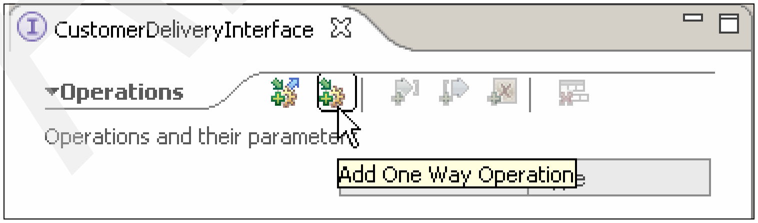
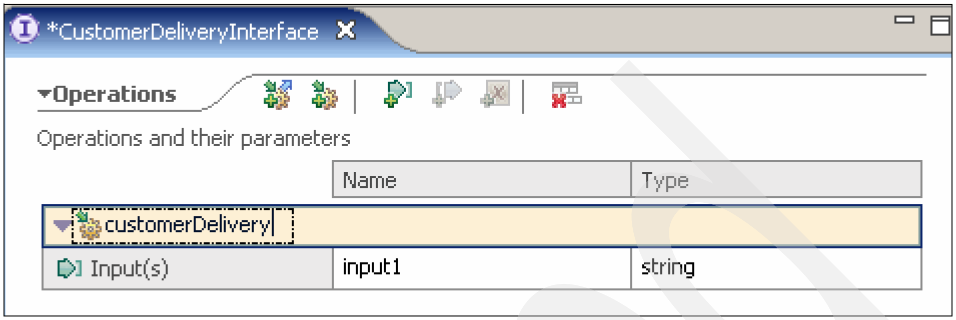


Figure 3-36 Add a one-way operation

2. Change the operation name from operation1 to a meaningful operation name (`customerDelivery` in this case) as shown in Figure 3-37. You can also change the name in the Properties view.



*CustomerDeliveryInterface

Operations

Operations and their parameters

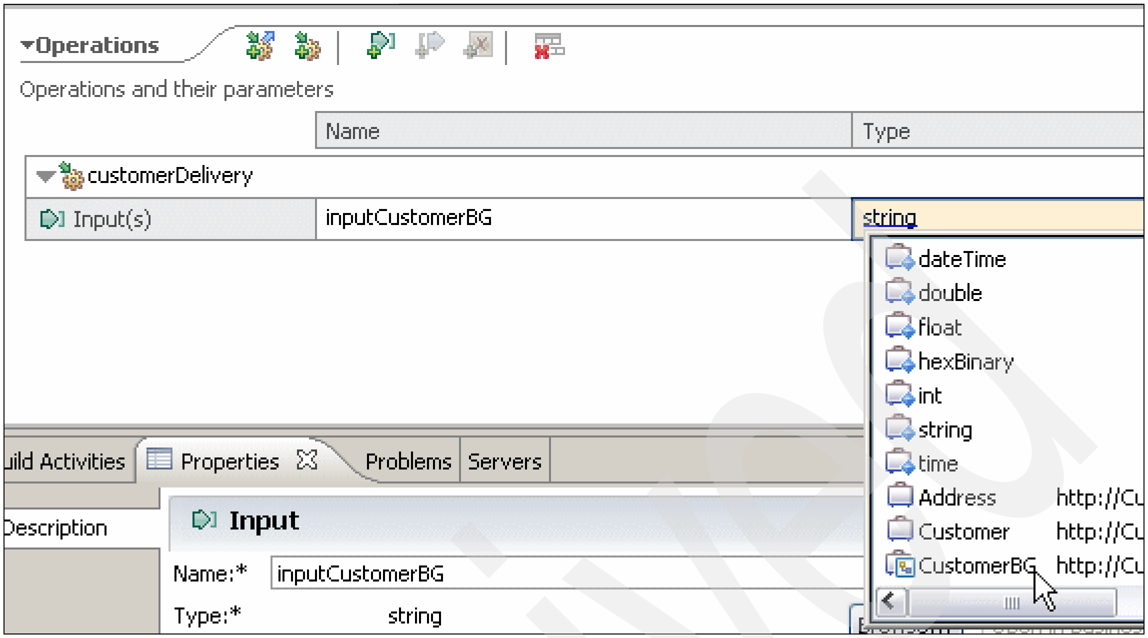
	Name	Type
customerDelivery		
Input(s)	input1	string

Figure 3-37 Interface operations editor

Depending on the type of operation, you have additional active icons to help add input , output , and fault fields and to delete a selected field or operation .

3. In the Input(s) row, click the default input of **input1**. Type over this name with a meaningful name (`inputCustomerBG`).You can also change the name of this field in the Properties view.

4. Click the string value in the Type field. Then, scroll down and select the business object or, as in this example, the business graph (**CustomerBG**). See Figure 3-38 on page 68.



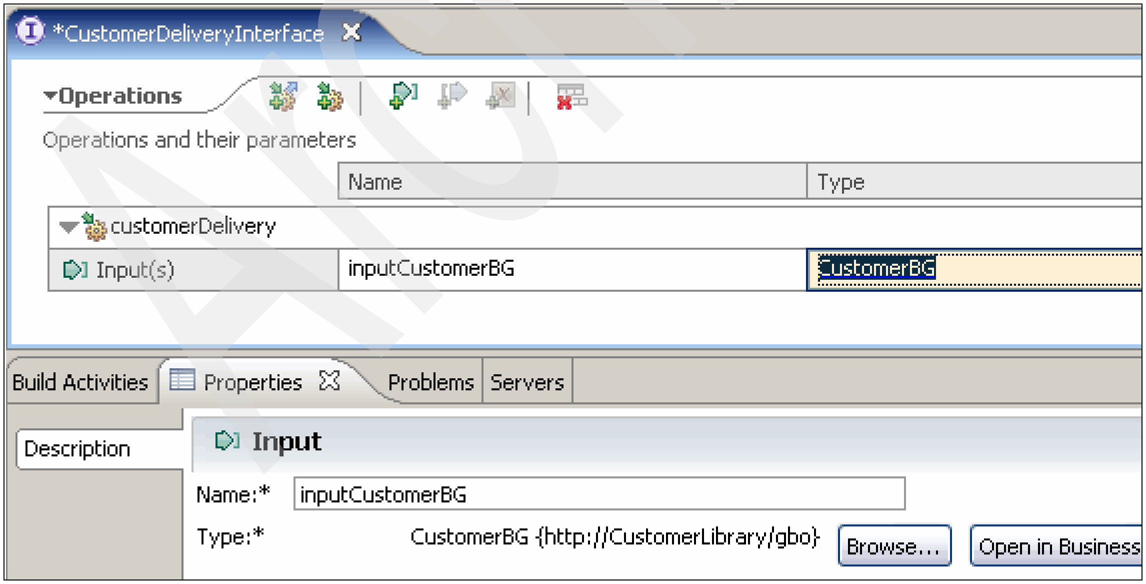
inputCustomerBG string

Name:* inputCustomerBG

Type:* string

Figure 3-38 Interface Operations wizard

5. With the new input field selected, you can view its settings in the Properties view as shown in Figure 3-39.



inputCustomerBG CustomerBG

Name:* inputCustomerBG

Type:* CustomerBG {http://CustomerLibrary/gbo}

Figure 3-39 Interface description

6. Save and close the new interface definition.

3.6 Module assembly

A module provides business services that are modeled as SCA components that are wired together in an assembly diagram. The WebSphere Integration Developer’s assembly editor lets you build applications by assembling the SCA components that make up the module.

The assembly diagram is created when you create the module. Business integration modules initially have an empty assembly diagram. Mediation modules initially have one component that is implemented as a mediation flow.

3.6.1 Module assembly diagrams

WebSphere Integration Developer provides a number of methods that are designed for convenience during the performance of different tasks for populating the assembly diagram. You will see some of these methods in the various examples throughout this book. To simplify the process for illustration purposes, consider the following flow for assembling the components in an assembly diagram:

1. Create a new module. This process also creates the assembly diagram.
2. Open the assembly diagram in the assembly editor by double-clicking **Assembly Diagram** for the module in the Business Integration view (Figure 3-40). Initially the diagram is empty.

   You can drop existing components to the assembly diagram, or you can use the palette to drop a new component on the diagram. Double-clicking a new component starts a wizard to help you build the implementation for the component.

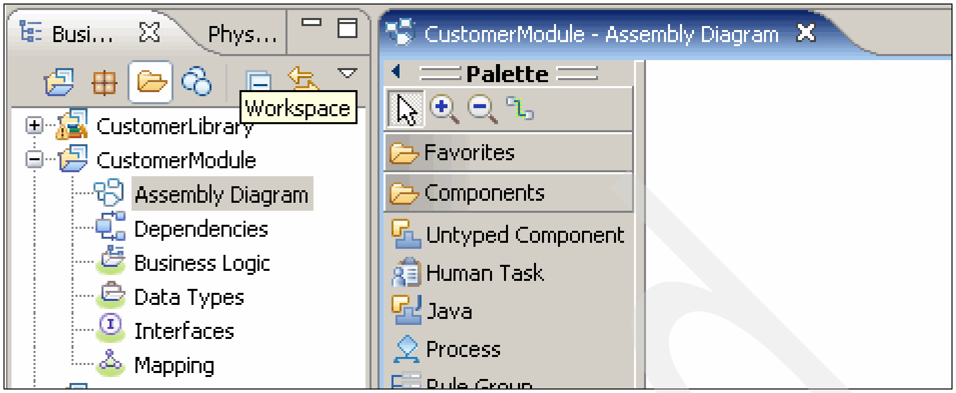


Figure 3-40 Initial assembly diagram

3. Add a new business process component. In the Business Integration view, right-click **Business Logic** and select **New** → **Business Process**. Complete the wizard to create the new business process component.

   During the process, you create a new interface or select an existing interface for this component. You also identify the operation in the interface that starts the process.

   At this point, the business process opens in a separate editor. See Figure 3-41.

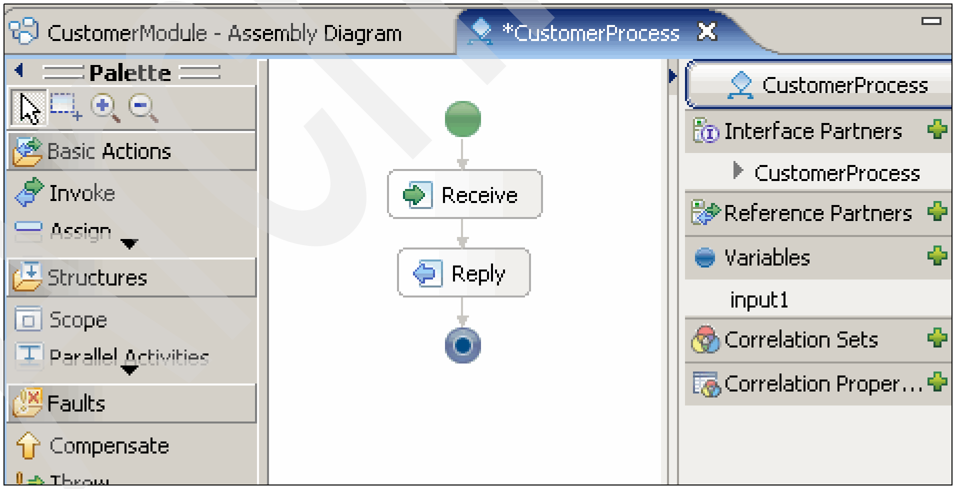


Figure 3-41 Business process logic

4. Close the business process logic.
5. Return to the assembly diagram and drag the new business process from the Business Integration view to the assembly diagram as shown in Figure 3-42.

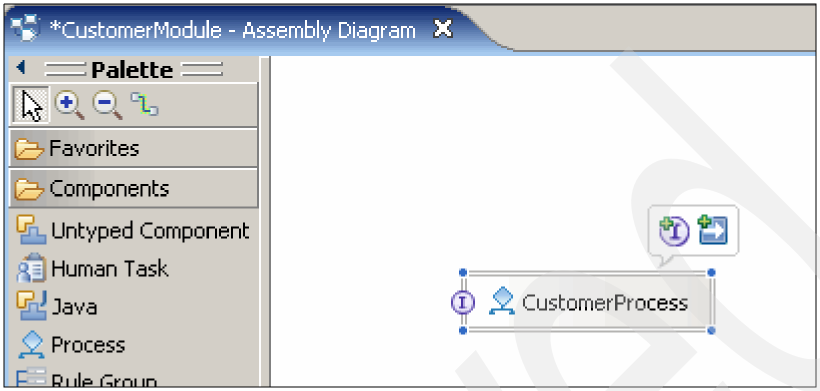


Figure 3-42 Add the business process logic to the assembly diagram

6. Add import components to the assembly diagram for services the process will call. You can add these to the diagram by dragging an existing export from this or another module, an interface from this module or a library, or a Web service port to the diagram. Alternatively, you can use the palette to create a new import component to the diagram.

   Dragging an existing artifact to the diagram to use as an import allows you to select the component type and binding. For example, when you drag an interface to the assembly diagram, you will have the options shown in Figure 3-43.

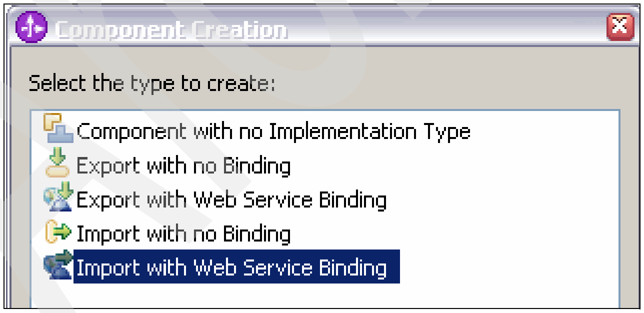

Component Creation
Select the type to create:
Component with no Implementation Type
Export with no Binding
Export with Web Service Binding
Import with no Binding
Import with Web Service Binding

Figure 3-43 Creating imports and exports

7. Wire the business process to the interface for the import component as shown in Figure 3-44.

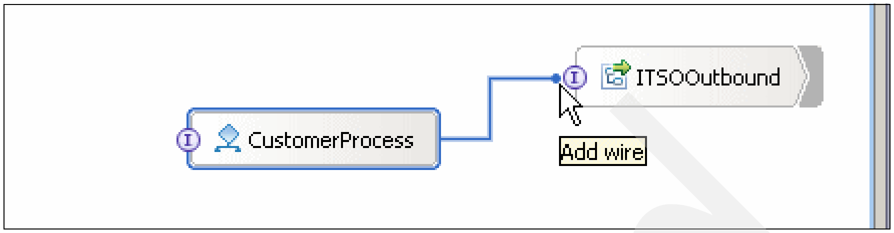


Figure 3-44 Wire the business process to the import component

This creates the reference as shown in Figure 3-45.

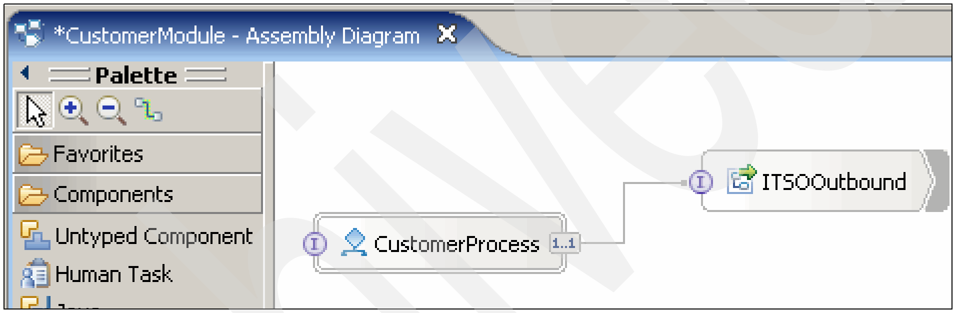


Figure 3-45 Wire the import component

8. Import components must have matching reference partners in the business logic implementation. If you add an import to the assembly diagram and no reference partner exists in the implementation, you see an error.

   Because we have not added any logic to the business process, the simplest way to fix the error is to regenerate the implementation for the business process component. Select the business process in the assembly diagram, right-click, and select **Regenerate Implementation**. The problem with this method is that you lose any work that you have done to the business process logic.

9. Add additional components as needed (for example interface maps, human tasks, adapters, and rule groups). Wire the business process to the interfaces for these components, creating references in the process. See Figure 3-46.

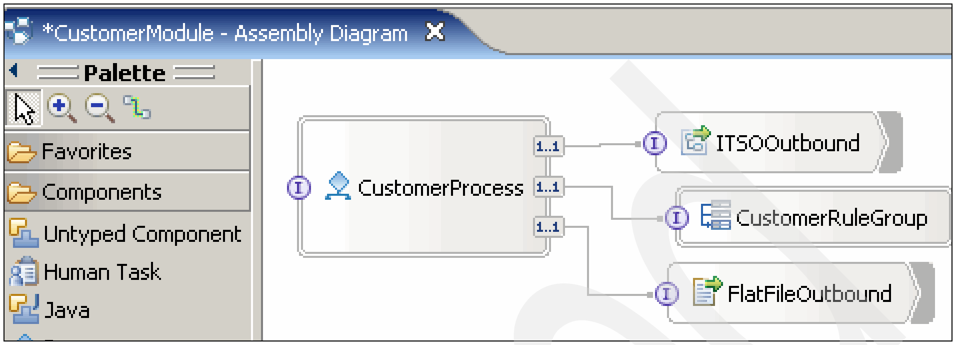


Figure 3-46 Assembly diagram

10. Generate an export for the business process. Right-click the business process, and select **Generate Export** → ***binding type***.
11. Generate bindings for any import and export components that do not already have them.

3.6.2 Mediation module assembly diagrams

The process for populating a mediation module assembly diagram is very similar to the process for a module, though somewhat simplified because you do not have as many choices for implementation.

A typical flow for assembling the components in an assembly diagram for a mediation module is:

1. Open the assembly diagram in the assembly editor by double-clicking the Assembly Diagram for the module in the Business Integration view. The assembly diagram has a mediation flow component.
2. Add import components to the assembly diagram for services that the mediation will call. You can add these to the diagram by dragging an existing export, Web service port, or interface to the diagram. Alternatively, use the Palette to create a new Import.
3. Wire references to the interfaces.
4. Generate an export component for the business process. Right-click the mediation flow in the assembly diagram, and select **Generate Export** → ***binding type***. You can elect to create the binding when you create the export, or later.

5. Generate bindings for import and export components that do not already have one generated.

3.6.3 Generating a binding

Each export and import component must have a binding generated for it before deployment. The binding determines the type of transport used for the message and the characteristics of the transport. To generate a binding:

1. Select the export or import component on the assembly diagram. Right-click and select **Generate Binding** → ***binding type***.

   Figure 3-47 shows the binding options for an export component.

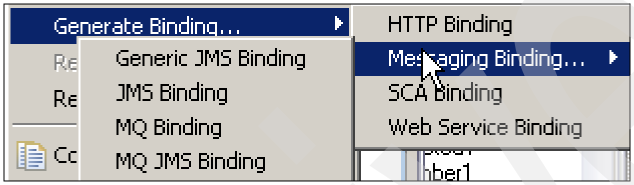


Figure 3-47 Export bindings

   Figure 3-47 shows the binding options for an import component.

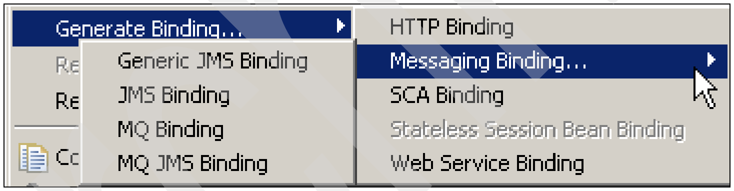


Figure 3-48 Import bindings

2. Use the Properties view to define the required transport settings.

JMS binding example

This example illustrates how to define a JMS binding for an export component, which allows a JMS client to call the process. In this example, we add an export component with a JMS binding to the business integration module shown in Figure 3-49. In this case, we have not yet created the export component. We will define the export component and binding at the same time.

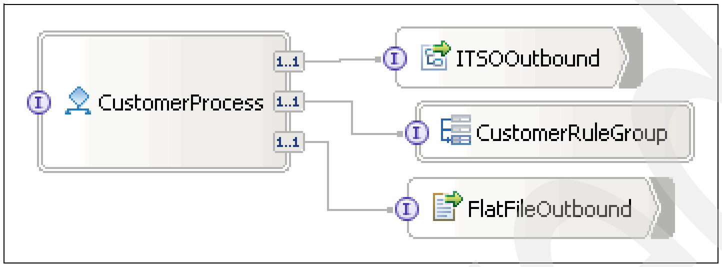


Figure 3-49 CustomerProcess assembly diagram

To define a JMS binding, follow these steps:

1. Right-click the business process and select **Generate Export** → **Messaging Binding** → **JMS Binding**.

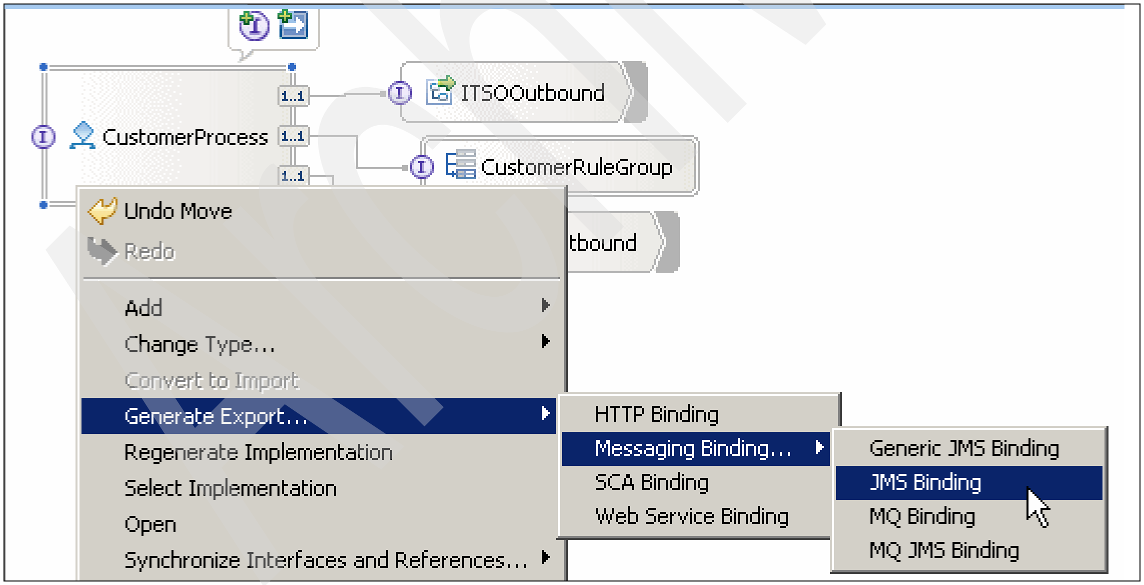
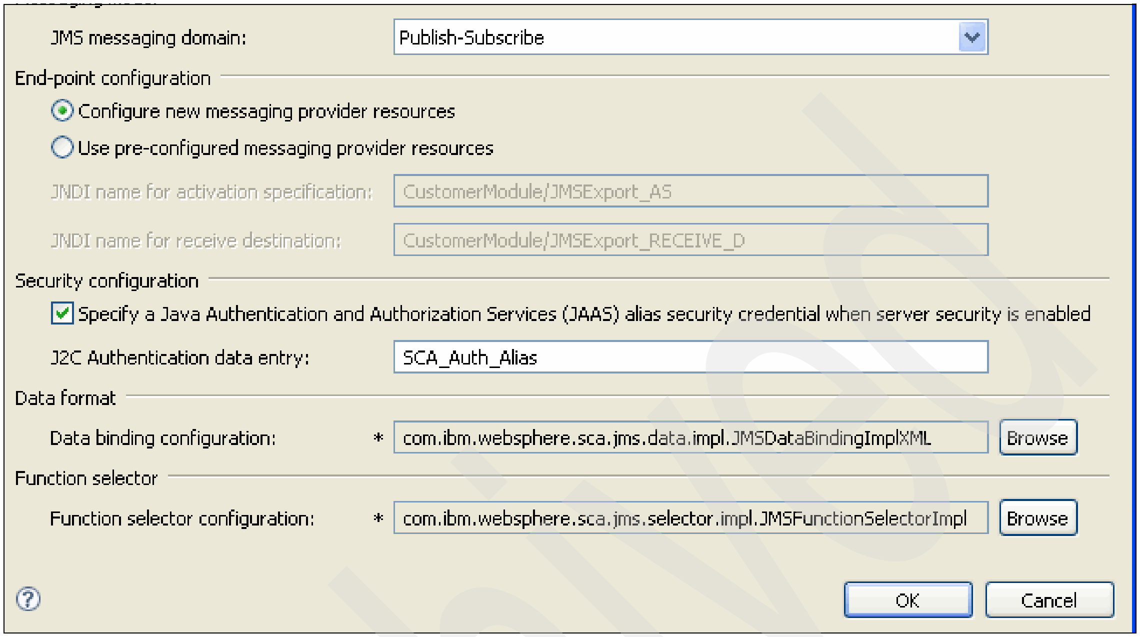


Figure 3-50 Generate an export component with a JMS binding

2. Complete the information that is required for the JMS binding as shown in Figure 3-51.



JMS messaging domain: Publish-Subscribe

End-point configuration

● Configure new messaging provider resources

○ Use pre-configured messaging provider resources

JNDI name for activation specification: CustomerModule/JMSExport_AS

JNDI name for receive destination: CustomerModule/JMSExport_RECEIVE_D

Security configuration

☑ Specify a Java Authentication and Authorization Services (JAAS) alias security credential when server security is enabled

J2C Authentication data entry: SCA_Auth_Alias

Data format

Data binding configuration: * com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML

Function selector

Function selector configuration: * com.ibm.websphere.sca.jms.selector.impl.JMSFunctionSelectorImpl

Figure 3-51 JMS binding options

The options are as follows:

a. The domain options depend on the type of operations that are defined on the interface. Interfaces with only one-way operations can support both publish/subscribe and point-to-point domains. Interfaces with request-response operations only support point-to-point.

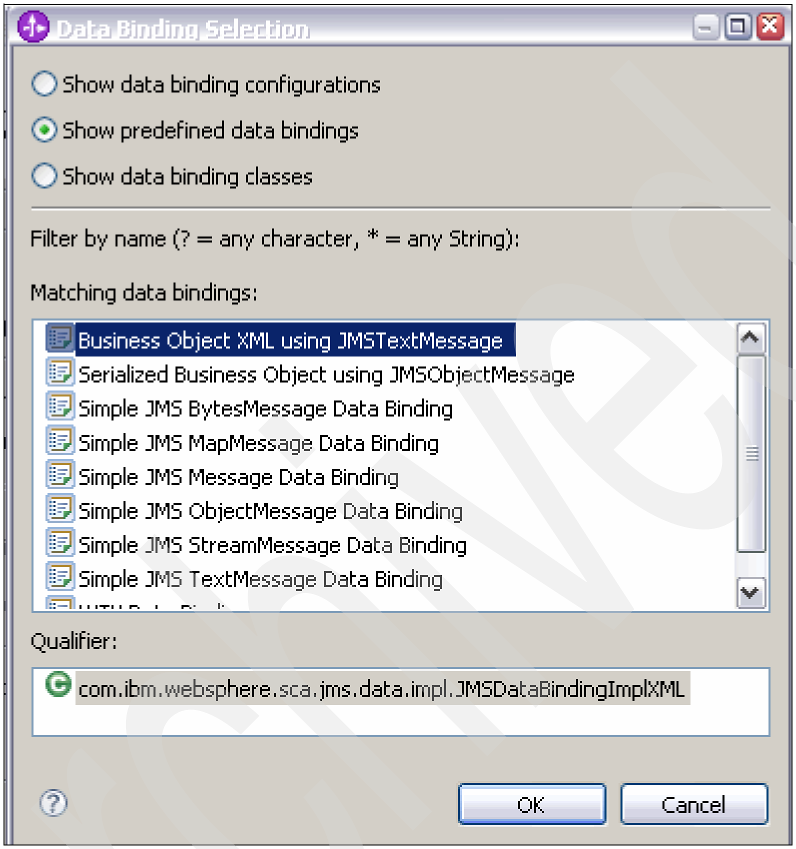
b. JMS bindings require a JNDI namespace entry that defines the messaging resources (activation specification, queues, and topics).

   You have an option to create the connections and destinations that are required for the JMS binding when the component is installed on your server, or you can specify the JNDI name of existing resources on the server.

c. JMS resources can require security authorization. You can specify the J2C authentication data entry to use for this purpose at run time.

d. A data binding provides a mapping between the format that is used by an external JMS message and the SDO representation that is used by the module.

Click **Browse** to open the Data Binding Selection window. Select **Show predefined data bindings**, and select the data binding. In this example, we serialize the data as XML. Then, click **OK**.



○ Show data binding configurations

● Show predefined data bindings

○ Show data binding classes

Filter by name (? = any character, * = any String):

Matching data bindings:

- Business Object XML using JMSTextMessage
- Serialized Business Object using JMSObjectMessage
- Simple JMS BytesMessage Data Binding
- Simple JMS MapMessage Data Binding
- Simple JMS Message Data Binding
- Simple JMS ObjectMessage Data Binding
- Simple JMS StreamMessage Data Binding
- Simple JMS TextMessage Data Binding

Qualifier:

com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML

Figure 3-52 Select the data binding

e. The function selector is used to determine which operation corresponds to an incoming message.

Click **Browse** to select a function. Select **Show predefined function selectors**, and select **JMS default Function Selector**. Click **OK**.

3. Click **OK** again to create the export component and binding. The export component is created with the same interface as the business process and are wired to the process. See Figure 3-53.

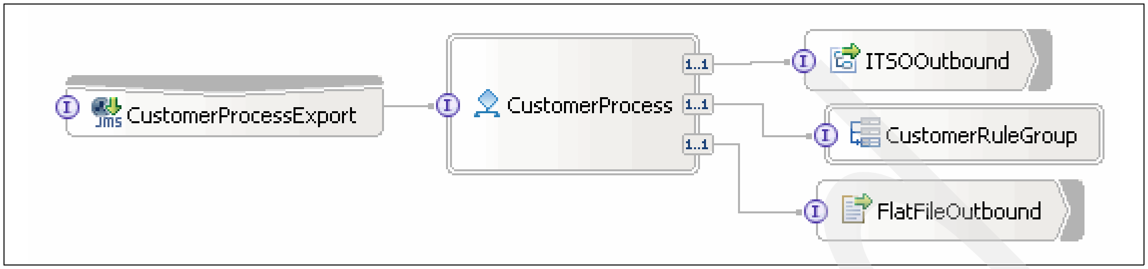


Figure 3-53 Full module assembly in the assembly diagram

4. You can view and modify the values that you specified for the bindings and additional settings in the Binding tab in the Properties view.

3.6.4 Using adapters

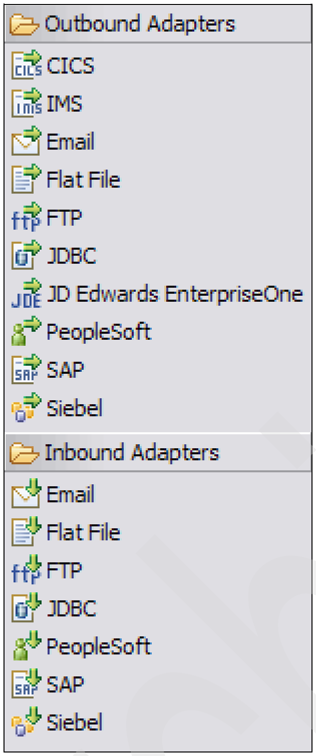
WebSphere Adapters provide a mechanism that allows for integration of existing EIS infrastructure with process integration applications. Adapters can be plugged into the process server to provide connectivity between the EIS, the process server, and the business integration application.

The external service discovery wizard discovers applications and data on an EIS and lets you generate services from the discovered applications and data. During this discovery, a resource adapter is imported and used to create a service to access the information from the external system. The generated artifacts include the interfaces and business objects, the import or export component, and an EIS binding to provide communication with the service on the EIS system.

To create an export component that uses an adapter, select an adapter from the Inbound Adapters folder on the assembly editor palette and drag it onto the canvas. Complete the fields in the wizard to generate the export.

To create an import component that uses an adapter, select an adapter from the Outbound Adapters folder on the assembly editor palette and drag it onto the canvas. Complete the fields in the wizard to generate the import.

Figure 3-54 shows the list of both application adapters (for example, PeopleSoft, SAP, and so forth) and technology adapters (for example, Email, JDBC, and so forth) available on the palette of assembly editor.


Outbound Adapters
- CICS
- IMS
- Email
- Flat File
- FTP
- JDBC
- JD Edwards EnterpriseOne
- PeopleSoft
- SAP
- Siebel

Inbound Adapters
- Email
- Flat File
- FTP
- JDBC
- PeopleSoft
- SAP
- Siebel

Figure 3-54 List of Outbound and Inbound Adapters

For more detailed discussion about adapters, refer to Chapter 7, “Using adapters” on page 471.

3.7 Business object maps

The business entities that travel through WebSphere Process Server are in the form of service data and business objects. A business object map provides the means to map each part of an input message to the corresponding part of an output message.

Note: Unlike interface maps, business object maps can also be used in WebSphere Enterprise Service Bus.

WebSphere Process Server data mapping support provides the following capabilities:

- Transforming data values from one or more fields in a source business object to one or more fields in a destination business object.
- Establishing and maintaining relationships between data entities that are equivalent but are represented differently and cannot be directly transformed.
- Enabling access to external mapping resources, such as databases for performing queries.

3.7.1 Data transformation types

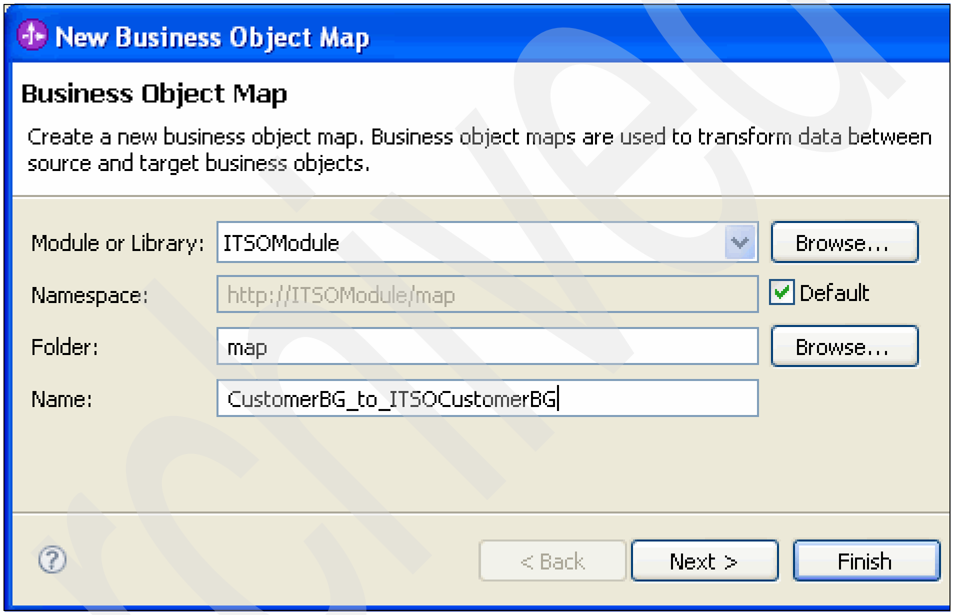
The data transformation types that you can use in business object maps are:

- **Move:** Assigns the value in the source to the target.
- **Extract:** The source value must be a string, which extracts a portion of the string and assigns it to the target. This is similar to the String.substring() method in Java.
- **Join:** Combines the values of two or more sources into one, and assigns it to the target. The target of a Join transform must be a string.
- **Submap:** The source and target must be business objects (that is, they must be complex types). The input and output of the specified business object map must be of the same type as the sources and targets of the transform.
- **Custom**: Specifies custom logic for mapping the inputs and outputs by using Java code.
- **Assign:** Sets the constant value to the output.
- **Relationship:** Performs relationship management. The source and target of a Relationship transform must also be complex types.
- **Relationship Lookup:** Performs relationship management between static cross-referencing data. The source and target of a Relationship Lookup transform must be simple types.
- **Custom Assign:** Similar to Custom, except that it does not take any input. It is also similar to Assign, except that you can use more logic in assigning the values using Java (using the text or visual editors).
- **Custom Callout:** Similar to Custom, except that it does not take any output. It can be useful for initialization before executing any transforms.

3.7.2 Creating a business object map

This example illustrates how to create a business object map. In this particular example, the map transforms a GBO (Customer) to an ASBO (ITSOCustomer). Follow these steps:

1. In the Business Integration view, expand the module or library. Right-click the **Mapping** folder and select **New** → **Business Object Map** from the context menu.
2. In the New Business Object Map dialog box, enter the name and location for the map as shown in Figure 3-55. Click **Next.**



New Business Object Map

Business Object Map

Create a new business object map. Business object maps are used to transform data between source and target business objects.

Module or Library: ITSOModule

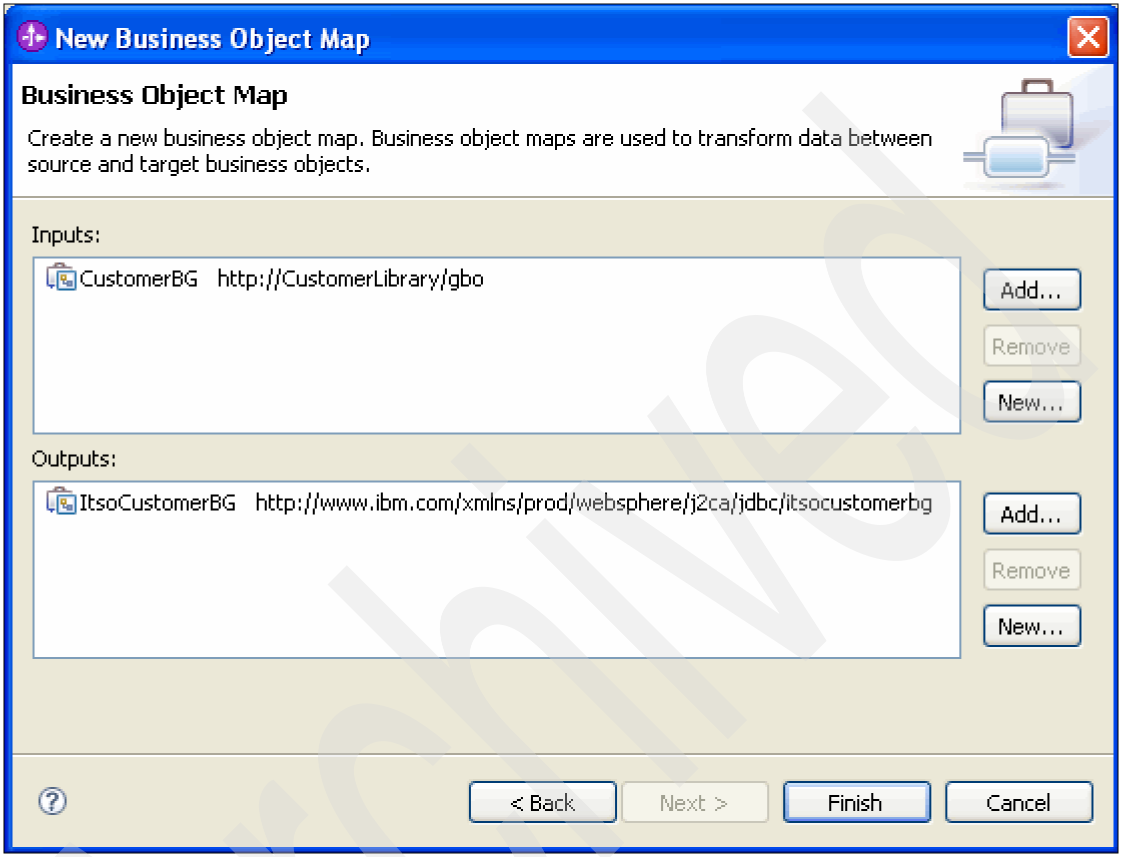
Namespace: http://ITSOModule/map ☑ Default

Folder: map

Name: CustomerBG_to_ITSOCustomerBG

Figure 3-55 New Business Object Map wizard

3. Click **Add** to select the input (source) and output (target) business objects click **Finish**. In Figure 3-56, two business graphs are selected.



Business Object Map

Create a new business object map. Business object maps are used to transform data between source and target business objects.

Inputs:

CustomerBG http://CustomerLibrary/gbo

Outputs:

ItsoCustomerBG http://www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/itsocustomerbg

Figure 3-56 New business object map wizard

The new map is created and opens in the editor.

Tip: The mapping editor requires a wide view to show all the business object fields. You can make the mapping editor full-screen within the perspective by double-clicking the top bar of the view. You can go back to the original perspective layout by double-clicking again. See Figure 3-57.

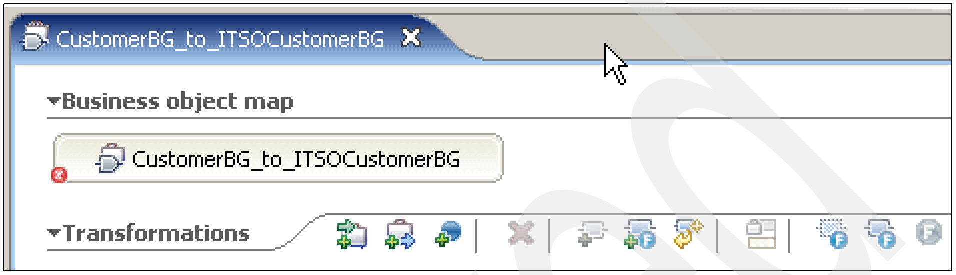


Figure 3-57 Double-click here to make the view full-screen

4. Map the source elements to the target.

   As you hover the mouse over an element in the source business object, an orange circle displays. Select the circle, and drag it to the target element.

   Connections made between two elements in a business object have the Move action as the default. You can change the transformation type by clicking **Move** and selecting a new transformation type from the drop-down menu.

   Connections made between the source and target at the business object level are created with Submap as the transformation type.

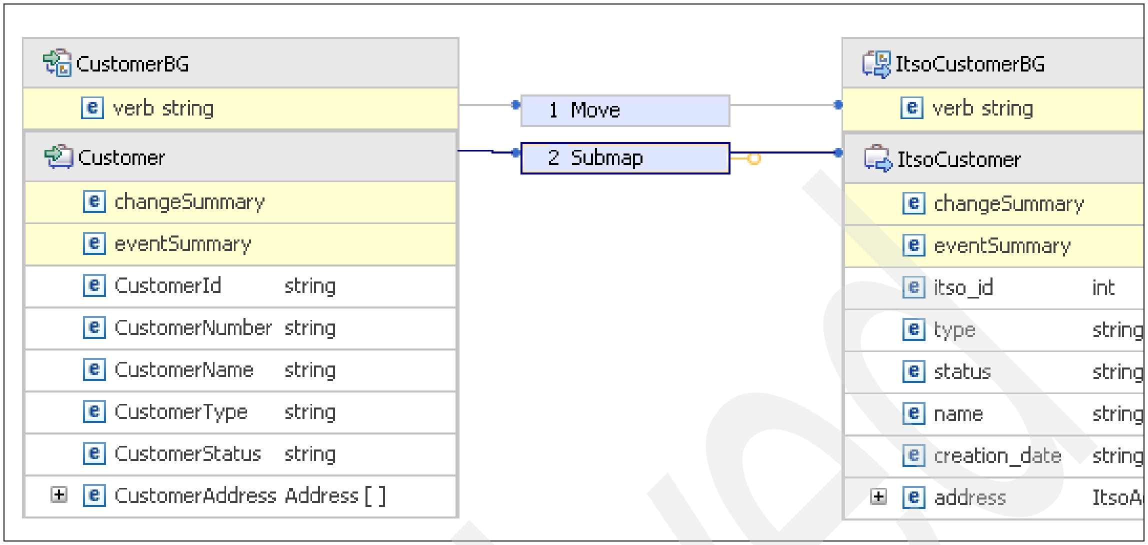
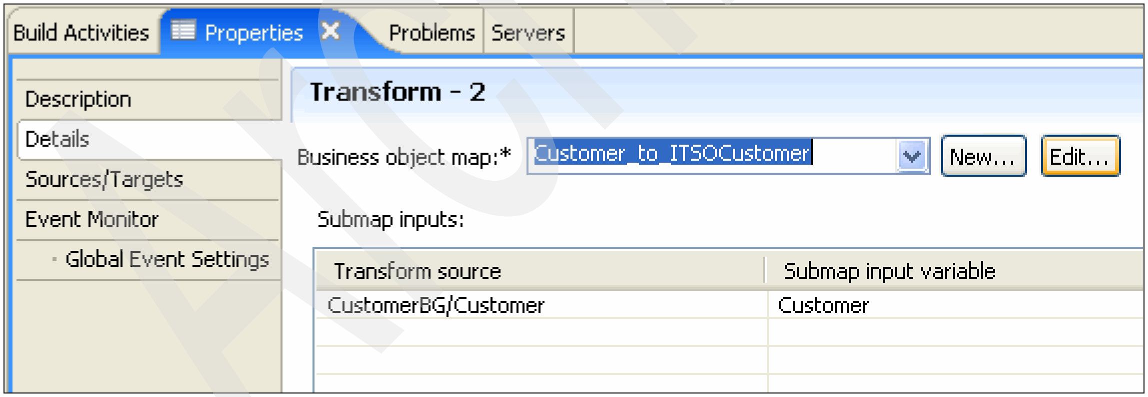


Figure 3-58 Mapping CustomerBG to ItsoCustomerBG

5. Submaps have an underlying map that defines the mapping between the fields in the source and target business objects. The submap mappings are creating in the Details tab of the Properties view for the submap. A submap is stored as a new map. To the right of the business object map field, click the **New** button to provide a name for the new map. See Figure 3-59.



Transform - 2

Business object map:* Customer_to_ITSOCustomer

Submap inputs:

Transform source	Submap input variable
CustomerBG/Customer	Customer

Figure 3-59 Properties tab to create the Customer_to_ITSOCustomer map

6. The new map opens. Map the fields as shown in Figure 3-60.

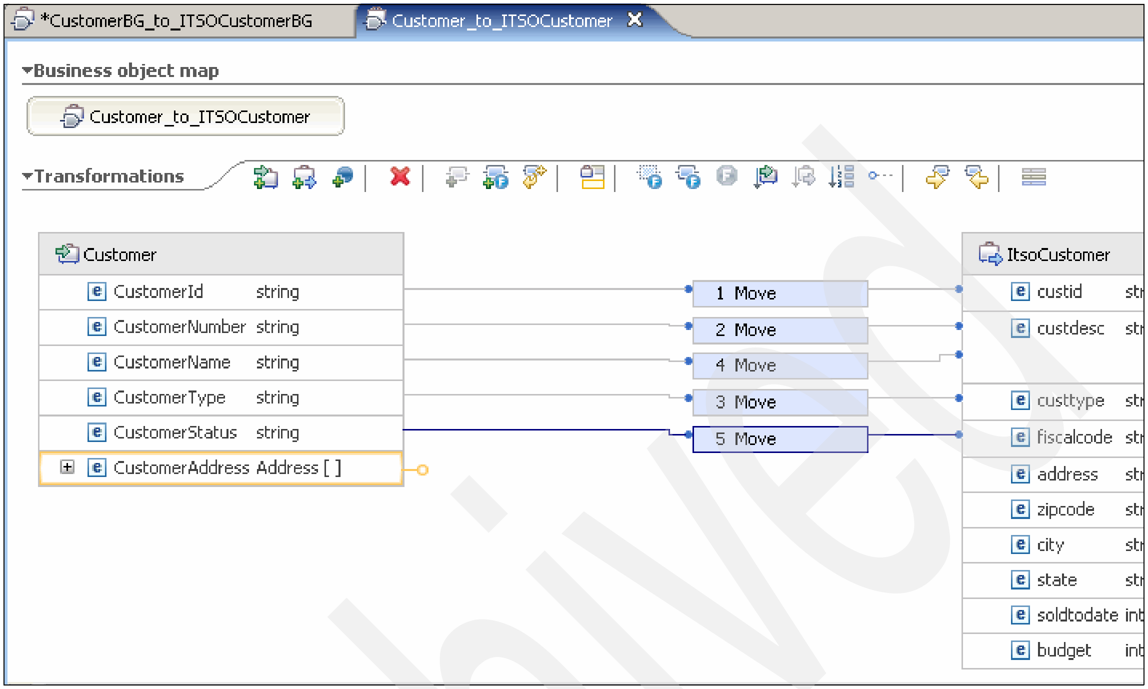


Figure 3-60 Submap

7. Close the submap and save the business object map.

3.8 Using a stand-alone reference

A stand-alone reference allows you to access a process from a non-SCA client (for example from a JSP in a J2EE application). The stand-alone reference is put in the assembly diagram.

> **Note:** Only one stand-alone Reference node available in each assembly diagram at a given time.

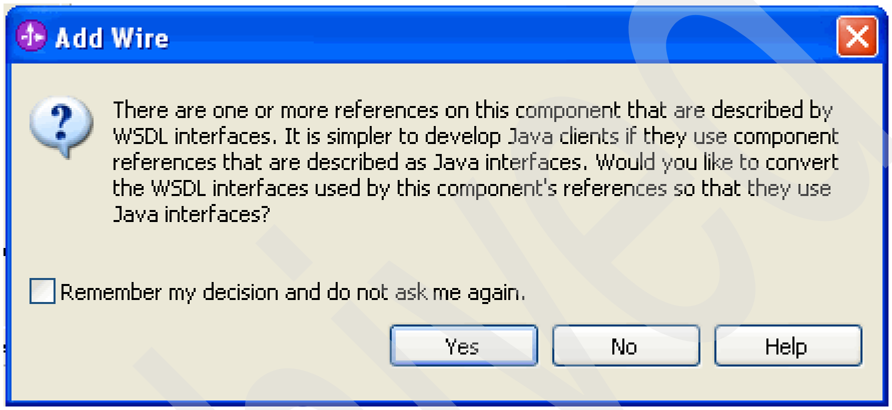
To add a stand-alone reference to the assembly diagram:

1. Select **References** from the Palette, and drop it on the canvas.
2. Wire the stand-alone reference to the process component.

If multiple interfaces exist, select the interface (**CustomerDeliveryInterface** in this example).

When prompted, if you want to convert the WSDL interface to Java interface, click **Yes** because you will be using a Java client (the JSP). See Figure 3-61.

> **Note:** In most applications, you use WSDL definition interface definitions, which is one of the few times that you use Java (the exception to the rule).



There are one or more references on this component that are described by WSDL interfaces. It is simpler to develop Java clients if they use component references that are described as Java interfaces. Would you like to convert the WSDL interfaces used by this component's references so that they use Java interfaces?

□ Remember my decision and do not ask me again.

Figure 3-61 Accepting CustomerDeliveryInterface wiring

The stand-alone reference is now wired with the process component as shown in Figure 3-62.

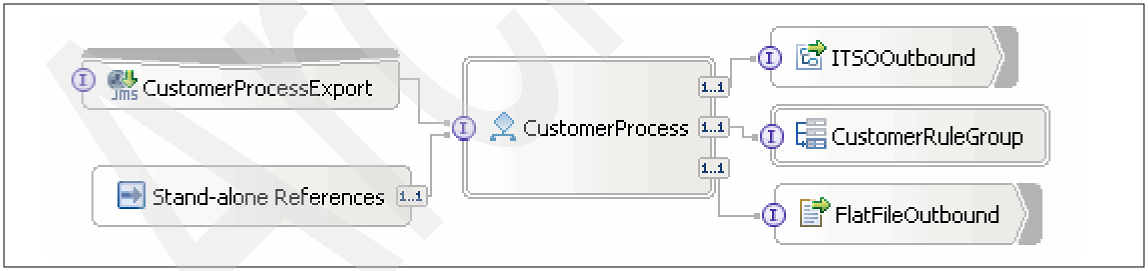


Figure 3-62 Assembly Diagram with Stand-alone Reference

3. Save the assembly diagram.

4. The Properties view for the stand-alone reference resemble those shown in Figure 3-63.

> **Note:** The name of the stand-alone reference is CustomerDeliveryInterfacePartner, which is the name of the non-SCA components that are used to locate the stand-alone reference. (You will see this name used later in the Java code of the client JSP.)

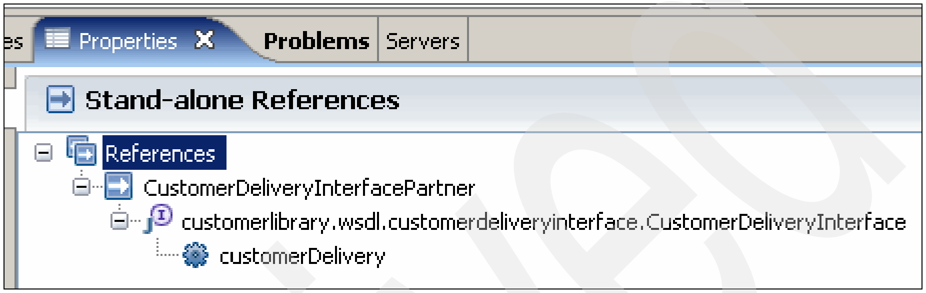

es
Properties
Problems
Servers
Stand-alone References
References
CustomerDeliveryInterfacePartner
customerlibrary.wsdl.customerdeliveryinterface.CustomerDeliveryInterface
customerDelivery

Figure 3-63 Stand-alone References Properties

In the Business Integration view, you see a new class file is generated with the name of the interface (CustomerDeliveryInterface) as shown in Figure 3-64.

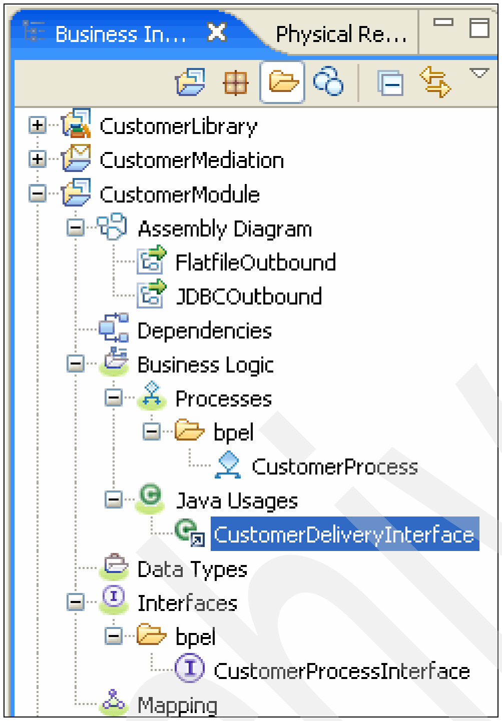
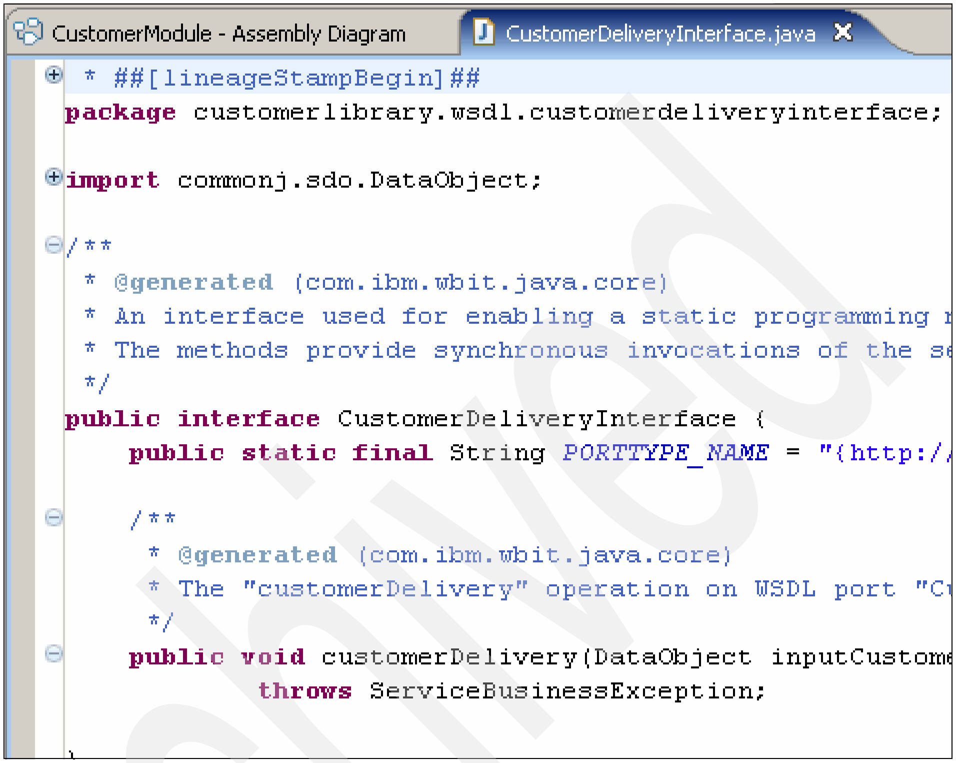


Figure 3-64 Java Usages class file

5. Double-click the **CustomerDeliveryInterface** class file and observe the generated file with customerDelivery method as shown in Figure 3-65.


CustomerModule - Assembly Diagram
CustomerDeliveryInterface.java

```
 * ##[lineageStampBegin]##
package customerlibrary.wsdl.customerdeliveryinterface;

import commonj.sdo.DataObject;

/**
 * @generated (com.ibm.wbit.java.core)
 * An interface used for enabling a static programming r
 * The methods provide synchronous invocations of the se
 */
public interface CustomerDeliveryInterface {
    public static final String PORTTYPE_NAME = "{http:/

    /**
     * @generated (com.ibm.wbit.java.core)
     * The "customerDelivery" operation on WSDL port "Cu
     */
    public void customerDelivery(DataObject inputCustome
            throws ServiceBusinessException;

}
```

Figure 3-65   Generated class file

3.8.1 Invoking the reference from a JSP

The following steps illustrate how to build a client to invoke an SCA component from a JSP:

1. Create a Web project and create a JSP in the project.
2. Add code in JSP to invoke the process (see Example 3-1).

Example 3-1   JSP code snippet to call the CustomerDeliveryInterfacePartner client code

```
<%@ page import="com.ibm.websphere.sca.ServiceManager" %>
<%@ page import="com.ibm.websphere.sca.Service" %>
<%@ page import="commonj.sdo.DataObject" %>
<%@ page import="com.ibm.websphere.bo.BOFactory" %>
<%@ page import="com.ibm.websphere.sca.scdl.*" %>
<%@ page import="com.ibm.websphere.sca.*" %>
<%@ page import="com.ibm.websphere.sca.sdo.*" %>
<%
 try {
   ServiceManager serviceManager = new ServiceManager();
   Service service = 
(Service)serviceManager.locateService("CustomerDeliveryInterfacePartner");

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

   DataObject inputCustomerBGBO = boFactory.create("http://CustomerLibrary/gbo", 
"CustomerBG");
   DataObject inputCustomerBO = boFactory.create("http://CustomerLibrary/gbo", 
"Customer");

   inputCustomerBO.setString("CustomerId","100");
   inputCustomerBO.setString("CustomerNumber","JS00");
   inputCustomerBO.setString("CustomerName","Jane Smith");
   inputCustomerBO.setString("CustomerType","Partner");
   inputCustomerBO.setString("CustomerStatus","Active");

   inputCustomerBGBO.setDataObject("Customer",inputCustomerBO);
   service.invoke("customerDelivery", inputCustomerBGBO);
}
catch (ServiceBusinessException e) {
   System.out.println("Exception occured: ServiceBusinessException" + e);
}
catch (Exception e) {
   System.out.println("Exception occured: Exception" + e);
}
%>
```

3. Open the dependencies for the business integration module with the stand-alone reference. Add a dependency for the Web project as a J2EE project.
4. Make sure that the “Deploy with Module” option is selected so that the Web project is added to the EAR file.

3.9 Working with databases in the workspace

WebSphere Integration Developer provides the Data perspective to help you integrate databases into a solution. This perspective provides tools for creating connections to and working with the following database types:

- Cloudscape®
- DB2 UDB
- DB2 for i5/OS
- DB2 for z/OS
- Derby
- Generic JDBC
- Informix®
- MySql
- Oracle
- SQL Server®
- Sybase

To open the perspective, select **Window** → **Data Perspective**. If the Data Perspective is not in the list, select **Other** and then **Data** in the Open Perspective panel.

3.9.1 Derby databases

Apache Derby is a Java relational database management system that is included with WebSphere Process Server 6.1 and, thus, WebSphere Integration Developer 6.1.

There are two approaches to working with Derby databases:

- Using WebSphere Integration Developer Database Explorer
- Using the ij tool

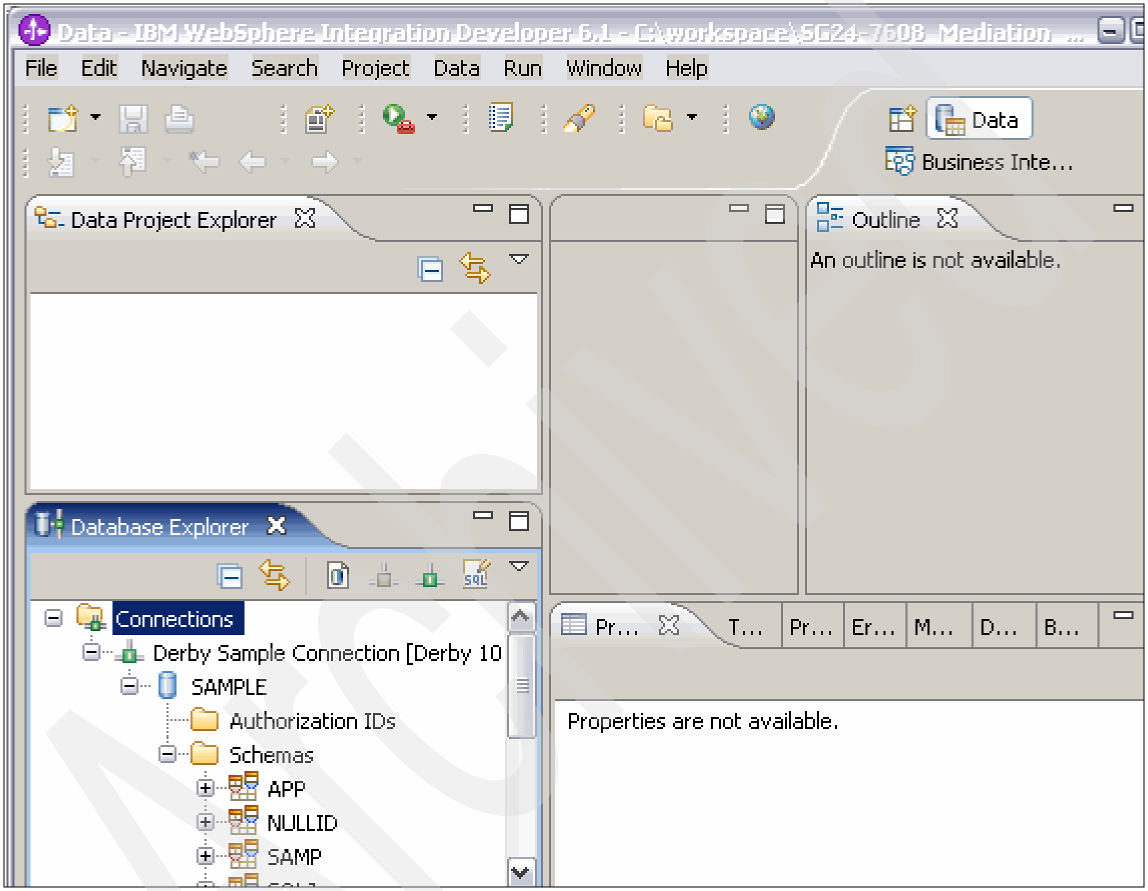
Two variations of Derby are included with WebSphere Process Server:

- Derby Embedded (for stand-alone servers)
- Derby Network (for distributed server environments)

Neither version is appropriate for a production environment.

3.9.2 Using the Database Explorer to connect to a database

The Data perspective includes the Database Explorer view that allows you to define connections to databases and to work with those databases (Figure 3-66). In the Database Explorer view click the plus (+) sign to view all available connections. By default, a sample connection for Derby is included.



An outline is not available.

Properties are not available.

Figure 3-66 Data perspective and the Database Explorer view

Connections are generally made to existing databases, but if you are working with Cloudscape or Derby, you can create a new database at the same time that you create the connection. Follow these steps:

1. Right-click **Connections**. and select **New Connection**. (You could also simply click the New Connection icon). See Figure 3-67.

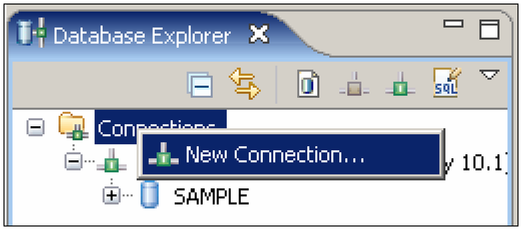
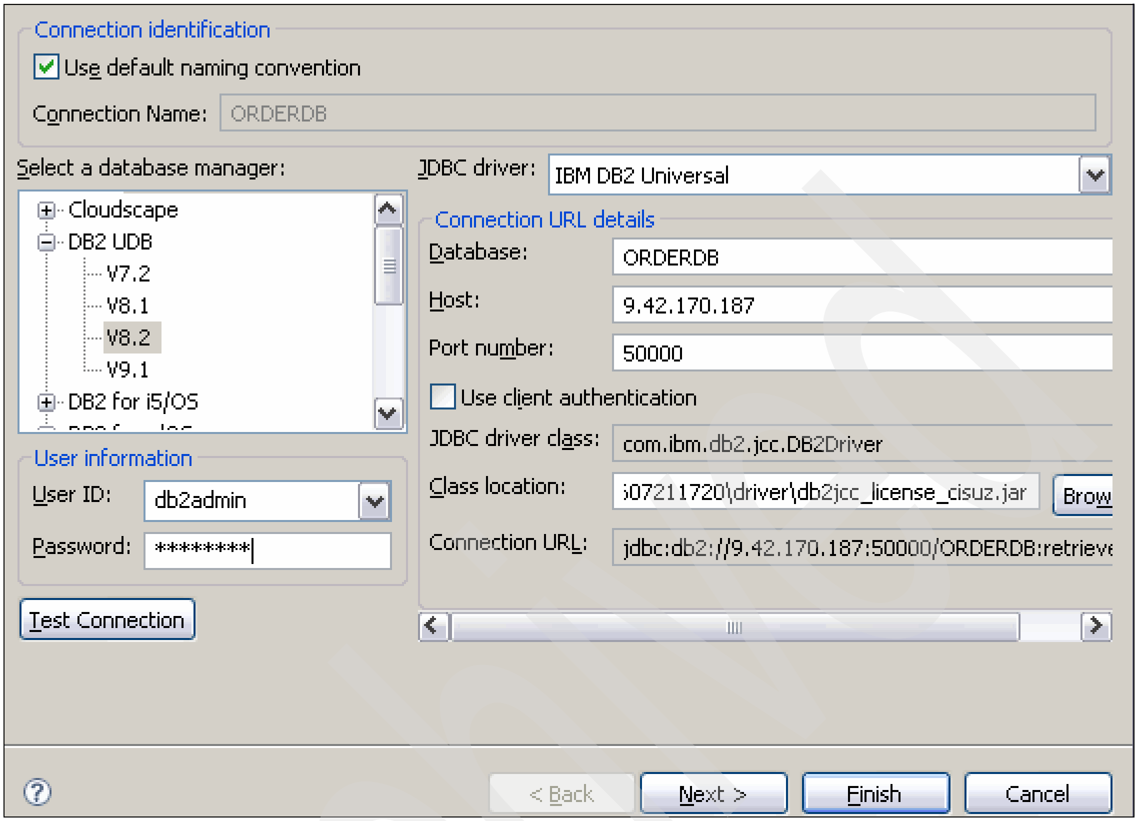


Figure 3-67 Add a new connection

2. On the New Connection panel, select the database type.
3. Enter the information that is required to connect to the database (as shown in Figure 3-68 on page 94). This information varies depending on the database server type that you select, but the basic properties remain the same. Enter:
   - A name for the connection
   - Information that is required to connect to (or create) the database



Connection identification

☑ Use default naming convention

Connection Name: ORDERDB

Select a database manager:

- ⊞ Cloudscape
- ⊟ DB2 UDB
  - V7.2
  - V8.1
  - V8.2
  - V9.1
- ⊞ DB2 for i5/OS

JDBC driver: IBM DB2 Universal

Connection URL details

Database: ORDERDB

Host: 9.42.170.187

Port number: 50000

□ Use client authentication

JDBC driver class: com.ibm.db2.jcc.DB2Driver

Class location: ;07211720\driver\db2jcc_license_cisuz.jar

Connection URL: jdbc:db2://9.42.170.187:50000/ORDERDB:retrieve

User information

User ID: db2admin

Password: *********

Figure 3-68 Create a new database connection

4. Use the **Test Connection** to test the connection to the database.

5. Click **Finish**. The new connection shows in the Database Explorer view. You can see the schema, tables, and other data for the database as shown in Figure 3-69.

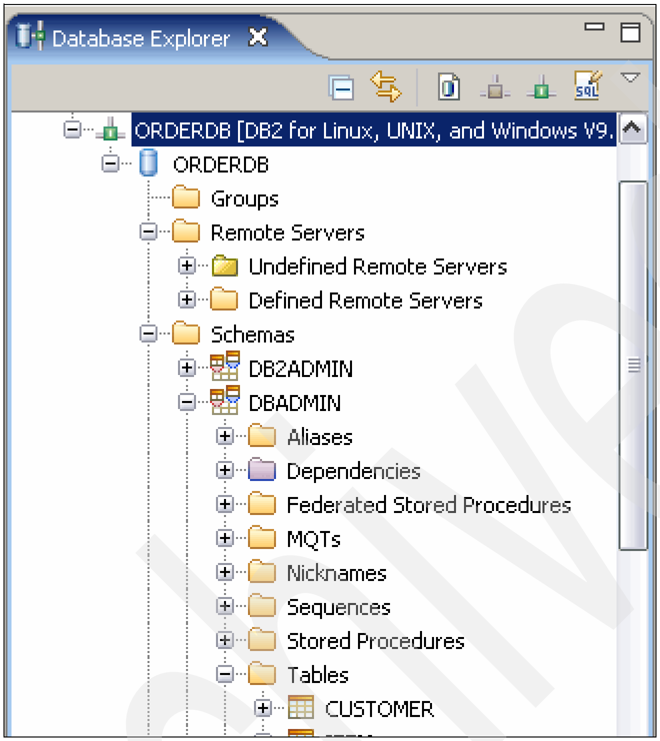


Figure 3-69 New connection

3.9.3 Using the SQL editor

You can use the SQL editor to execute SQL statements against the database to which you are connected:

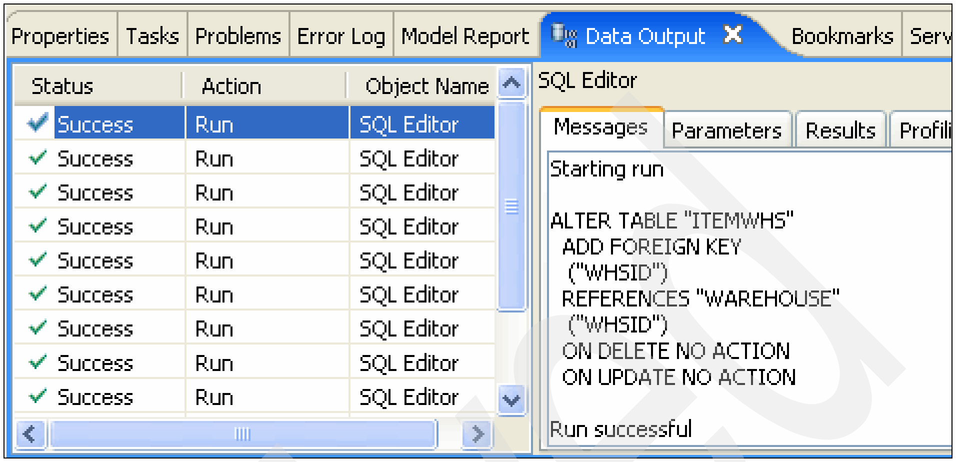
1. Click the Open SQL Editor icon SQL in the toolbar of the Database Explorer view to open a New_Statement_1 tab where you can type or paste SQL statements.

2. When you have entered the statements, right-click anywhere in SQL editor panel, and select **Run SQL** from the pop-up menu.

3. In the Connection Selection panel select **Use an existing connection**, and select the connection.

4. Click **Finish**.

You see the DataOutput report shown in Figure 3-70. Each SQL statement is executed, and its result is reported in the Status column.



Status	Action	Object Name
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor

```
Starting run

ALTER TABLE "ITEMWHS"
 ADD FOREIGN KEY
  ("WHSID")
 REFERENCES "WAREHOUSE"
  ("WHSID")
 ON DELETE NO ACTION
 ON UPDATE NO ACTION

Run successful
```

Figure 3-70 SQL output

3.9.4 Loading data into tables

Data can be loaded into tables from a file that contains the data. For example, the data for the CUSTOMER table is stored in a file (see Example 3-2)

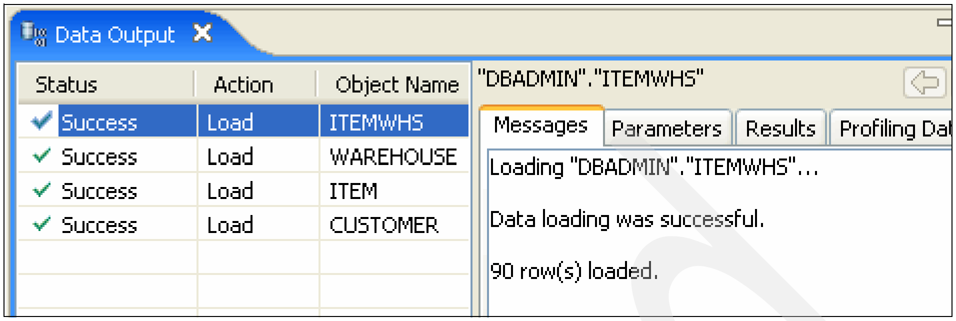
Example 3-2 Database data to load

```
"10001","John Smith Corp.","G","ABC0000000001231","245 South
Road","12601","Poughkeepsie","NY",1000,50000
"10002","Jane Doe Inc.","G","ERT0000000001235","3039 E. Cornwallis
Road","27709","RTP","NC",50000,15000
"10003","YXZ itso Corp.","F","LKJ0000000001232","4205 S. Miami
Blvd","27709","RTP","NC",10000,5000
"10101","John Smith Corp.","P","ABC0000000001231","245 South
Road","12601","Poughkeepsie","NY",1000,50000
```

To load the data into the table:

1. Expand the schema, then expand Tables.
2. Select the table, right-click, select **Data** from the pop-up menu, and then select **Load**.
3. On the Load Data panel select the input file and review the file format parameters to make sure that they correspond to the format that is used in the data file that you plan to load. Then, click **Finish**.

You can see the results in the DataOutput window as shown in Figure 3-71.



Status	Action	Object Name
✓ Success	Load	ITEMWHS
✓ Success	Load	WAREHOUSE
✓ Success	Load	ITEM
✓ Success	Load	CUSTOMER

```
Loading "DBADMIN"."ITEMWHS"...

Data loading was successful.

90 row(s) loaded.
```

Figure 3-71 Results

3.9.5 Viewing and editing data in the tables

Working with the data in the tables is easy with the Database Explorer. Using the CUSTOMER table as an example, to view and edit data in the tables:

1. On Database Explorer window right-click the table, and select **Data** → **Edit** as shown in Figure 3-72.

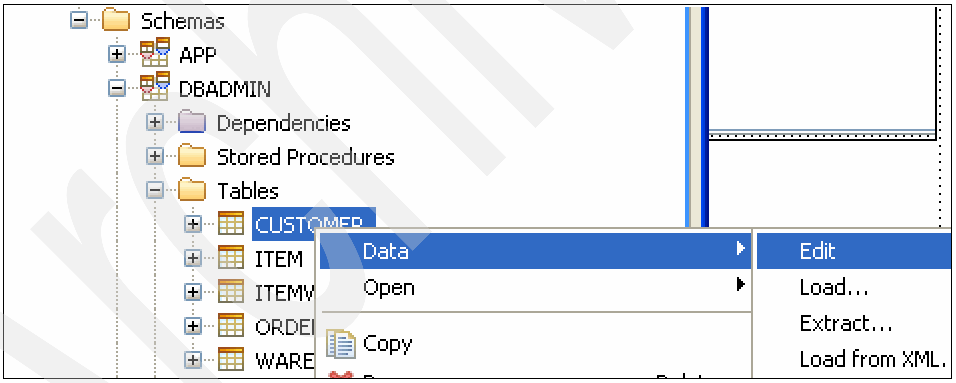
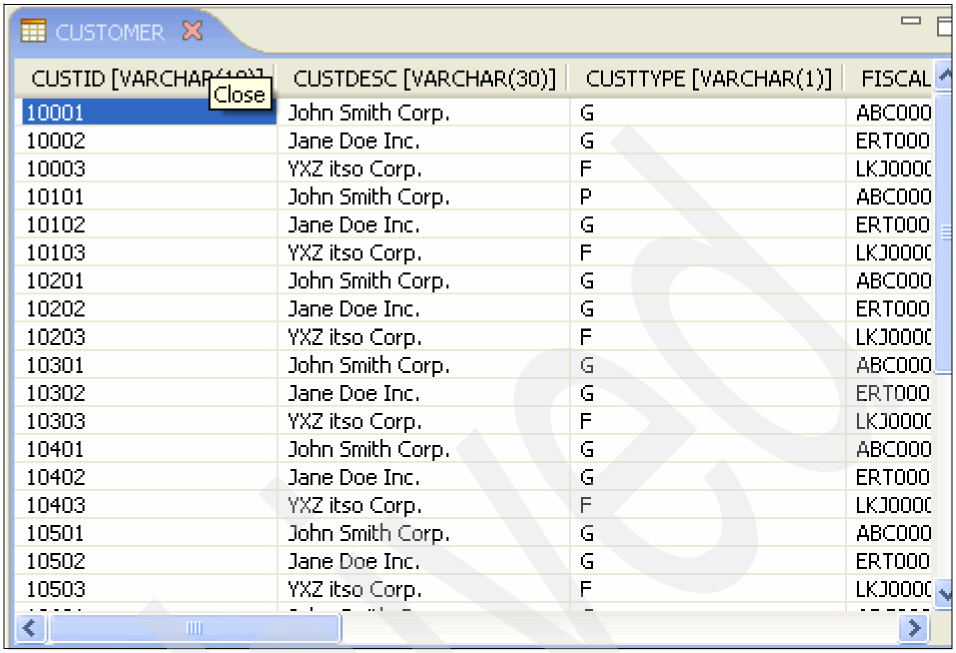


Figure 3-72 Edit tables

A new tab opens that shows the contents of the table (Figure 3-73).



CUSTID [VARCHAR(10)]	CUSTDESC [VARCHAR(30)]	CUSTTYPE [VARCHAR(1)]	FISCAL
10001	John Smith Corp.	G	ABC000
10002	Jane Doe Inc.	G	ERT000
10003	YXZ itso Corp.	F	LKJ000
10101	John Smith Corp.	P	ABC000
10102	Jane Doe Inc.	G	ERT000
10103	YXZ itso Corp.	F	LKJ000
10201	John Smith Corp.	G	ABC000
10202	Jane Doe Inc.	G	ERT000
10203	YXZ itso Corp.	F	LKJ000
10301	John Smith Corp.	G	ABC000
10302	Jane Doe Inc.	G	ERT000
10303	YXZ itso Corp.	F	LKJ000
10401	John Smith Corp.	G	ABC000
10402	Jane Doe Inc.	G	ERT000
10403	YXZ itso Corp.	F	LKJ000
10501	John Smith Corp.	G	ABC000
10502	Jane Doe Inc.	G	ERT000
10503	YXZ itso Corp.	F	LKJ000

Figure 3-73 Table contents

2. To save the updates press **Ctrl+S**, or right-click and select **Save**.

   If you want to restore the changes that you just made, right-click and select **Revert**. You can insert and delete rows as well.

3. To exit, click the *X* at the top of the tab.

3.9.6 Closing connections

The context menu for the connection has options to disconnect and reconnect to the database.

To gracefully close the database connection, select the connection, right-click, and select **Disconnect** (Figure 3-74). The icon to the left of the connection changes color from green to yellow.

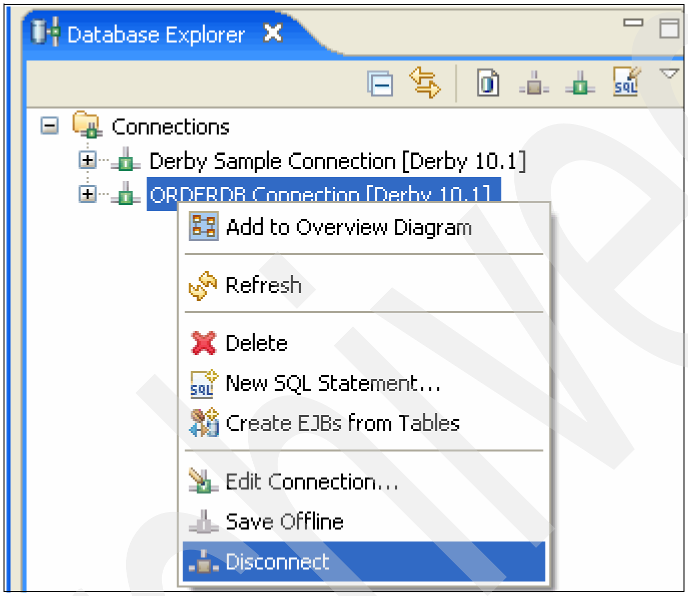


Figure 3-74 Disconnect from the Derby database

To reconnect select the connection, right-click and select **Reconnect**.

3.10 Derby ij tool

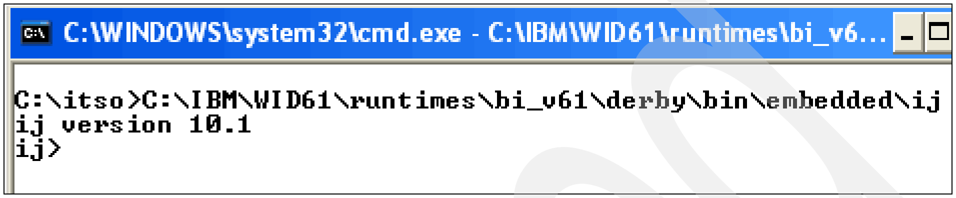
Another way to work with Derby database is to use scripts and the ij scripting tool that comes with WebSphere Integration Developer 6.1, WebSphere Process Server 6.1, and WebSphere Enterprise Service Bus 6.1. You can find more information about ij at:

http://db.apache.org/derby/papers/DerbyTut/ij_intro.html

Here, we provide just a quick demo on how to get started with ij. To start the tool, open a Command Prompt window and type:

```
WID_root\runtimes\bi_v61\derby\bin\embedded\ij
```

If you are working with a WebSphere Process Server or WebSphere Enterprise Service Bus installation rather than WebSphere Integration Developer, replace WID_root with WPS_root or WESB_root respectively.


C:\WINDOWS\system32\cmd.exe - C:\IBM\WID61\runtimes\bi_v6...

```
C:\itso>C:\IBM\WID61\runtimes\bi_v61\derby\bin\embedded\ij
ij version 10.1
ij>
```

Figure 3-75 Starting ij

Now, you are ready to work with ij.

Note: Every **ij** command ends with semicolon (;).

You can use:

- The **help;** command to learn more about the tool.
- The **run '*filename*';** command to execute scripts.

3.10.1 Create a database

Databases are created by running scripts that contain the SQL that is required to create the database and tables. For example, Example 3-3 shows a portion of the script that is used to build the ORDERDB database, create the tables, and import data to the tables.

Example 3-3 ConnectCreateLoadORDERDBTablesLoad.ddl

```
-- ConnectCreateLoadORDERDBTablesLoad.ddl
connect
'jdbc:derby:C:\itso\sampleDB\ORDERDB;create=true;user=dbadmin;password=
dbadmin';

CREATE TABLE "DBADMIN"."ITEM"  (
     "ITEMID" VARCHAR(10) NOT NULL ,
     "ITEMNAME" VARCHAR(30) ,
     "PRICE" INTEGER NOT NULL )  ;
```

```
ALTER TABLE "DBADMIN"."ITEM"
  ADD PRIMARY KEY
   ("ITEMID");
... (more here) ..
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE (null,
'ITEM','C:\itso\RedBookData\Item.data',null,null,null,0);
```

Example 3-4 shows the **ij** command to run this script and the output.

```
run ‘C:\junk\testDerby_ddl\ConnectCreateLoadORDERDBTablesLoad.ddl’
```

Note: If the command hangs, be sure that you entered the semicolon (;) at the end of the command.

Example 3-4 The ij command and output

```
C:\WID61\runtimes\bi_v61\derby\bin\embedded>ij
ij version 10.1
ij> run 'C:\itso\RedBookData\ConnectCreateLoadORDERDBTablesLoad.ddl';
ij> -- ConnectCreateLoadORDERDBTablesLoad.ddl
connect
'jdbc:derby:C:\itso\sampleDB\TESTDB;create=true;user=dbadmin;password=d
b
admin';
ij> ----------------------------------------------
-- DDL Statements for table "ORDERHEADER"
------------------------------------------------

 CREATE TABLE "DBADMIN"."ORDERHEADER"  (
     "ORDID" VARCHAR(40) NOT NULL,
     "CUSTID" VARCHAR(20) ,
     "AMOUNT" INTEGER ,
     "SUBMITTERID" VARCHAR(30) ,
     "STATE" VARCHAR(10) ,
     "CREATIONDATE" TIMESTAMP ,
     "COMPLETIONDATE" TIMESTAMP ,
     "ITEMID" VARCHAR(10) NOT NULL ,
     "ITEMQTY" INTEGER NOT NULL ) ;
0 rows inserted/updated/deleted
ij> -- DDL Statements for primary key on Table ORDERHEADER

ALTER TABLE "DBADMIN"."ORDERHEADER"
  ADD PRIMARY KEY
   ("ORDID");
```

```
0 rows inserted/updated/deleted

... more output here ..

ij> CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE (null,
'ITEMWHS','C:\itso\RedBookData\Ite
mwhs.data',null,null,null,0);
0 rows inserted/updated/deleted
ij>
```

Note that the output indicates:

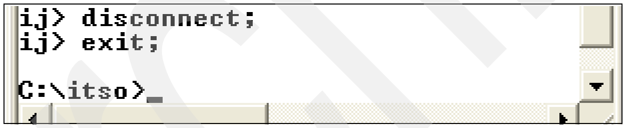
```
0 rows inserted/updated/deleted
```

Data was actually loaded into the tables, which can be verified with Database Explorer. However, note that you have to disconnect from the database in the ij session before accessing the same database from the Database Explorer. Otherwise, you will get an error.

3.10.2 Database disconnect and reconnect

Use the following commands to disconnect from a Derby database (as shown in Figure 3-76):

```
disconnect;
exit;
```



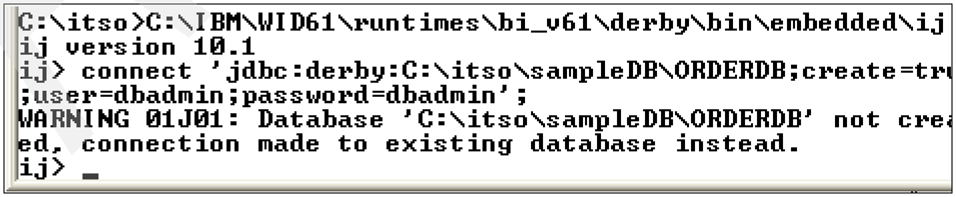
```
ij> disconnect;
ij> exit;

C:\itso>_
```

Figure 3-76 The command to disconnect from a Derby database

To reconnect to a database, use the following command (as shown in Figure 3-77):

```
connect 'jdbc:derby:file;create=true;user=userID;password=pw';
```



```
C:\itso>C:\IBM\WID61\runtimes\bi_v61\derby\bin\embedded\ij
ij version 10.1
ij> connect 'jdbc:derby:C:\itso\sampleDB\ORDERDB;create=tru
;user=dbadmin;password=dbadmin';
WARNING 01J01: Database 'C:\itso\sampleDB\ORDERDB' not crea
ed, connection made to existing database instead.
ij> _
```

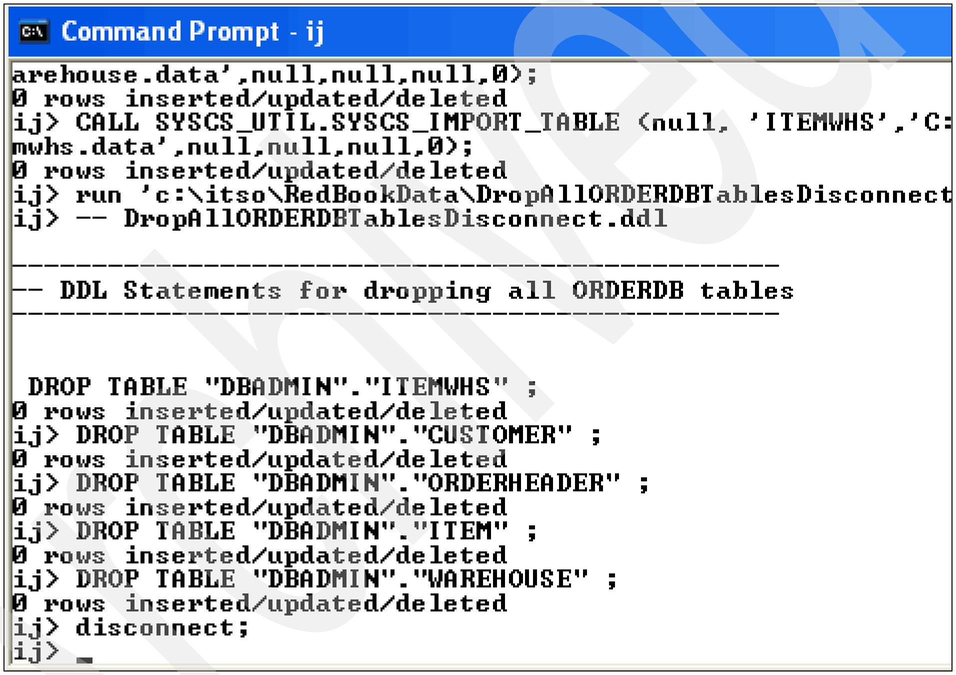
Figure 3-77 The command to reconnect to a database

3.10.3 Drop tables

To drop tables and disconnect from the database, create a script with the commands and run it from the ij tool.

Note: The order in which the ORDERDB tables are dropped is important because of imposed constraints. The script that is used to drop ORDERDB is called DropAllORDERDBTablesDisconnect.ddl.

Figure 3-78 shows the commands that are included in this file and that are executed by the script.



```
arehouse.data',null,null,null,0);
0 rows inserted/updated/deleted
ij> CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE (null, 'ITEMWHS','C:
mwhs.data',null,null,null,0);
0 rows inserted/updated/deleted
ij> run 'c:\itso\RedBookData\DropAllORDERDBTablesDisconnect
ij> -- DropAllORDERDBTablesDisconnect.ddl

-------------------------------------------------
-- DDL Statements for dropping all ORDERDB tables
-------------------------------------------------


 DROP TABLE "DBADMIN"."ITEMWHS" ;
0 rows inserted/updated/deleted
ij> DROP TABLE "DBADMIN"."CUSTOMER" ;
0 rows inserted/updated/deleted
ij> DROP TABLE "DBADMIN"."ORDERHEADER" ;
0 rows inserted/updated/deleted
ij> DROP TABLE "DBADMIN"."ITEM" ;
0 rows inserted/updated/deleted
ij> DROP TABLE "DBADMIN"."WAREHOUSE" ;
0 rows inserted/updated/deleted
ij> disconnect;
ij> _
```

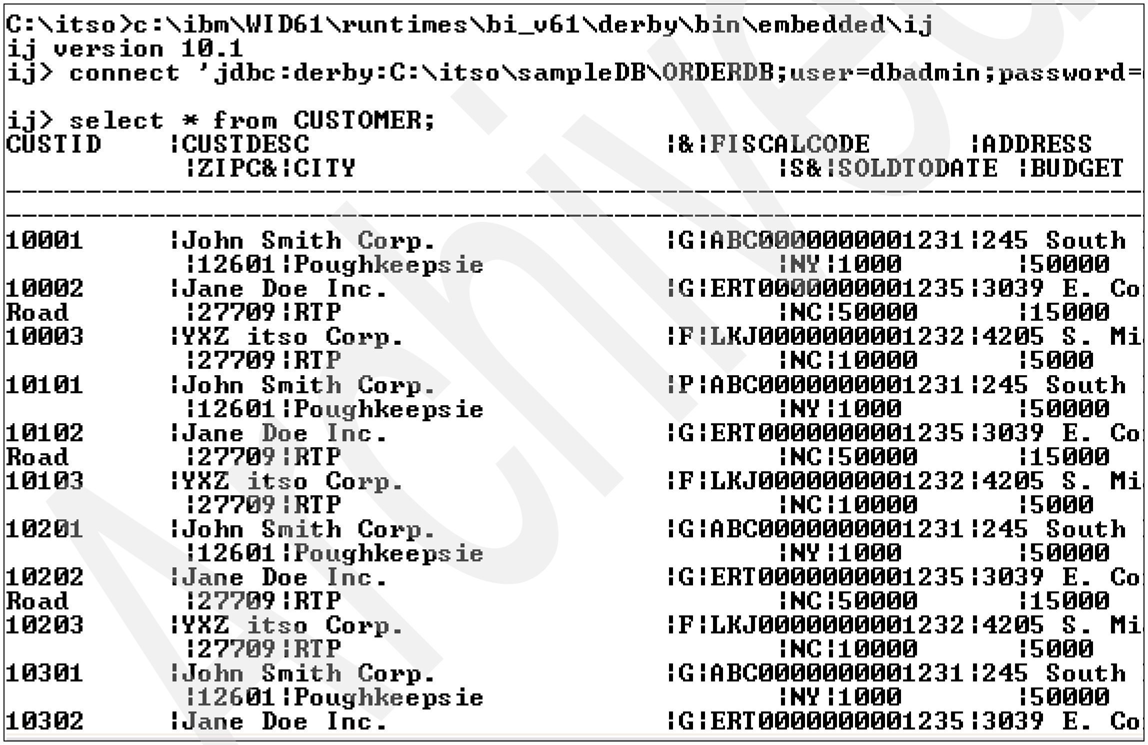
Figure 3-78 The DropAllORDERDBTablesDisconnect.ddl script

3.10.4 Viewing a database

You can type SQL statements directly in ij to interact with a database. For example:

```
connect 'jdbc:derby:file;create=true;user=userID;password=pw';
SELECT * FROM table;
(view output)
disconnect;
exit;
```

For example, if you want to verify whether the CUSTOMER table was initialized correctly with application data, you type in the SQL statements shown in Figure 3-79.



```
C:\itso>c:\ibm\WID61\runtimes\bi_v61\derby\bin\embedded\ij
ij version 10.1
ij> connect 'jdbc:derby:C:\itso\sampleDB\ORDERDB;user=dbadmin;password=

ij> select * from CUSTOMER;
CUSTID    |CUSTDESC                      |&|FISCALCODE      |ADDRESS
           |ZIPC&|CITY                          |S&|SOLDTODATE |BUDGET
-----------------------------------------------------------------------
-----------------------------------------------------------------------
10001     |John Smith Corp.              |G|ABC0000000001231|245 South
           |12601|Poughkeepsie                  |NY|1000       |50000
10002     |Jane Doe Inc.                 |G|ERT0000000001235|3039 E. Co
Road       |27709|RTP                           |NC|50000      |15000
10003     |YXZ itso Corp.                |F|LKJ0000000001232|4205 S. Mi
           |27709|RTP                           |NC|10000      |5000
10101     |John Smith Corp.              |P|ABC0000000001231|245 South
           |12601|Poughkeepsie                  |NY|1000       |50000
10102     |Jane Doe Inc.                 |G|ERT0000000001235|3039 E. Co
Road       |27709|RTP                           |NC|50000      |15000
10103     |YXZ itso Corp.                |F|LKJ0000000001232|4205 S. Mi
           |27709|RTP                           |NC|10000      |5000
10201     |John Smith Corp.              |G|ABC0000000001231|245 South
           |12601|Poughkeepsie                  |NY|1000       |50000
10202     |Jane Doe Inc.                 |G|ERT0000000001235|3039 E. Co
Road       |27709|RTP                           |NC|50000      |15000
10203     |YXZ itso Corp.                |F|LKJ0000000001232|4205 S. Mi
           |27709|RTP                           |NC|10000      |5000
10301     |John Smith Corp.              |G|ABC0000000001231|245 South
           |12601|Poughkeepsie                  |NY|1000       |50000
10302     |Jane Doe Inc.                 |G|ERT0000000001235|3039 E. Co
```

Figure 3-79 Viewing a database

3.11 Deploying modules

Deploying modules involves placing the files that comprise modules and adapters, if used, into an operational environment for production or testing. The two deployment environments include:

- WebSphere Integration Developer integrated test environment
- WebSphere Process Server production environment

3.11.1 Deploying the module for testing

WebSphere Integration Developer integrated test environment incorporates the runtime environment support for WebSphere Process Server, WebSphere Enterprise Service Bus, or both, depending on the test environment profiles that are installed when WebSphere Integration Developer is installed.

In WebSphere Integration Developer, you can deploy the modules to one or more servers in the test environment. This method is typically the most common practice for running and testing business integration modules.

3.11.2 Deploying the module for production

Deploying a module built in WebSphere Integration Developer to WebSphere Process Server in a production environment is a two-step process:

- Export the module in WebSphere Integration Developer as an enterprise archive (EAR) file.
- Deploy the EAR file using the WebSphere Process Server administrative console.

3.12 Test tools

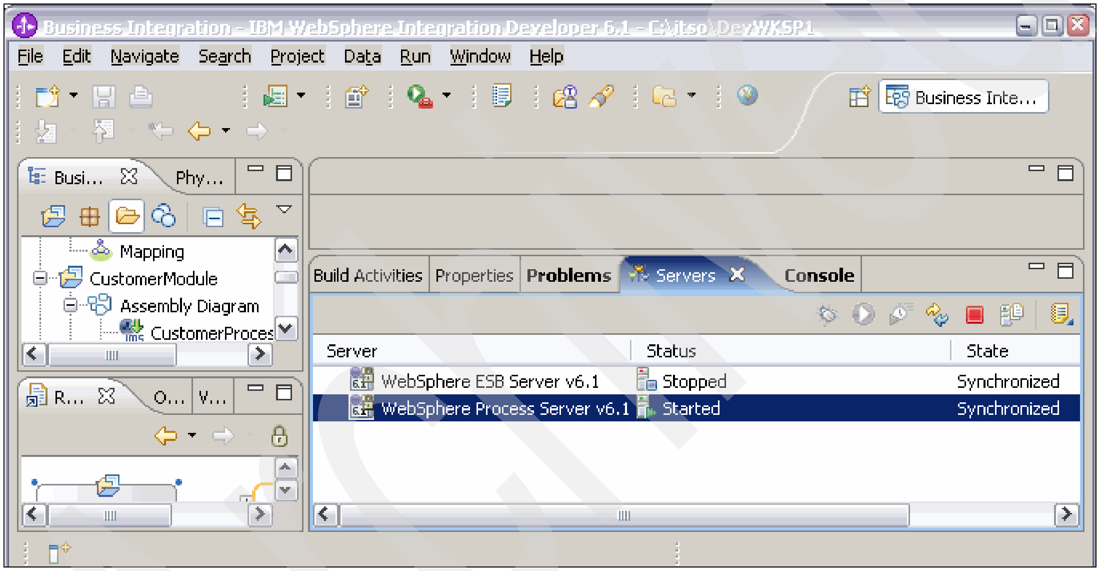
WebSphere Integration Develop provides a variety of tools to help you test your SCA application as well as J2EE applications. This section discusses the most commonly used features for testing for business integration and mediation modules.

3.12.1 Integrated test environment

WebSphere Integration Developer provides a WebSphere Process Server and a WebSphere Enterprise Service Bus runtime environment. The two server types are predefined when you open a workspace. The Servers view and Console view are the primary views that you use to work with servers.

Servers view

The Servers view, shown in Figure 3-80, allows you to start and stop the servers and to publish (deploy) to the server. The toolbar at the top of the view has icons to help you do these actions quickly. You can also select options from the context menu (right-click) of the server.



Server	Status	State
WebSphere ESB Server v6.1	Stopped	Synchronized
WebSphere Process Server v6.1	Started	Synchronized

Figure 3-80 Server view

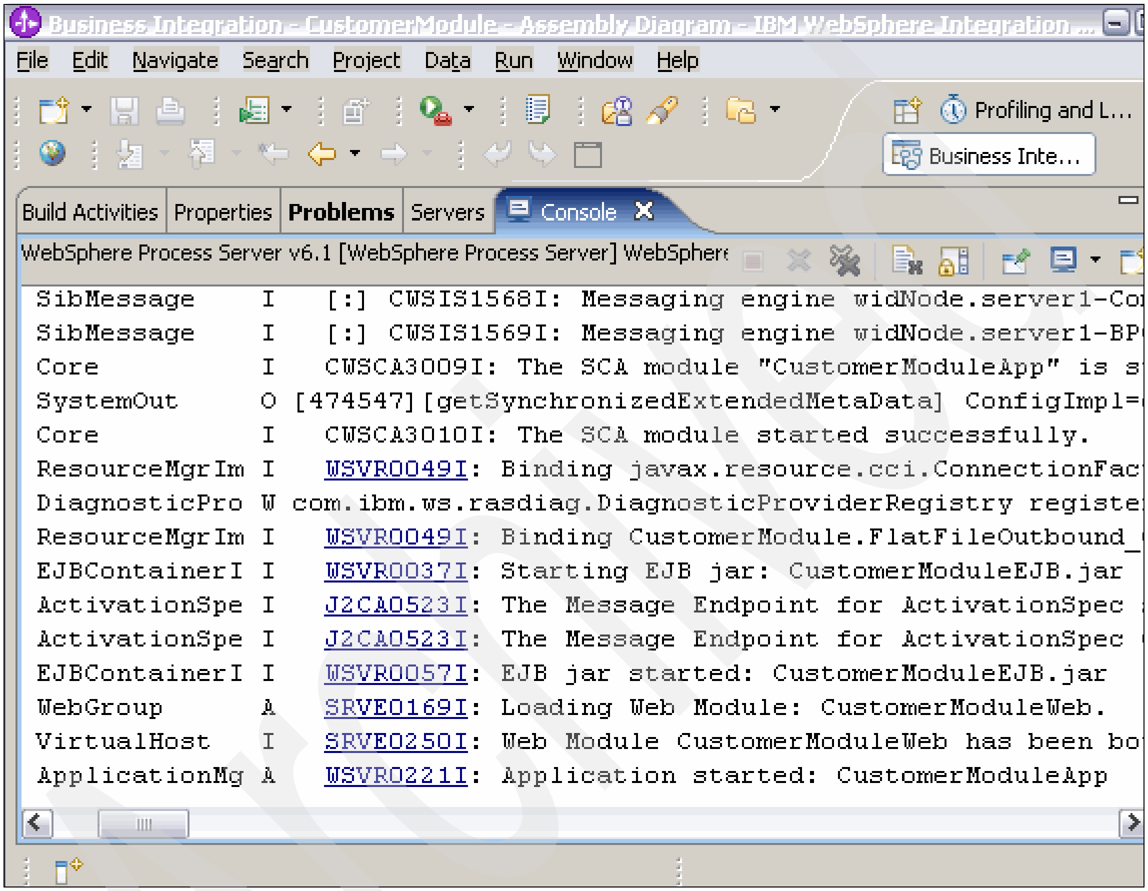
To start a server, select the server, and click the Start icon . The server starts, and you are switched to the Console view where you can monitor the progress of the startup.

To stop a server, select the server, and click the Stop icon . The server stops, and you are switched to the Console view where you can monitor the progress.

When you make changes to deployed applications, you need to republish the applications to the server. To do this, select the server, and click the Publish icon . You need to publish changes only if the server state is Republish. If the state is synchronized, the server has the latest copy of the application.

Console view

The Console view, shown in Figure 3-81, shows you SystemOut messages from the server. Because the messages tend to be very long, double-click the bar above the view (to the right of the Console tab) to make it a full-screen view.


Business Integration - CustomerModule - Assembly Diagram - IBM WebSphere Integration ...
File
Edit
Navigate
Search
Project
Data
Run
Window
Help
Profiling and L...
Business Inte...
Build Activities
Properties
Problems
Servers
Console
WebSphere Process Server v6.1 [WebSphere Process Server] WebSphere

```
SibMessage     I   [:] CWSIS1568I: Messaging engine widNode.server1-Co
SibMessage     I   [:] CWSIS1569I: Messaging engine widNode.server1-BP
Core           I   CWSCA3009I: The SCA module "CustomerModuleApp" is s
SystemOut      O [474547][getSynchronizedExtendedMetaData] ConfigImpl=
Core           I   CWSCA3010I: The SCA module started successfully.
ResourceMgrIm I   WSVR0049I: Binding javax.resource.cci.ConnectionFac
DiagnosticPro W com.ibm.ws.rasdiag.DiagnosticProviderRegistry registe
ResourceMgrIm I   WSVR0049I: Binding CustomerModule.FlatFileOutbound_
EJBContainerI I   WSVR0037I: Starting EJB jar: CustomerModuleEJB.jar
ActivationSpe I   J2CA0523I: The Message Endpoint for ActivationSpec
ActivationSpe I   J2CA0523I: The Message Endpoint for ActivationSpec
EJBContainerI I   WSVR0057I: EJB jar started: CustomerModuleEJB.jar
WebGroup       A   SRVE0169I: Loading Web Module: CustomerModuleWeb.
VirtualHost    I   SRVE0250I: Web Module CustomerModuleWeb has been bo
ApplicationMg A   WSVR0221I: Application started: CustomerModuleApp
```

Figure 3-81 Console view

You can clear the console so that you only see new messages by clicking the Clear console icon .

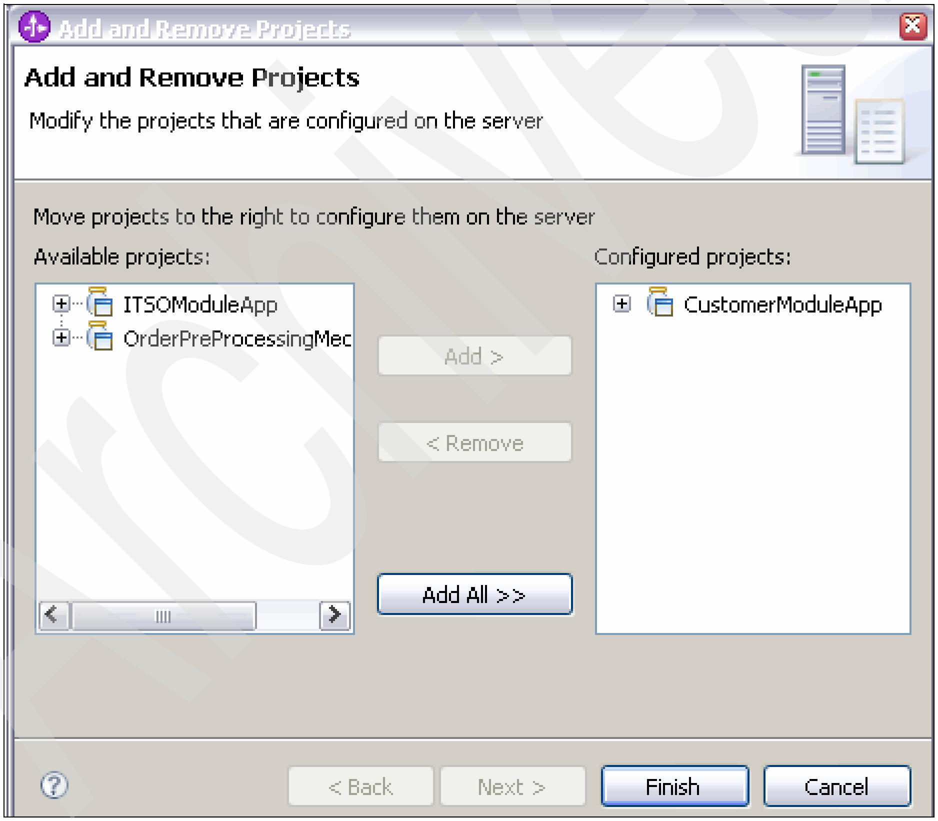
You can switch between consoles of active servers by clicking the Display selected console icon and selecting the server from a drop-down list.

Deploying and undeploying applications

To deploy or undeploy applications to a server, select the server in the Servers view, right-click, and select **Add and remove projects**.

Component testing: When you run a component test, the application is deployed for you automatically. However, you might want to deploy the application and any dependent applications ahead of time to make sure all the components that you need are available.

Move the projects to deploy from the Available projects column to the Configured projects column, and click **Finish**. Note that projects are packaged as applications automatically. The project names you see will be the EAR file name. See Figure 3-82.



Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

- ITSOModuleApp
- OrderPreProcessingMec

Configured projects:

- CustomerModuleApp

Figure 3-82 Add and remove projects

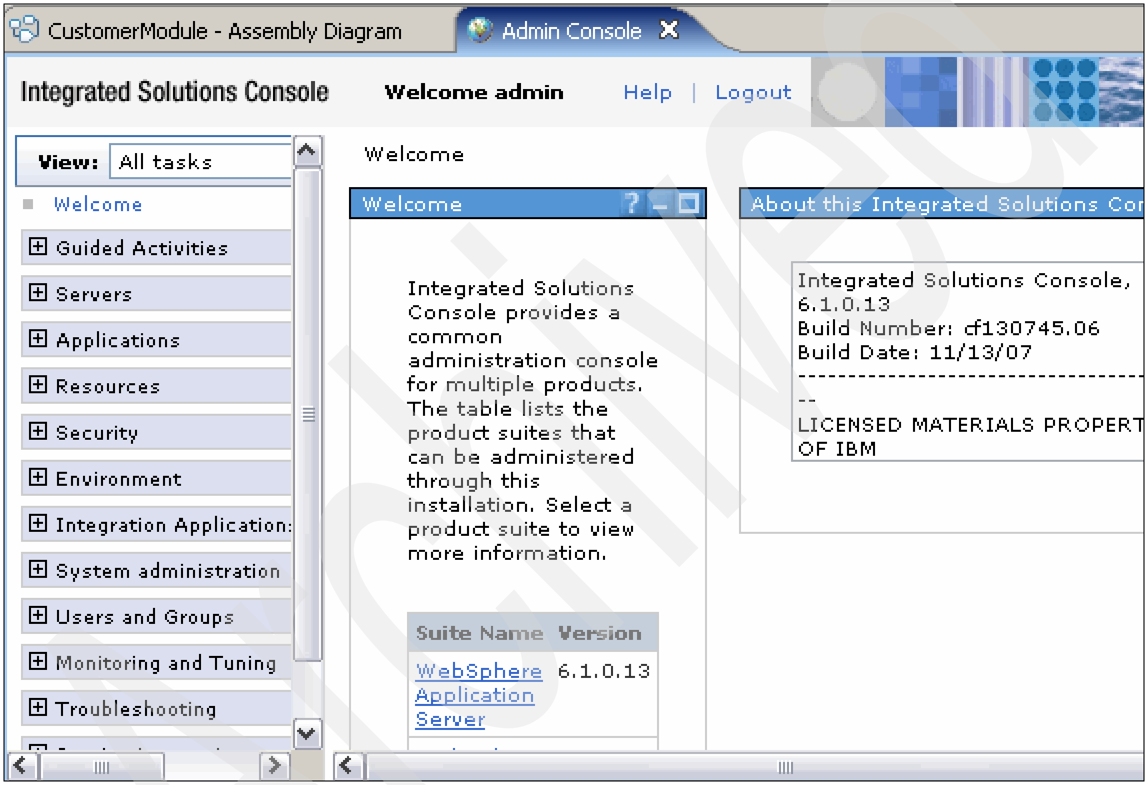
The server must be started for an application to deploy. If the server is stopped, this process starts it and deploys the application.

Administrative console

WebSphere Process Servers and WebSphere Enterprise Service Bus servers are managed through administrative tools. The primary tool is the Web-based administrative console, shown in Figure 3-83.

To open the console in the workspace, select the server in the Servers view, right-click and select **Run administrative console**.

Log in to the console using `admin` as both the user ID and password.



Integrated Solutions Console provides a common administration console for multiple products. The table lists the product suites that can be administered through this installation. Select a product suite to view more information.

Suite Name	Version
WebSphere Application Server	6.1.0.13

Integrated Solutions Console, 6.1.0.13
Build Number: cf130745.06
Build Date: 11/13/07

--
LICENSED MATERIALS PROPERT
OF IBM

Figure 3-83 Administrative console

Logs

The SystemOut and SystemErr logs for the servers is stored, by default, in the *WID_root*\pf*profile*\logs*server* directory, where profile is “wps” or “esb” depending on the server.

You can also view these logs through the administrative console by selecting **Troubleshooting** → **Logs and Trace** → ***server*** → **JVM™ logs**. Select the Runtime tab.

3.12.2 Component testing

WebSphere Integration Developer provides an integration test client that is intended to test modules, individual components, and inter-module components.

The test client provides *emulation* services so that you can test individual components without having all the components available. An emulator can be defined for references, components, or imports in the module. Emulators are manual by default, meaning the test stops at the emulator and allows you to enter the response parameters for the emulated artifact before continuing. You can also have programmatic emulators that use a response file to emulate the artifact.

You can open the integration test client from the Business Integration view or assembly diagram by right-clicking the module or component that you want to test and selecting **Test Module** or **Test Component**.

A typical test sequence for a component is as follows:

1. Deploy the modules to the server.
2. Open the module in the assembly editor.
3. Right-click the component, and select **Test Component**. The integration test client opens to the Events tab.
4. Select the module, component, interface, and operation that you want to test. The test starts with the component that you select, so you can test by providing input at any component interface in the module.
5. Enter the data that you want to provide as input data in the Initial request parameters area. You have several options for entering this data, in addition to entering it manually:

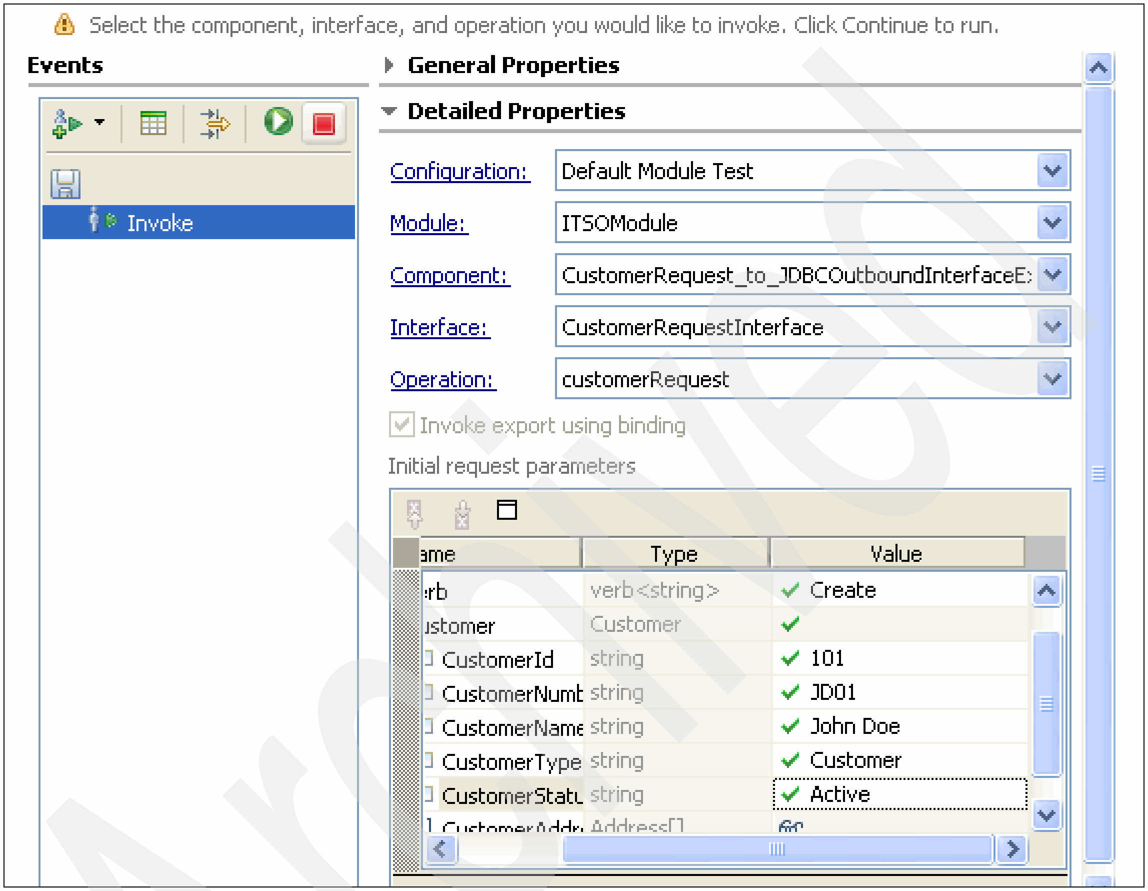
   – Test data can be stored in XML files for import. Attributes in the XML that match the test data attributes are imported. The simplest way to create the XML files is to enter the data into the initial request parameters. Then, right-click the business object in the table, and select **Export to XML file** from the context menu. To use the file in a subsequent test, right-click the business object in the initial request parameters table and select **Import from XML file**.

     This allows you to populate all the input parameter fields easily with pre-set values.

   – Store the data in the data pool. Enter the data manually the first time that you start the test. Then, right-click the field that you want to save and select **Add value to pool** from the context menu. You can retrieve the value for this field in subsequent tests by right-clicking the field and selecting **Use value from pool**.

This allows you to populate individual fields with pre-set values.

– You can also use the **Copy value** and **Paste value** functions from the context menu of each field to copy data from one field to another.



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Events

Invoke

General Properties

Detailed Properties

Configuration: Default Module Test

Module: ITSOModule

Component: CustomerRequest_to_JDBCOutboundInterfaceE:

Interface: CustomerRequestInterface

Operation: customerRequest

☑ Invoke export using binding

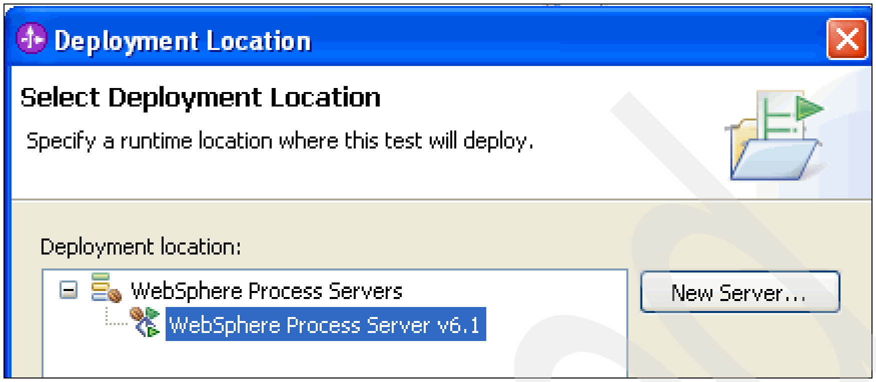
Initial request parameters

ame	Type	Value
rb	verb<string>	✓ Create
ustomer	Customer	✓
CustomerId	string	✓ 101
CustomerNumb	string	✓ JD01
CustomerName	string	✓ John Doe
CustomerType	string	✓ Customer
CustomerStatu	string	✓ Active
CustomerAddr	Address[]	

Figure 3-84 The integration test client populated with Customer info

6. Switch to the Configuration tab to verify, define or remove emulators.
7. Switch back to the Events tab and click the Continue icon ▶ to start the test.

8. Select the deployment location (WebSphere Process Server v6.1) as shown in Figure 3-85 and click **Finish**.



Deployment Location

Select Deployment Location

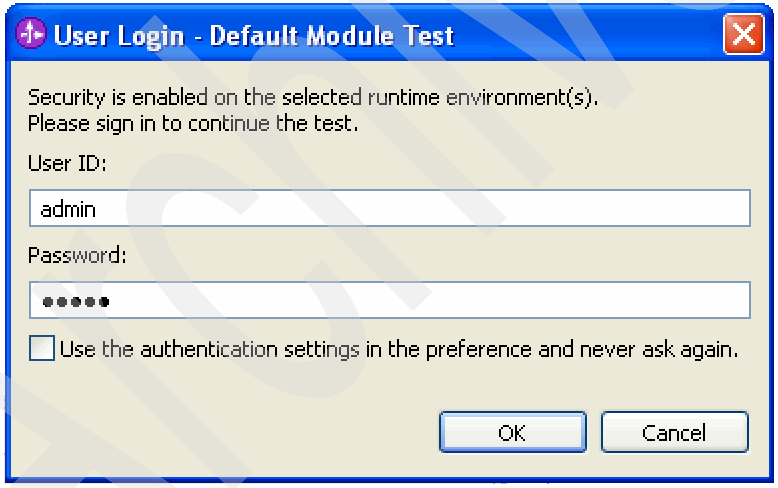
Specify a runtime location where this test will deploy.

Deployment location:

- WebSphere Process Servers
  - WebSphere Process Server v6.1

Figure 3-85 Deployment Location wizard

9. Enter the administrative user ID and password (`admin` for both) and click **OK** as shown in Figure 3-86.



User Login - Default Module Test

Security is enabled on the selected runtime environment(s).
Please sign in to continue the test.

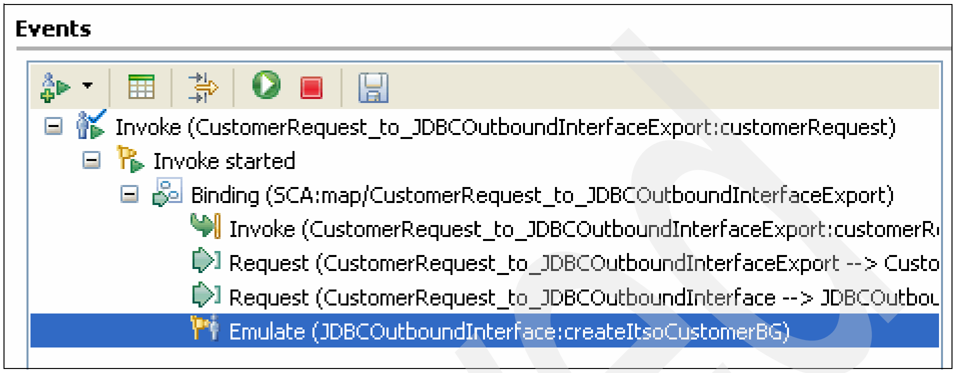
User ID: admin

Password: ●●●●●

□ Use the authentication settings in the preference and never ask again.

Figure 3-86 User Login for Test Module

10. The invocation of the test runs with the data that you provided. Each step through the components is recorded in the Events pane (Figure 3-87), which lets you verify that the path that you expected the module to take is what actually happened.



Events

- Invoke (CustomerRequest_to_JDBCOutboundInterfaceExport:customerRequest)
  - Invoke started
    - Binding (SCA:map/CustomerRequest_to_JDBCOutboundInterfaceExport)
      - Invoke (CustomerRequest_to_JDBCOutboundInterfaceExport:customerR
      - Request (CustomerRequest_to_JDBCOutboundInterfaceExport --> Custo
      - Request (CustomerRequest_to_JDBCOutboundInterface --> JDBCOutbou
      - Emulate (JDBCOutboundInterface:createItsoCustomerBG)

Figure 3-87 Values retuned from the emulation

11. If a component implementation is missing (that is, if the reference in the assembly diagram is not wired), an emulator is defined for that reference. When the module reaches that point, the test client stops with an Emulate event. Select the event, and fill in the data the emulator is to provide as output. Then, click the Continue icon again.

12. At the end of the run, the output data is displayed in the Output parameters window.

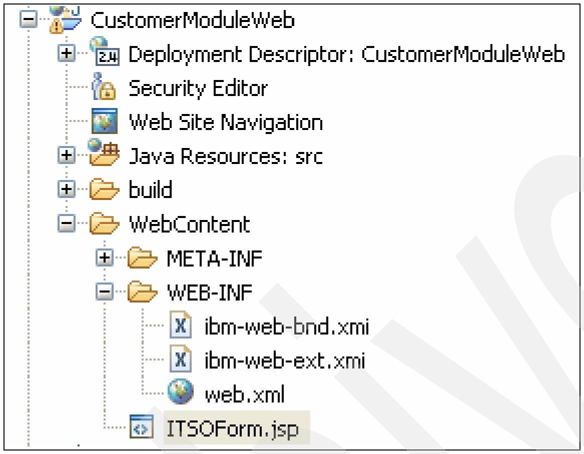
13. To run a new test with different input parameters, click the Invoke icon . Complete the new input parameters, and click the Continue icon.

    To repeat a test using the same input data, right-click the Invoke event and select **Rerun**.

3.12.3 JSP component testing

To test a JSP component:

1. Switch to the J2EE perspective or one of the Java perspectives.
2. In the Physical Resources view, navigate to the Web module and locate the JSP as shown in Figure 3-88.



```
CustomerModuleWeb
  Deployment Descriptor: CustomerModuleWeb
  Security Editor
  Web Site Navigation
  Java Resources: src
  build
  WebContent
    META-INF
    WEB-INF
      ibm-web-bnd.xmi
      ibm-web-ext.xmi
      web.xml
    ITSOForm.jsp
```

Figure 3-88 CustomerModuleWeb in the J2EE perspective

3. Right-click the JSP and select **Run As** → **1. Run on Server** (Figure 3-89).

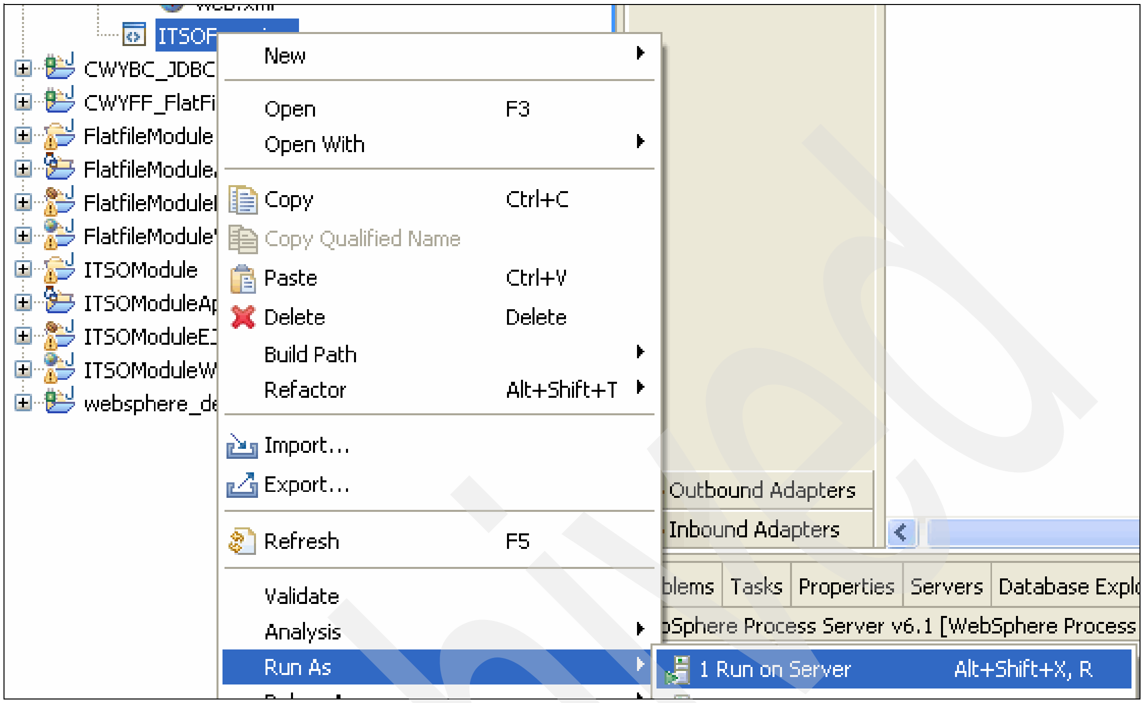


Figure 3-89 Starting JSP page from J2EE perspective

4. Select **WebSphere Process Server v6.1** and click **Finish**.
5. Open a browser, enter the URL for the JSP, and test the execution.

3.13 Team development

WebSphere Integration Development supports team development with Concurrent Versions System (CVS) and Rational ClearCase®. This support allows developers to coordinate complex development projects and tasks (for example building and reuse, sharing of components, and version control) in a team environment.

3.13.1 Sharing your integration project

The single user environment is the basic environment for authoring a project. In this environment, a single user at a time builds, stores, validates, and analyzes the application. The single user environment is ideal for small scale projects in which parallel (or concurrent) development of the project does not often occur.

The multi-user environment supports larger projects with several users contributing to build the project. Parallel development occurs when two or more users work on the project simultaneously. The danger with parallel development in the single user environment is that data loss can occur.

If the team is using importing and exporting artifacts to share the project, the team members must cooperate and coordinate closely to avoid overwriting each other's changes and causing data loss.

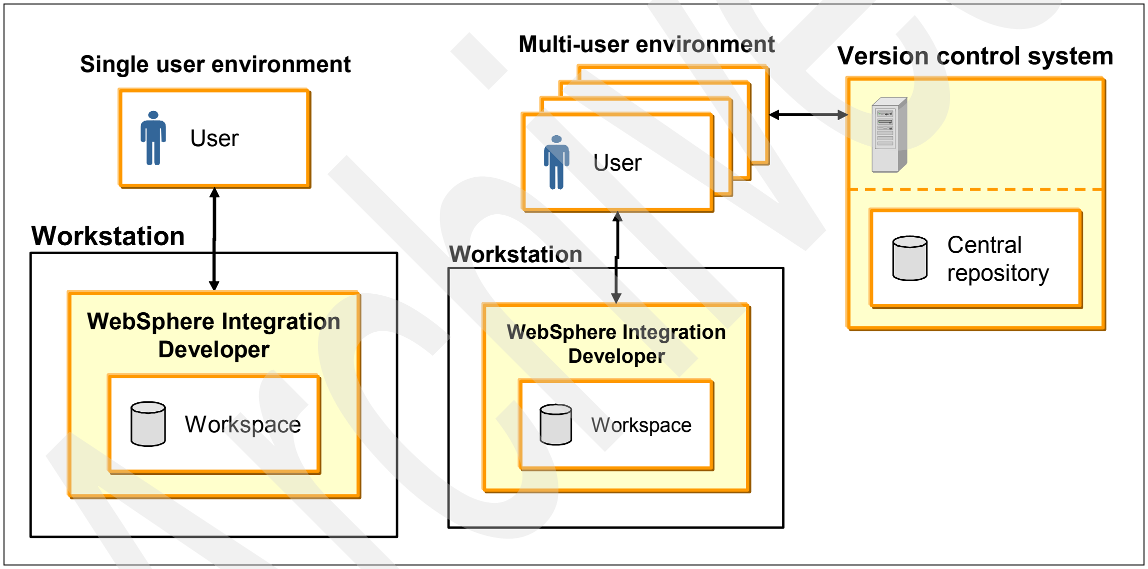


Figure 3-90 Shared project integration

The following are reasons for using versioning tools:

- Efficient, reliable, and layered security
- Diversity of support for multiple platforms
- Central and distributed code
- Backup and multiple layers of saved changes
- Only differences are stored on server—not copies
- *Concurrent* access to the same file by multiple users

3.13.2 Using CVS

CVS is a program that allows multiple users to access, modify, and update projects and their contents simultaneously. It lets a team of developers share control of different versions of files in a common repository of files. CVS provides options to record the history of the source files in such way that several developers can work on the same project concurrently, where developers can edit files within their own working copy of the project and send (or check in) modifications to the server.

Table 3-1 lists common CVS terminology.

Table 3-1 Common CVS terminologies

Components	Descriptions
Commit (check in)	The user makes changes to an artifact that is checked out by CVS and uploads the changes to the CVS repository.
Update	The user synchronizes the workspace with the repository. The user realizes that the local files are unmodified but that newer versions are available on the server. The user's local copy is out of date, and the user must extract the most recent copy from CVS.
Conflict	The user is trying to commit a file that has already been modified and committed by a different user. In other words, the version currently in CVS is not the version the user originally checked out. The user's artifacts are out of synch with the remote CVS server.
Share	Connect a project to a version control system.
Disconnect	Stop sharing from a version control system.
Check out	Copy to different local machine to make changes.
Synchronize	To determine the differences between the local copy and the central repository and when the user is ready to commit work and to synchronize with the repository.

3.13.3 Install CVS and create a repository

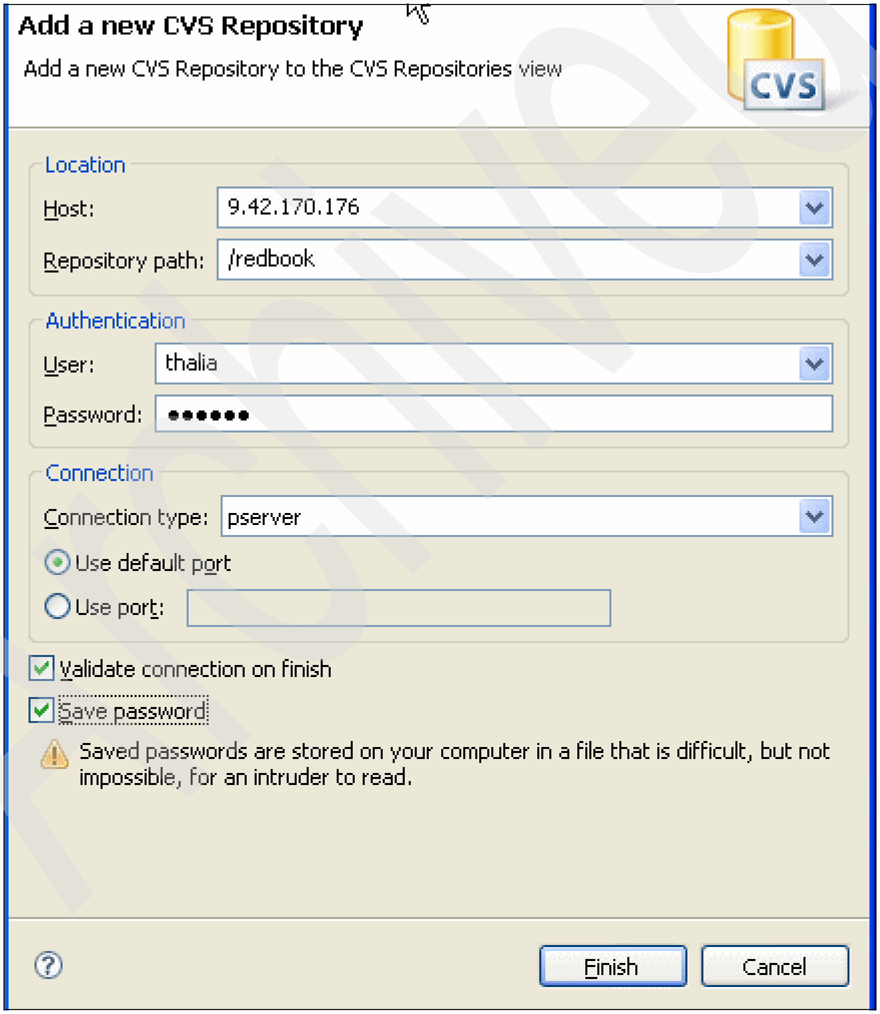
The first step to using a CVS repository is to download the code and to install the server. This process is relatively simple, so we do not cover it here. For more information about downloading and installing a CVS server, see:

http://www.nongnu.org/cvs/

3.13.4 Adding a CVS repository to the workspace

You must define the CVS repository to your workspace in WebSphere Integration Developer. Each development team member must perform the following steps to define the repository on the workspace:

1. Open the CVS Repository Exploring Perspective. Anywhere in the CVS Repositories pane, right-click **New** → **Repository Location**.
2. Enter the information that defines the location of the repository, the user ID with which to connect, and the connection type. Click **Finish**. See Figure 3-91.


Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: 9.42.170.176

Repository path: /redbook

Authentication

User: thalia

Password: ●●●●●●●

Connection

Connection type: pserver

● Use default port

○ Use port:

☑ Validate connection on finish

☑ Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

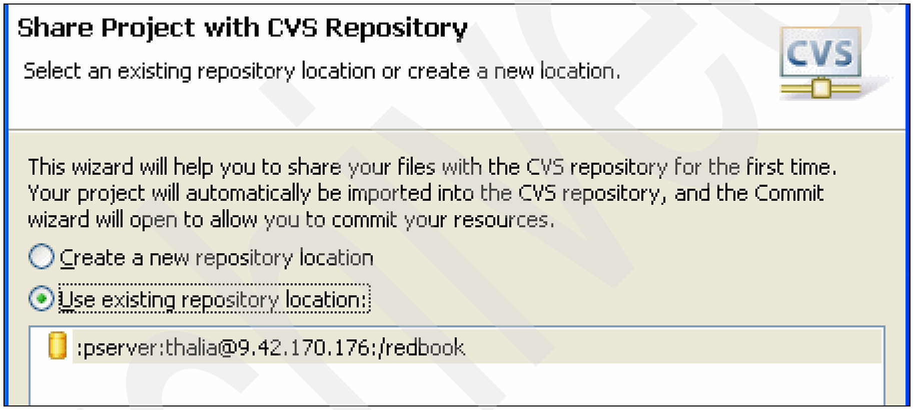
Figure 3-91 Add new CVS Repository wizard

3.13.5 Sharing a project with CVS

Use this option if you have not shared this project before and if this project does not exist in the CVS repository.

To share a project with CVS:

1. Open the Business Integration perspective.
2. In the Business Integration view, right-click the project that you want to share, and then select **Team** → **Share Project**.
3. Select **CVS** as the repository type and click **Next**.
4. Select **Use existing repository location** and select the repository that you added as shown in Figure 3-92. Then, click **Next**.



Share Project with CVS Repository

Select an existing repository location or create a new location.

This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Commit wizard will open to allow you to commit your resources.

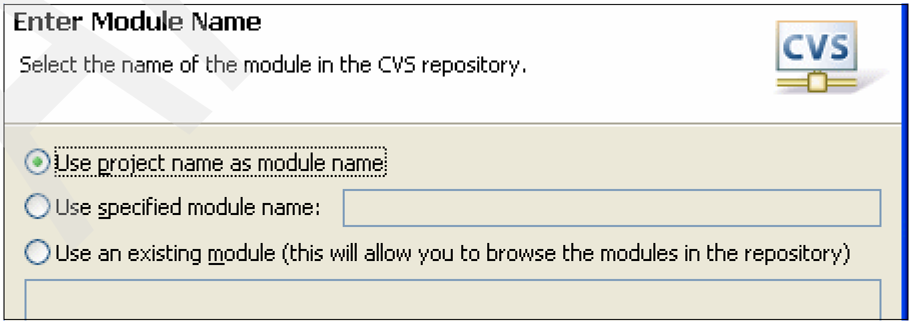
○ Create a new repository location

● Use existing repository location:

:pserver:thalia@9.42.170.176:/redbook

Figure 3-92 Share Project with CVS Repository

5. Accept the default for the module name (**Use project name as module name**) as shown in Figure 3-93, and then click **Next**.



Enter Module Name

Select the name of the module in the CVS repository.

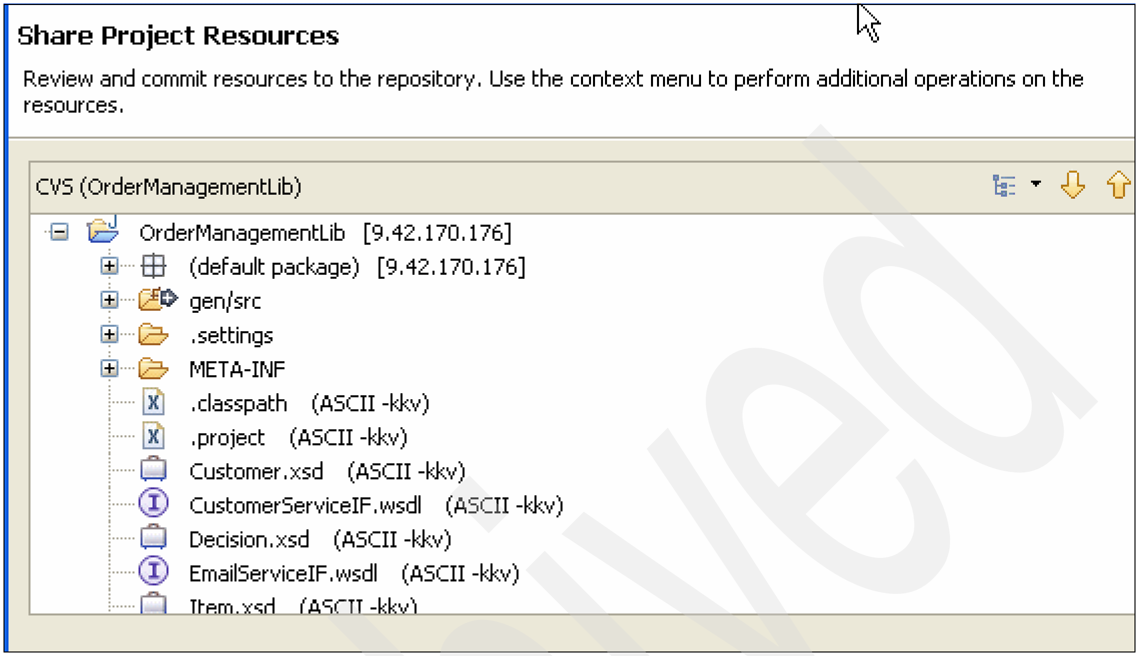
● Use project name as module name

○ Use specified module name:

○ Use an existing module (this will allow you to browse the modules in the repository)

Figure 3-93 Enter Module Name

6. Accept the defaults, as shown in Figure 3-94, and then click **Finish**.



Share Project Resources

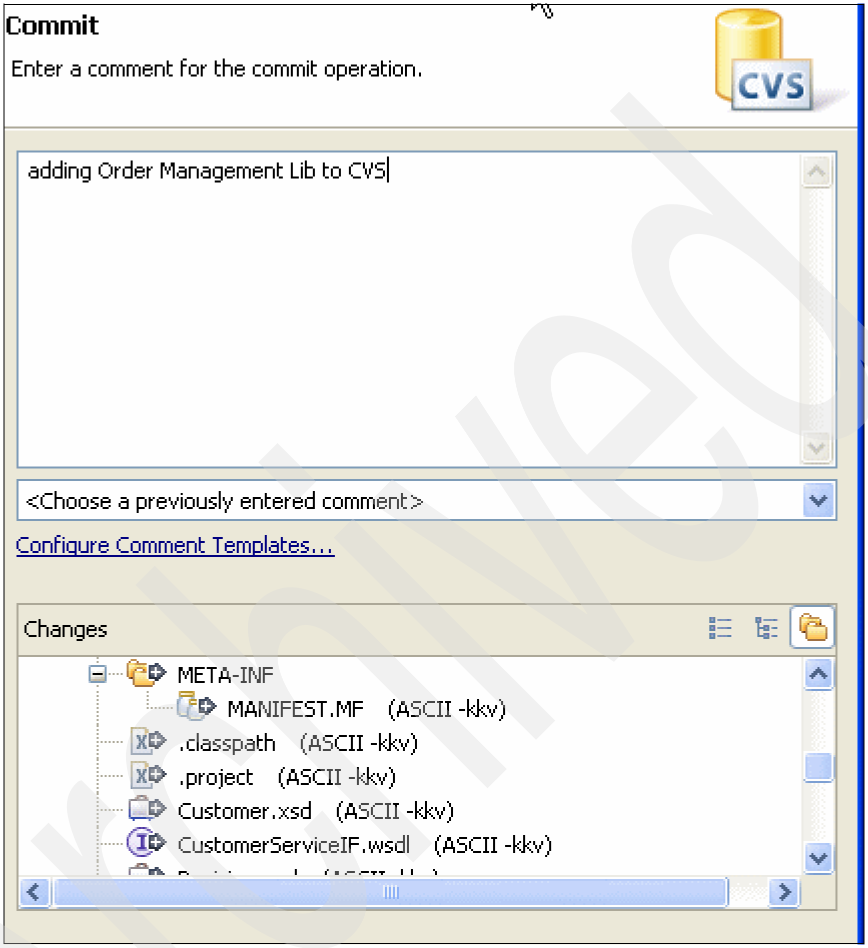
Review and commit resources to the repository. Use the context menu to perform additional operations on the resources.

CVS (OrderManagementLib)

```
OrderManagementLib  [9.42.170.176]
    (default package)  [9.42.170.176]
    gen/src
    .settings
    META-INF
    .classpath   (ASCII -kkv)
    .project   (ASCII -kkv)
    Customer.xsd   (ASCII -kkv)
    CustomerServiceIF.wsdl   (ASCII -kkv)
    Decision.xsd   (ASCII -kkv)
    EmailServiceIF.wsdl   (ASCII -kkv)
    Item.xsd   (ASCII -kkv)
```

Figure 3-94   Share Project Resources

7. Enter a comment to the commit operation as shown in Figure 3-95, and then click **Finish**.



Commit

Enter a comment for the commit operation.

adding Order Management Lib to CVS

<Choose a previously entered comment>

Configure Comment Templates...

Changes

META-INF
MANIFEST.MF (ASCII -kkv)
.classpath (ASCII -kkv)
.project (ASCII -kkv)
Customer.xsd (ASCII -kkv)
CustomerServiceIF.wsdl (ASCII -kkv)

Figure 3-95 Commit files

3.13.6 Checking out a project from CVS

If other team members already have projects checked into CVS (for example, the shared library), you can use the following steps to check these projects out of CVS to start your own development:

1. Open the CVS Repositories view, expand the server, then expand **HEAD** as shown in Figure 3-96.

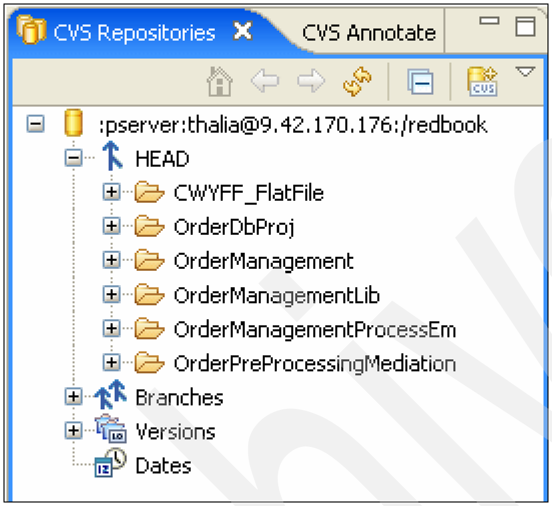


Figure 3-96 CVS server showing HEAD expanded

2. Right-click the project that you want to check out and select **Check Out**.
3. If you switch back to the Business Integration perspective, you see the Project that you just checked out from CVS.

3.13.7 Checking in changes to a project to CVS

As you develop your modules, you will want to check in changes periodically into CVS. The instructions in this section assume that you have either shared your project or that you have checked out a project from CVS.

For example, if you have checked in the module EmailOutbound previously in CVS and made additional changes, you check in new changes with the following steps:

1. The Business Integration perspective shows when a module or file has changes that are not in CVS. A > symbol is prefixed to the artifact name as shown in Figure 3-97.

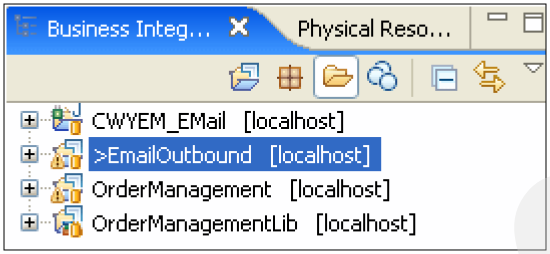

Business Integ...
Physical Reso...
CWYEM_EMail [localhost]
>EmailOutbound [localhost]
OrderManagement [localhost]
OrderManagementLib [localhost]

Figure 3-97 Business Integration perspective showing differences with CVS

2. To check in the changes to EmailOutbound, right-click **EmailOutbound** → **Team** → **Synchronize with Repository.** Confirm that you want to switch to the Team Synchronizing perspective by clicking **Yes**.
3. The Synchronize view has a toolbar at the top with icons that allow you to work with the changes (Figure 3-98).

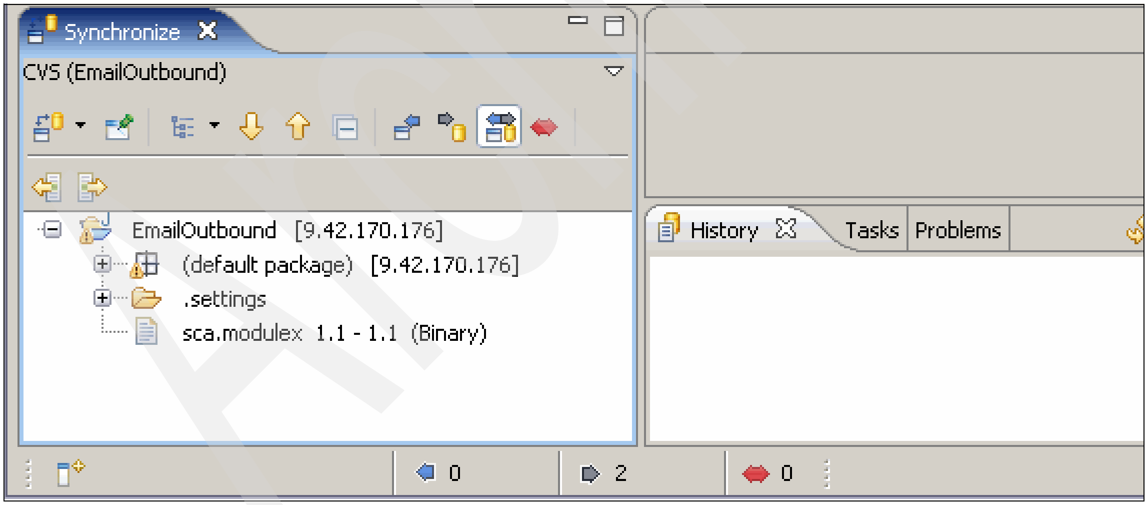
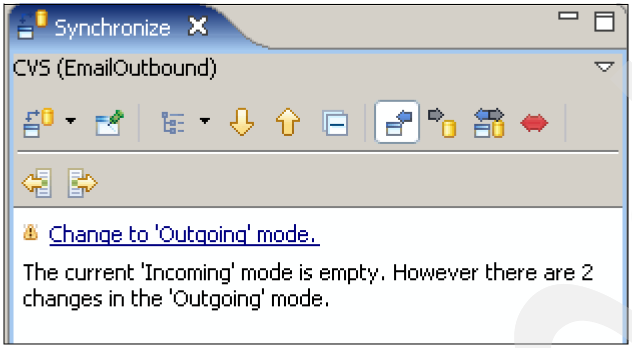

Synchronize
CVS (EmailOutbound)
EmailOutbound [9.42.170.176]
(default package) [9.42.170.176]
.settings
sca.modulex 1.1 - 1.1 (Binary)
History
Tasks
Problems
0
2
0

Figure 3-98 Synchronize view

The status bar at the bottom of the screen shows a total for the incoming, outgoing, and conflicting changes. In this example, there are, respectively, 0 incoming, 2 outgoing, and 0 conflicting changes.

4. Click the Incoming Mode icon to see the changes that are incoming from the repository. In this case, there are no incoming changes as shown in Figure 3-99.

s



Change to 'Outgoing' mode.

The current 'Incoming' mode is empty. However there are 2 changes in the 'Outgoing' mode.

Figure 3-99 Incoming mode

5. Click the Outgoing Mode icon to see the changes that are outgoing (Figure 3-100). You can expand the folders to examine your changes.

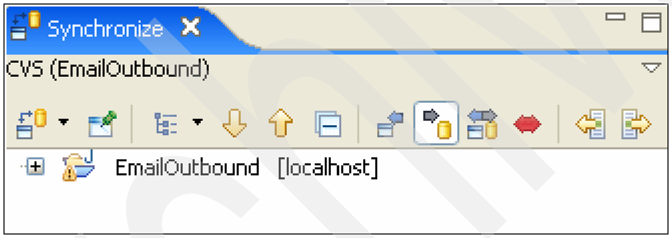


Figure 3-100 Outgoing mode

6. Click the Incoming/Outgoing Mode icon to see both (Figure 3-101).

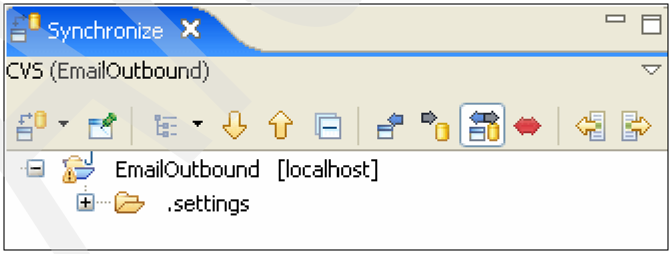


Figure 3-101 Incoming and Outgoing modes

7. To check in your changes to CVS, right-click **EmailOutbound** in the Synchronize view and select **Commit** as shown in Figure 3-102.

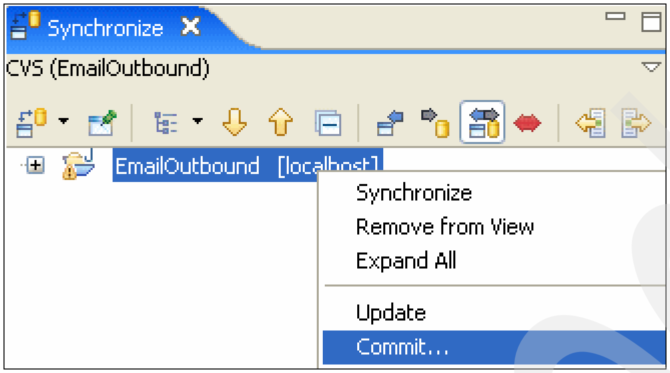


Figure 3-102 Checking in changes into CVS

8. Enter a comment for the commit operation, then click **Finish**.

4

Building business integration applications

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

This chapter discusses the details of creating business integration applications and their supporting artifacts. It includes the following topics:

- Typical development flow
- Service components for modules
- Using Java objects
- Business processes
- Human tasks
- Administering processes and tasks
- Business state machines
- Business rules
- Selectors
- Interface maps

4.1 Typical development flow

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

In Chapter 3, “Basics of development” on page 29, we discussed the typical development flow for modules and mediation modules, which is as follows:

1. Start WebSphere Integration Developer and open a workspace (as described in 3.2.2, “Start WebSphere Integration Developer” on page 42).
2. Switch to the Business Integration perspective for development (as described in 3.2.3, “Using the Business Integration perspective” on page 43).
3. Create a library to store artifacts, such as business objects and interfaces that are shared among multiple modules (as described in 3.3.1, “Libraries” on page 44).
4. Create a new module or mediation module (as described in 3.3.2, “Modules and mediation modules” on page 46).
5. Create the business objects to contain the application data, for example, customer or order data (as described in 3.4, “Business objects” on page 53).
6. Create the interface and define the interface operations for each component. The interface determines what data can be passed from one component to another (as described in 3.5, “Interfaces” on page 63).
7. **Create and implement the service components.**
8. Build the module assembly by adding the service components, imports, and exports to the assembly diagram. Wire the components together. Bind the imports and exports to a protocol (as described in 3.6, “Module assembly” on page 69).
9. Test the module in the integrated test environment (as described in 3.12, “Test tools” on page 105).
10. Deploy the module to WebSphere Process Server. We discuss this step in *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus, Part 3: Run time,* SG24-7643.
11. Share the tested module with others on the team by putting it in a repository (as described in 3.13, “Team development” on page 115).

This chapter focuses on step 7, the implementation of the service components for a business integration module.

4.2 Service components for modules

When you create a module, an assembly diagram for the module is created. As you add components to the module, you populate the assembly diagram with these components. The assembly editor is used to visually compose the integrated application by using elements from the palette or from the tree in the Business Integration view.

The implementations of components that are used in a module assembly reside within the module. Components in other modules can be used in the assembly by publishing the external service as an export, and then dragging the exports into the required assembly diagram to create the import to call the service.

Figure 4-1 shows an example of an assembly diagram.

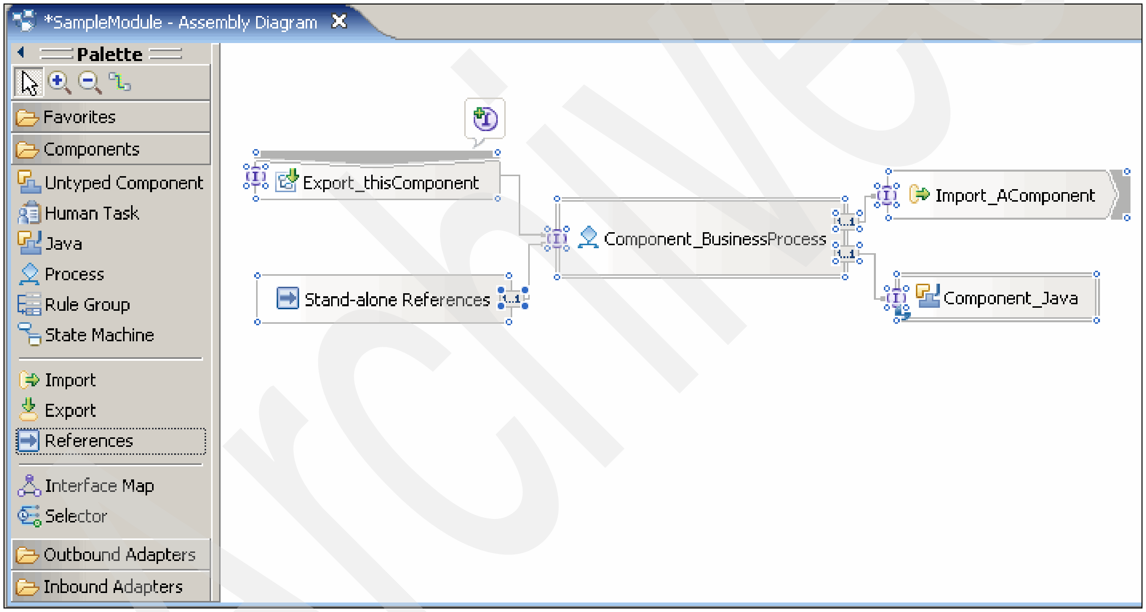


Figure 4-1 Assembly diagram of a sample module

In Figure 4-1, the main component (in the center) is implemented using a business process. The process has two references:

- The first invokes a Java component.
- The second invokes an external service through an import component.

The process is made available to clients through the export or stand-alone reference (on the left).

The service component's implementation contains the logic of the process. The various implementation types that are supported in WebSphere Integration Developer for components of a business integration module include:

- Java objects
- Business processes
- Human tasks
- State machines
- Rule groups (Business rules)
- Selectors
- Interface maps

In the assembly diagram, the component's implementation type is represented by an icon in the component. Figure 4-1 on page 129 shows all the implementation types listed with their symbols on the assembly editor palette.

In the remaining sections, we discuss how to build the important implementation types to implement the business logic.

4.3 Using Java objects

An implementation of a component in Java is referred to as a *Java object*. This implementation is one of the common implementations for a service component. This Java implementation type is sometimes termed as a *plain old Java object* or *POJO*. Generally, this implementation has a WSDL interface type, although this implementation can also have a Java interface.

When working with a Java object, the code remains hidden from you within the context of the editors. A POJO can also be used in a mediation module. So, it can be deployed to either a WebSphere Process Server or a WebSphere Enterprise Service Bus.

4.3.1 Creating a Java component

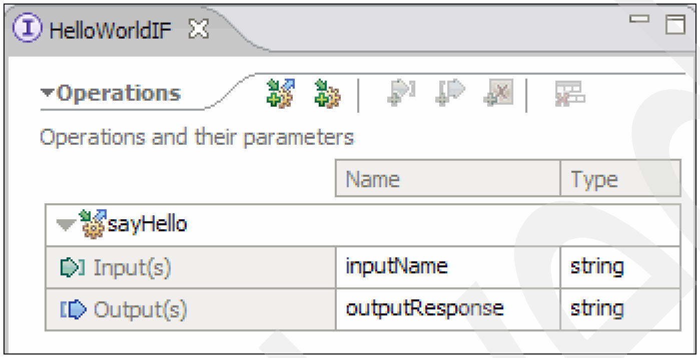
Follow any of these methods to create a new Java component:

- Click **Java** on the palette menu of the assembly editor, move the cursor to the canvas, and click again to drop a Java component into the assembly diagram.
- Drag a Java class onto the assembly editor canvas. A Java class becomes a Java component in the assembly diagram.
- Apply a Java implementation to a component that has no implementation type.

4.3.2 Creating a “HelloWorld” sample

Follow these steps to create a Java component and implement using a POJO:

1. Create a module, HelloWorldModule, and an interface, HelloWorldIF, in the module, that looks similar to the one shown in Figure 4-2.



	Name	Type
sayHello		
Input(s)	inputName	string
Output(s)	outputResponse	string

Figure 4-2 HelloWorldIF interface

2. Expand the HelloWorldModule, and then double-click **Assembly Diagram** to open it in assembly editor. Click **Java** on the palette menu of the assembly editor, move the cursor to the canvas, and click again to drop a Java component into the assembly diagram. Now, the assembly diagram looks similar to the one shown in Figure 4-3.

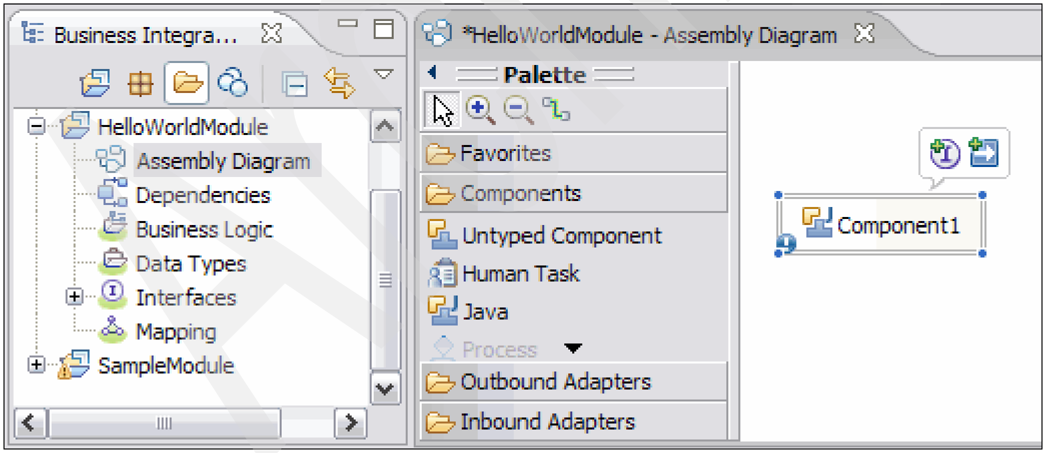
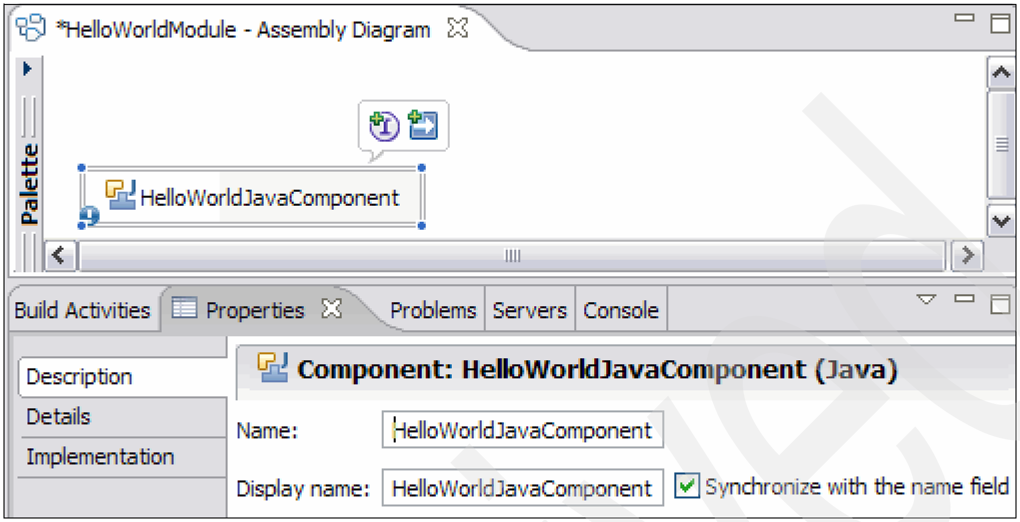


Figure 4-3 Java component on the assembly diagram

3. Use the Properties view to rename the component as `HelloWorldJavaComponent` as shown in Figure 4-4.



Component: HelloWorldJavaComponent (Java)

Name: HelloWorldJavaComponent

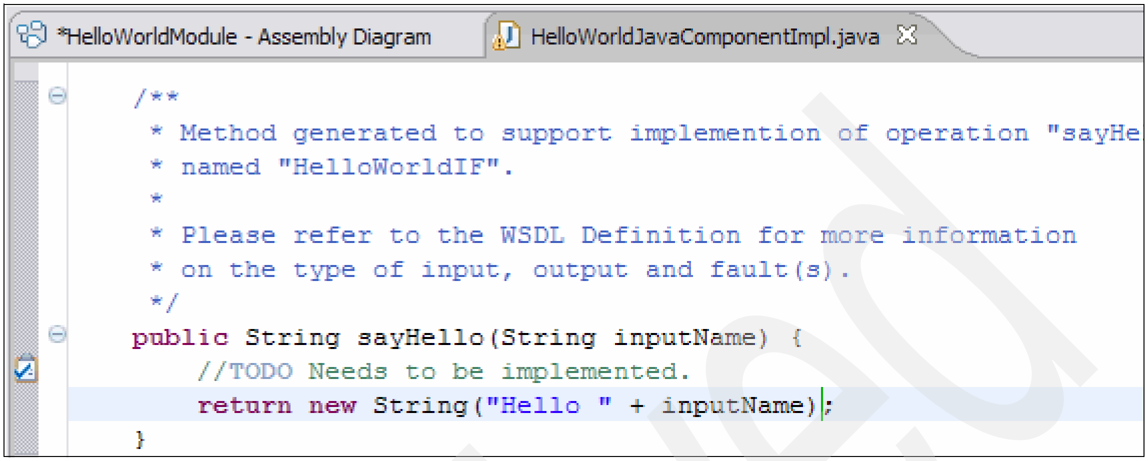
Display name: HelloWorldJavaComponent ☑ Synchronize with the name field

Figure 4-4 Renaming Java component

4. Add the `HelloWorldIF` interface to the component:
   a. Select the component.
   b. Right-click it, and select **Add** → **Interface**.
   c. Select **HelloWorldIF** from the Add Interface wizard, and click **OK**.
5. Add an implementation to the component:
   a. Click the component to select it, and select **Generate Implementation**.

      Alternatively double-click the component, and click **Yes** on the Open dialog box to implement it.
   b. Click **OK** on the Generate Implementation dialog box to use default package.

c. Implement the sayHello method in HelloWorldJavaComponentImpl.java as shown in the Figure 4-5.


*HelloWorldModule - Assembly Diagram
HelloWorldJavaComponentImpl.java

```
    /**
     * Method generated to support implemention of operation "sayHe
     * named "HelloWorldIF".
     *
     * Please refer to the WSDL Definition for more information
     * on the type of input, output and fault(s).
     */
    public String sayHello(String inputName) {
        //TODO Needs to be implemented.
        return new String("Hello " + inputName);
    }
```

Figure 4-5 Implementing Java component

6. Save the assembly diagram and all other resources. Now, the component looks similar to the one shown in Figure 4-6.

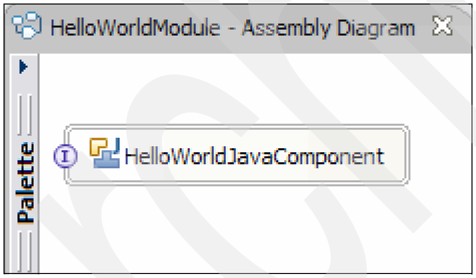


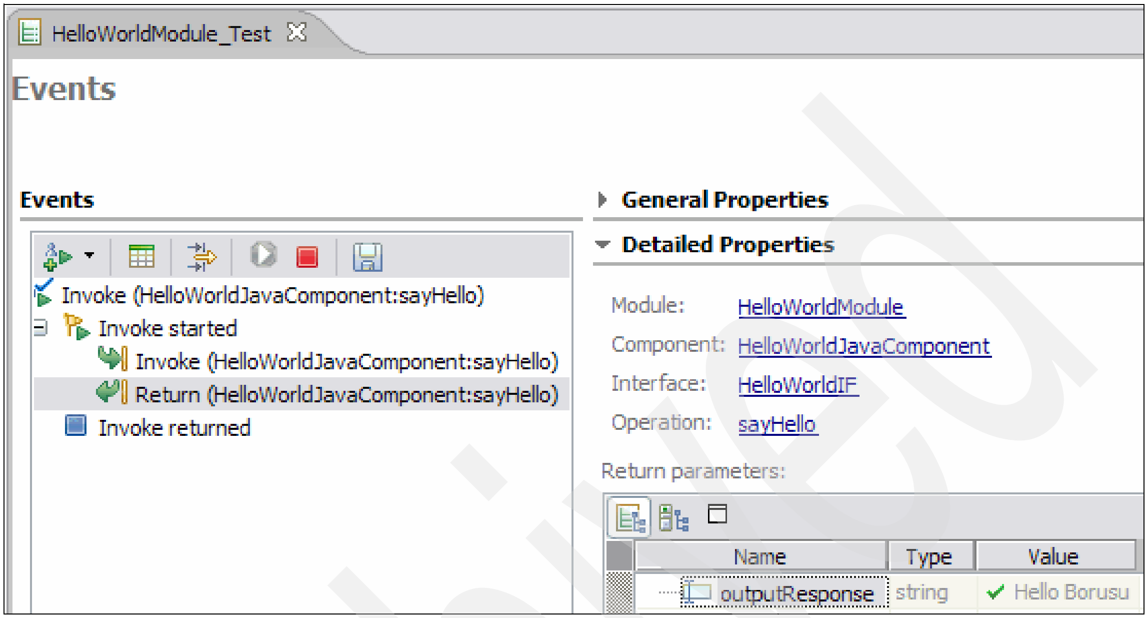
Figure 4-6 Implemented Java component in assembly diagram

7. Deploy the module by right-clicking WebSphere Process Server in the Servers view of WebSphere Integration Developer and by selecting **Add and Remove Projects**. Move the new application from the Available projects list to the Configured projects list, and click **Finish.**

8. Test the component in the integration test client by right-clicking **HelloWorldModule** in the Business Integration View and by selecting **Test** → **Test Module**. Supply a string value for the input (`Borusu` in this example).

9. In the Events area of the test client, a message is sent to the HelloWorldJavaComponent, and then the Java component returns a

response after a moment. In the Value column of the value editor, the text Hello Borusu is returned for the outputResponse, as shown in Figure 4-7.



Invoke (HelloWorldJavaComponent:sayHello)
Invoke started
Invoke (HelloWorldJavaComponent:sayHello)
Return (HelloWorldJavaComponent:sayHello)
Invoke returned

Module: HelloWorldModule
Component: HelloWorldJavaComponent
Interface: HelloWorldIF
Operation: sayHello

Return parameters:

Name	Type	Value
outputResponse	string	✓ Hello Borusu

Figure 4-7 Response from HelloWorldJavaComponent in integration test client

10. Now the test is complete and successful. Close the integration test client and click **No** when asked whether you want to save the changes.

4.4 Business processes

A *business process* consists of a series of activities or individual tasks that are executed in a specific order, sequentially or in parallel, to achieve a larger business goal.The business process editor is used to build business processes based on BPEL standards.

4.4.1 Types of business processes

Business processes can be either long-running processes or microflows. You select the type of process when you create it. You can change the type from the Properties view for the business process in the business process editor.

Long-running processes

A *long-running process* executes over an extended period of time and is asynchronous in nature. It is used most commonly with services that might not respond immediately, in particular human tasks. Interruptible business processes and asynchronous business processes are examples of long running processes.

When a business process is interruptible, it is long running, and execution stops at specific activities and does not continue until an appropriate action takes place.

Typically, components that are implemented as long running processes are called *asynchronous*, which means that a client calls them and then proceeds to do other work while waiting for a reply.

When a process is paused, it is waiting. You decide what it waits for. For example, you might decide that the process needs instruction from a human before continuing, or you might decide that it cannot proceed until it has specific input from a partner.

Microflows

A *microflow* is a non-interruptible business process and runs within a single transaction. Because it runs automatically from start to finish, it is ideal for situations where the user is expecting an immediate response and does not require the use of a human task. Microflows are an IBM extension to BPEL.

A microflow has the following characteristics:

- Runs in one transaction or activity session
- Normally runs for a short time
- Is in a transient state and, therefore, is not stored in the runtime database
- Typically invokes services synchronously
- Can have only non-interruptible child processes
- Cannot contain:
  - Human tasks
  - Wait activities
  - Non-initiating receive activities or pick activities

Microflows are faster than long-running processes. Thus, you need to use them when possible. For example, you can attempt to decompose a long-running process into multiple processes using microflows when possible.

4.4.2 New, enhanced business process features with WebSphere Process Server V6.1

This section describes the new or enhanced business process capabilities for business integration applications that are available with WebSphere Process Server V6.1.

> **Note:** We discuss some of these new features in detail in the subsequent sections.

The new or enhanced features include:

- Generic JMS interface for the Business Flow Manager allows for programmatic interaction with business process templates and instances (Figure 4-8).

  In WebSphere Process Server V6.0.2, there is support for Enterprise Java Bean (EJB) clients and Web Services clients to interact generically with business process templates and instances.

  In V6.1, this interface is also exposed to JMS clients. It allows custom JMS clients to:

  – Query and retrieve processes and activities
  – Invoke and send messages to processes
  – Life cycle operations (Restart/Delete/Suspend/Resume instances)
  – Repair a business process
  – Create and work with stored queries
  – Handle variable
  – Handle Input/Output/Fault

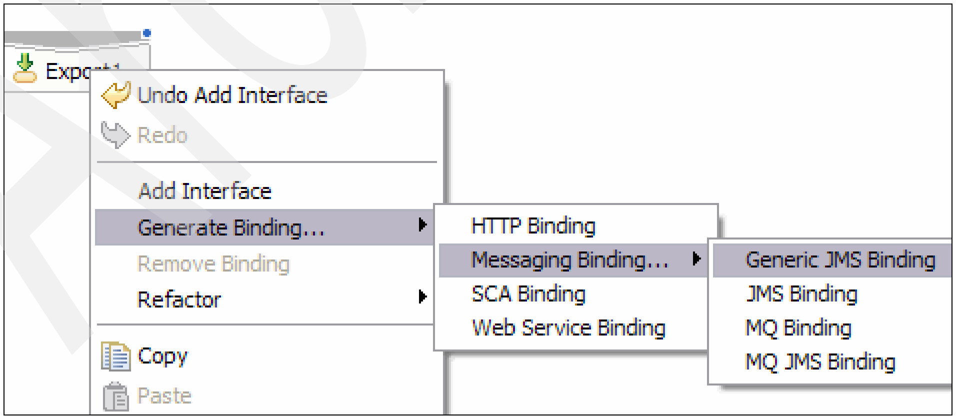


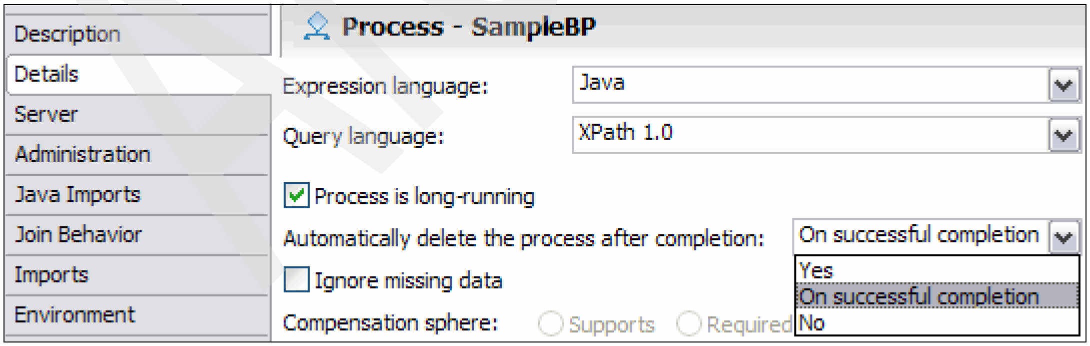
Figure 4-8 Generic JMS binding

- Extensions to the Generic Web Services Interface for Business Flow Manager and new runtime capabilities.
- The ForEach activity allows processing a dynamic number of multiple branches (either in parallel or serially).

  The ForEach activity is an iterator that repeats its contained activity a specified number of times, either serially or in parallel. It allows you to specify a completion condition to be used if not all branches are required to complete (for example, when parallel requests are sent out and a sufficiently large subset of the recipients respond). The remaining branches are terminated if the completion condition is met.
- Suspend capabilities are extended to allow specifying that process instances resume automatically.

  Suspending a business process instance means the navigation or execution of the process instance is put on hold. In all releases of WebSphere Process Server V6.0, processes can be suspended using a suspend operation, and are resumed explicitly calling a resume operation. In V6.1, suspending a process instance allows you to specify a duration or point in time if the process instance is to be resumed automatically.
- Automatic deletion can be restricted to processes that completed successfully, allowing you to keep only the process instances that need further analysis or repair (Figure 4-9).

  In all releases of WebSphere Process Server V6.0, the process-level attribute autoDelete specifies whether a long-running process instance is deleted automatically upon its completion. In V6.1, an additional option is introduced that allows to delete only successfully completed process instances automatically. This additional option enables you to keep only those process instances that might need further attention, for example those processes that need to be analyzed or even repaired and restarted.



Process - SampleBP

Expression language: Java

Query language: XPath 1.0

☑ Process is long-running

Automatically delete the process after completion: On successful completion

□ Ignore missing data

Compensation sphere: ○ Supports ○ Required

Figure 4-9 Automatic deletion on successful completion of a process

- Additional data-handling option ignores missing data during access instead of throwing faults.
- Back links in single-threaded flows are supported.

  In WebSphere Process Server V6.0.2, a flow cannot contain back links. Arbitrary cycles can be circumvented using (nested) loop constructs (while activities). In V6.1, graph-oriented modeling of business processes is enhanced by enabling arbitrary cycles (also known as *back links*).

  This capability is introduced by providing the cyclic flow activity, also known as *single threaded graph*. With arbitrary back links or cycles, it is possible to return to a prior activity in the flow. In particular, this capability is useful for human workflows to allow to go back to a prior step in the workflow.
- Support for wildcards and unconstrained content in interfaces, variables, and assign activities.
- Allow accessing an activity within a process.

  In WebSphere Process Server V6.1, an additional variant of method activityInstance is introduced to allow accessing of not only the current activity but also an activity that is identified using its name. It returns null if no activity is found.

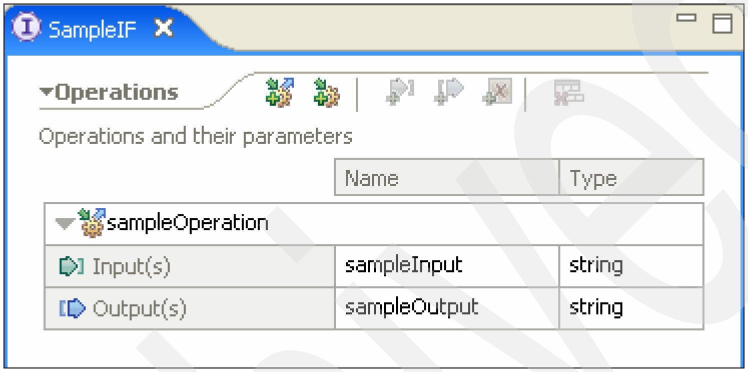
  If there is no activity with the specified name or if the name is not unique within the scope of the process, a StandardFaultException of type bpws:selectionFailure is thrown.

4.4.3 Creating a business process

Follow these steps to create a new business process:

1. Create a new module. In this example, the module name is *SampleModule*.
2. Create a new interface for the module. The interface, SampleIF, looks similar to that shown in Figure 4-10.

   We discuss how to create interfaces in 3.5.1, “Creating a new interface” on page 65.



	Name	Type
sampleOperation		
Input(s)	sampleInput	string
Output(s)	sampleOutput	string

Figure 4-10 SampleIF in Interface Editor

3. Select **File** → **New** → **Business Process**. In the New Business Process window, shown in Figure 4-11:

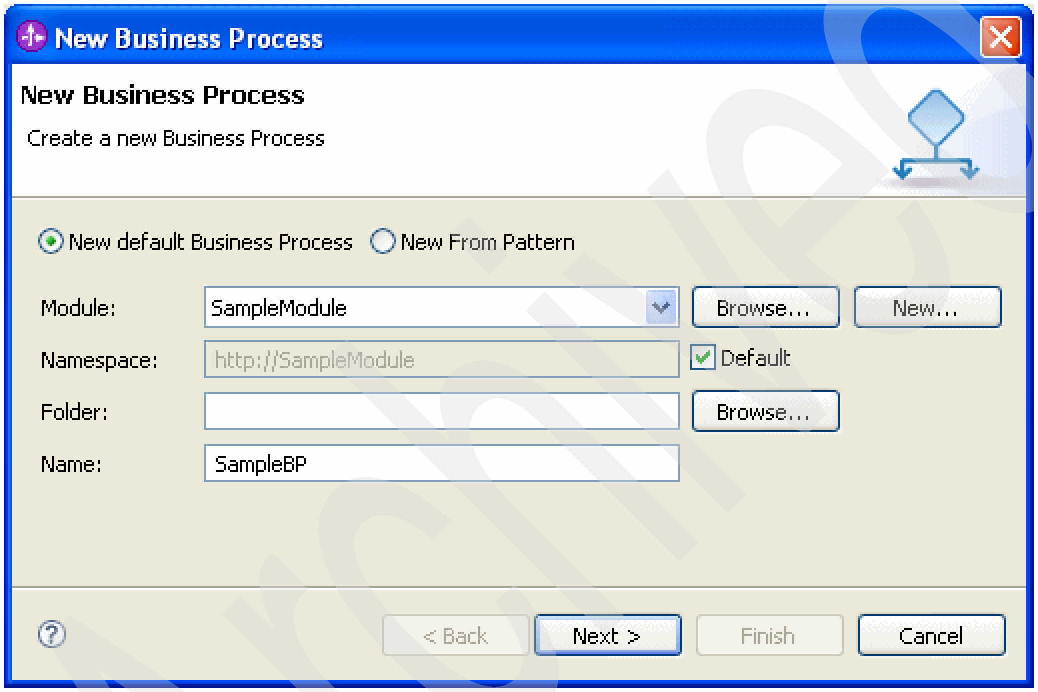
   a. Select **New default Business Process**.

   b. Browse to the module (or click **New** to create one).

   c. Specify a folder (optional).

   d. Specify a name for the new process (for example, `SampleBP`).

   e. Click **Next**.



New Business Process

Create a new Business Process

● New default Business Process ○ New From Pattern

Module: SampleModule

Namespace: http://SampleModule ☑ Default

Folder:

Name: SampleBP

Figure 4-11 New Business Process Wizard

4. Select a type for the business process, as shown in Figure 4-12.

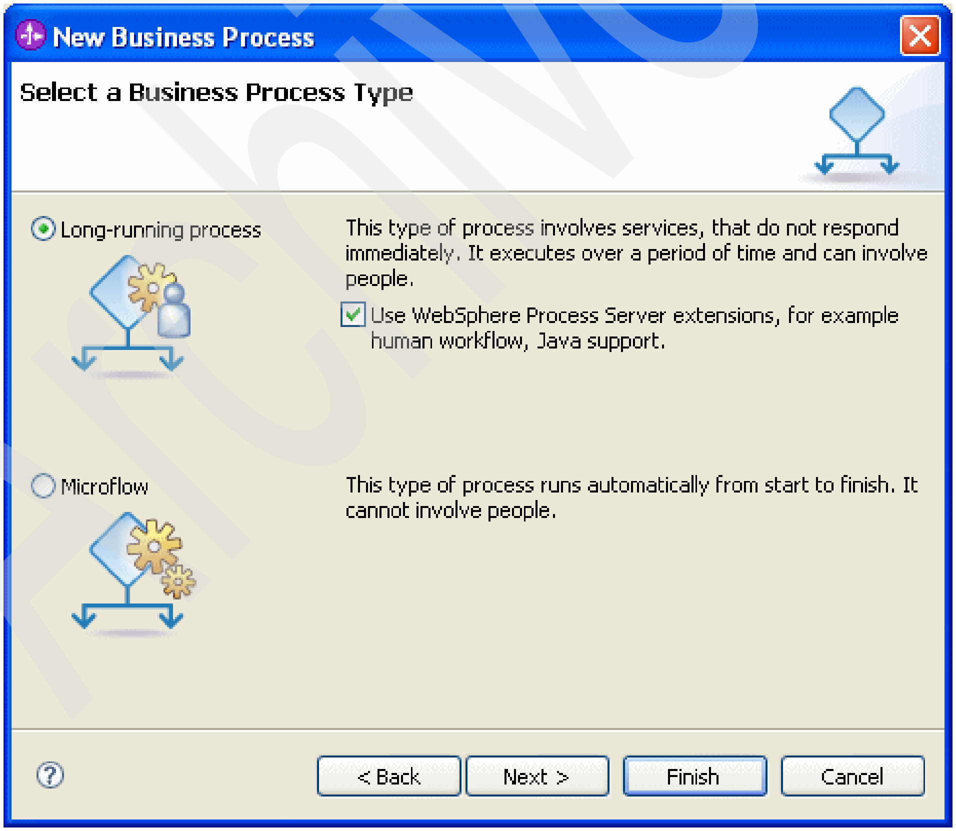
   You have the following process types:

   – *Long-running process*: Executes over an extended period of time and is asynchronous in nature. It is used most commonly with services that might not respond immediately, in particular human tasks.

   – *Microflow*: A non-interruptible business process that runs within a single transaction. Because it runs from start to finish automatically, a microflow process is ideal for situations where the user is expecting an immediate response and does not require the use of a human task.

   WebSphere Integration Developer enhances BPEL capabilities through the use of extensions. If you want to create a process without the use of these extensions, then disable the “Use WebSphere Process server extensions” option. This option is not applicable to microflows because they are a BPEL extension.

   Click **Next**.



New Business Process

Select a Business Process Type

● Long-running process

This type of process involves services, that do not respond immediately. It executes over a period of time and can involve people.

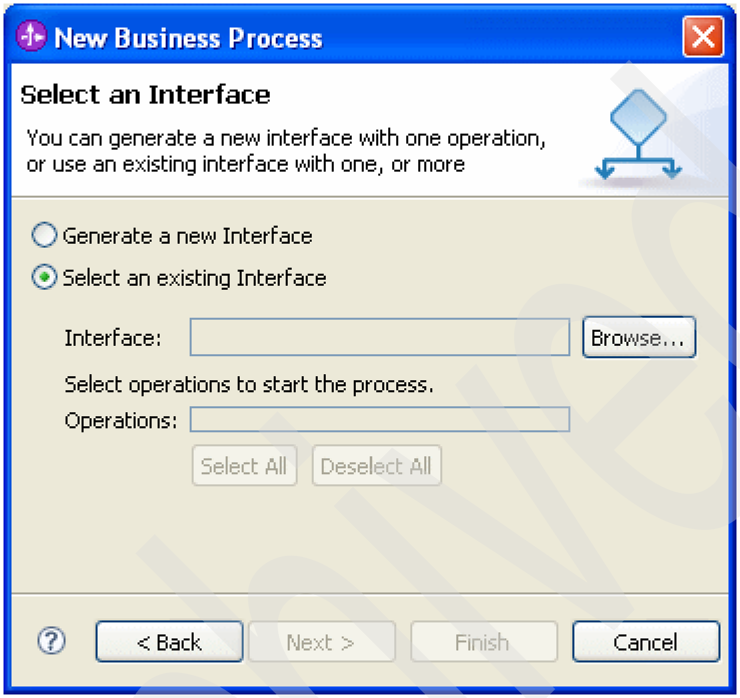
☑ Use WebSphere Process Server extensions, for example human workflow, Java support.

○ Microflow

This type of process runs automatically from start to finish. It cannot involve people.

Figure 4-12 Select business process type in the New Business Process wizard

5. Click **Select an existing Interface** as shown in Figure 4-13. Alternatively, you can select **Generate a new Interface** to generate a new interface to be used with this process.



New Business Process

Select an Interface

You can generate a new interface with one operation, or use an existing interface with one, or more

○ Generate a new Interface

● Select an existing Interface

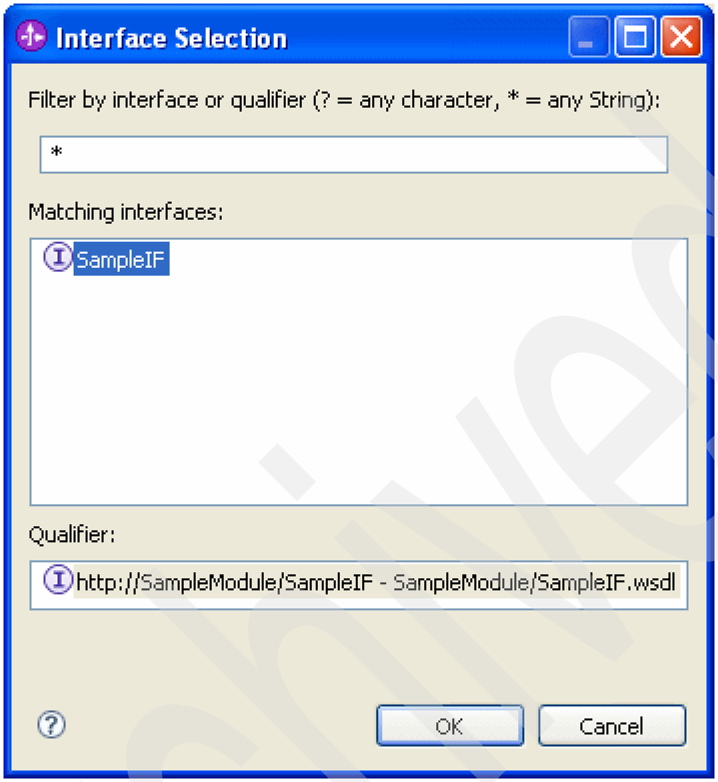
Interface:

Select operations to start the process.

Operations:

Figure 4-13 Select an interface in the New Business Process wizard

6. To select an existing interface, click **Browse** and select the interface as shown in Figure 4-14. Then, click **OK**.



Interface Selection

Filter by interface or qualifier (? = any character, * = any String):

*

Matching interfaces:

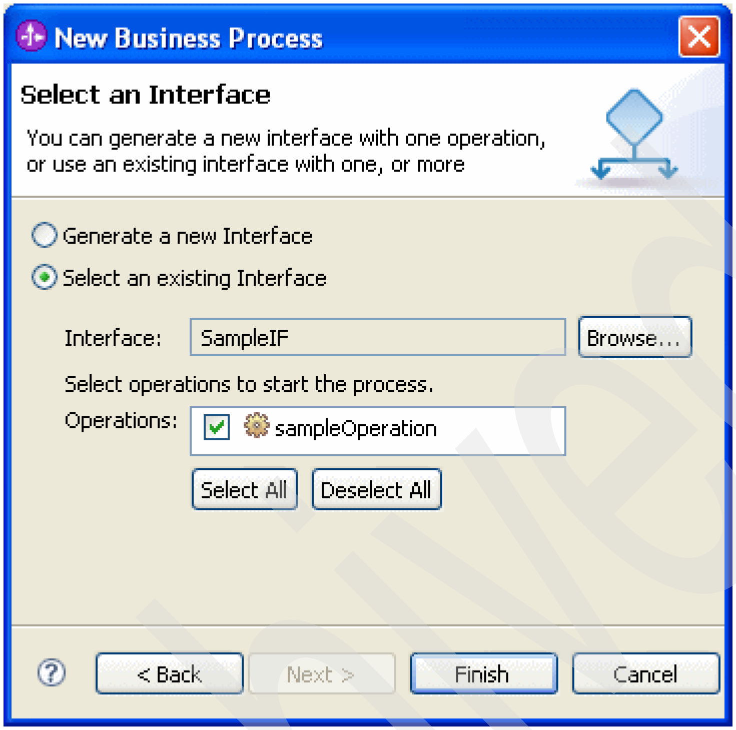
SampleIF

Qualifier:

http://SampleModule/SampleIF - SampleModule/SampleIF.wsdl

Figure 4-14 Interface Selection wizard

7. Click **Finish** (Figure 4-15).



New Business Process

Select an Interface

You can generate a new interface with one operation, or use an existing interface with one, or more

○ Generate a new Interface

● Select an existing Interface

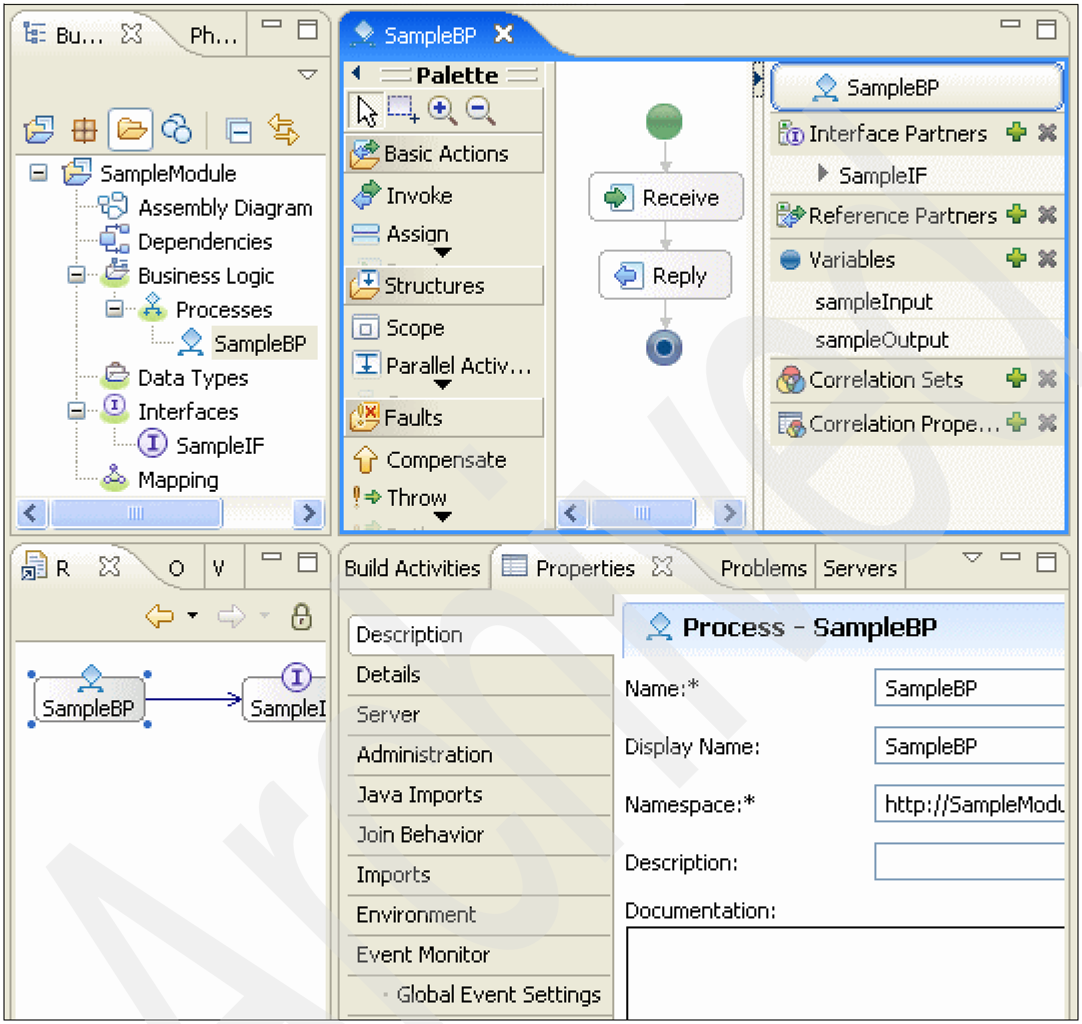
Interface: SampleIF

Select operations to start the process.

Operations: ☑ sampleOperation

Figure 4-15 Selecting an existing interface on New Business Process wizard

The business process is created and opens in the business process editor as shown in Figure 4-16.



Name:* SampleBP

Display Name: SampleBP

Namespace:* http://SampleModu

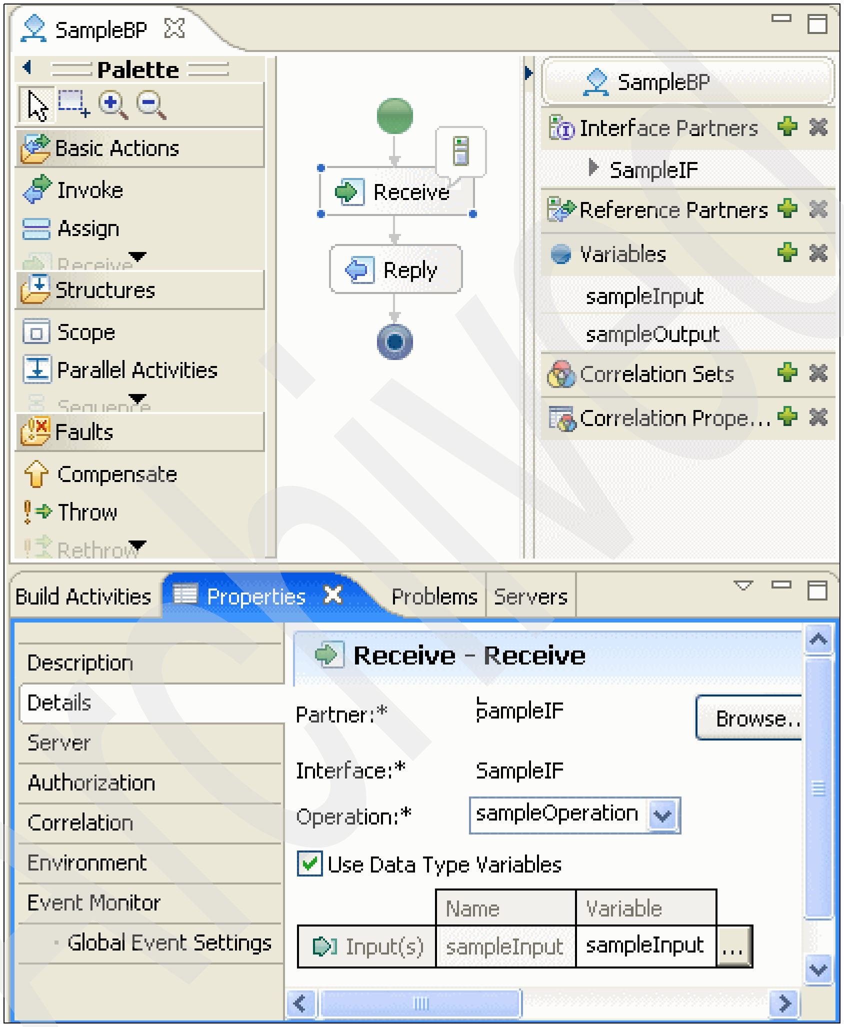
Description:

Documentation:

Figure 4-16 SampleBP Business Integration View

4.4.4 Business process editor

The business process editor allows you to visually build and manipulate business processes. The editor consists of several distinct areas, including the palette, canvas, action bar, tray, and Properties view, as shown in Figure 4-17.



Receive - Receive

Partner:* SampleIF

Interface:* SampleIF

Operation:* sampleOperation

☑ Use Data Type Variables

	Name	Variable	
Input(s)	sampleInput	sampleInput	...

Figure 4-17 The Business Process Editor

The palette

The *palette* is the shaded area to the left of the canvas that houses the objects that can be dropped onto the canvas to build the process. The icons are categorized under several different headings. Click the heading once, and the icons remain hidden until you click that heading again.

To increase the size of the icons, right-click the palette and select **Use Large Icons**.

The canvas

The *canvas* is the white area in the middle of the editor that is used to assemble the activities to build the business process. When you click and drag an activity from the palette onto the canvas, the icon beside the cursor has a plus symbol, and you can decide where to drop the activity. When the cursor becomes a crossed out circle, continue moving the cursor until it becomes a plus sign again.

The action bar

The *action bar* is a miniature dialog box that opens beside certain activities when you select them. It contains a series of one or more icons that represent the actions that are relevant to that activity. For example, when you select the Receive activity, the **Set Partner** icon displays as shown in Figure 4-17 on page 146.

The tray

The *tray* displays the interface partners, reference partners, variables, correlation sets, and correlation properties that are associated with the process.

To see the interfaces and operations associated with the partners, click the small gray arrow (▸) beside the partner’s name. To create a new partner, variable, correlation set or correlation properties, click the corresponding green plus ✚ icon, or to remove one, highlight it and click the red *X* ✖ icon.

The Properties view

This area of the business process editor displays properties of the object that are currently selected on the canvas or the tray. Click the tabs to the left of this view to toggle through various property pages. Properties marked with an asterisk (*) are mandatory.

4.4.5 Building blocks of a business process

Several types of building blocks are used to compose a business process using the business process editor:

- *Activities*: The individual business tasks within the process.
- *Correlations*: The records that are used to keep track of multiple partners and messages and to correlate them with a specific instance of a business process at any point of time.
- *Elements*: The objects that are used to support or configure activities.
- *Handlers*: Consist of a group of activities that represent a specific course of action that is associated with a particular activity. A handler runs when certain situations occur within the parent activity (that is, the activity with which the handler is associated).
- *Partners*: The external parties (users or services) that interact with the business process. Within a process, the term *partner* describes the other services that might be calling the interfaces or that it might be calling.
- *Variables*: Store the data that is used within a business process. A variable belongs to the scope in which it is declared. When you create a variable, you need to declare what type it is before using it.

4.4.6 Using partners in a business process

The two types of partners that are available for use in a business process include:

- *Interface partners*: A direct link to the interface where the partner is configured. An interface partner is the process interface and exposes operations that can be called by external partners.
- *Reference partners*: The reference is not an interface but instead tells the process where to find operations that it can invoke. More to the point, it specifies the interface that is used in the invocation of another component.

When a client calls your process, that call comes from an interface partner. When the process calls another service, it does so using a reference partner. So, an interface partner contains incoming operations and a reference partner performs the outgoing operations.

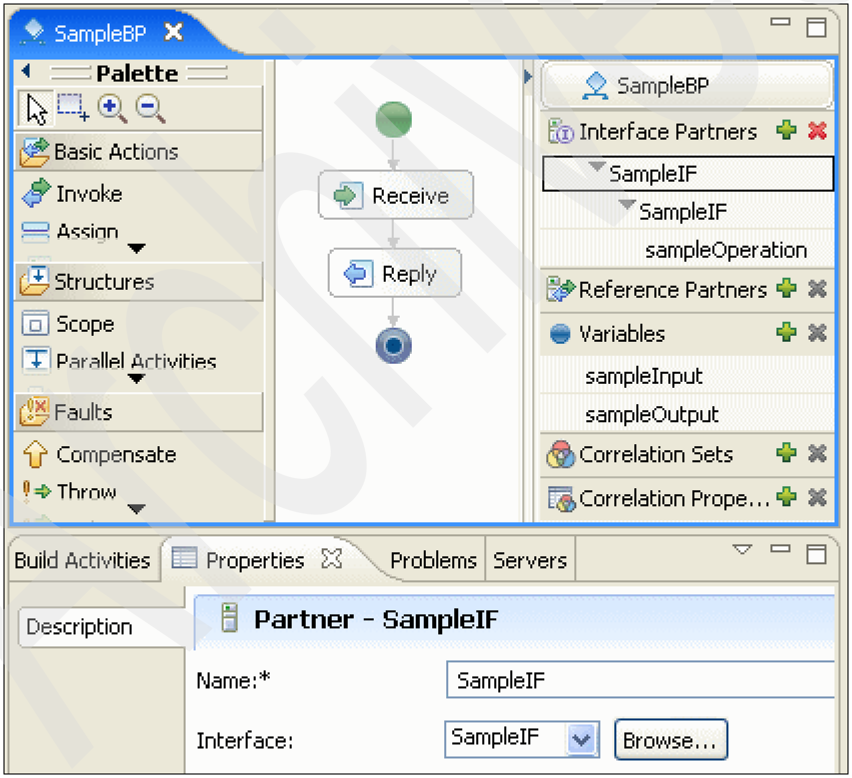
Adding a partner to a business process

To create a partner in a business process using the business process editor:

1. Click the plus icon ✚ besides the partner's (**Interface Partners** or **Reference Partners**) section on the tray.
2. Specify an appropriate name for the partner.
3. Click the Description tab in the properties area.
4. Select an interface from the drop-down list, or click **Browse** to choose an interface with the Interface Selection wizard.

When a business process is created, the interface partner that is associated with the Receive activity is created automatically. If you need any additional interface partners, you can add them by following the steps described here.

Figure 4-18 shows the interface partner, `SampleIF`, on the tray and its properties.



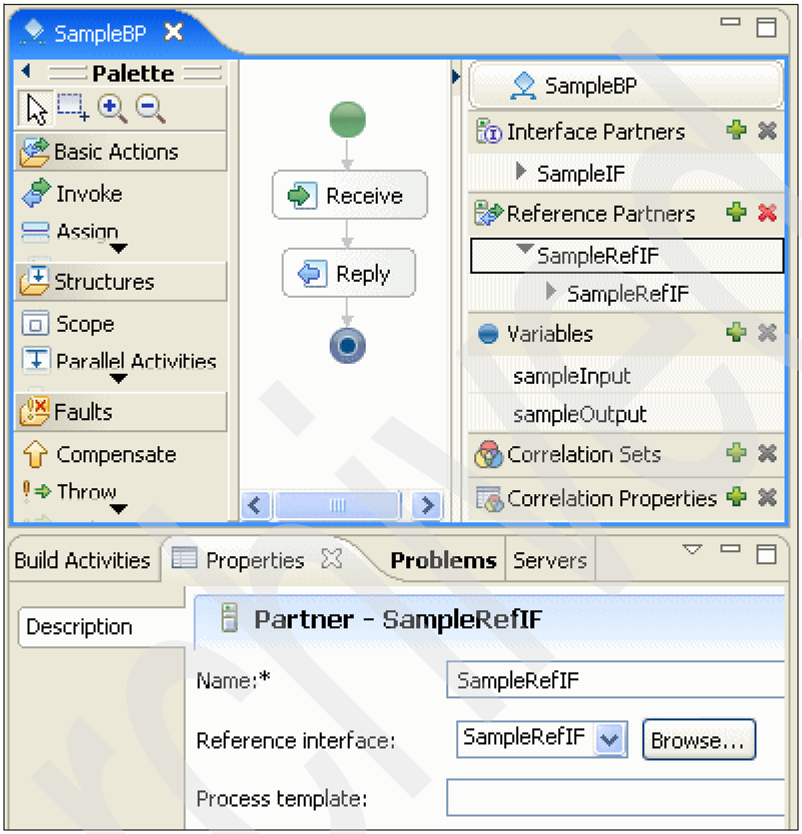
Partner - SampleIF

Name:* SampleIF

Interface: SampleIF

Figure 4-18 Properties of an interface Partner

You can create a reference partner following these same steps. Alternatively, you can select an interface from the Business Integration view and drag it onto the canvas. Figure 4-19 shows a reference partner, `SampleReferenceIF`, on the tray and its properties, which were created using this approach.



Partner - SampleRefIF

Name:* SampleRefIF

Reference interface: SampleRefIF

Process template:

Figure 4-19 Properties of a reference partner

Use the Process template field to invoke another process from this process. The connection is resolved dynamically in the runtime environment. The calling process always picks up the current valid version of the process that it is invoking. In addition, if this is a subprocess binding, the called process is subject to life cycle operations of the calling process. For example, when the parent process is terminated, it will terminate the child process, too.

4.4.7 Using variables in a business process

Variables store the data that is used within a business process. The two types of variables that are available to use in a business process include:

- *Data type variables*: Can be either a business object or a simple type, such as string or integer.
- *Interface variables*: Uses either an input or output parameter as defined within an interface.

A variable belongs to the scope in which it is declared. If it is created in the global process scope, then it is a global variable and, thus, is visible to all objects and activities within the process. Those variables that are created within nested scopes are called *scoped* or *local variables* and are visible only to the objects within the scope in which they were declared.

Adding a variable to a business process

To create a variable in a business process using the business process editor:

1. To create a global variable, click a blank area of the canvas or any other activity that is not a scope.
2. Then, click the plus icon (✚) beside the Variables section on the tray. Alternatively, you can right-click in the Variables section and select **Add Variable**.
3. Specify an appropriate name for the variable.
4. Click the Description tab in the properties area.
5. Select the type for the variable by selecting either **Data type variable** or **Interface variable**.

   If you select **Data type variable**, click **Browse** to choose a type or business object from the Data Type Selection wizard.

   If you select **Interface variable**, follow these steps:

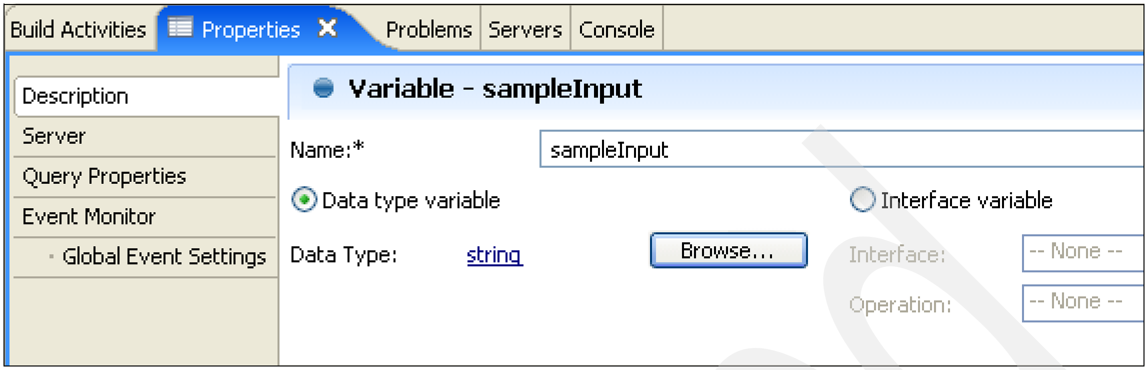
   a. Select an interface from the drop-down list, or click **Browse** to choose one with the Interface Selection wizard.

   b. Select an operation from the drop-down list.

   c. Select a direction by selecting **Input** or **Output**.

Note: Unlike Data type variables, when you use an Interface variable to send data to a partner, the variable must be initialized before sending it. You can use an Assign activity or a Java snippet for this purpose.

Figure 4-20 shows the properties of sampleInput variable.



Variable - sampleInput

Name:* sampleInput

● Data type variable ○ Interface variable

Data Type: string

Interface: -- None --

Operation: -- None --

Figure 4-20 Properties of sampleInput variable

When you create a business process, global variables are created automatically. These variables are associated with the corresponding input and output parameters of the operation that is associated with the respective Receive and Reply activities of this business process. (Refer to Figure 4-23 on page 155 and Figure 4-24 on page 156.)

4.4.8 Using activities in a business process

The various categories of activities that are available on the palette of the business process editor, as shown in Figure 4-21, include:

- *Basic Actions*: Primitive activities that do not contain other activities. These activities represent an individual task within a business process.
- *Structures*: Define the order in which a collection of activities takes place. Structured activities are comprised of one or more basic activities and are used to express control patterns, data flow, and coordination of message exchanges between process instances.
- *Faults*: Anticipated errors that can occur in situations that arise which prevent a business process from reaching completion. Fault activities deal with these cases.

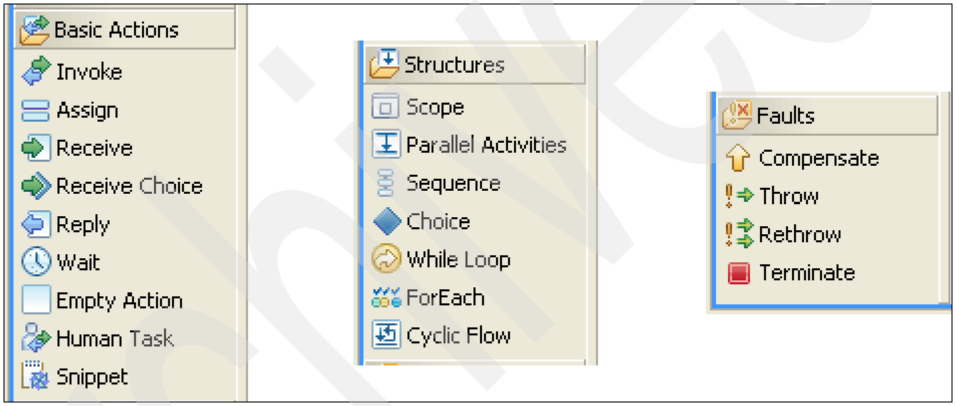


Figure 4-21 Snippets of Palette Menu of Business Process Editor

Adding an activity to a business process

To add an activity to a business process using the business process editor:

1. Click an activity icon on the palette and move the cursor onto the canvas.

   The icon beside the cursor has a plus symbol when the cursor is at a place where you are allowed to drop the activity. When the cursor becomes a crossed out circle, continue moving the cursor until it becomes a plus sign again. If there are already activities on the path, then a black line displays as you hover over the path showing where you can drop this new activity (as shown in Figure 4-22 on page 154).

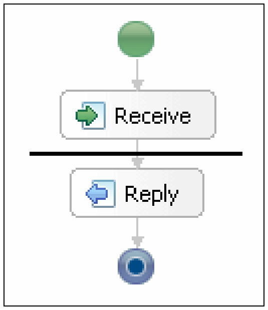


Figure 4-22 Adding an activity to a business process

2. Click the area of the canvas where you want to drop the activity.
3. If the activity added is a structured activity (that contains other activities), you can expand or collapse it by clicking the plus (⊞) or minus (⊟) icons or by double-clicking the structured activity itself.
4. Configure the activity as necessary in the properties area of the business process editor.

Receive activity

The Receive activity allows a business process to wait for a message (external input) to arrive and channels it into the process. It can have one or more associated reply activities if it is used in request-response operations.

A Receive activity is an entry point to a process; it is the point where the process starts or continues. You can associate a specific operation of an interface to a Receive activity using an interface partner. Figure 4-23 on page 155 shows the properties of a sample Receive activity.

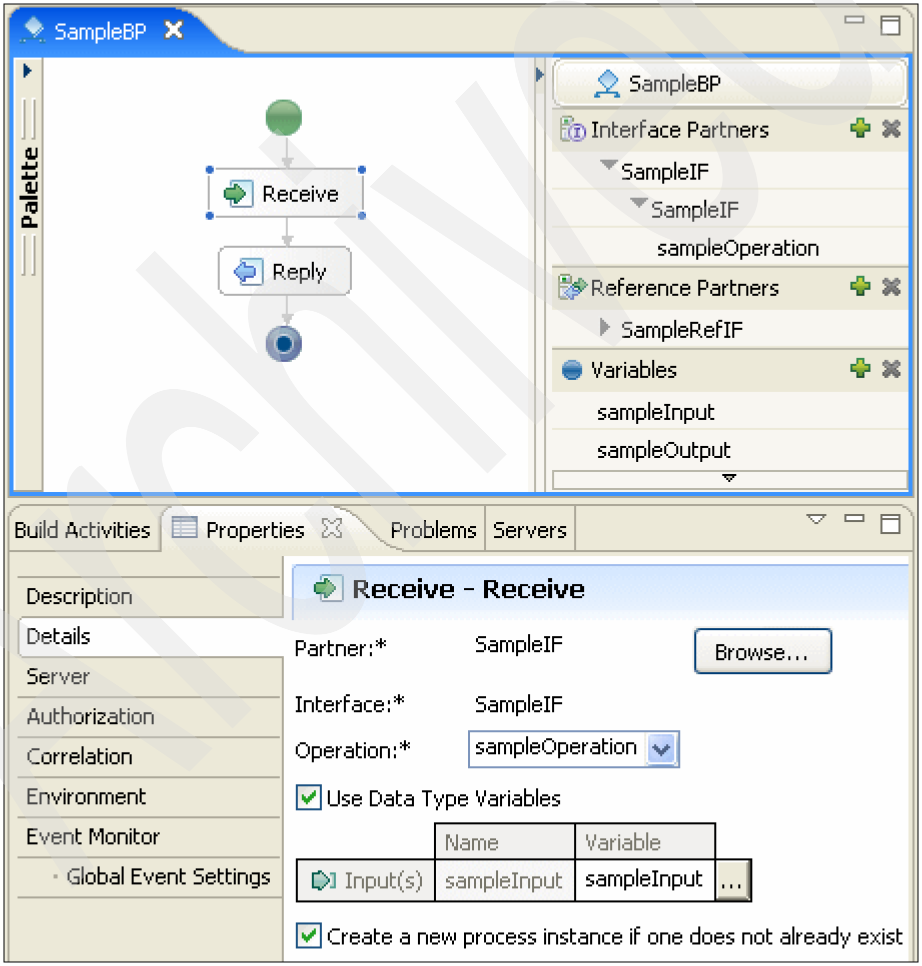
When a call is made to this process's operation, the corresponding Receive activity accepts the call, and the process continues running from there. A Receive activity can also occur in the middle of a business process. In this case, if the process encounters a Receive activity while it is running, the process stops and waits for the corresponding operation to be invoked.

When you create a business process, Receive and Reply activities are created automatically and are associated with the operation of the interface for the process. The interface partners and global variables are also created automatically. (Refer to Figure 4-23 on page 155 and Figure 4-24 on page 156.)

Details tab of the Properties view of Receive activity

The Details tab of the Properties view allows you to choose the interface partner and the operation. The table that displays below the "Use Data Type Variables" options shows all of the inputs of the selected operation. If you want to use data type variables, select one from the list, or clear the option to use interface type variables. Then, browse to the appropriate interface variables with input direction. We discuss the interface variables in 4.4.7, “Using variables in a business process” on page 151.

The “Create a new process instance if one does not already exist” option determines how to proceed in cases where an instance of this process has not yet been created in the runtime environment. Enabling this option allows the creation of a new instance of the process. Clearing it denies such authority. There must be one receive-type activity or element in the process that has this setting enabled, usually it is the first receive of the process.



Receive - Receive

Partner:* SampleIF

Interface:* SampleIF

Operation:* sampleOperation

☑ Use Data Type Variables

	Name	Variable
Input(s)	sampleInput	sampleInput

☑ Create a new process instance if one does not already exist

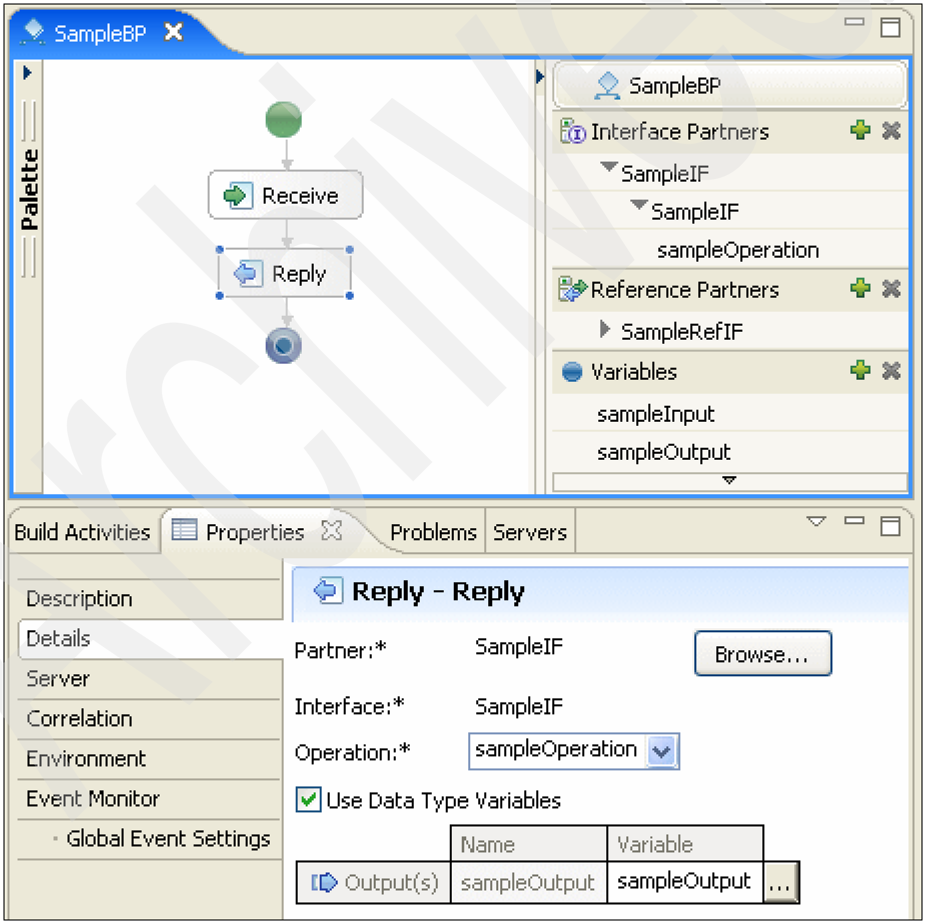
Figure 4-23 Receive activity of a business process

Reply activity

When a Receive activity belongs to a request-response operation, a Reply activity returns the output of the operation. The Reply activity specifies the same partner implementation as the corresponding Receive activity. A reply is always sent to the same partner from which a message was received previously.

A Reply activity does not necessarily need to be at the end of the process. A process can start with a Receive activity and then return a response before proceeding to do other work. You can have more than one Reply activity for each Receive activity, such as in the case where a process has multiple paths. The idea is that when another component calls a request-response operation of a process' interface, it needs to eventually get a response for that operation.

Figure 4-24 shows the properties of a sample Reply activity.



Reply - Reply

Partner:* SampleIF

Interface:* SampleIF

Operation:* sampleOperation

☑ Use Data Type Variables

	Name	Variable
Output(s)	sampleOutput	sampleOutput

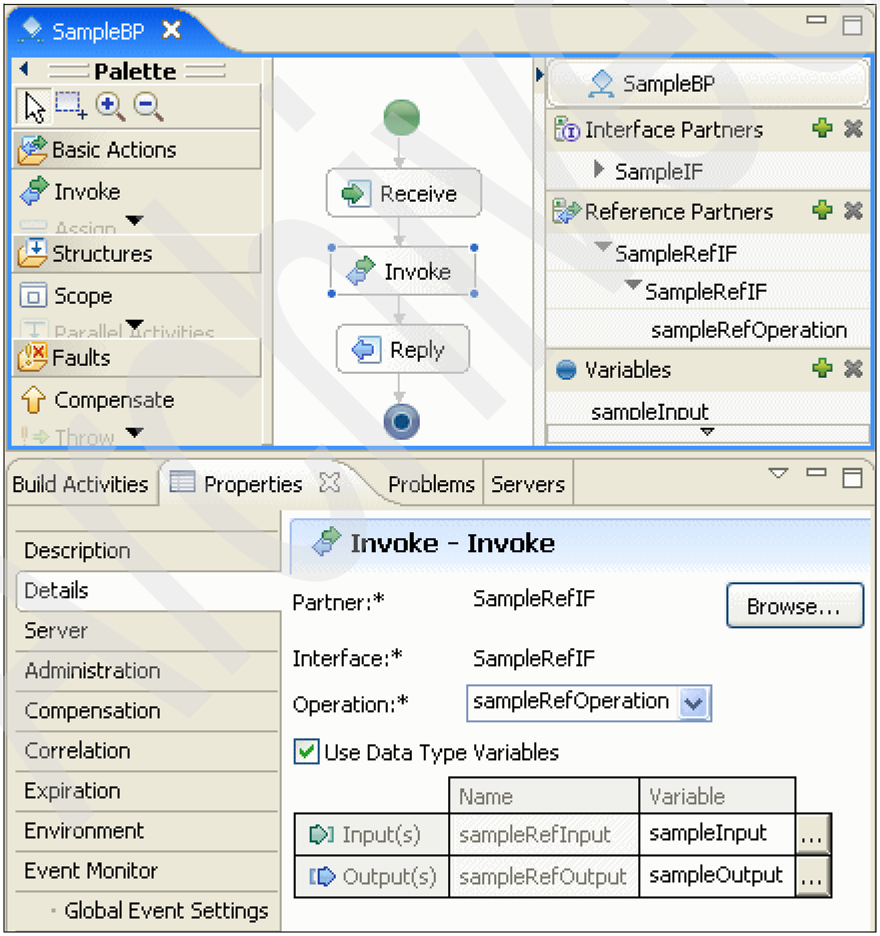
Figure 4-24 Reply activity of a business process

Invoke activity

The Invoke activity allows a business process to call a one-way operation (asynchronous) or request-response operation (synchronous) on a specific partner. The components and artifacts that can be invoked include:

- A Web service
- A Java class
- An EJB
- Another process
- Another SCA component

Figure 4-25 shows the implementation details of an Invoke activity, where the activity, labelled *Invoke*, is configured to invoke the sampleRefOperation of the reference partner, SampleRefIF.



Invoke - Invoke

Partner:* SampleRefIF

Interface:* SampleRefIF

Operation:* sampleRefOperation

☑ Use Data Type Variables

	Name	Variable
Input(s)	sampleRefInput	sampleInput
Output(s)	sampleRefOutput	sampleOutput

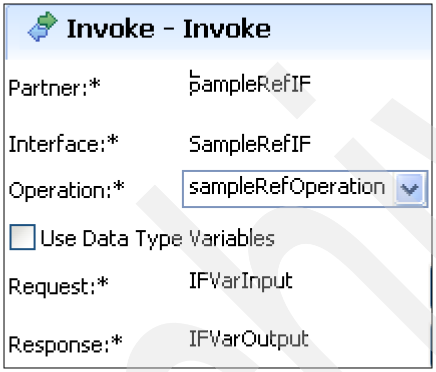
Figure 4-25 Properties of Invoke activity

Assign activity

The Assign activity allows basic data manipulation through the use of expressions to map service endpoint references to or from partner links or to copy some form of information from one part of your process to another. For example, the Assign activity can used to copy values from one variable to another or to initialize variables.

The following example demonstrates one of the usages of the Assign activity. In this example, two Assign activities are used to populate request and response interface type variables:

1. Create a new business process as shown in Figure 4-25 on page 157.
2. Configure the properties of Invoke activity to use interface variables as shown in Figure 4-26.



Invoke - Invoke

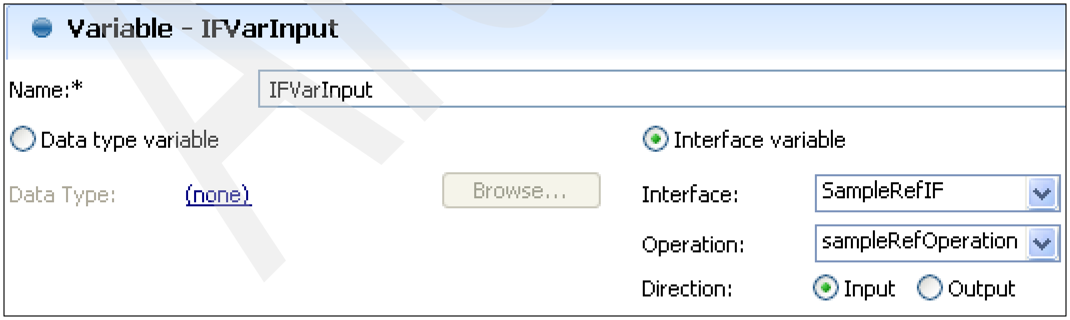
Partner:*	SampleRefIF
Interface:*	SampleRefIF
Operation:*	sampleRefOperation

□ Use Data Type Variables

Request:*	IFVarInput
Response:*	IFVarOutput

Figure 4-26 Properties of SampleRefIF invoke activity

You need to create two interface variables called *IFVarInput* and *IFVarOutput*. Figure 4-27 shows the properties of IFVarInput.



Variable - IFVarInput

Name:* IFVarInput

○ Data type variable

Data Type: (none)

● Interface variable

Interface:	SampleRefIF
Operation:	sampleRefOperation
Direction:	● Input ○ Output

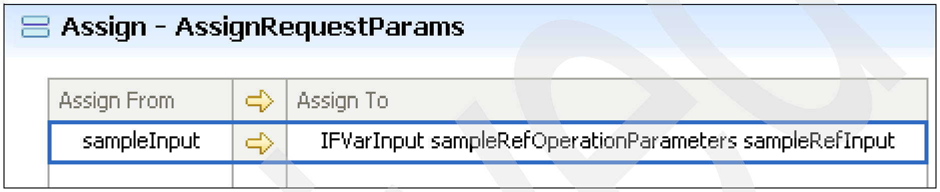
Figure 4-27 Properties of an interface variable, IFVarInput

3. Add an Assign activity between the existing Receive and Invoke activities, and change its properties as follows:

   a. Select the Assign activity and go to the Description tab of the Properties view. Change both the Name and Display Name fields to `AssignRequestParams`.

   b. Select the Details tab, and click **Select From** under the Assign From column. Select **sampleInput** by clicking it in the content assist dialog box.

   c. Click **Select To** under the Assign To column and expand IFVarInput on the content assist dialog box. Select **sampleRefInput**. Figure 4-28 shows the results.

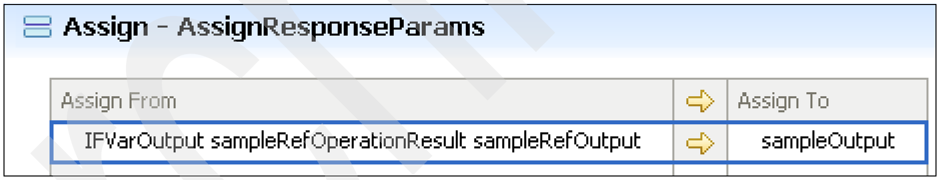


Assign - AssignRequestParams

Assign From	⇨	Assign To
sampleInput	⇨	IFVarInput sampleRefOperationParameters sampleRefInput

Figure 4-28 Properties of AssignRequestParams, an Assign activity

4. Add one more Assign activity between the existing Invoke and Reply activities to assign the response from Invoke to the response to be send back by Reply. Configure its properties as shown in Figure 4-29.



Assign - AssignResponseParams

Assign From	⇨	Assign To
IFVarOutput sampleRefOperationResult sampleRefOutput	⇨	sampleOutput

Figure 4-29 Properties of AssignResponseParams, an Assign activity

Your completed business process looks similar to the one shown in Figure 4-30.

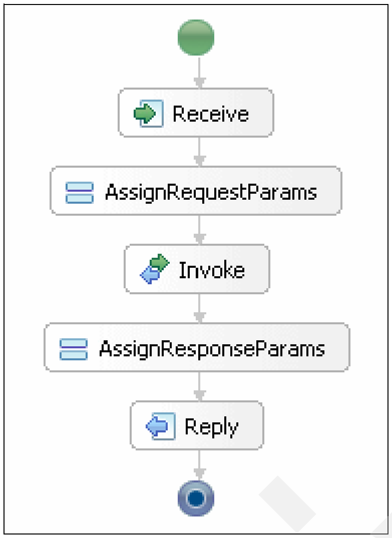


Figure 4-30 Assign example business process

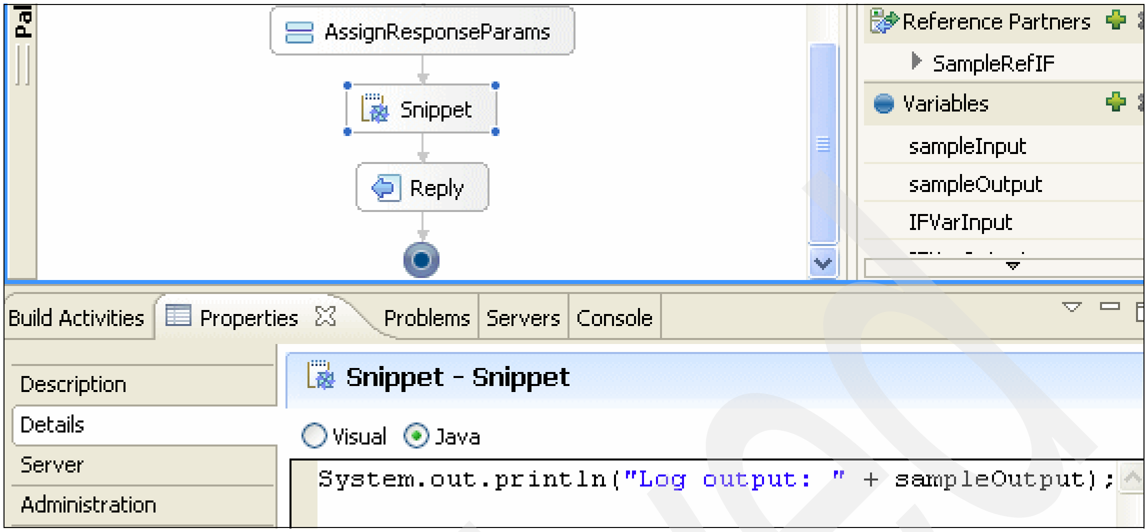
Human Task activity

A Human Task activity is used when work is to be performed by a person. This activity sends a process-related task out to a human for completion. It is referred to as *inline* because the task is implemented within a business process. We discuss human tasks in more detail in 4.5, “Human tasks” on page 184.

Snippet activity

The Snippet activity allows you to implement custom behavior into a business process. You implement the task of this activity using a Java programming snippet. You can either write a Java code or use the built-in graphical editor, *visual snippet editor*, to compose visual expressions and snippets that can be generated into valid Java code that can be used internally in the run time.

As shown in Figure 4-31 on page 161, you type in the Java code snippets in the Details tab of snippet’s properties by selecting the Java editor and by clicking Java radio button.



○ Visual ● Java

```
System.out.println("Log output: " + sampleOutput);
```

Figure 4-31 Properties of a Snippet activity

For a more detailed discussion about using the visual snippet editor to compose Java code, refer to:

- *Customizing behavior with visual snippets*

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.610.help.activity.ui.doc/topics/cactint.html

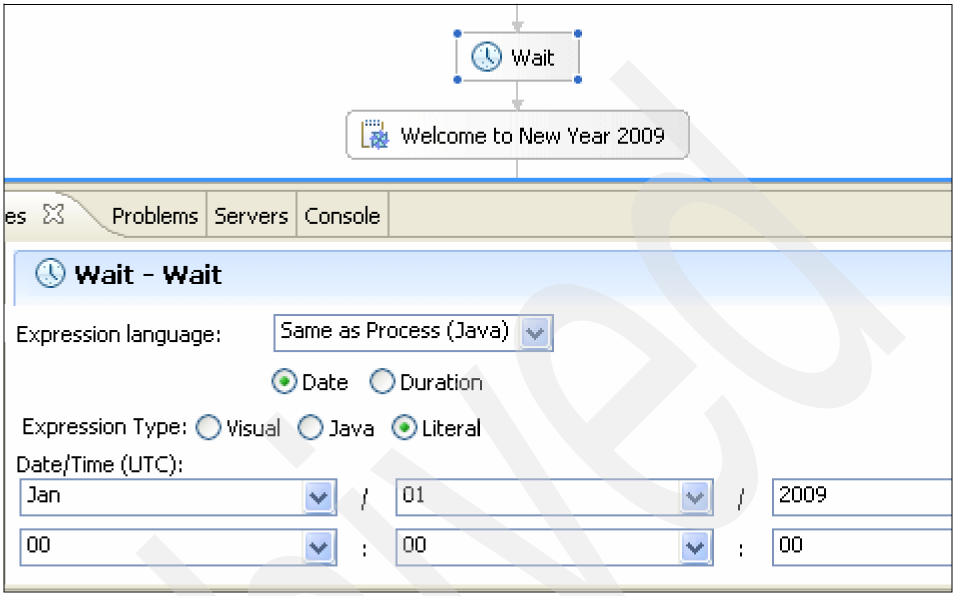
- IBM WebSphere Developer Technical Journal: *A guided tour of WebSphere Integration Developer—Part 4, Unleashing visual snippets and business state machines in your service-oriented application*

  http://www.ibm.com/developerworks/websphere/techjournal/0606_gregory/0606_gregory.html#VSE

Wait activity

The Wait activity pauses the process execution for a specified period of time. You configure this activity either by specifying a duration to wait, or by specifying a specific time and date when the process should continue to run. You can even use the visual snippet editor to compute the duration to wait.

Figure 4-32 shows a Wait activity configured to wait until 01 January 2009, 00 a.m., to proceed executing the next activity, which is named *Welcome to New Year 2009*.



Expression language: Same as Process (Java)

● Date ○ Duration

Expression Type: ○ Visual ○ Java ● Literal

Date/Time (UTC):

Jan / 01 / 2009

00 : 00 : 00

Figure 4-32 Properties of a Wait activity

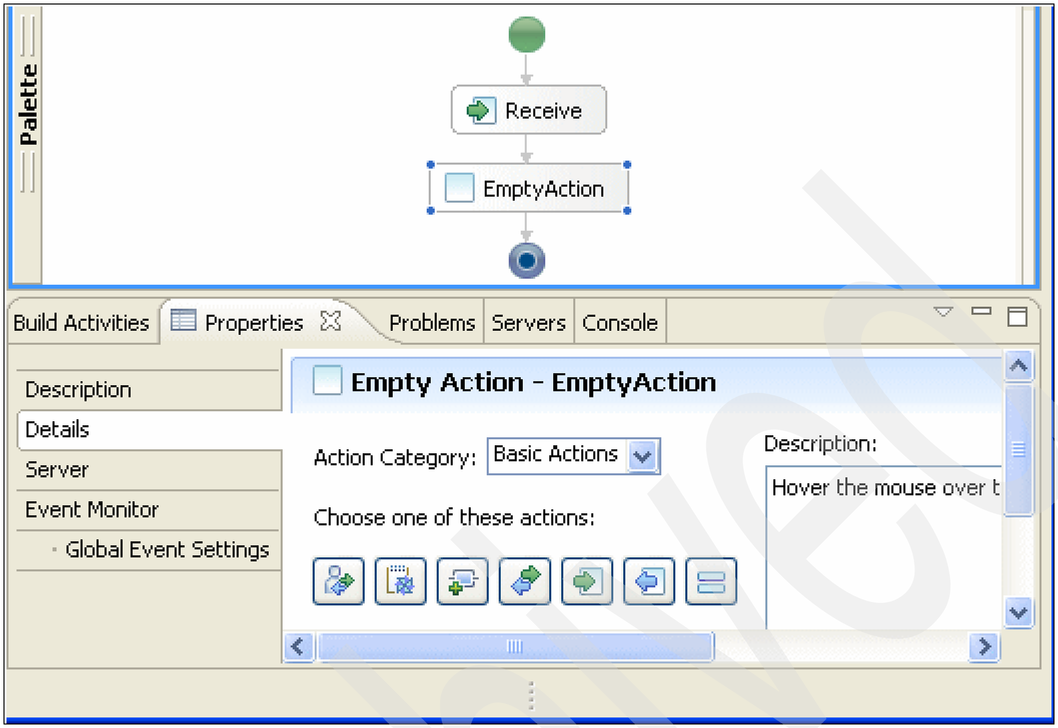
Empty Action activity

The Empty Action activity is an activity which does nothing. Use this activity when you have a construct that requires an activity but there is no work to be done.

For example, you might want to ignore certain faults and just carry on with the process. Because a fault handler must contain an activity, you can insert an empty activity to suppress the fault.

You can also use this activity as an undefined object to act as a placeholder within your process. You might do this if you are designing a process that you expected somebody else to implement or if you are trying to synchronize the activities within a parallel activity. Alternatively, these placeholders can be the activities whose details you will fill in later.

You can also change an Empty Action activity to a different activity using the Details tab of properties of this activity (Figure 4-33). Simply click the appropriate icon on the Details tab, and the Empty Action changes accordingly.



Action Category: Basic Actions

Choose one of these actions:

Description:

Hover the mouse over t

Figure 4-33 Properties of EmptyAction, Details tab

Sequence activity

The Sequence activity is a structured activity that contains one or more other activities that are performed sequentially, in the order in which they are placed within the sequence activity. The editor canvas is actually one big Sequence activity (that is hidden) where you add more simple or structured activities.

Figure 4-34 shows a simple Sequence activity that contains three other activities to be executed in sequence.

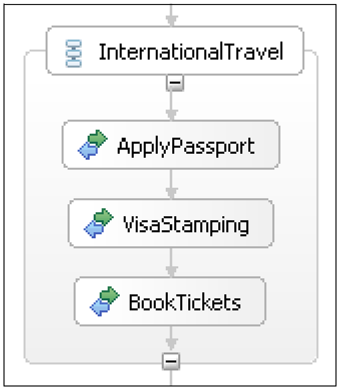


Figure 4-34 A Sequence activity

Parallel Activities activity

Use this structured activity to nest other activities that run concurrently. The nested activities run sequentially in an order that is dictated by links and transition conditions. (When no links are present, all activities will be executed concurrently.) When activities are arranged on separate control paths, the paths run concurrently.

This activity is equivalent to the concept of fork/join—all the paths within the activity run simultaneously and the target activity does not fire until all paths have completed.

A link is used to connect the activities within a Parallel Activities activity to form individual control paths. It is directional so that you can draw from a source activity to a target activity. When the source activity finishes running, the target activity at the other end of the link runs. You can create links from a single source activity to multiple target activities or from multiple source activities to a single target activity.

You can specify a transition condition on the link. If the link condition evaluates to false and if it is the only link between a source and a target activity, the target activity does not run.

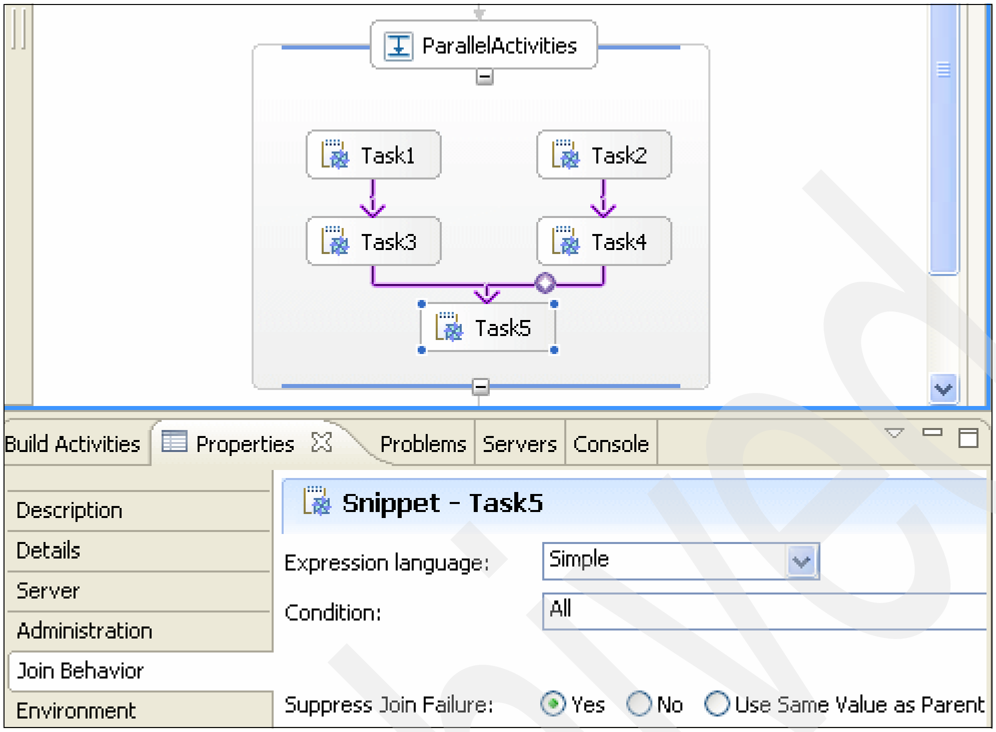
To understand these concepts better, let us look at the business process in Figure 4-35 on page 166. This example shows a parallel activity that contains five activities. (For simplicity, we used snippets for all these activities, which might not be the case in the real world.)

Task1 → Task3 and Task2 → Task4 are two paths that run concurrently followed by Task5. More specifically, Task1 and Task2 run concurrently, where as Task3 and Task4 run only after the completion of Task1 and Task2 (as dictated by the Links). There is a link, Link3 from Task3 to Task5 that is followed, and the other link, Link4 from Task4 to Task5, is followed only if the condition (shown as bubble; select the link to see the condition in properties area) on Link4 is satisfied. Task5 is a special activity that is a target for multiple (two in this case) incoming links.

By default, the target activity runs when any one of the incoming links is followed. A *join condition* specifies when the target of one or more links should run. These join conditions can be created using Java code, a visual snippet, XPath, or by selecting from a list of simple choices: True, False, Any, and All.

If the link condition on Link4, the one from Task4 to Task5, evaluates to `false`, that link is not followed. So in this case, the join condition, `All` for Task5 as set in the Join Behavior tab cannot be satisfied, and a join failure occurs. Join failures for Task5 are suppressed as indicated by the selected setting.

> **Note:** If a join condition is not satisfied, then a join failure fault is thrown. This can be suppressed to allow the process execution to skip the activity and carry on to the next one by selecting **Yes** for **Suppress Join Failure** on the Join Behavior tab of the activity's properties.



Snippet - Task5

Expression language: Simple

Condition: All

Suppress Join Failure: ● Yes ○ No ○ Use Same Value as Parent

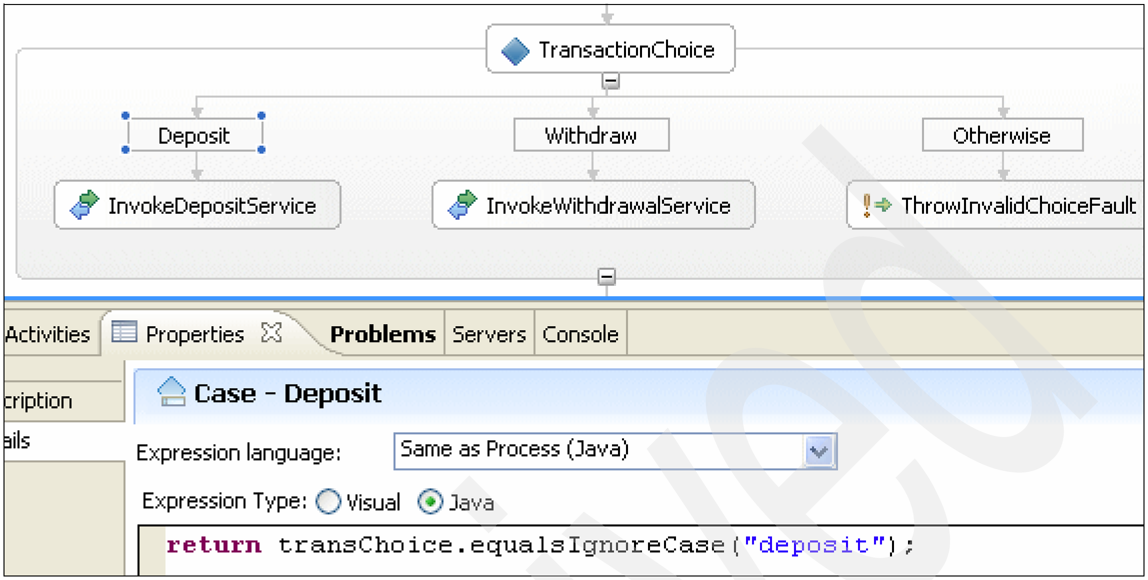
Figure 4-35 A parallel activity

Choice activity

The Choice activity is a structured activity that contains a set of activities that are organized onto individual control paths. The Choice activity decides the path of execution based on a condition.

A Choice activity contains case elements that include expressions that evaluate to either true or false, followed by a sequence of activities. During execution, the Choice activity evaluates the conditions on the paths in order (case elements) and follows the first path that evaluates to true. A Choice activity can also contain an otherwise element to be taken when no case element evaluates to true.

The business process shown in Figure 4-36 shows the usage of Choice activity.



Expression language: Same as Process (Java)

Expression Type: ○ Visual ● Java

```
return transChoice.equalsIgnoreCase("deposit");
```

Figure 4-36 A Choice activity

Receive Choice activity

The Receive Choice activity is a structured activity that halts the process in order to wait for an operation to be called on it or for a time-out alarm to go off. A Receive Choice activity is similar to a Choice activity. The difference is that instead of case elements, there are receive case elements that represent Receive activities and there is no otherwise element. Instead the Receive Choice activity can have a timeout element.

Each of the receive case elements in a Receive Choice activity accepts a particular type of message (an operation of the process's interface). When a Receive Choice activity is encountered, the process execution stops and waits to receive a message (operation). The difference between a normal Receive activity and a Receive Choice activity is that the Receive Choice activity can receive any one of the multiple operations of the associated interface. It follows the control path that is appropriate to the first message (operation) that it receives. The first activity in any path can be either a receive case element or a timeout element.

A timeout element is used within a Receive Choice activity to create a control path that is executed when a specified time either has been reached or has elapsed. During process execution, this path is chosen when no operation is received within this time period (duration) or by the specified date.

Figure 4-37 shows an example of a Receive Choice activity called *ArithmeticOperations*. When this activity is encountered, the process execution is paused and awaits a call to any one of the arithmetic operations (that is add, subtract, multiply, and divide) as configured on the receive case elements. Whichever is called first, the corresponding path is executed. In Figure 4-37, add is called first, then a DoAddition activity is invoked, and the process ends after sending back the reply through a Reply-add activity.

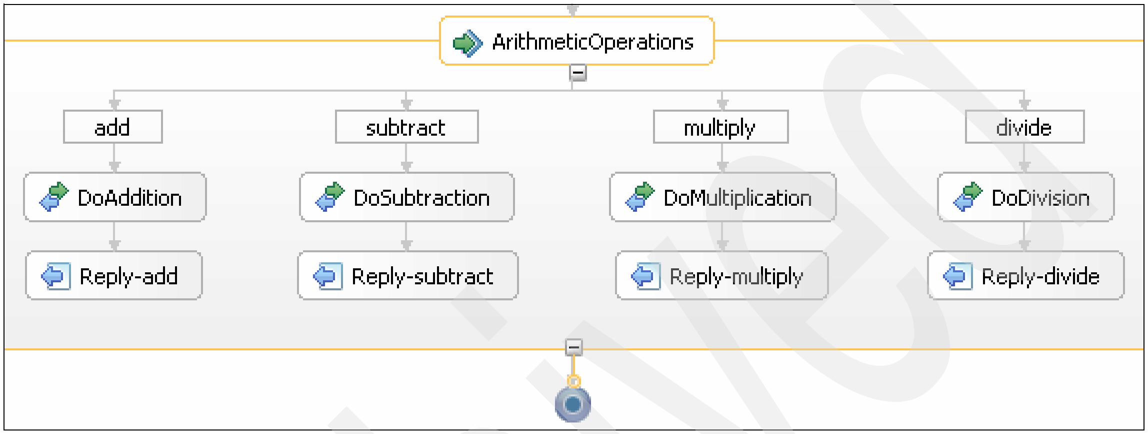


Figure 4-37 A Receive Choice activity

While Loop activity

The While Loop activity is a structured activity that repeatedly executes one or more activities as long as specific success criteria or conditions are met. This structured activity contains a group of other activities and a condition as part of its configuration. If the condition evaluates to true, then the activities within this While Loop activity are executed. When the condition evaluates to false, the loop terminates, and the process execution continues with the next activity.

The condition is checked before the first iteration of the loop, so it might happen that none of the nested activities are executed if the condition evaluates to false. The condition can be a Java or visual expression, a simple true or false, or an XPath.

Scope activity

A Scope activity is a structured activity that acts as a behavioral container for one or more activities in the process. By design, a process as a whole is contained within a single global scope, and you can create and nest other scopes within it, forming a hierarchy.

A Scope activity allows you to define local variables, local correlation sets, and various handlers. A variable is visible only in the scope in which it is created. If it is created in the global process scope, then it is a global variable and, thus, is visible to the process as a whole. Those variables that are created within nested scopes can only be seen by activities or objects within that scope.

Each scope has a primary activity that defines its normal behavior. You can define fault, event, or compensation handlers for a particular scope (see 4.4.11, “Using handlers in a business process” on page 179).

A scope can be configured either as isolated or compensable:

- Choose *isolated* when you want to control simultaneous access to variables that are shared across various scopes and activities.
- Choose *compensable* if you want to allow compensation related activities on this scope. If this option is clear, then this scope is transparent to all compensation logic.

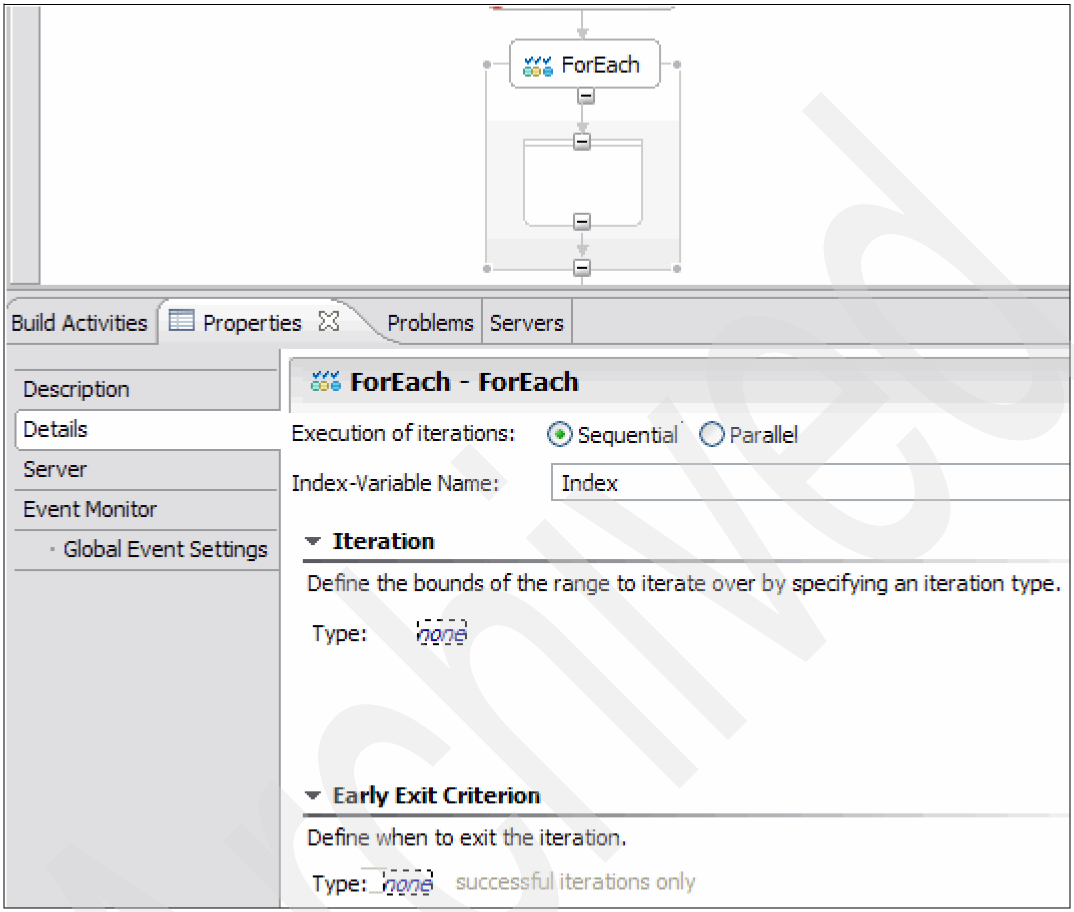
ForEach activity

The ForEach activity is an iterator that repeats the execution of the activities that it contains either sequentially or in parallel for a specified number of iterations. It is possible to define an early exit condition if not all branches are required to complete.

The ForEach activity is very useful in scenarios where you want to interact with a set of partners in parallel and where the partners are identified only at run time. Usually, this activity executes a part of the process multiple times in parallel, without having the knowledge of the actual number of parallel branches at the time that the process is defined. Instead, this number is determined dynamically during the process execution.

For example, you might want to get a quote for a particular product from all available supplier services and can determine only at run time which services are available. Or, you might want to design a process to perform the technical review of an article prior to publication by several independent reviewers in parallel and then consolidate the review comments. In this case, you can add an early exit criterion that specifies that the business process continues with the next activity of review comments consolidation when a reply is received from at least two of the reviewers.

Figure 4-38 lists the properties to be configured when adding a ForEach activity to a business process.



ForEach - ForEach

Execution of iterations: ● Sequential ○ Parallel

Index-Variable Name: Index

Iteration

Define the bounds of the range to iterate over by specifying an iteration type.

Type: *none*

Early Exit Criterion

Define when to exit the iteration.

Type: *none* successful iterations only

Figure 4-38 Properties of a ForEach activity in a business process

The properties include:

- **Execution of iterations**
  - Choose *Sequential* if you want the iterations (to execute group of activities) run one after the other.
  - Choose *Parallel* if you want the iterations run simultaneously.
- **Index-Variable Name**

  Use this field to specify a name for the index variable. For each iteration, the value of this integer variable is increased by one. You can use this variable to determine which iteration you are in or to access an array element.

- **Iteration**

  Use this section to specify the iteration type (how you want the ForEach activity to iterate). This property implicitly defines the bounds of the range values for start and end. The various iteration options are as follows:

  – *Array* (dynamic bounds): A simple option to define bounds of the range. When you choose this option, the ForEach activity then iterates through the elements of this array. The index variable's value starts with 1 and is increased in each iteration up to the size of the selected array.

  – *Integer* (static bounds): Use this option when you know the number of iterations. When you choose this option, two text fields displays into which you enter the start and end values of the iteration.

  – *Expression*: The most flexible option to specify expressions for start and end values. These expressions can be Java or XPath expressions.

- **Early Exit Criterion**

  You can specify an early exit criterion if you do not want to wait for all iterations to complete before the business process execution continues with next activity. When the criterion is met, the business process terminates the remaining iterations and continues with the next activity after the ForEach.

  To specify an early exit criterion, you can use either an expression (Java or XPath) or an integer value.

  Count successful iterations only: Select this option to count only successful iterations for the early exit criterion. An iteration is deemed successful if no fault occurs within the scope that is nested directly inside the ForEach activity.

Cyclic Flow activity

The Cyclic Flow activity (also known as *single threaded graph*) is used to nest activities on individual, customized control paths. This activity enhances the modeling of a business process by enabling arbitrary cycles (also known as *back links*). With arbitrary back links or cycles, it is possible to return to a prior activity in the flow. In particular, this activity is useful for human workflows to allow to go back to a prior step in the workflow.

This activity is equivalent to the concept of split/merge. Although there are multiple paths within the activity, only the path that evaluates to true is followed, and the target activity starts the moment that path is complete.

Use the link within a Cyclic Flow activity to connect nested activities and to form individual control paths. This activity is directional so that you can draw from a source activity to a target activity. When the source activity finishes running, the target activity at the other end of the link runs. You can create links from a single source activity to multiple target activities or from multiple source activities to a single target activity.

You can specify a transition condition on the link. If the link condition evaluates to false and if it is the only link between a source and a target activity, the target activity does not run.

Figure 4-39 shows a section of a business process with a Cyclic Flow activity. The process execution can go back to a prior activity, PerformCodeChanges is based on the reviewer comments as modelled using the back link.

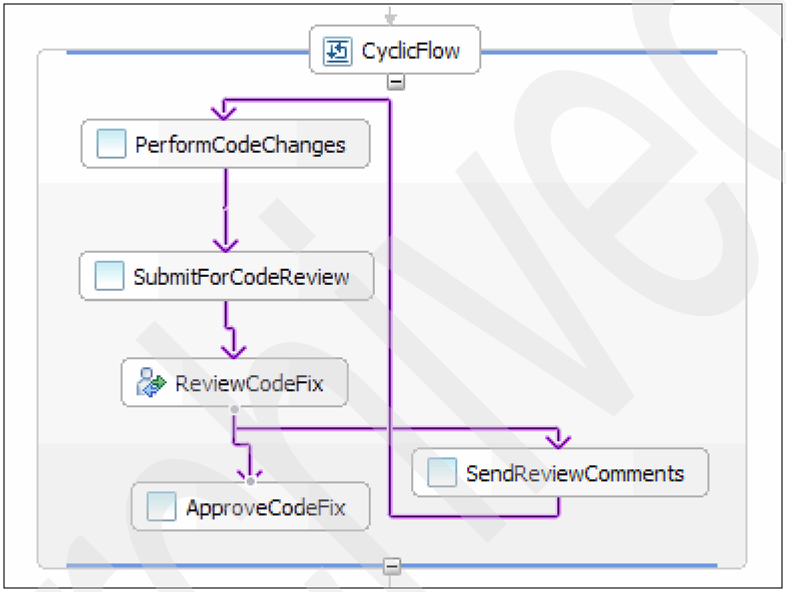


Figure 4-39 A sample Cyclic Flow activity

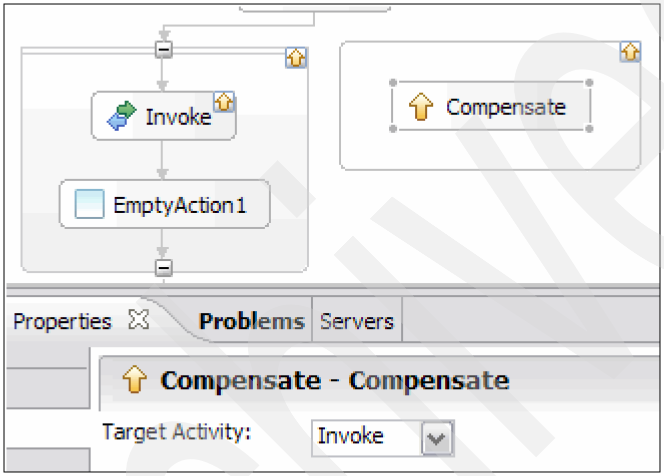
Compensate activity

You use the Compensate activity within a scope's fault or compensation handler to invoke a specific compensation handler within the scope. We discuss compensate handlers in detail in 4.4.11, "Using handlers in a business process" on page 179.

Compensation is an "undo" action for work that has completed successfully. For example, suppose that your process involves shipping orders after payment is received and that an activity to accept payment from a customer completes successfully. Then, something goes wrong, and the complete order cannot be shipped. A compensation handler can perform an action such as reimburse the customer for the unshipped order.

You must configure the Compensation activity with a target activity for the compensation. The target activity specifies the activity whose compensation handler or compensation operation executes when this compensation activity is invoked. This field lists only targets a single activity or a scope that are downstream from this activity (in other words, scopes that are nested and, therefore, can be called). You can leave the field empty. Then, all enclosing successfully completed activities are compensated.

Figure 4-40 shows a snippet of a business process with a Compensate activity for the Scope activity. This snippet is configured to invoke the compensation handler that is defined for the Invoke activity contained within the Scope activity.



Compensate - Compensate

Target Activity: Invoke

Figure 4-40 Target Activity for a Compensate Activity

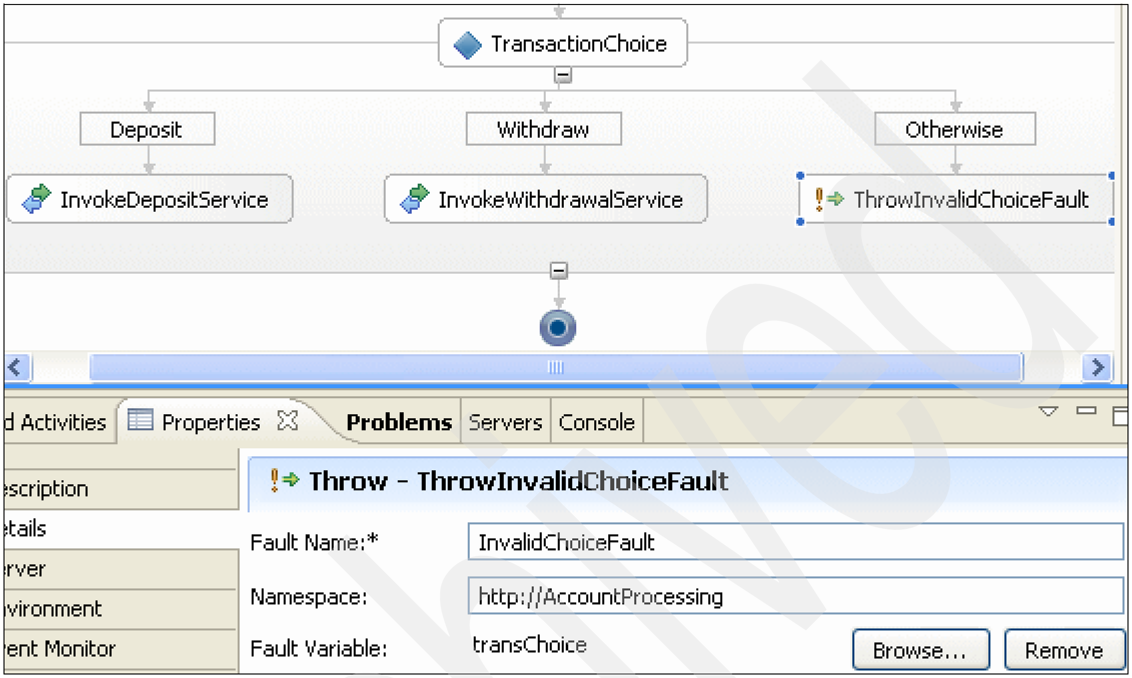
Throw activity

A Throw activity is used to signal an error that can occur that might prevent a business process from reaching completion. A fault thrown by a Throw activity can be caught and handled within the business process using a fault handler.

If a fault that is thrown by a Throw activity is not handled within a process with a request-response operation, it is returned as a runtime exception to the process caller. A fault must have a name and, optionally, can contain a variable that holds information related to the error.

Note: You cannot return a fault with a Throw activity. You must use a Reply activity to return a fault to the process caller. A Reply activity can only return a fault that is defined on the interface that the process implements.

As shown in Figure 4-41, an InvaildChoiceFault is thrown for transaction choices other than deposit and withdraw, which are not covered by any of the case elements.



Throw - ThrowInvalidChoiceFault

Fault Name:* InvalidChoiceFault

Namespace: http://AccountProcessing

Fault Variable: transChoice

Figure 4-41 A Throw activity

Rethrow activity

You can use a Rethrow activity in a fault handler to rethrow the fault to the next enclosing scope. This activity might be useful when you want to do some fault handling on the current scope, such as triggering specific compensation handlers, and still want to make the enclosing scopes aware of this issue. You can also use a Rethrow activity when the current fault handler cannot handle the fault and wants to propagate it to an outer-scoped fault handler.

In the absence of a Rethrow activity, a fault propagated to a higher level using a Throw activity is a new fault instance. When a Rethrow activity is invoked, the fault is the same instance. The fault is rethrown exactly as it was caught by the fault handler (that is, any modifications of the associated fault data are ignored).

Terminate activity

A Terminate activity halts the execution of a process. When used, all activities that are currently active are stopped without any fault handling or compensation behavior.

Terminology for activity labels per BPEL specification

The palette menu shown in Figure 4-42 uses labels for activities per the BPEL specification.

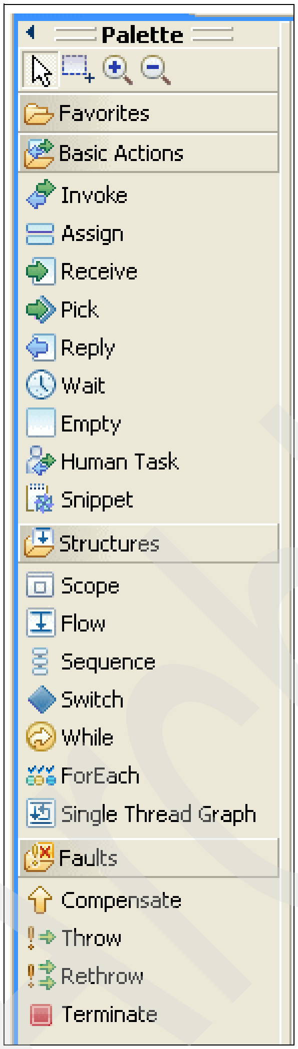


Figure 4-42 Palette of Business Process Editor

Many of the labels that are used for activities and elements in the process editor differ from those used in the BPEL specification. Table 4-1 lists the original terms and how these terms map to those used in this editor.

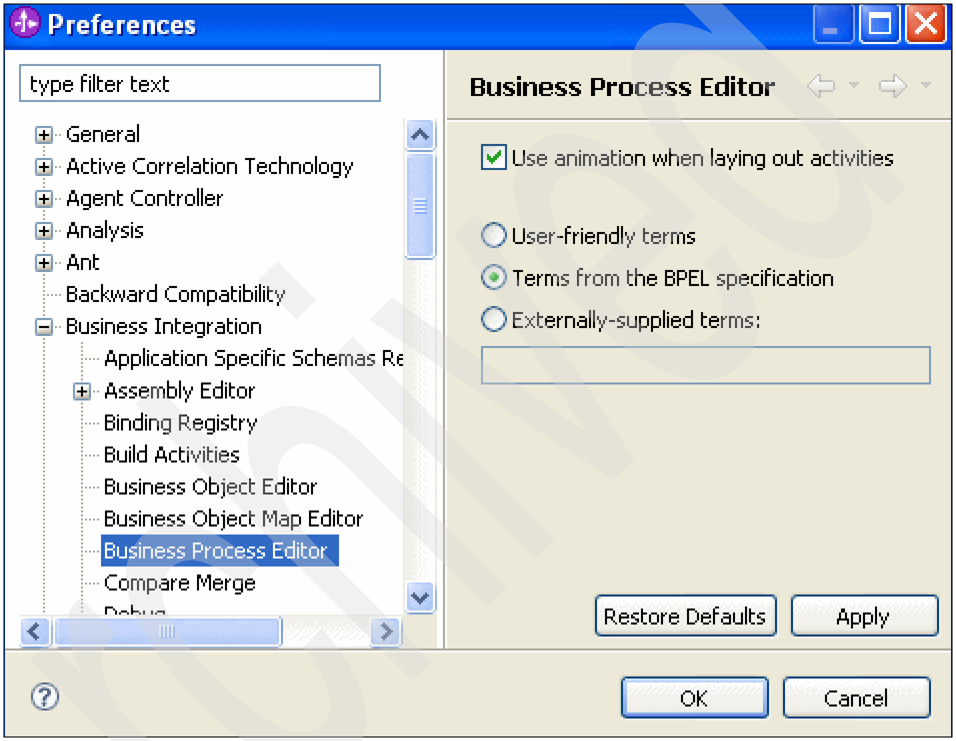
Table 4-1 Terminology used in the process editor and in the BPEL spec

Business process editor terminology	**BPEL terminology**
Receive Choice activity	Pick activity
Empty Action activity	Empty activity
Parallel Activities activity	Flow activity
Choice activity	Switch activity
While Loop activity	While activity
Cyclic Flow activity	Single Thread Graph activity
Receive case element	OnMessage element
Partner	Partner Link

The business process editor uses labels for its activities and elements that are meaningful and easy to understand, but you can change them.

To modify the terminology, follow these steps:

1. Select **Window** → **Preferences**.
2. In the preferences window, expand **Business Integration** → **Business Process Editor** as shown in Figure 4-43.



Business Process Editor

☑ Use animation when laying out activities

○ User-friendly terms

● Terms from the BPEL specification

○ Externally-supplied terms:

Figure 4-43 Business Process Editor Preferences window

3. Select the “Terms from the BPEL specification” option to use labels that are identical to those that are defined in the BPEL specification.
4. Click **OK**.

4.4.9 Using elements in a business process

Elements are the objects that are used to support or configure activities. The simple way to add elements to an activity is to use the action bar menu. Also, an alternative is to right-click the activity and then choose the add element option from the drop-down menu. The elements and handlers that display in the action bar vary according to what activity is currently selected.

Figure 4-44 illustrates these two methods of adding elements for the Choice activity.

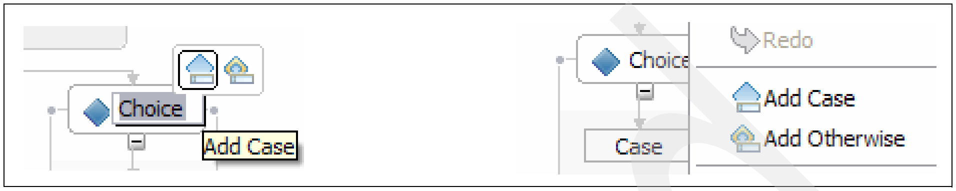


Figure 4-44 Adding elements to Choice activity

You can choose Add Case or Add Otherwise, as follows:

- *Case element*: This element within a Choice activity is used to create a control path and define the conditions that causes this path to run. During execution, the process evaluates the conditions in each of the case elements and follows the first condition that evaluates to true.
- *Otherwise element*: This element within a Choice activity is used to create a control path that runs when none of the other cases evaluate to true. Use this element on only one of the paths within a choice activity. When run, the process evaluates the conditions in each of the case elements and if none of the conditions evaluate to true, it runs the activities in this path.

4.4.10 Using correlation in a business process

Business process interactions involve stateful conversations between processes. In practice, these conversations can occur over a long period of time, possibly days or months. Message correlation is used to match returning consumers or partners to long-running business processes. When a request is issued by a partner, it is necessary to identify whether a new instance of the business process needs to be created or whether the request needs to be directed to an existing process instance. Instead of using a special instance ID, WS-BPEL correlation reuses identifying information from the existing business messages.

Correlation are the records that are used to keep track of multiple partners or messages and that correlate them with a specific and same instance of a business process at any point of time.

The two types of correlation artifacts include:

- *Correlation sets*: Used in runtime environments where there are multiple instances of the same process running. The sets allow two partners to initialize a business process transaction, temporarily suspend activity, and then recognize each other again when that transaction resumes.

- *Correlation properties*: Similar to correlation sets, correlation properties make it possible for the runtime engine to recognize a specific process in an environment where multiple ones are running.

4.4.11 Using handlers in a business process

Handlers consists of a group of activities that represent a specific course of action, which is associated with a particular activity. A handler runs when certain situations occur within the parent activity (that is, the activity with which the handler is associated).

The various types of handlers include:

- Compensation handlers
- Event handlers
- Fault handlers

Faults are any exceptional conditions that can change the normal processing of a business process. A well-designed process should consider faults and handle them whenever possible. Compensation and fault handlers are the two ways of handling faults.

Fault handlers

Use a fault handler to handle partial and unsuccessful work that is a result of a fault (problem or exceptional situation during process execution). A fault handler is a collection of specific activities that run when a fault is thrown on a particular activity with which the handler is associated. When a fault occurs in a process, the navigation moves to the fault handler.

Fault handlers can be added on an invoke or Scope activity. A fault handler can catch a specific fault name, fault type, or both using one or more catch elements. Each path within the fault handler is preceded by either a catch or a catch all element. You can add one catch element for each fault that can potentially occur within the scope or the invoke activity. Each catch is succeeded by a set of activities to deal with that particular fault. You can also add a catch all element to deal with any other faults that are not caught by any of the existing catch elements in the fault handler.

Adding a fault handler

To add a fault handler, follow these steps:

1. Click the scope or invoke activity for which you want to add a fault handler.
2. In the action bar, click the Add Fault Handler icon as shown in Figure 4-45. A fault handler is created with one default catch element and displays on the canvas besides the parent activity.

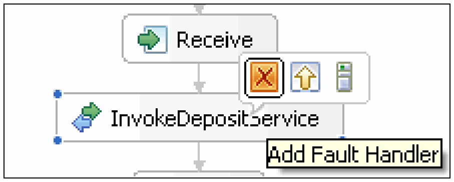
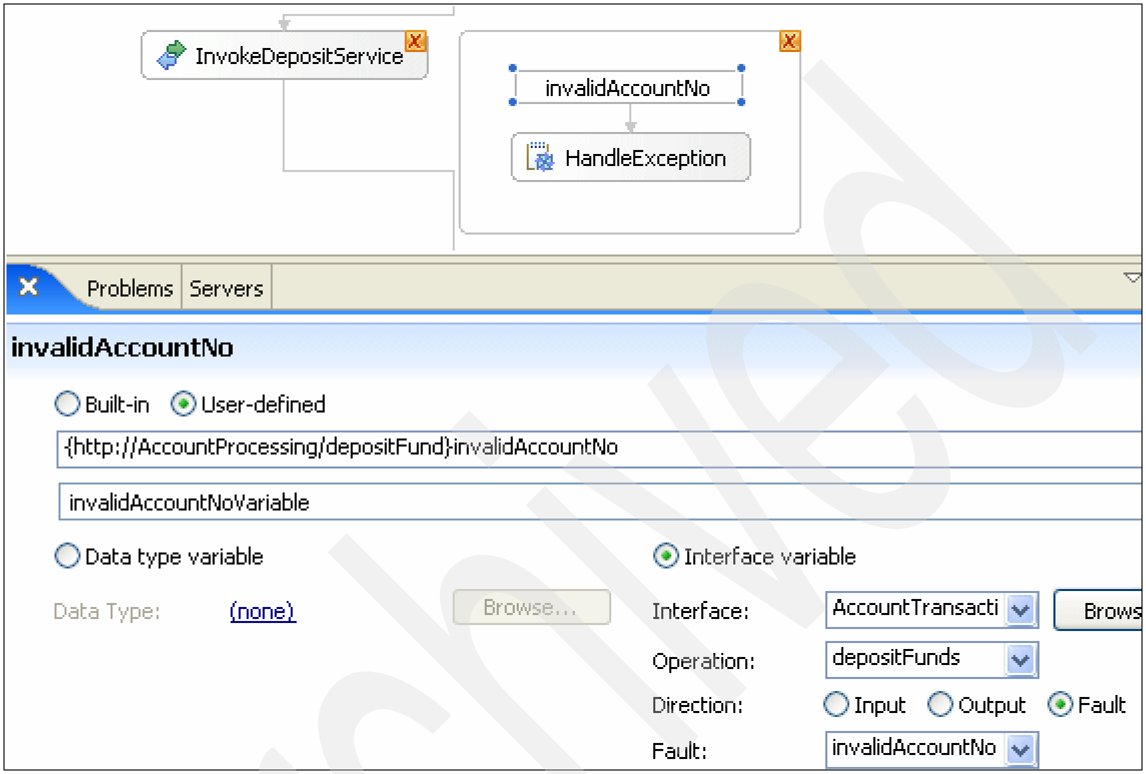


Figure 4-45 Action bar showing Add Fault Handler icon

3. Add to the existing catch path all the activities that are appropriate to deal with the specific exception that has occurred (as shown in Figure 4-46, the properties of the catch element, invalidAccountNo).



invalidAccountNo

○ Built-in ● User-defined

{http://AccountProcessing/depositFund}invalidAccountNo

invalidAccountNoVariable

○ Data type variable ● Interface variable

Data Type: (none) Browse...

Interface: AccountTransacti Brows

Operation: depositFunds

Direction: ○ Input ○ Output ● Fault

Fault: invalidAccountNo

Figure 4-46 Properties of Catch element

4. To add another path, click the fault handler to launch the action bar, and choose the appropriate element (that is Catch or Catch All) as shown in Figure 4-47.

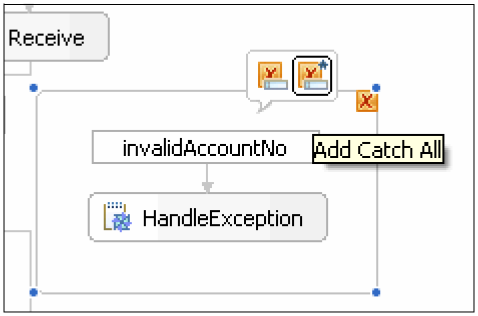


Figure 4-47 Action bar showing Add Catch All

Compensation handlers

Compensation lets you to undo a completed activity. The goal of a compensation handler is to return a failed process to a balanced state by executing a series of associated activities necessary to reverse the completed work. The handler runs only when a fault is thrown and after the parent activity is committed.

You can add a compensation handler to an Invoke or Scope activity. The activity that defines the compensation handler must complete before the compensation handler can run. Therefore, if a the activity throws a fault, compensation for that activity cannot run because it never completed its work. When you define a compensation handler for an activity, you can invoke the handler using a compensate activity.

Figure 4-48 shows the compensation handler for an invoke activity in the scope, and it contains an UndoInvoke activity.

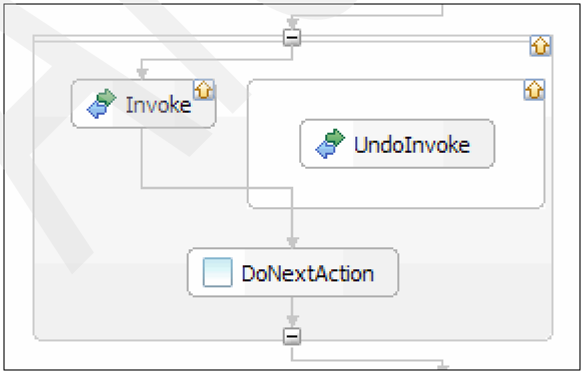


Figure 4-48 Compensation Handler for an Invoke activity

Event handlers

Event handlers enable a running business process to react to events that occur independently and asynchronously. They can respond to events that happen at any time during an application's lifetime or as many times as those events repeat. There might be zero or multiple events at any time.

The advantage of using event handlers

In the GUI environment, events usually signify that the user has made a demand on the system, and the application must respond to it appropriately. In such cases, a Receive or Pick activity can usually be used, but these activities have limitations. For example, they can be used only during normal execution of a process, and they can be implemented only once. These limitations effectively mean that you have to know ahead of time how many events to expect and when to expect them.

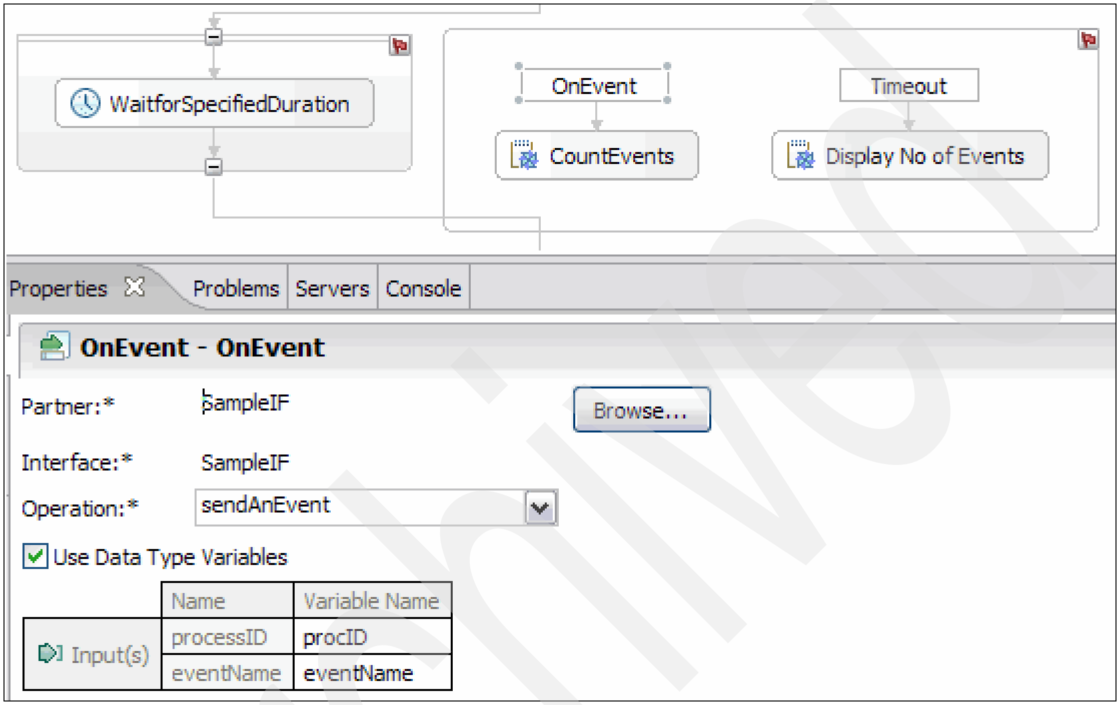
Event handlers can be associated with either a scope or with the business process (which in turn is also a scope). When a scope starts, all associated event handlers are enabled. The event handlers belonging to a scope are disabled when the scope ends. If the scope ends with a fault, the processing of the event handler is terminated.

While a scope is active, the event handlers that are associated with that scope wait for specified events. If no event occurs while the scope is active, the event handler does nothing and is disabled when the scope completes. This behavior is different from a Receive or Pick activity. Receive or pick activities must encounter the message for which they are waiting before processing can continue. Event handlers stop waiting after the associated scope is complete.

There are two elements to support the events:

- *OnEvent element*: Use to create a control path and to specify the operation that causes this path to execute. These events are the incoming messages that correspond to a WSDL operation. A correlation must be specified for the incoming messages.
- *Timeout element*: Use to create a control path that is executed when a specified time either is reached or has elapsed. This element is used on a single path and is configured to specify either a specific date or period of time. During execution, this path is chosen when no input is received within this time period or by the specified date.

Figure 4-49 shows an event handler that is configured for the Scope activity. The Scope activity is designed to wait for specified duration. Whenever a sendAnEvent operation is invoked, the event handler is triggered, and the events are counted by CountEvents snippet. When timeout happens, the count of the events occurred display as designed in Display No of Events snippet.



OnEvent - OnEvent

Partner:* SampleIF

Interface:* SampleIF

Operation:* sendAnEvent

☑ Use Data Type Variables

	Name	Variable Name
Input(s)	processID	procID
	eventName	eventName

Figure 4-49 Event handler configured for the Scope activity

4.5 Human tasks

Human interaction is key to many business integration applications. Human interaction can be required for input and initiation of a process or for review and approval of an item or activity in a business process. Human interaction can also be required to review and correct an error or exception situation in a process that is not fully automated. Even straight-through processing processes invariably have exceptions that cannot be handled automatically. So human interaction is still needed, even with the very capable fault handlers or compensation capability that currently exists.

There are many challenges that are involved when you add human interaction to a process integration solution, such as:

- How to make sure that the correct people are involved?
- How to add capabilities that model real-life human task management, where tasks are claimed, transferred, and completed or are overdue and need escalation?
- How do you make sure that the human interaction provides a suitable interface without disrupting the flexibility to change the business process?

A *human task* is an activity that needs the interaction of a human user. Whenever it is not possible to code logic or to automate part of a process, a human task is the way to enable the staff of a company to interact with a business process. The human task component recognizes the reality that many processes require human intervention for tasks such as reviewing, researching, and approving.

A human task implements a task that is carried out by a person. Human tasks are service components that either can assign work to users or staff or can invoke other services. Human task components include built-in support for role-based task assignment, scheduling, and escalation policies in case a task is not processed within in a predefined time limit.

The Human Task Manager in WebSphere Process Server supports the creation and tracking of tasks during run time. You can use existing Lightweight Directory Access Protocol (LDAP) directories (as well as operating system repositories and the WebSphere user registry) to access staff information. WebSphere Process Server also supports multi-level escalation for human tasks, including e-mail notification and priority aging.

WebSphere Process Server includes an extensible Web client that can be used to work with tasks of processes. This Web client is based on a set of reusable Java Server Faces (JSF) components that you can use to create custom clients or to embed human task functionality into other Web applications.

4.5.1 Implementations and types of human tasks

There are two methods to implement a human task, depending on the usage scenario, and four main types of tasks.

Implementations of human tasks

Human tasks can be implemented as:

- *Inline tasks*: Defined within an implementation of a business process. A inline task can be implemented either directly in the process using a human task activity or as a property of an Invoke, Receive, Receive Choice, event handler, or OnMessage activity. This task does not appear outside of the process and cannot be reused.
- *Stand-alone tasks*: Implemented as an independent component implementation that can be wired to any other components. This task exists independently of a business process and implements human interaction as a service, which is reusable.

The main differences between these two methods of implementation are as follows:

- Inline tasks are tied to a particular process, whereas stand-alone tasks can be reused across multiple processes.
- Stand-alone tasks can be replaced by other SCA implementations, for example business rules, without altering the original process.
- Inline tasks have access to the context information of the process in which they contained and any other inline task within the same process. Stand-alone tasks have access only to the context information of the individual human task.
- Inline tasks can be used for server-driven page flow applications using the completeAndClaimNext functionality. Stand-alone tasks are independent of each other.
- Inline tasks are deleted when their parent process is deleted.

Types of human tasks

The four main types of human tasks are:

- To-do task
- Invocation task
- Collaboration task
- Administration task

To-do task

A *to-do task* (machine-to-human or *M2H*) is a human task that is invoked by a service component, which assigns a task to a human to do something. A to-do task can be either inline or stand-alone. In earlier releases of WebSphere Process Server V6.1, this task is referred to as a *participating task*.

Invocation task

An *invocation* task (human-to-machine or *H2M*) is a human task that is triggered by a human who assigns a task to a service component. In this case, a human invokes a service component such as a business process. An invocation task can be either inline or stand-alone.

When this task is inline, an invocation task allows humans to invoke the operations that a business process exposes through activities such as Receive, Receive Choice, or event handlers. Through this task, a user can start a process and define authorization for its inbound activities. In earlier releases of WebSphere Process Server V6.1, this task is referred as an *originating task*.

Collaboration task

A *collaboration* task (human-to-human or *H2H*) is a human task that is triggered by a human who assigns a task to another human. A collaboration task is stand-alone in that there is no interaction between it and any other component. This task is self-contained and implements a stand-alone human interaction without any reference or interface to another service. In earlier releases of WebSphere Process Server V6.1, this task is referred to as a *pure human task*.

Administration task

An *administration* task is a human task that is created by components to offer an interface for a human administrator. This type of task grants to a human the right to perform administrative actions, such as suspend, terminate, restart, force-retry, or force-complete a business process. Administration tasks can be set up on either an Invoke activity or the process as a whole. This type of task is available only within a business process (inline task).

These tasks have the interaction patterns shown in Figure 4-50.

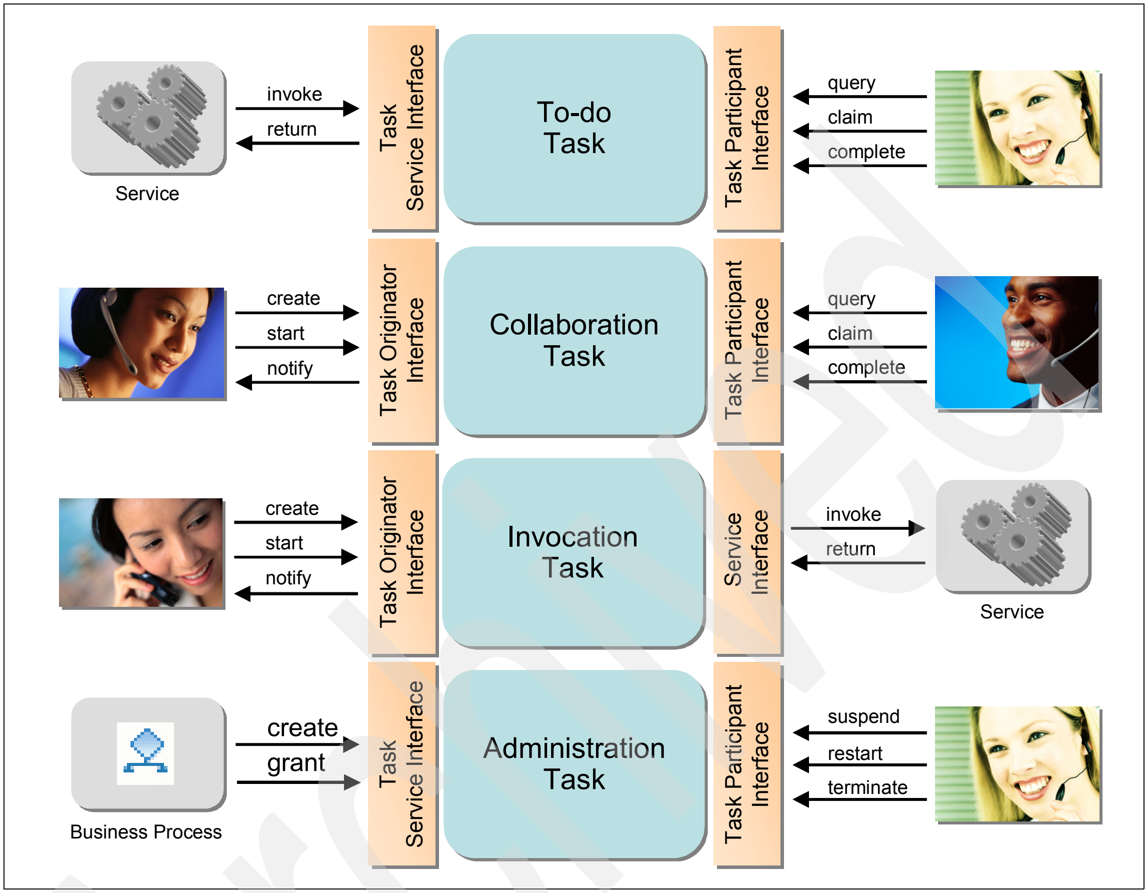
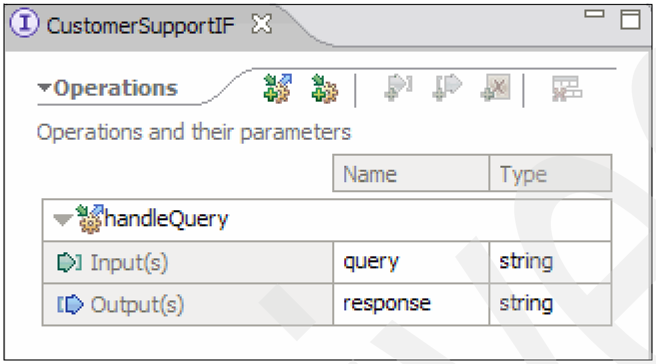


Figure 4-50 The interactions of the main types of human tasks

4.5.2 Creating a human task

To create a new human task, follow these steps:

1. Create a module. In this example, the module is *CustomerSupportModule*.
2. Create an interface in the module. In this example, we use *CustomerSupportIF*, as shown in Figure 4-51.



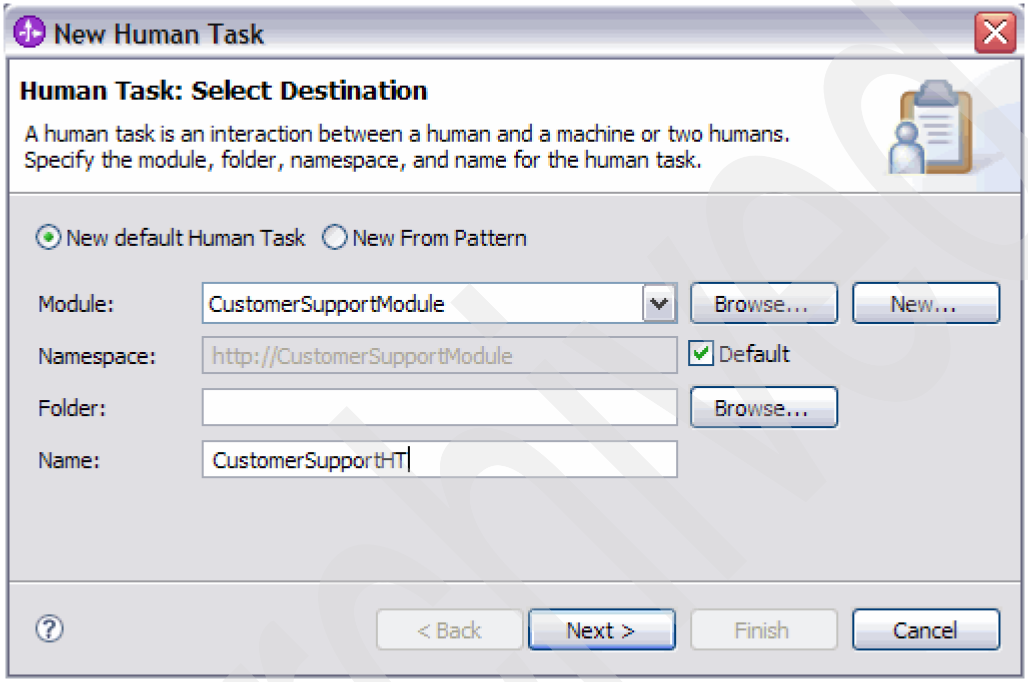
Operations and their parameters

	Name	Type
handleQuery		
Input(s)	query	string
Output(s)	response	string

Figure 4-51 CustomerSupportIF for a new human task

3. Select **File** → **New** → **Human Task**. In the New Human Task window, shown in Figure 4-52:

   a. Select **New default Human Task**
   b. Browse to an existing module (or click **New** to create one).
   c. Specify a folder (optional).
   d. Specify a name for the new human task (for example, `CustomerSupportHT`).
   e. Click **Next**.



New Human Task

Human Task: Select Destination

A human task is an interaction between a human and a machine or two humans. Specify the module, folder, namespace, and name for the human task.

● New default Human Task ○ New From Pattern

Module: CustomerSupportModule

Namespace: http://CustomerSupportModule ☑ Default

Folder:

Name: CustomerSupportHT

Figure 4-52 New Human Task Wizard

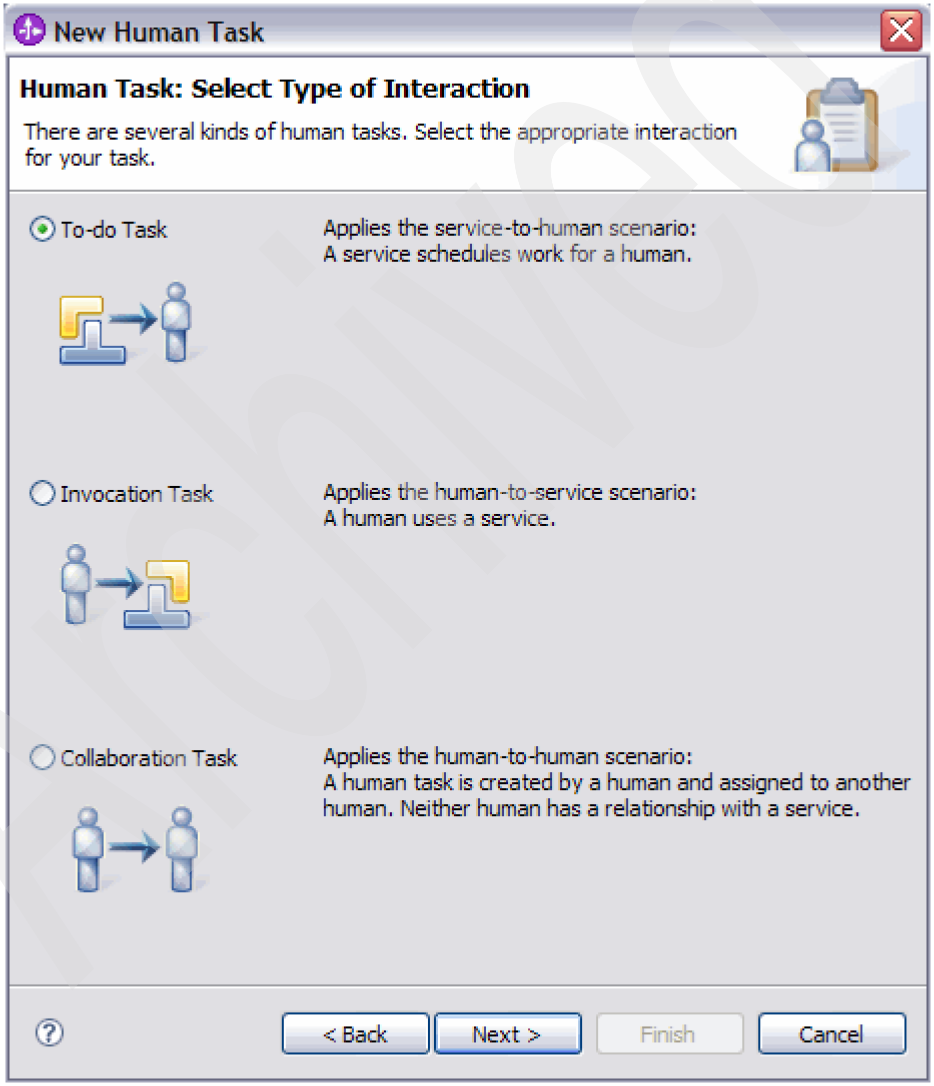
4. Select the type of interaction for the human task. As shown in Figure 4-53, you have the following options in a stand-alone human task implementation:

   To-do task A service component assigns a task to a human to do something.

   Invocation task A human can “assign” a task to a service component.

   Collaboration task A human assigns a task to another human.

   In this example, click **To-do Task**, and then click **Next**.



New Human Task

Human Task: Select Type of Interaction

There are several kinds of human tasks. Select the appropriate interaction for your task.

● To-do Task — Applies the service-to-human scenario: A service schedules work for a human.

○ Invocation Task — Applies the human-to-service scenario: A human uses a service.

○ Collaboration Task — Applies the human-to-human scenario: A human task is created by a human and assigned to another human. Neither human has a relationship with a service.

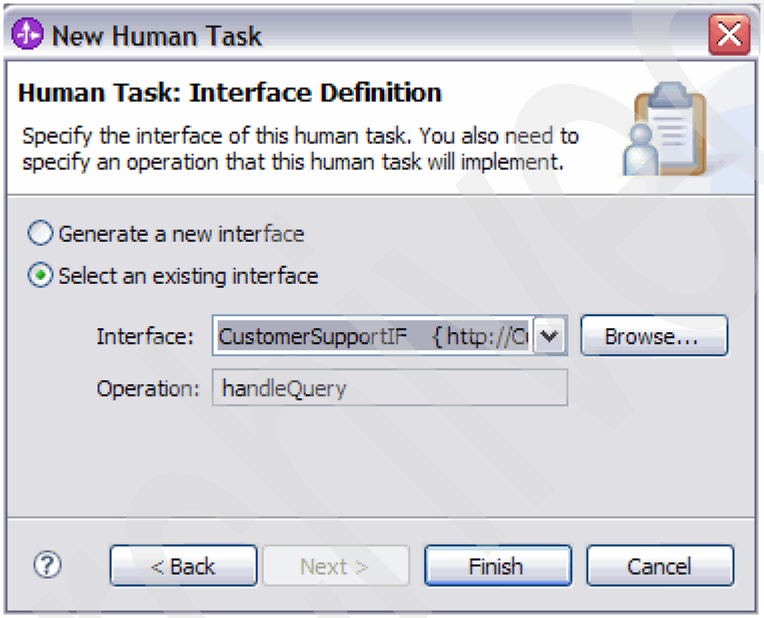
Figure 4-53 Select type of interaction for human task

5. In the New Human Task dialog box, shown in Figure 4-54:

   a. Click **Select an existing Interface**. Alternatively, you can select **Generate a new Interface** to generate a new interface to be used with this human task.

   b. Select the existing interface, **CustomerSupportIF**, from the drop-down menu. Alternatively, you can click **Browse** and select an interface in Interface Selection wizard as shown in Figure 4-55 on page 193.

   c. Click **Finish**.



New Human Task

Human Task: Interface Definition

Specify the interface of this human task. You also need to specify an operation that this human task will implement.

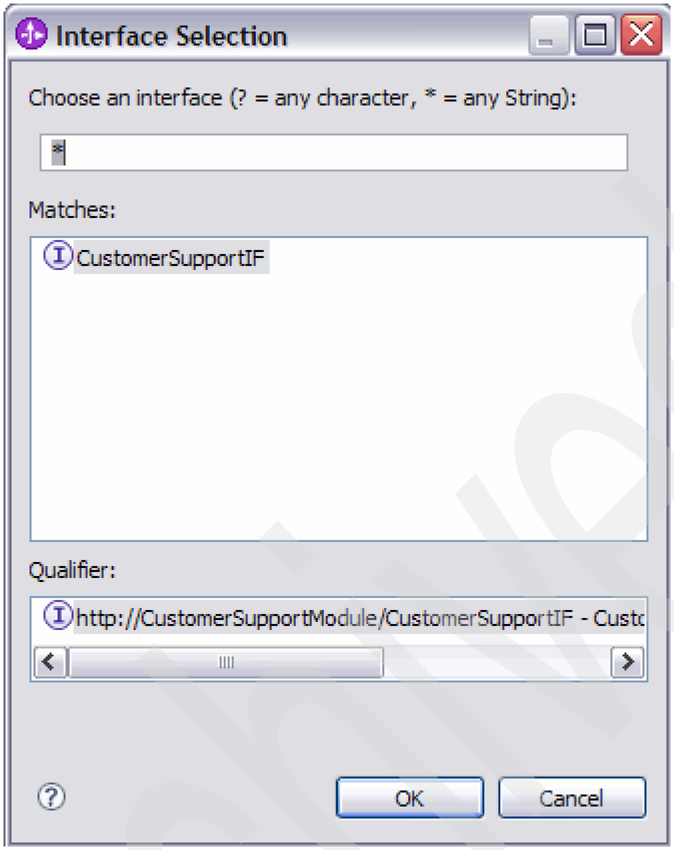
○ Generate a new interface

● Select an existing interface

Interface: CustomerSupportIF {http://C

Operation: handleQuery

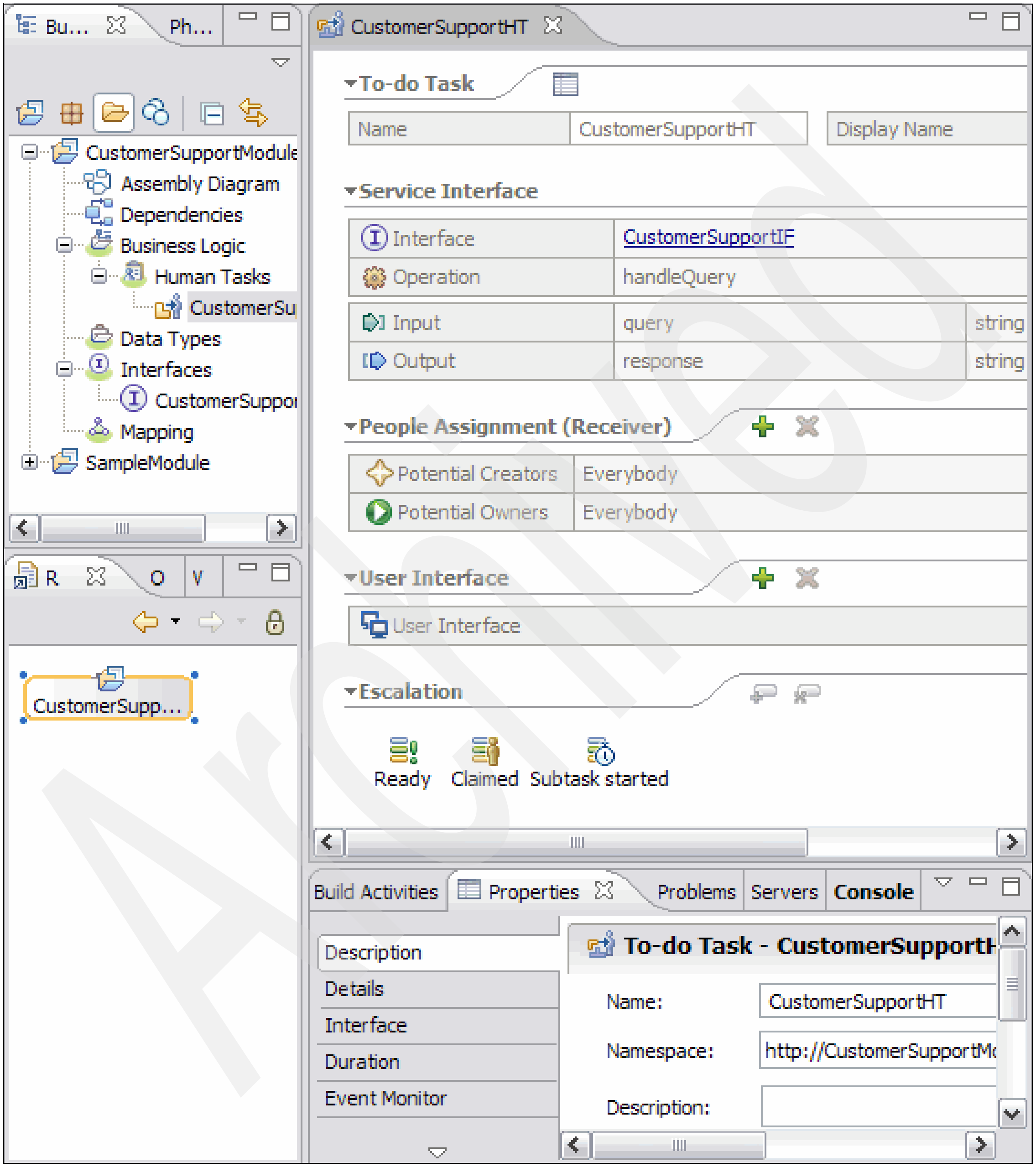
Figure 4-54 Select an interface in the New Human Task wizard



Interface Selection
Choose an interface (? = any character, * = any String):
*
Matches:
CustomerSupportIF
Qualifier:
http://CustomerSupportModule/CustomerSupportIF - Custc
OK
Cancel

Figure 4-55 Interface Selection wizard for human task

The CustomerSupportHT is created and opens in the human task editor as shown in Figure 4-56.



Name	CustomerSupportHT	Display Name

Interface	CustomerSupportIF	
Operation	handleQuery	
Input	query	string
Output	response	string

Potential Creators	Everybody
Potential Owners	Everybody

User Interface

Name: CustomerSupportHT

Namespace: http://CustomerSupportMc

Description:

Figure 4-56 CustomerSupportHT Business Integration View

4.5.3 Human task editor

The human task editor, shown in Figure 4-57 on page 196, is a is a GUI-based tool. It helps you to visually configure the interaction between a service and its associated human participants. The human task editor includes the following areas:

- The service interface area

  This area shows the interface and the operation, which is associated with this human task and the corresponding inputs and outputs. You can click the interface name link to launch the interface editor to make any changes to the interface.

- The people assignment area

  This area shows a list of the people assignments (roles with criteria) who can interact with the task, providing the “Add roles to define access rights to the task” icon (✚) to add a new people assignment. To remove an authorization role, highlight a role and click the delete (✖) icon.

- The user interface area

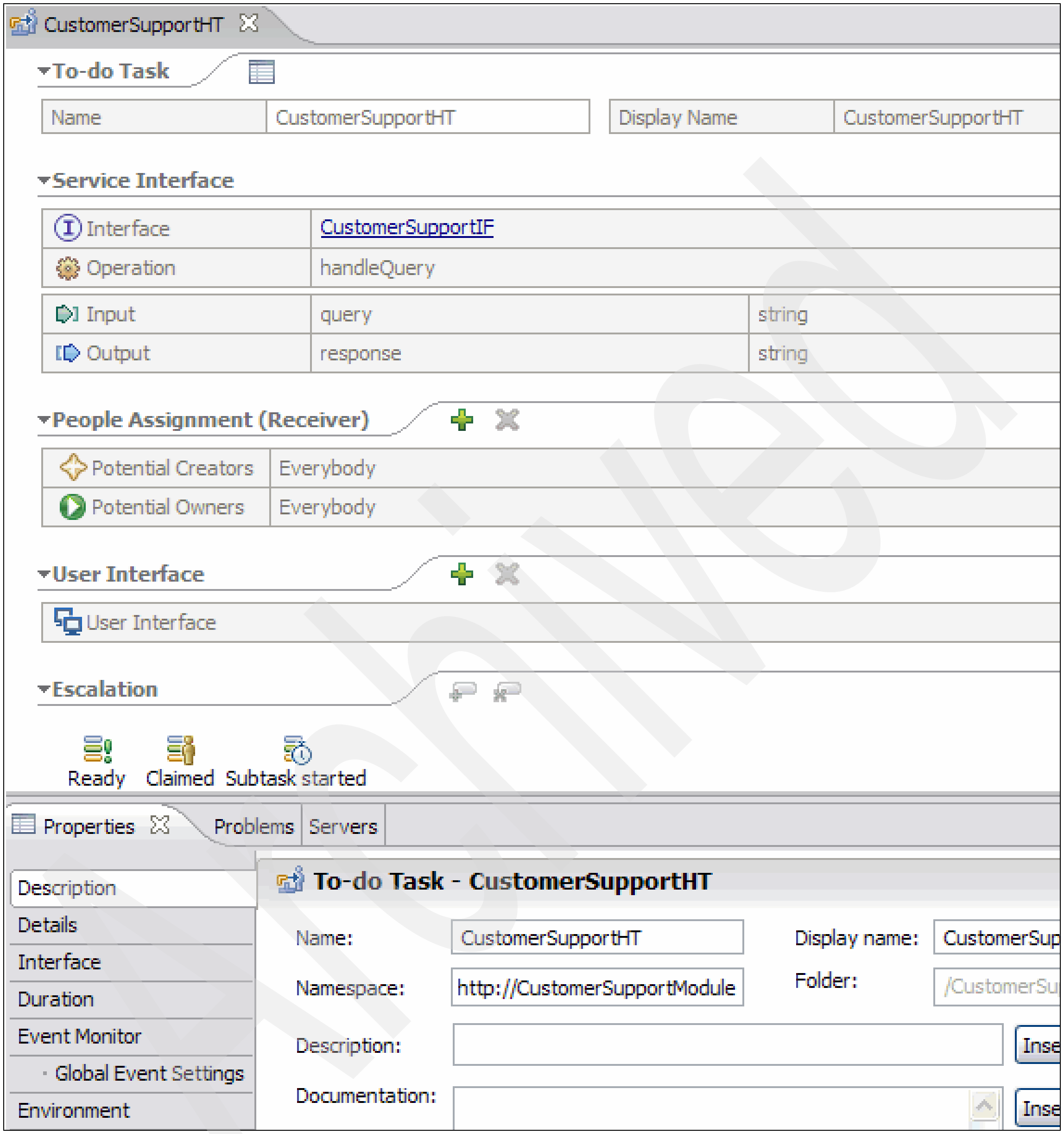
  Use this area to add additional, client type specific information to a human task. The clients that are listed here are the clients that can interact with the task in the runtime environment. Click the “Add definitions to be used by clients” icon (✚) to add a new user interface and then to configure it in the Details tab of Properties view.

- The escalation area

  The escalation settings define how a task is handled when an expected action has not been performed within a predefined time period.

- The Properties view

  This area displays properties of the object that are currently selected in the editor. Click the tabs to the left of this view to toggle through various property pages. You can add or modify any of the properties by clicking the appropriate field shown on GUI.



CustomerSupportHT

To-do Task

Name	CustomerSupportHT	Display Name	CustomerSupportHT

Service Interface

Interface	CustomerSupportIF	
Operation	handleQuery	
Input	query	string
Output	response	string

People Assignment (Receiver)

Potential Creators	Everybody
Potential Owners	Everybody

User Interface

User Interface

Escalation

To-do Task - CustomerSupportHT

Name: CustomerSupportHT

Display name: CustomerSup

Namespace: http://CustomerSupportModule

Folder: /CustomerSu

Description:

Documentation:

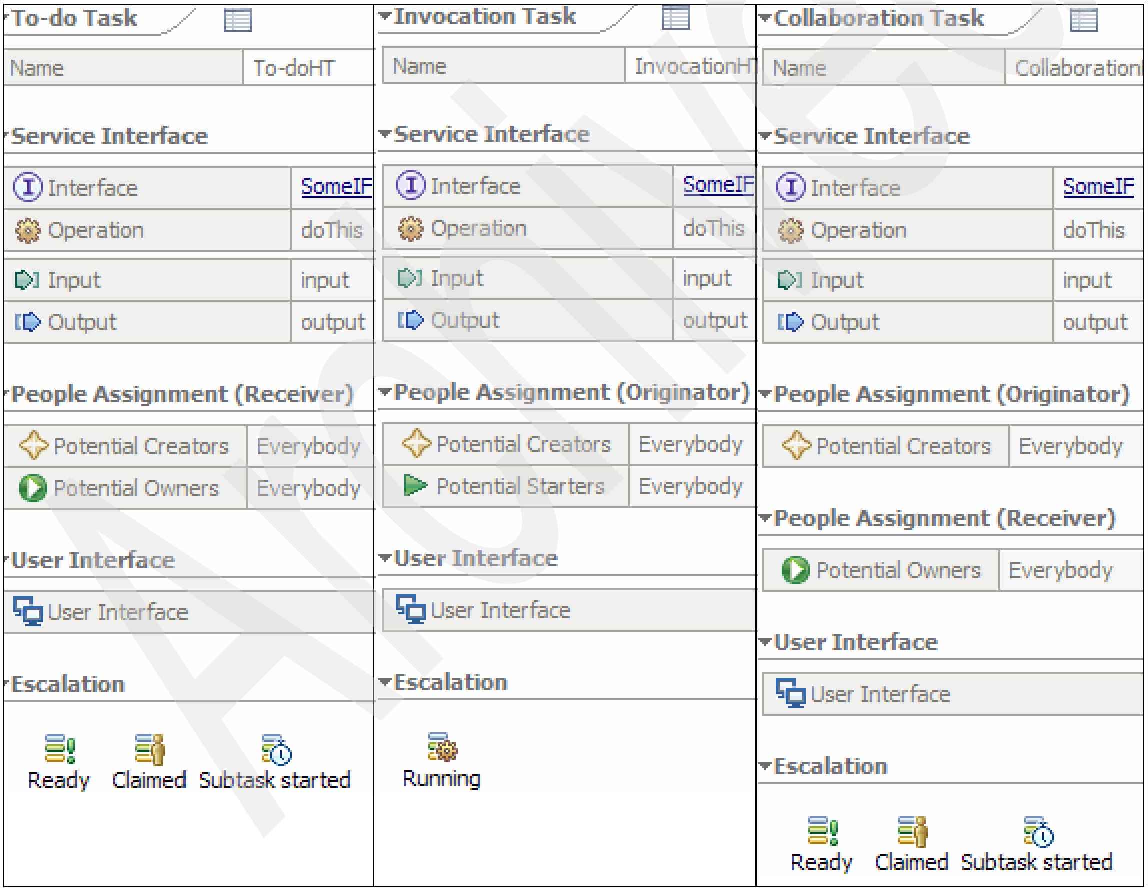
Figure 4-57 The human task editor

4.5.4 Building blocks of a human task

A human task definition is comprised of the following building blocks:

- Type and name of the task (the type can be to-do, invocation, or collaboration)
- Service interface (what needs to be done and how)
- People assignment (who can do the task)
- User interface (how the human interacts with the task)
- Escalation (what happens when the task takes too long)

You can configure or manipulate each of these settings using the human task editor. Figure 4-58 shows the building blocks for each type of human task. The figure contains a small portion of what you see when you open each of these three task types in the human task editor.



To-do Task

Name	To-doHT

Service Interface

Interface	SomeIF
Operation	doThis
Input	input
Output	output

People Assignment (Receiver)

Potential Creators	Everybody
Potential Owners	Everybody

User Interface

User Interface

Escalation

Ready Claimed Subtask started

Invocation Task

Name	InvocationHT

Service Interface

Interface	SomeIF
Operation	doThis
Input	input
Output	output

People Assignment (Originator)

Potential Creators	Everybody
Potential Starters	Everybody

User Interface

User Interface

Escalation

Running

Collaboration Task

Name	Collaboration

Service Interface

Interface	SomeIF
Operation	doThis
Input	input
Output	output

People Assignment (Originator)

Potential Creators	Everybody

People Assignment (Receiver)

Potential Owners	Everybody

User Interface

User Interface

Escalation

Ready Claimed Subtask started

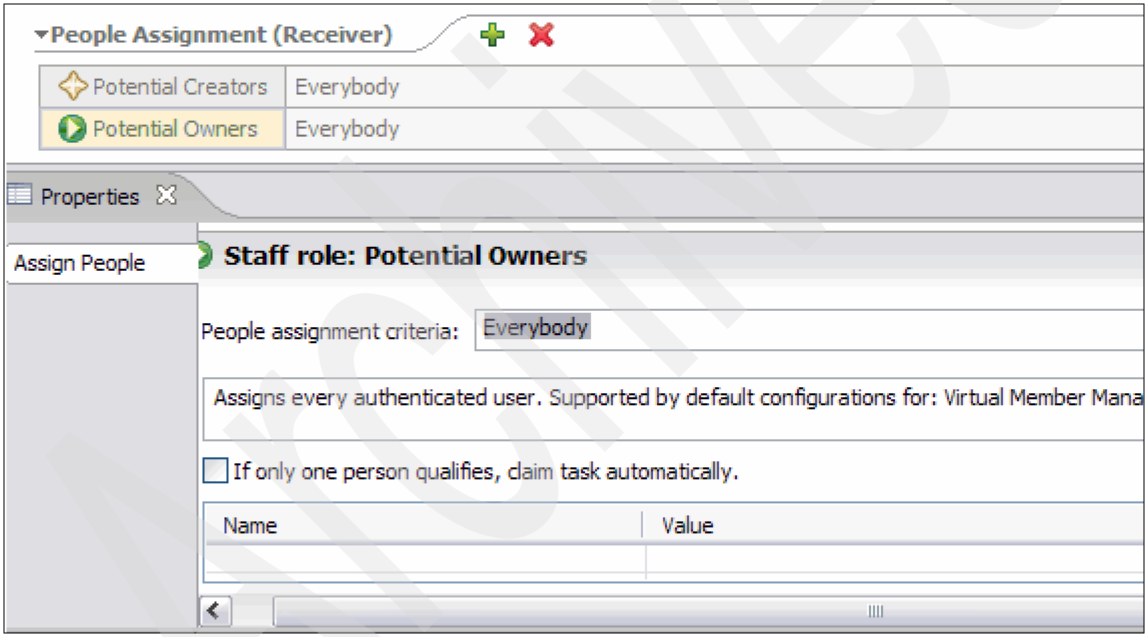
Figure 4-58 To-do, invocation, and collaboration tasks in the human task editor

4.5.5 People assignments

People assignment roles and the criteria that is defined for a task identifies the staff members to interact with the task based on their access rights. More specifically, *authorization roles* determine what the members are allowed to do in the runtime environment, and *people assignment criteria* defines who is a member of an authorization role.

The people assignment area of a task in human task editor facilitates the configuration of authorization roles and people assignment criteria.

A task is assigned to a role, not an individual. Each staff member who belongs to this role group has permissions assigned to the role as a whole. Figure 4-59 shows the assigned roles (for example, Potential creators, potential owners) under the people assignment area of the human task editor.



Potential Creators	Everybody
Potential Owners	Everybody

Staff role: Potential Owners

People assignment criteria: Everybody

Assigns every authenticated user. Supported by default configurations for: Virtual Member Mana

□ If only one person qualifies, claim task automatically.

Figure 4-59 People assignment roles and criteria

People assignment criteria (of a specific role) further refines the list of members of the specific role group who can work with the task. Figure 4-59 shows the people assignment criteria for a role (for example, Potential Owners as Everybody) that can be changed on the Assign People tab of the Properties view of the human task editor.

Authorization roles for human tasks

Authorization is the mechanism by which certain people (staff of an organization) are enabled to perform selected actions on tasks. Authorization roles are used to define sets of actions that are available to specific roles. Role-based authorization requires that administrative and application security is enabled for the application server. A role can be:

- A system-level J2EE role
- An instance-based role

System-level J2EE roles

System-level J2EE roles are set up when the Human Task Manager is configured. The authority level that is implied by these roles is valid for all tasks and escalations. You can change the assignment of users and groups to these roles using the administrative console of WebSphere Process Server. The following J2EE roles are supported:

- *TaskSystemAdministrator*: Users assigned to this role have all privileges. This role is also referred to as the *system administrator* for human tasks.
- *TaskSystemMonitor*: Users assigned to this role can view the properties of all of the task objects. This role is also referred to as the *system monitor* for human tasks.

Instance-based roles

Instance-based roles are valid for tasks and escalation instances or for the templates that are used to create task or escalation instances. The association of users to instance-based roles is determined either by people assignment criteria or as the result of task actions.

People are assigned to roles at run time by people assignment criteria that is based on the user and user group information that is stored in a people directory, namely, virtual member manager, LDAP, user registry, and system (operating system), as shown in Table 4-2.

Table 4-2 Authorization roles determined by people assignment criteria

Role name	Description
Potential Creators	Members of this role can create an instance of the task but cannot start it.
Potential Starters	Members of this role can start an existing task instance. This role is associated only with an invocation task.
Potential Owners	Members of this role can claim, work on, and complete tasks.

Role name	Description
Readers	Members of this role can view tasks but cannot work on them. This role can be used in situations where an employee wants to monitor a task without taking any action on it.
Editors	Members of this role can work with the content of a task but cannot claim or complete it.
Administrators	Members of this role can administer tasks. They have the authority to perform higher authority duties, such as suspend, terminate, restart, force-retry, and force-complete.
Escalation receiver	Members of this role have the authority of a reader for the escalation and the escalated task.

The roles shown in Table 4-3 are associated with only one user and are assigned as the result of a task action.

Table 4-3 Authorization roles determined by task actions

Role name	Description
Originator	The person who created the task. A person with a potential creator role becomes an originator after creating a task. The person with this role has administrative rights until the task starts. When the task starts, the originator has the authority of a reader and can perform some administrative actions, such as suspending tasks, resuming tasks, and transferring work items.
Starter	The person who started the task. A person with a potential starter role becomes a starter after starting a task. The person with this role has the authority of a reader and can perform some administrative actions, such as transferring work items.
Owner	The person who claimed the task. A person with a potential owner role becomes an owner after claiming a task. The person with this role works on and completes a task.

People assignment criteria

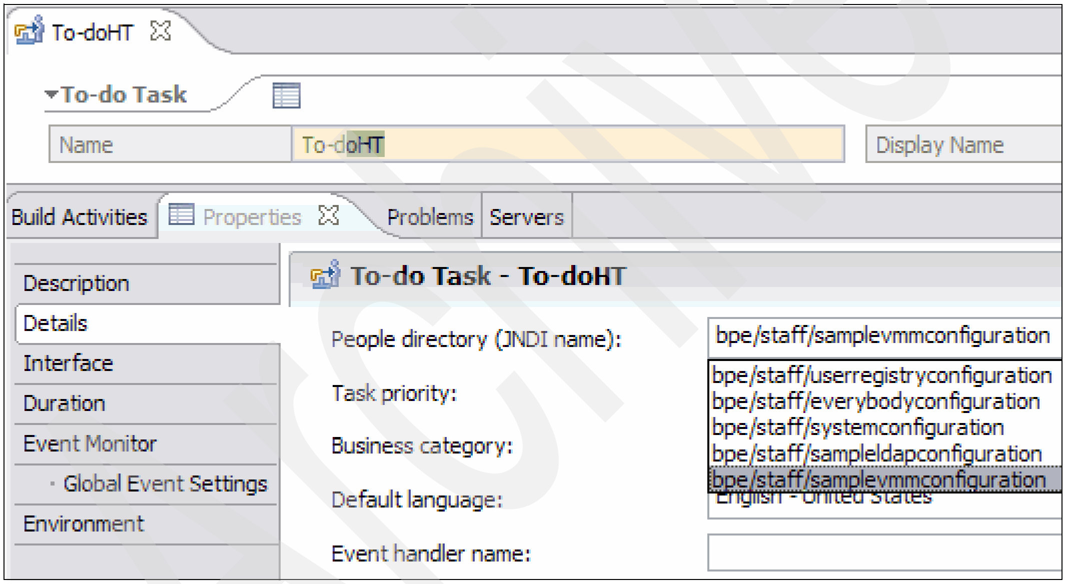
People assignment criteria are constructs that are used with human tasks to identify sets of people that can be assigned to an instance-based authorization role. At run time, this criteria is used to retrieve sets of users from people directories. The criteria can be composed using predefined staff verbs or keywords, such as Everybody, Group, and so forth, and replacement expressions.

People directories

People directories store user and groups information that is used for people resolution based on the people assignment criteria. The following people directories are supported:

- *Virtual member manager* (VMM), also referred to as *federated repositories*: This is the default people directory that is supported by WebSphere Application Server.
- LDAP directory
- Local operating system user registry
- WebSphere Application Server user registry

Figure 4-60 shows the people directory configuration property page for a to-do task, using a preconfigured sample VMM configuration. The drop-down list in the figure is expanded to capture the JNDI names of other people directories.



Name: To-doHT
Display Name

People directory (JNDI name): bpe/staff/samplevmmconfiguration

bpe/staff/userregistryconfiguration
bpe/staff/everybodyconfiguration
bpe/staff/systemconfiguration
bpe/staff/sampleldapconfiguration
bpe/staff/samplevmmconfiguration

Task priority:

Business category:

Default language: English - United States

Event handler name:

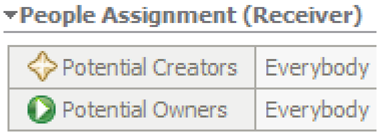
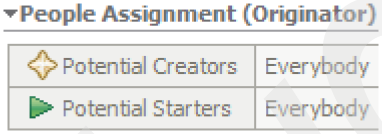
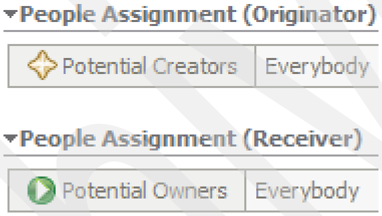
Figure 4-60 People directory configuration page for a to-do task

Assigning a role to a human task

To add a role to a task using the human task editor, follow these steps:

1. Open the task in the human task editor. Table 4-4 shows the roles that are added by default during creation for various types of human tasks.

Table 4-4 People assignment settings for various human tasks

Task Type	People Assignment Display	Description
To-do	 People Assignment (Receiver) Potential Creators Everybody Potential Owners Everybody	Person is a receiver. In this case, you can configure the roles for the people who can claim and work on this task.
Invocation	 People Assignment (Originator) Potential Creators Everybody Potential Starters Everybody	Person is an originator. In this case, you can configure the roles for the people who can initiate the task.
Collaboration	 People Assignment (Originator) Potential Creators Everybody People Assignment (Receiver) Potential Owners Everybody	Person-to-person (that is, the person is both a receiver and an originator). Because a collaboration task is assigned to one person from another, it has both receiver and originator settings.

2. Click the "Add roles to define access rights to the task" icon (✚) to add a new role and to select one from the drop-down list.

> **Note:** The drop-down list shows only those roles that are applicable and still not added. The roles that are listed as images in Table 4-2 on page 199 are the ones that you can add to the tasks under people assignment area.

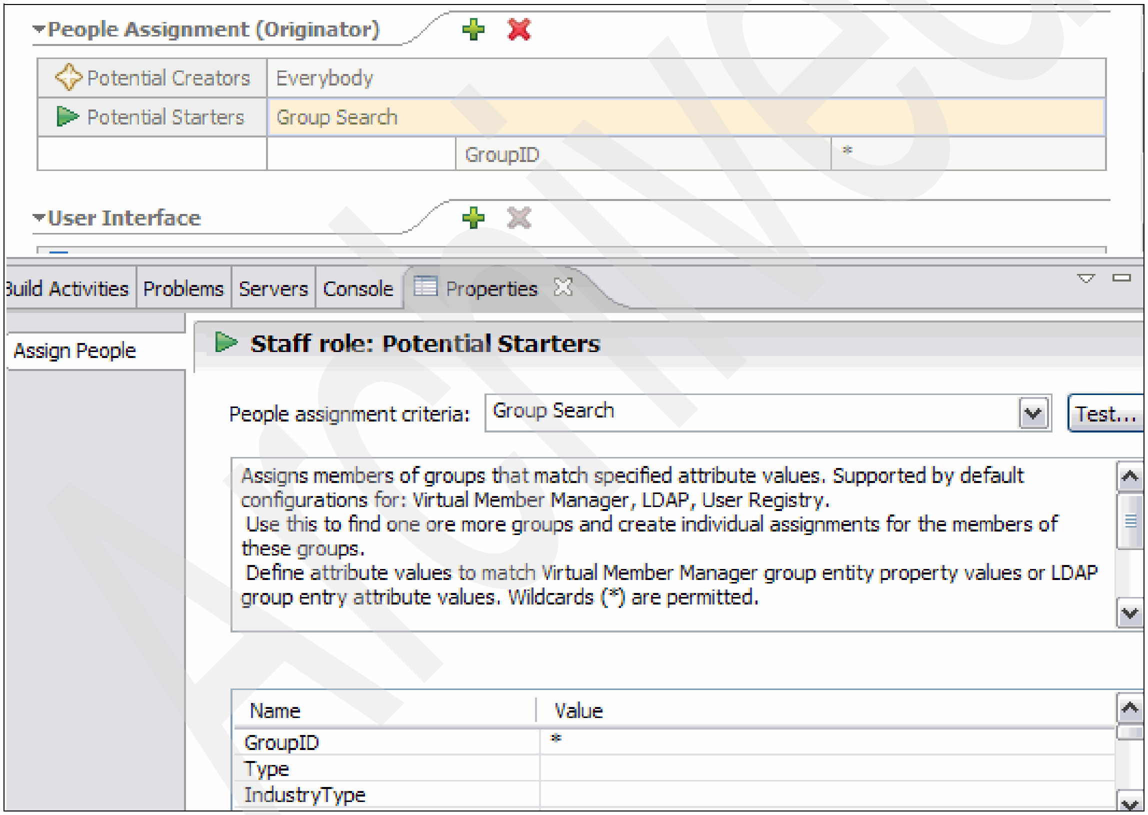
3. Configure the people assignment criteria settings on the Assign People tab of Properties view of the human task editor (Figure 4-59 on page 198).

Testing people assignment criteria

You can test the people assignment criteria that is used in a human task to make sure that the correct people are retrieved from the people directory. Follow these steps:

1. Open a human task in the human task editor.
2. Select a role.
3. Click **Test** in the Assign tab of the properties page as shown in Figure 4-61).

> **Note:** This button is available for all people assignment criteria except *everybody* and *nobody*.



Potential Creators	Everybody	
Potential Starters	Group Search	
	GroupID	*

Staff role: Potential Starters

People assignment criteria: Group Search

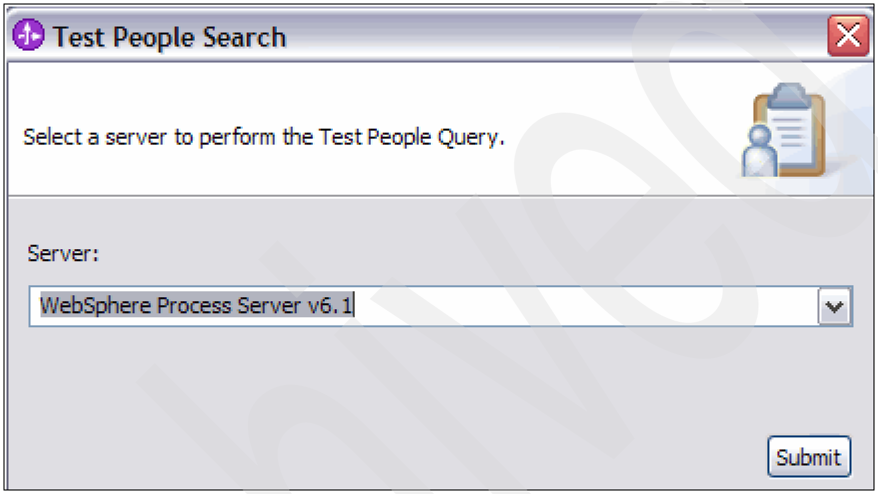
Assigns members of groups that match specified attribute values. Supported by default configurations for: Virtual Member Manager, LDAP, User Registry.
Use this to find one ore more groups and create individual assignments for the members of these groups.
Define attribute values to match Virtual Member Manager group entity property values or LDAP group entry attribute values. Wildcards (*) are permitted.

Name	Value
GroupID	*
Type	
IndustryType	

Figure 4-61 Testing people assignment criteria

4. Select one of the running WebSphere Process Server instances from the drop-down menu on the Test People Search window (shown in Figure 4-62), and click **Submit**.

> **Note:** If the people assignment criteria contains one or more replacement variables, the dialog box includes an input field for each variable to enter some input data to be used in the query.



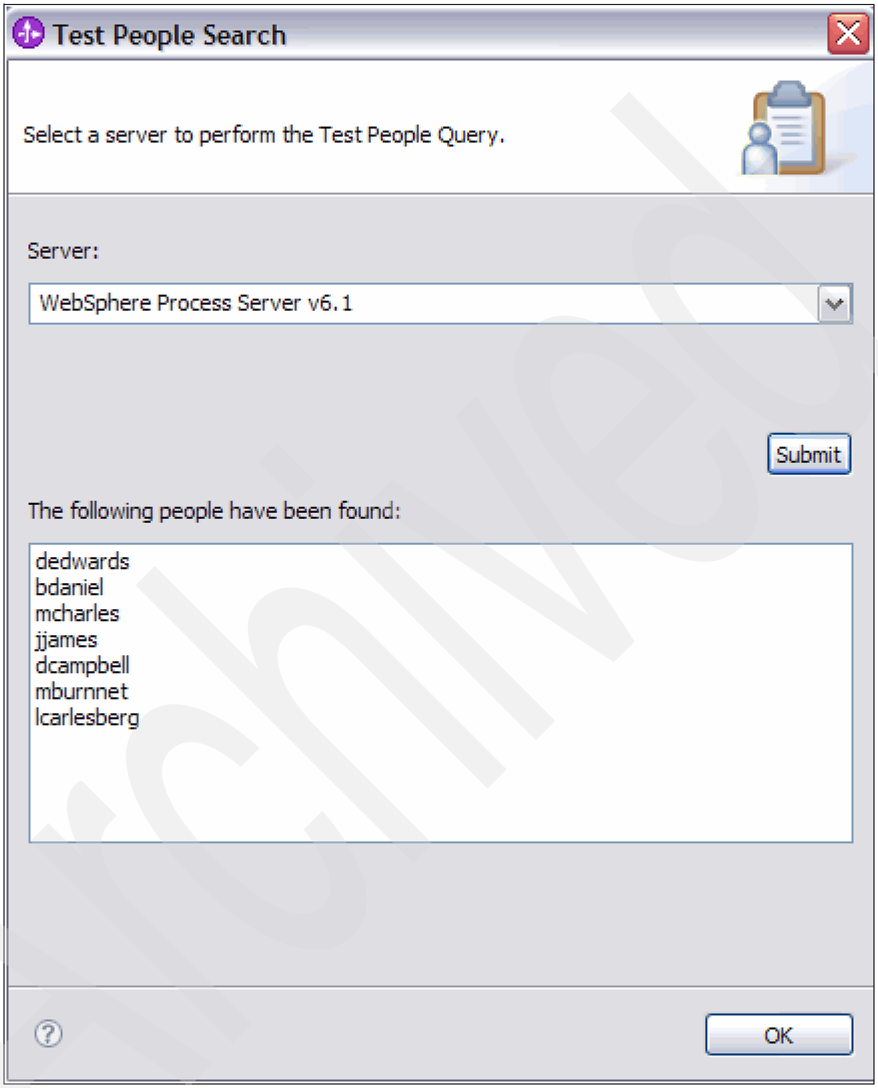
Test People Search

Select a server to perform the Test People Query.

Server: WebSphere Process Server v6.1

Figure 4-62 Test people search window

5. Figure 4-63 shows an example of the query results. Click **OK** to close the window.



Select a server to perform the Test People Query.

Server: WebSphere Process Server v6.1

The following people have been found:

dedwards
bdaniel
mcharles
jjames
dcampbell
mburnnet
lcarlesberg

Figure 4-63 People names queries in test people search

4.5.6 User interfaces

The User Interface area in the human task editor lists the clients that are used by people to interact with the task in the runtime environment. Click the “Add definitions to be used by clients” icon (✚) to add a new user interface, and then configure it in the Details tab of Properties view.

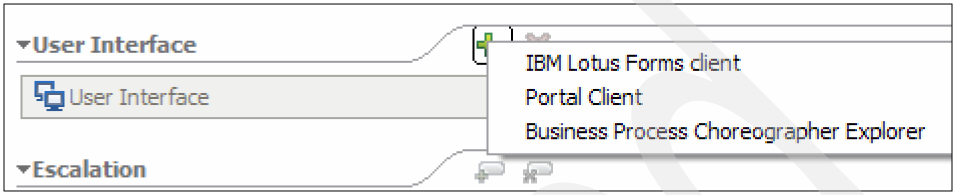


Figure 4-64 User interface clients for a human task

There are three user interfaces:

- *IBM Lotus Forms client*: Select IBM Lotus Forms client to present information to the user with the Lotus Forms Client.
- *Portal client*: Select the portal client to specify a client that is executed on WebSphere Portal.
- *Business Process Choreographer Explorer*: Select the Business Process Choreographer Explorer to use the standard Web client that is provided with the product.

4.5.7 Escalations

An *escalation* is a notification or an alert that can be raised automatically when a human task has not been actioned in the specified amount of time. Escalations for human tasks are optional but are very useful to ensure that tasks do not go overdue or get forgotten. For example, you can use an escalation to alert a supervisor or manager when a staff member is unable to complete a task by the defined deadline.

Escalations allow WebSphere Process Server run time to request an action automatically if a task has not been actioned (for example, claimed and completed) in a certain amount of time.

> **Note:** The *escalation time values* in WebSphere Process Server human tasks are not the same as the *task duration values*. Although a human task might exceed its duration and so become due, this does not trigger an escalation event automatically if no escalation details are defined.
>
> If you want a task to be escalated, you must explicitly define an escalation for it.

Escalations gets activated at a certain task state and escalate only if the task is not moved to the expected task state when the defined time limit for the escalation expires. You can set up escalations to escalate a task to users or groups that are defined by any of the staff verbs and can use varying durations. The time period until the escalation is calculated from the point of time when the task is set to the particular state.

You can define escalations for tasks, and they get activated when the task reaches the task states shown in Figure 4-65.

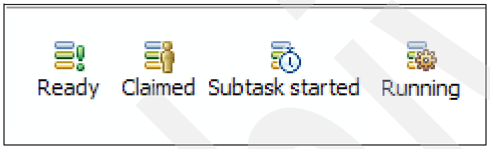


Figure 4-65 Escalation task states

These task states are as follows:

- **Ready**

  For tasks in the ready state, you can define escalations for the following situations:

  – Escalate when the task is not claimed in time using the expected task state of claimed.

  – Escalate when the task is not completed in time using the expected task state of ended.

- **Claimed**

  For to-do tasks or collaboration tasks in the claimed state, you can define escalations for the following situations:

  – Escalate when the task is not completed in time using the expected task state of ended.

  – Escalate when the subtasks of this task are not completed in time using the expected task state of all subtasks ended.

- **Subtask started**

  In the subtasks started sub-state of a to-do task or collaboration task, you can escalate the task when its subtasks are not completed in time using the expected task state of all subtasks ended. Note that it is the parent task that gets escalated, not the subtasks. For more information about subtasks, see 4.5.8, “Ad-hoc tasks” on page 209.

- **Running**

  In the running state of an invocation task, you can escalate when the invoked service does not return in time using the expected task state of ended.

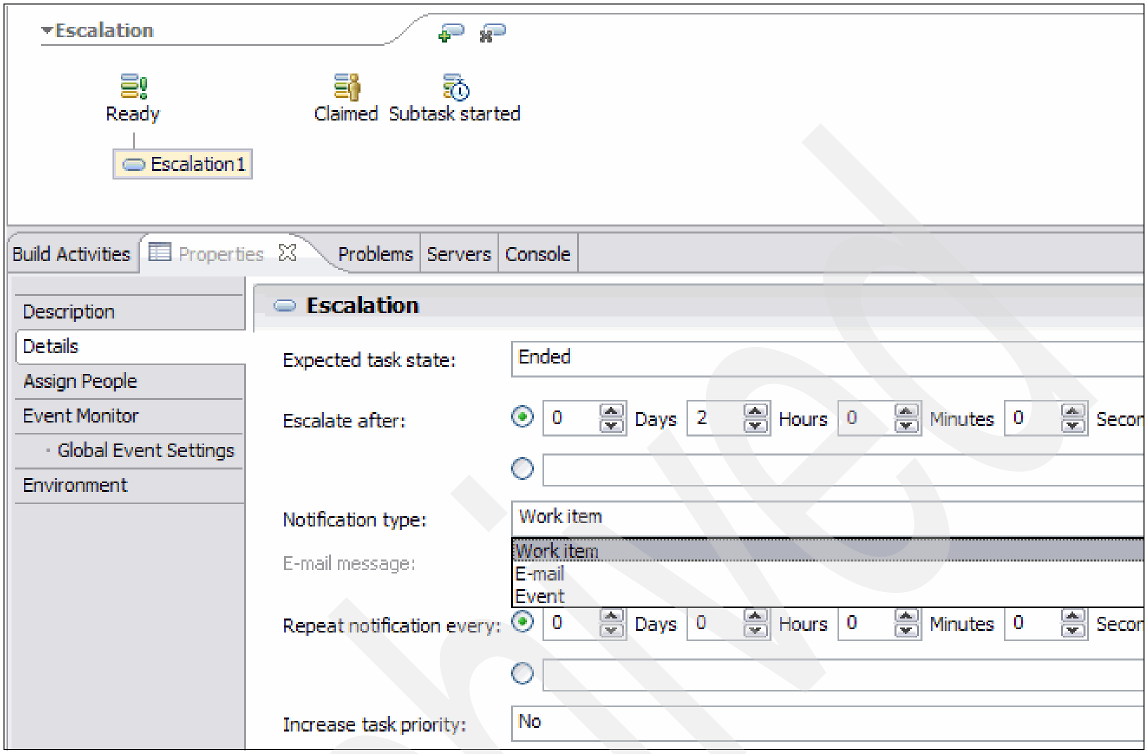
When an escalation is raised, the people that are affected by the escalation (the escalation receivers) receive work items. Depending on the definition of the escalation, the escalation receivers might also receive an e-mail that notifies them that the task is escalated. The list of users who are notified is defined by people assignment criteria on the Assign People tab in the Properties view of the escalation.

The e-mail functionality is especially beneficial when you are notifying users who only occasionally interact with the business process and alerts them to the fact that they have a task to perform in the form of an escalation. The e-mail can contain the escalated task name, the state that the task should have reached, and a description of the task.

You can define repeating escalations. These escalations check the same expected task state at every timeout, and perform the defined escalation action until the expected task state is reached.

You can configure the escalation to increase the priority of the escalated task using the increase task priority property. The priority can be increased automatically either for this time only or for every iteration of the escalation.

These configuration properties are shown in Figure 4-66.



Escalation

Expected task state: Ended

Escalate after: ● 0 Days 2 Hours 0 Minutes 0 Secon

○

Notification type: Work item

- Work item
- E-mail
- Event

E-mail message:

Repeat notification every: ● 0 Days 0 Hours 0 Minutes 0 Secon

○

Increase task priority: No

Figure 4-66 Escalation properties

4.5.8 Ad-hoc tasks

Ad-hoc tasks are created “on-the-fly” in the runtime environment and represent parts of the original task or partially completed task. These tasks can be used when the application is stand-alone and when the task is either to-do or collaboration.

When you work on a task in the user interface of the runtime environment, you can define a task dynamically either as a subtask or as a follow-on task.

Subtasks

In the runtime environment, if a person who claims a task finds that the task cannot be completed, that person can delegate portions of that original task to other people in the form of *subtasks*. You can create subtasks from stand-alone task templates that are stored in the Business Process Choreographer database, from task templates created at run time, or by providing a new task model at run time. The parent task can be a to-do task or a collaboration task, and it must have the supportsSubtask attribute set to true. The subtasks that you create can be either collaboration tasks or invocation tasks. These subtasks can, in turn, have subtasks or follow-on tasks.

There can be more than one subtask for any given parent task and the parent task owner has the control over the overall result. The parent task owner must then consolidate the information received from the subtasks back into the parent task. Escalation is still available on the parent task, and if the parent task fails to complete, the owner can see that it is because there are still subtasks to complete. So the owner of the parent task still has responsibility for the completion of the parent task.

Follow-on tasks

In the runtime environment, if a person who claims a task finds that the task cannot be completed, that person can assign the remaining work to other people in the form of a *follow-on task*. Follow-on tasks can be only collaboration tasks. You can start a collaboration task from a to-do task or a collaboration task that has the supportsFollowOnTask attribute set to true.

The follow-on task is similar in concept to the subtask; however, the data in the follow-on task must match the data in the original task from which you are following. This time, the original task is put into a forwarded state, and the original task owner does not get the response back. On completion of the follow-on task, the process continues as usual. Also, the follow-on task can have its own escalation. So, in this case the owner of the follow-on task has responsibility for completing the follow-on task.

4.5.9 New human task features in WebSphere Process Server V6.1

The new or enhanced human task capabilities for business integration applications available with WebSphere Process Server V6.1 include:

- Extended people directory support

  The Human Task Manager uses VMM as a people directory. *VMM* is a new component in WebSphere Application Server V6.1 to integrate customer specific people directories. This component provides the following functionality:

  - A common schema that offers powerful people query functionality.
  - Various external repositories.
  - People repository federation, that is multiple directories are represent as a single directory with unified schema. For example, it allows federating multiple LDAP directories.

  - Extending the built-in schema by adding additional people properties. Data from schema extensions is stored in VMM's look-aside repository.
  - Plugging in custom repositories with powerful query functionality

- Pre-configured people directory

  The WebSphere Integration Developer test environment allows for rapid prototyping of human tasks. This functionality is built upon VMM's file repository and supports testing of people resolution in the integrated test environment.

- The ability to test the people assignment criteria in the runtime environment.
- Participant substitution capabilities

  Substitution capabilities allow users of the runtime applications to temporarily delegate work when they are unavailable. Human Task Manager allows users to specify substitutes for users.

  People can specify a list of substitutes. If users notify the system of their absence, then work is assigned to substitutes:

  - If someone is absent, then the tasks for that person are assigned to the appropriate person on that person's substitute list.
  - If the first substitute is absent also, then the second substitute is used, and so on.

  Tasks that are assigned to unavailable users are re-assigned automatically to their specified substitutes. Substitution occurs during people resolution and is re-evaluated upon refresh of people assignments. Substitution takes advantage of VMM and is supported only when using VMM as the people directory.

- Enhanced performance for task list queries

  In systems with large numbers of human tasks (hundreds of thousands) and large numbers of users (many thousands), the performance of task list queries might become an issue. *Materialized views*, a technique known from database management systems, are introduced to optimize the performance of task list queries.

- Batch processing support

  This support allows you to perform operations on batches of human tasks and work items. This operation leads to improved performance when dealing with a large number of items as only one client. Server interaction is required. New bulk HTM APIs are introduced to support batch processing.

- Auto deletion can be restricted (optional) to tasks that completed successfully.

- Forms created using IBM Lotus Forms Designer (integrated into WebSphere Integration Developer) can be used as the user interface for human tasks and processes.

4.6 Administering processes and tasks

Business processes and human tasks are part of modules that are deployed and installed as part of an enterprise application. You can use the administrative console or the administrative commands to administer process templates and task templates, and you can use Business Process Choreographer Explorer to work with process instances and task instances. You can use Business Process Choreographer Observer to report on business processes and human tasks.

4.6.1 Business process templates and instances

Process templates define business processes within an enterprise application that can run on WebSphere Process Server. You can think of the template as the cookie-cutter that lets you create copies of business processes with the same definition. The user who wants to initiate a process must have permission to create an instance of that process from its template. If you want to control who can start and stop your process, you do that by controlling access to its template.

When an enterprise application that contains process templates is installed or deployed and started, the process templates are put into the started state. When a process template is started, new instances of the template can be started. The process template must be stopped before the business process application can be uninstalled. When a process template is in the stopped state, no new instances of this template can be started.

A *process instance* is an instance of a process template and can be a long-running process or a microflow.

The Business Process Choreographer Explorer displays information about process templates and process instances or acts on process instances. These actions can be, for example, starting process instances and, for long-running processes, other process life cycle actions, such as suspending, resuming, or terminating process instances, or repairing activities.

4.6.2 Human task templates and instances

A *task template* contains the definition of a deployed human task that was created using WebSphere Integration Developer or at run time using the Business Process Choreographer APIs.

The template contains properties, such as the task name and priority, and aggregates artifacts, such as escalation templates, custom properties, and people query templates. When an enterprise application that contains task templates is installed or deployed and then started, the task templates are put into the started state.

When a task template is started, new instances of the template can be started. The task template must be stopped before the human task application can be uninstalled. When a task template is in the stopped state, no new instances of this template can be started.

A *task instance* is an instance of a task template and represents a running human task.

4.6.3 Administering process and task templates

You can use the administrative console that is provided by the WebSphere Process Server runtime environment to administer process or task templates.

When an enterprise application that contains process or task templates is installed or deployed and then started, the process or task templates are put into the started state. You can use the administrative console or the administrative commands to stop and start these templates. The process or task templates that are started are shown in Business Process Choreographer Explorer.

Follow these steps to start or stop the templates:

1. Select the module that you want to manage from the administrative console by clicking **Applications** → **SCA modules** → ***<module_name>***.
2. In the Configuration page for the module under Additional Properties:
   a. Click **Business processes**, and then select a process template.

      or

   b. Click **Human tasks**, and then select a task template.
3. To start the template, click **Start**. To stop the template, click **Stop**.

Existing instances of the stopped templates continue to run until they end normally. However, you cannot create instances from a stopped template.

4.6.4 Business Process Choreographer Explorer

The Business Process Choreographer (BPC) Explorer is a Web client that facilitates a user to communicate with the runtime environment of WebSphere Process Server. It provides a user interface for administering business processes and human tasks or to work with assigned tasks.

When you start Business Process Choreographer Explorer, the objects that you see in the user interface and the actions that you can take depend on your user role. You can use Business Process Choreographer Explorer to perform the following tasks:

- If you are a business process administrator, you can manage the life cycle of business processes and human tasks, as well as manage work assignments. You can view information about process and task templates, process instances, task instances, and their associated objects. You can also act on these objects (for example, you can start new process instances, create and start tasks, repair and restart failed activities, manage work items, and delete completed process instances and task instances).
- If you are a business user, you can view and work with your assigned tasks.

You can configure the Business Process Choreographer Explorer using a script or through the WebSphere Process Server administrative console.

Launching Business Process Choreographer Explorer

Note: Before you can start using Business Process Choreographer Explorer from a Web browser, you must have installed the business process container, human task container, and the Business Process Choreographer Explorer application, and the application must be running.

To launch the Business Process Choreographer Explorer:

1. Open the Business Process Choreographer Explorer URL from a Web browser:

   ```
   http://<app_server_host>:<default_host_port_no>/context_root
   ```

   A sample URL in the address bar might be:

   ```
   http://localhost:9080/bpc
   ```

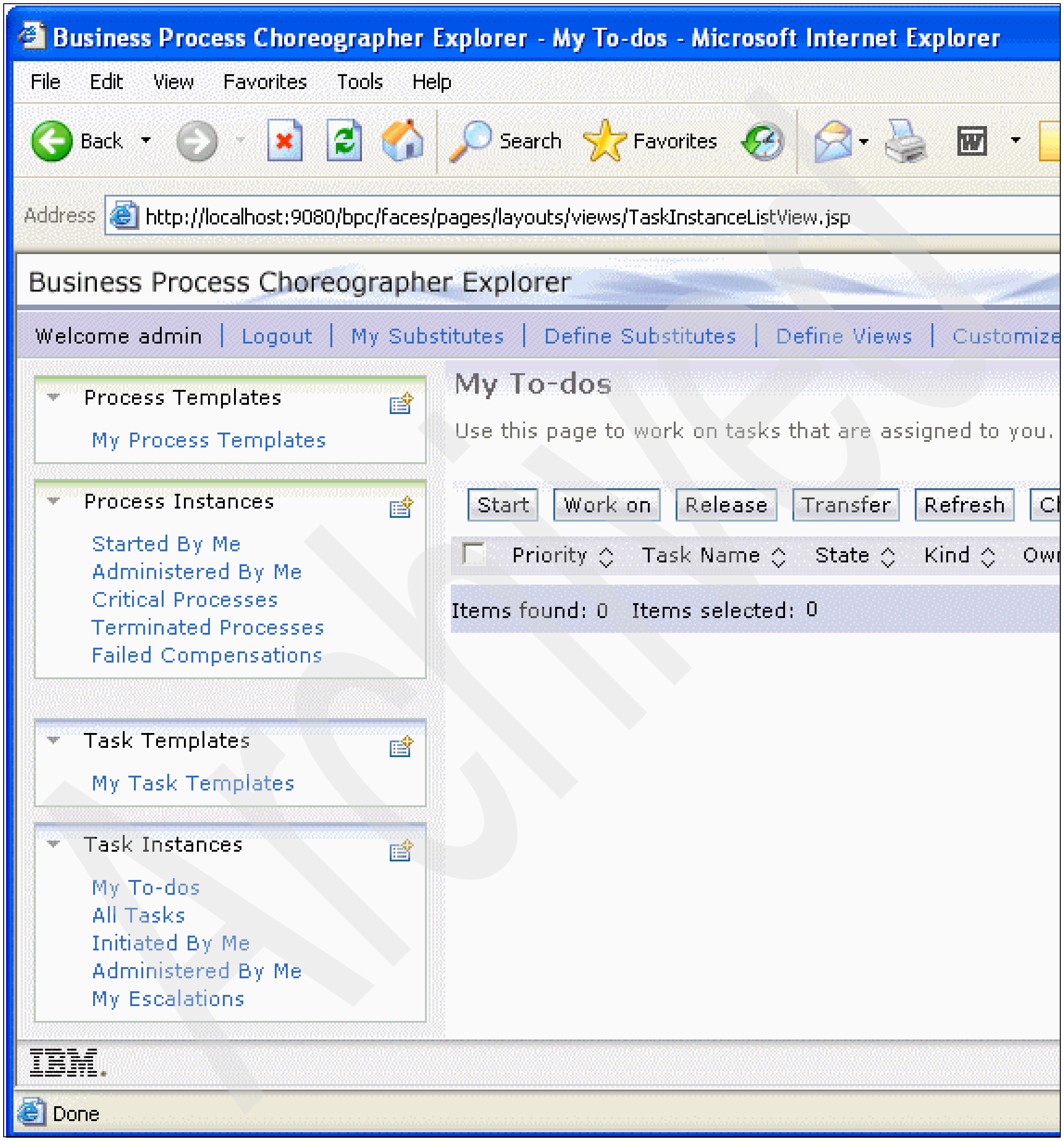
2. If security is enabled, you must enter a user name and password, and then click **Login**.

Launching the Business Process Choreographer Explorer from WebSphere Integration Developer

Alternatively, if you are using WebSphere Integration Developer test environment, you can perform the following steps to start the Business Process Choreographer explorer:

1. Switch to the Business Integration perspective.
2. Click the Servers view tab. Otherwise, select **Windows** → **Show View** → **Servers**.
3. Select a server and start it.
4. Right-click the server and select **Launch** → **Business Process Choreographer Explorer**.

The initial page of the Business Process Choreographer Explorer displays as shown in Figure 4-67.


Business Process Choreographer Explorer - My To-dos - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address

http://localhost:9080/bpc/faces/pages/layouts/views/TaskInstanceListView.jsp

Use this page to work on tasks that are assigned to you.

Items found: 0 Items selected: 0

Figure 4-67 My To-dos page of the Business Process Choreographer Explorer

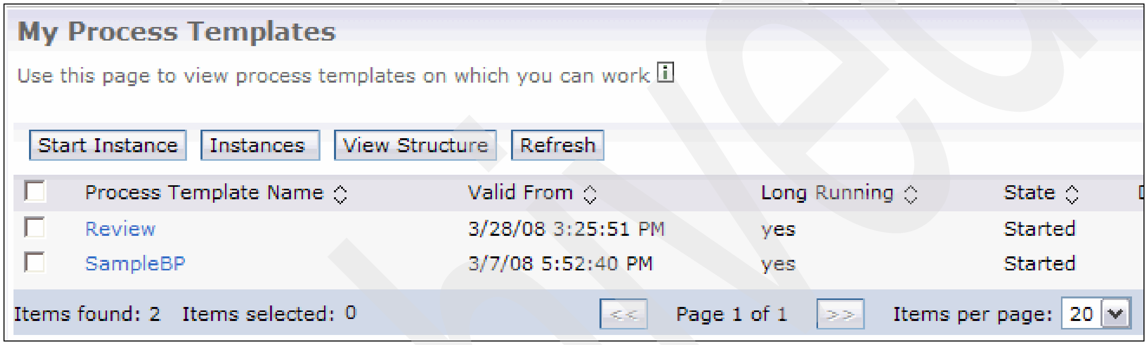
Process and task templates

The process and task templates that are started are shown in Business Process Choreographer Explorer.

My Process Templates view

This view under the process templates group lists the process templates. From this view, you can view the information about the process template and its structure, navigate to running process instances that are associated with a template, and start process instances.

Figure 4-68 shows the My Process Templates view of Business Process Choreographer Explorer.



My Process Templates

Use this page to view process templates on which you can work

□	Process Template Name	Valid From	Long Running	State
□	Review	3/28/08 3:25:51 PM	yes	Started
□	SampleBP	3/7/08 5:52:40 PM	yes	Started

Items found: 2 Items selected: 0 Page 1 of 1 Items per page: 20

Figure 4-68 My Process Templates view of Business Process Choreographer Explorer

My Task Templates view

This view under the task templates group lists the task templates. From this view, you can create and start a task instance and navigate to running task instances that are associated with a template.

4.6.5 Administering process and task instances

The various views that are available under process instances and task instances groups on the navigation pane of the Business Process Choreographer Explorer facilitate the administration of process and task instances.

Process instances

The process instances group includes the following views:

- Started By Me

  This view lists the process instances that you started. From this view, you can monitor the current state or progress of the process instance and list the activities, processes, or tasks that are related to it.

- Administered By Me

  This view lists the process instances that you are authorized to administer. From this view, you can act on the process instance (for example, suspend and resume a process or monitor the progress of the activities in a process instance).

- Critical Processes

  This view lists process instances in the running state that contain activities in the stopped state. From this view, you can act on the process instances or list the activities and then act on them.

- Terminated Processes

  This view lists process instances that are in the terminated state. From this view, you can act on these process instances.

- Failed Compensations

  This view lists the compensation actions that have failed for microflows.

Task instances

The task instances group includes the following views:

- My To-dos

  This view lists the task instances with which you are authorized to work. From this view, you can work on a task instance, release a task instance that you claimed, or transfer a task instance to another user.

- All Tasks

  This view lists all of the tasks for which you are the owner, potential owner, or editor. From this view, you can work on a task instance, release a task instance that you claimed, or transfer a task instance to another user.

- Initiated By Me

  This view lists the task instances that you initiated. From this view, you can work on a task instance, release a task instance that you claimed, or transfer a task instance to another user.

- Administered By Me

  This view lists the task instances that you are authorized to administer. From this view, you can act on the task instance (for example, suspend and resume a task, create work items for the task instance, or display a list of the current work items for the task instance).

- My Escalations

  This view lists all the escalations for the logged on business user.

4.6.6 Business Process Choreographer Observer

The Business Process Choreographer (BPC) Observer is a Web application that generates reports about the execution of business processes and human tasks. The Business Process Choreographer Observer allows you to:

- Observe state and evolution of processes.
- Observe overall duration and actual work time of activities.
- Provide customizable reports and graphical charts of historical and accumulated data of processes.
- Retrieve statistical data on processes and activities through flexible drill downs.

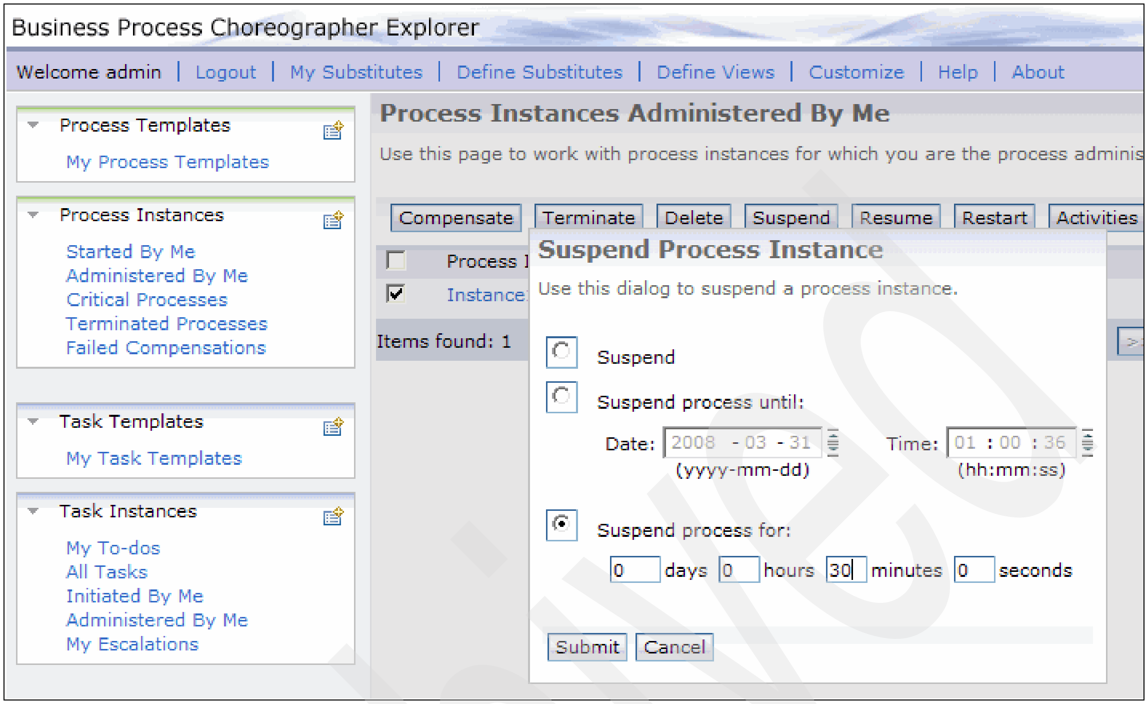
4.6.7 New, enhanced Business Process Choreographer features with WebSphere Process Server V6.1

This section describes the new or enhanced features of Business Process Choreographer Explorer and Business Process Choreographer Observer that are available with WebSphere Process Server V6.1.

The new and enhanced features include:

- You can now perform the following tasks with the Business Process Choreographer Explorer enhanced capabilities:
  - Handle absence and substitution of users.
  - Use the “Suspend until” option for business processes and human tasks.

    In V6.1, suspending a process or task instance allows you to specify a duration or point in time, if the process or task instance has to be resumed automatically. Figure 4-69 shows this option.



Process Instances Administered By Me

Use this page to work with process instances for which you are the process adminis

□ Process I

☑ Instance

Items found: 1

Suspend Process Instance

Use this dialog to suspend a process instance.

○ Suspend

○ Suspend process until:

Date: 2008 - 03 - 31 (yyyy-mm-dd) Time: 01 : 00 : 36 (hh:mm:ss)

● Suspend process for:

0 days 0 hours 30 minutes 0 seconds

Figure 4-69 Suspend process until option for a process instance in BPC explorer

- – View and edit XML source data.
- – Use extended custom views to sort and control the amount of data that is presented to the application users.
- – Combine filter criteria across processes and tasks with their definitions and instances.
- – Navigate between related tasks (subtasks and follow-on), as well as administer and view information about specific ad-hoc tasks.
- – Include human task priority and business category as filter criteria and list columns.
- – Edit custom properties.
- – Use an improved graphical process view.

- ► You can now export reports in the Business Process Choreographer Observer for further analysis in tools such as Microsoft® Excel®, and you can save the reports for automatic generation at a later time based on schedule or on demand.

4.7 Business state machines

A *state machine* is an event driven business application in which external operations trigger changes that guide the state machine from one discrete mode to another. Each mode is an individual state, and this mode determines what activities and operations can occur. The state machines are service components which represent business processes based on states and events instead of a sequential business process model.

You create business state machines in IBM WebSphere Integration Developer using the business state machine editor.

An example of a state machine

In this section, we describe a a simple example to understand business state machine simulation and usage—purchasing a snack from a vending machine. Figure 4-70 on page 222 shows the various interactions and activities that are involved during this purchase process.

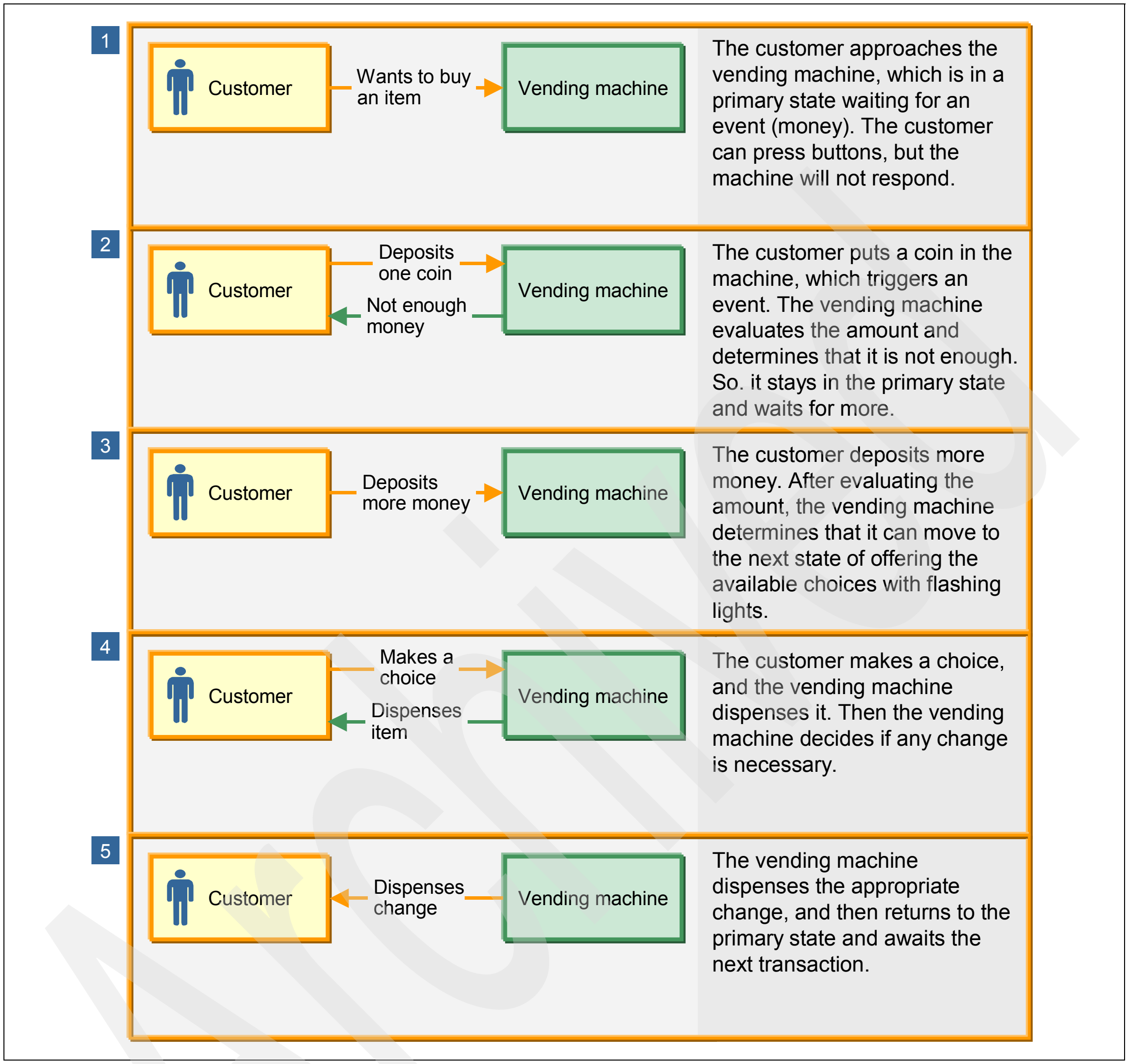

1
Customer
Wants to buy an item
Vending machine
The customer approaches the vending machine, which is in a primary state waiting for an event (money). The customer can press buttons, but the machine will not respond.
2
Customer
Deposits one coin
Not enough money
Vending machine
The customer puts a coin in the machine, which triggers an event. The vending machine evaluates the amount and determines that it is not enough. So. it stays in the primary state and waits for more.
3
Customer
Deposits more money
Vending machine
The customer deposits more money. After evaluating the amount, the vending machine determines that it can move to the next state of offering the available choices with flashing lights.
4
Customer
Makes a choice
Dispenses item
Vending machine
The customer makes a choice, and the vending machine dispenses it. Then the vending machine decides if any change is necessary.
5
Customer
Dispenses change
Vending machine
The vending machine dispenses the appropriate change, and then returns to the primary state and awaits the next transaction.

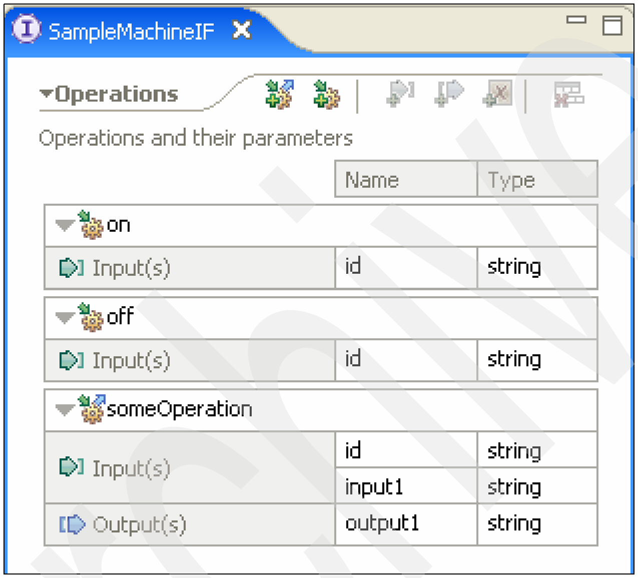
Figure 4-70 State machine use

In this example, a customer must go through a sequence of interactions to purchase a snack from a vending machine. The vending machine does not respond until money is deposited. Even then, the vending machine does not proceed to the next state until a certain amount of money is deposited. In each of the states, the actions that the customer can perform is unique from any other state. For example, when it is in the primary state waiting for money, the customer can press buttons, but doing so has no effect on the transaction.

This example helps you to understand the building blocks of a state machine that we discuss in subsequent topics. In the last topic, 4.7.7, “Vending machine sample” on page 236, we show the implemented state machine for this vending machine example.

4.7.1 Creating a state machine

This example assumes that a module called *SampleModule* and an interface for the module called *SampleMachineIF* have been created as shown in Figure 4-71.



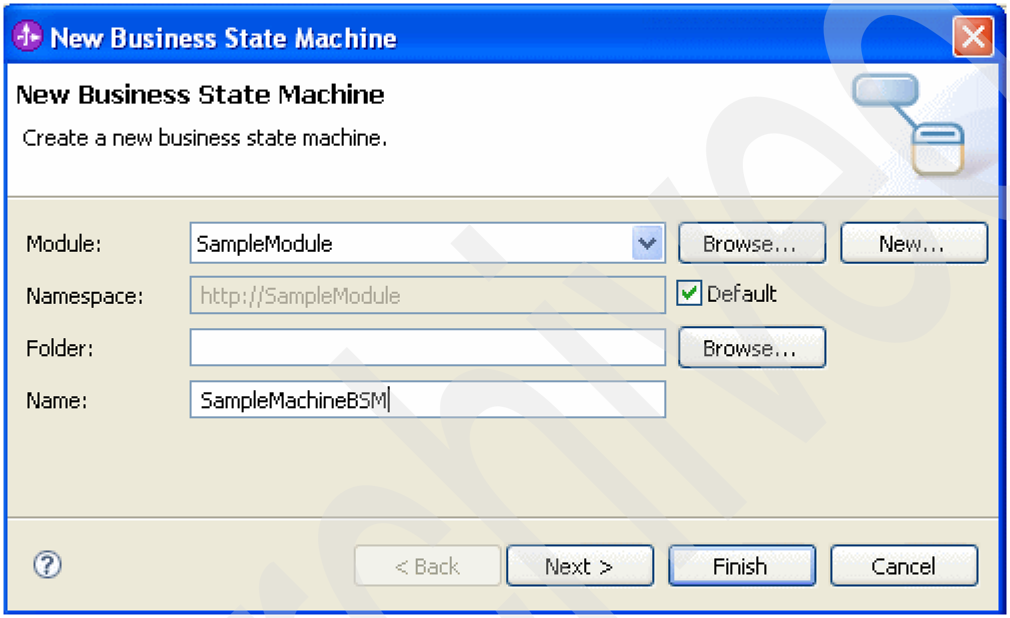
Operations and their parameters

	Name	Type
on		
Input(s)	id	string
off		
Input(s)	id	string
someOperation		
Input(s)	id	string
	input1	string
Output(s)	output1	string

Figure 4-71 SampleMachineIF for the state machine

Follow these steps to create a new state machine:

1. Select **File** → **New** → **Business State Machine**.
2. In the New Business State Machine window (shown in Figure 4-72):
   a. Browse to an existing module (or click **New** to create one).
   b. Specify a folder (optional).
   c. Specify a name for the new state machine (for example, SampleMachineBSM).
   d. Click **Next**.



New Business State Machine

New Business State Machine

Create a new business state machine.

Module: SampleModule

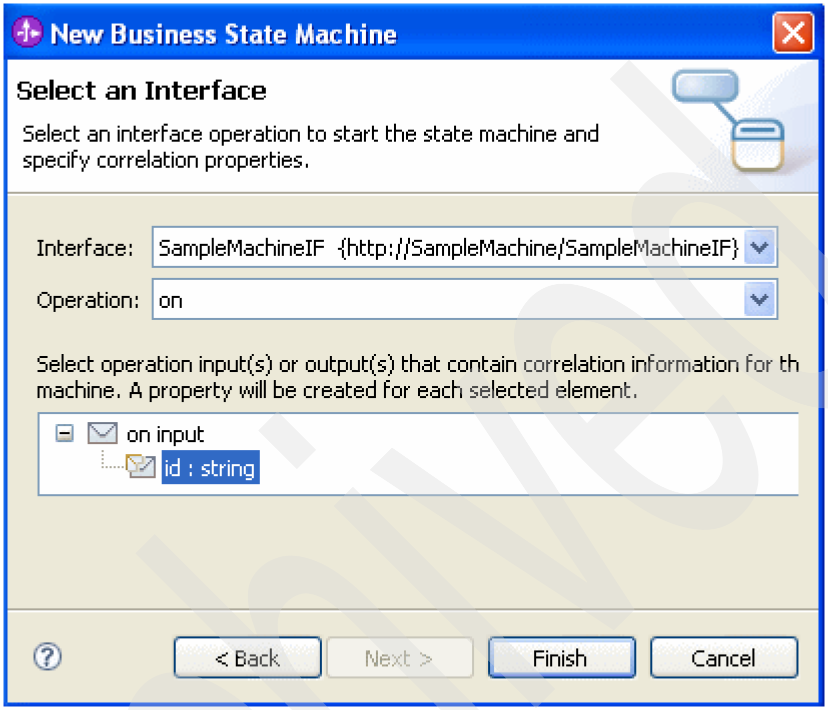
Namespace: http://SampleModule ☑ Default

Folder:

Name: SampleMachineBSM

Figure 4-72 New Business State Machine wizard

3. Select an appropriate interface and operation to start the state machine, and then select the necessary operation inputs and outputs for correlation information as shown in Figure 4-73. Click **Finish**.



Select an Interface

Select an interface operation to start the state machine and specify correlation properties.

Interface: SampleMachineIF {http://SampleMachine/SampleMachineIF}

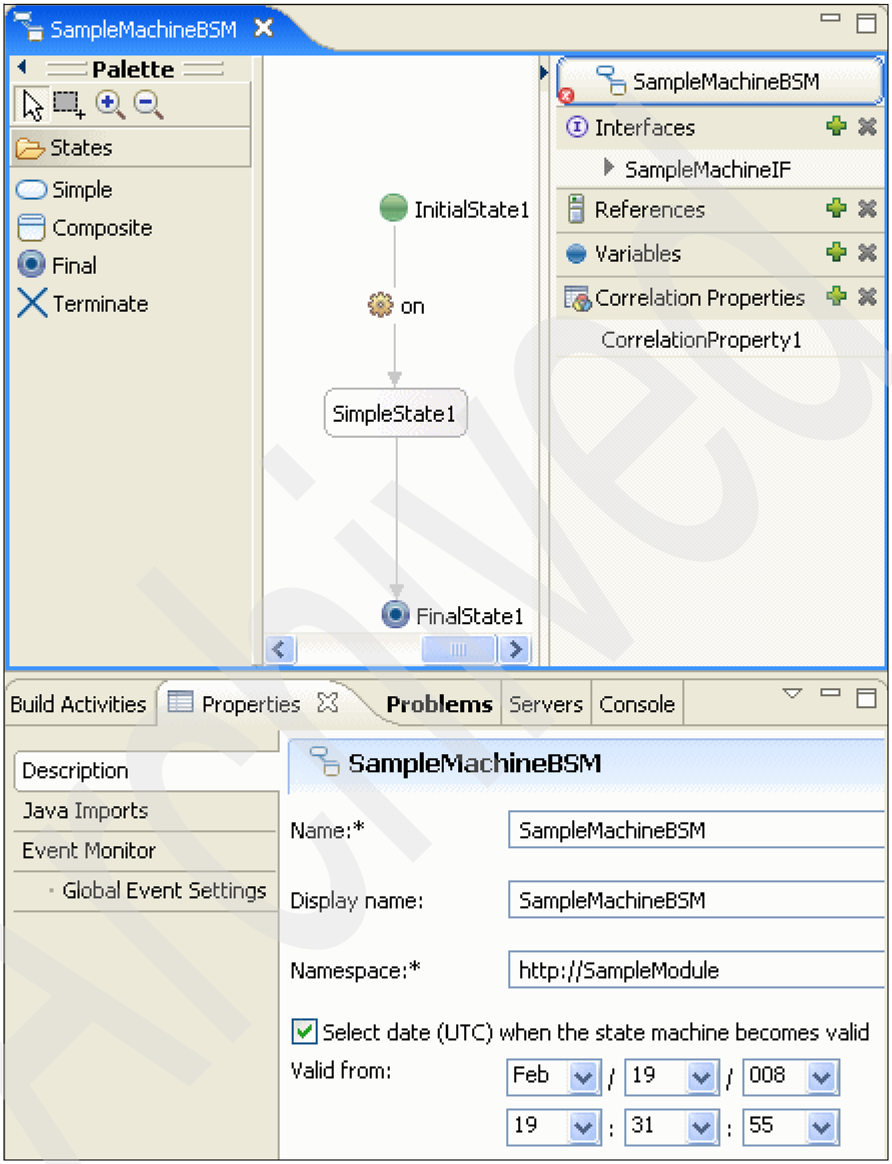
Operation: on

Select operation input(s) or output(s) that contain correlation information for th machine. A property will be created for each selected element.

on input
id : string

Figure 4-73 Select an interface for new business state machine

The business state machine is created and opens in the business state machine editor as shown in Figure 4-74.



Name:* SampleMachineBSM

Display name: SampleMachineBSM

Namespace:* http://SampleModule

☑ Select date (UTC) when the state machine becomes valid

Valid from: Feb / 19 / 008

19 : 31 : 55

Figure 4-74 Business state machine editor

4.7.2 Business state machine editor

You can use the business state machine editor to build and manipulate state machines. This editor consists of several distinct areas, including the canvas, palette, tray, Properties view, and action bar. The white space is called the *canvas*. It is populated with objects pulled from the palette on the left and with references to the objects in the tray on the right to build a state machine. The Properties view displays pertinent details about whatever object is currently selected on the canvas or the tray.

The only two areas whose functionality and appearance differ when compared with those of the business process editor (as explained in the topic, 4.4.4, “Business process editor” on page 146) are the palette and the action bar, which we describe in the sections that follow.

The palette

The *palette* is the shaded area to the left of the canvas that houses the states that you click and drag onto the canvas to build a state machine. The icons that are available on the palette represent various states of a state machine, as shown in Figure 4-75.

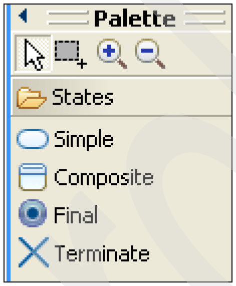
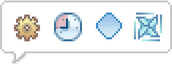


Figure 4-75 States on the palette of the business state machine editor

The action bar

The *action bar* is a miniature dialog box that displays beside certain objects when you select them. It contains a series of one or more elements that can be added to that object. For example, this dialog box, , displays when a transition is selected.

4.7.3 Building blocks of a state machine

You use the following building blocks to compose a business state machine using the business state machine editor:

- *Interface*: A set of operations to which the state machine accepts and responds.
- *Reference*: Not an interface, but instead it tells the state machine where to find operations that it can invoke. More specifically, it points to the interface that is used in the invocation of another component.
- *Variables*: Store the data that is used within a state machine and can be either a business object or a simple type.
- *Correlation properties*: Used to distinguish one instance of a state machine from another within a runtime environment.
- *State*: One of several discrete individual stages that represent a business transaction.
- *Transition*: The movement from one state to the next by recognizing an appropriate triggering event. The movement is based on the evaluation of the conditions that are necessary for control to flow through it. During the transition, the associated actions are executed.
- *Event*: What (an external prompt) triggers a transition from one state to another.
- *Condition*: Guards the transition and only allows processing when and if it evaluates to true. Otherwise, the current state is maintained.
- *Action*: An activity that is executed when a state is entered, exited, or on a transition within a business state machine.

We discuss interface, reference partners, and variables in detail in 4.4, “Business processes” on page 134. The following sections provide more information about artifacts that are either unique in state machines or that must be configured in a different manner than in business processes.

4.7.4 Using correlation properties in a state machine

You use *correlation properties* to distinguish one instance of a state machine from another within a runtime environment. For each operation (event) to which the state machine responds, a property alias locates the input that corresponds to each correlation property that is defined.

This correlation is done in two steps:

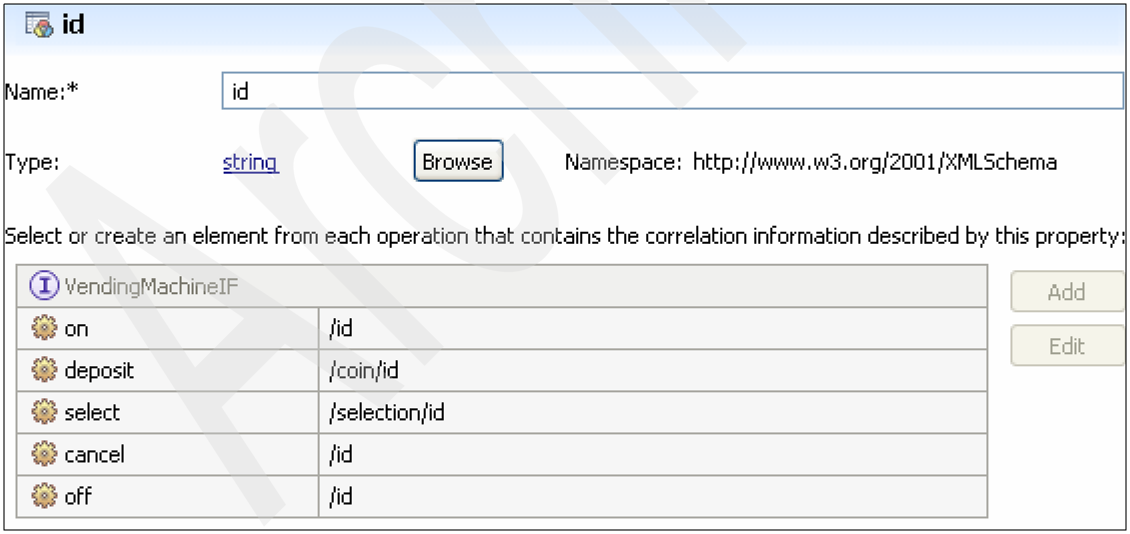
1. Identify a property that will be present in all operations.
2. Identify, for each operation, the part that will be used to supply the value for the property.

Adding a correlation property to a state machine

To create a correlation property in a state machine using the business state machine editor:

1. Click the plus icon ✚ besides the Correlation Properties section on the tray.
2. In the Add Correlation Property dialog box:
   a. Specify an appropriate name.
   b. Select the data type.
   c. Click **OK**.
3. Click the Description tab in the properties area, and then:
   a. Select each operation.
   b. Click **Add** and specify property aliases.

Figure 4-76 shows the correlation property that is used by one of the sample business state machine that we discuss in 4.7.7, "Vending machine sample" on page 236.



id

Name:* id

Type: string Browse Namespace: http://www.w3.org/2001/XMLSchema

Select or create an element from each operation that contains the correlation information described by this property:

VendingMachineIF	
on	/id
deposit	/coin/id
select	/selection/id
cancel	/id
off	/id

Figure 4-76 Correlation property of a sample business state machine

4.7.5 Using states in a state machine

A *state* is one of several discrete individual stages that represent a business transaction. An *action* is an activity that is executed when a state is entered or exited or on a transition within a business state machine. Typically, a state has the following life cycle:

- The state begins with the running of any existing entry actions.
- The state then waits and listens to a particular set of events.
- When an event occurs, an appropriate path through the state machine is chosen.
- An exit action, if any is executed before the state machine transitions to another state.

The various types of states are:

- Initial state

  Initial state is the first state in which a state machine starts. When used in a primary state machine, an initial state has one outbound transition that defines the operation that starts the state machine. When used in a composite state, the outbound transition must be automatic. In both cases, there can be only one outbound transition, and it cannot be guarded.

- Simple state

  Simple state is an individual discrete stage of a state machine. This state can have an action associated with state entry and exit. These actions occur whenever the state is entered or exited, respectively. When the state is entered, it enables all of its outbound transitions.

- Composite state

  Composite state is an aggregate of two or more states. This state is used to decompose a complex state machine diagram into an easy to comprehend hierarchy of state machines or to facilitate exception and error handling. These composite states can have one or more outbound exception transitions or a single default transition.

- Final state

  Final state is the state where the state machine comes to a normal or expected completion. When it is contained in a composite state, the movement to final state fires the composite state's default transition. This state can have only an entry action not an exit action.

- Terminate

  Terminate is the state where the state machine comes to an abnormal or unexpected end. When this state is reached, entire activity within the state machine halts. This state can have only an entry action but not an exit action.

Adding a state to a state machine

To add a state to a state machine using the business state machine editor:

1. Click a state icon on the palette.
2. Drag the cursor onto the canvas. The icon beside the cursor has a plus symbol when you are at a place where you are allowed to drop the state. When the cursor becomes a crossed out circle, continue moving the cursor until it becomes a plus sign again.
3. Click the area of the canvas where you want to drop the state.

Figure 4-77 shows a simple state, SimpleState1, added to a state machine.

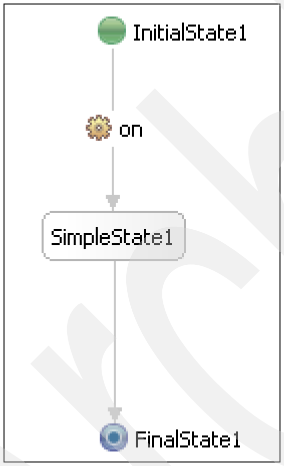


Figure 4-77 SimpleState of a state machine

Adding an entry or an exit to a state

An *entry* is an activity which is executed when entering a state, while the *exit* is executed while leaving the state. To add an entry or an exit to a state or a composite state using the business state machine editor, follow these steps:

1. Click the state for which you want to add an entry or an exit.
2. In the action bar, click the **Add an Entry** or **Add an Exit** icons that display as shown in Figure 4-78.

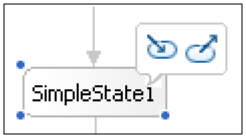
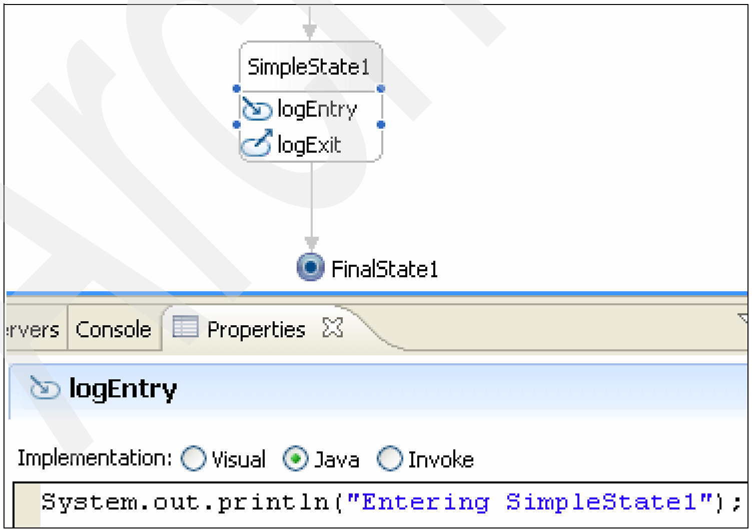


Figure 4-78 Action bar showing Entry and Exit icons

3. Click the Details tab in the Properties area.
4. Select one of the following options to implement the entry or exit:
   - **Visual**: Choose this option to use the visual snippet editor to graphically compose Java code.
   - **Java**: Choose this option to write the Java code yourself.
   - **Invoke**: Choose this option to invoke an operation on a reference.

Figure 4-79 shows a state with entry and exit actions.



Implementation: ○ Visual ● Java ○ Invoke

```
System.out.println("Entering SimpleState1");
```

Figure 4-79 A sample state with entry and exit

4.7.6 Using transitions in a state machine

A *transition* is the movement from one state to the next by recognizing an appropriate triggering *event*. The movement is based on the evaluation of the *condition* that is necessary for control to flow through it. During the transition, the associated actions are executed.

Events

An *event* is what (an external prompt which) triggers a transition from one state to another. There are three types of events:

- *Call event*: The transition is triggered when the correct operation is called.
- *Timer event*: The transition is triggered when the timer expires.
- *Completion event*: The transition is triggered when the source state is entered and its entry action, if any, is executed.

The call events are the operations, so the events can have attributes. The attributes are the input parameters of the operation. If the transition has an action, these attributes are available to the action.

Conditions

A *condition* guards the transition and allows movement to the next state only if the condition evaluates to true. Otherwise, the current state is maintained.

Life cycle guiding principles

A transition has a life cycle that is characterized by the following principles:

- A transition is disabled when the current state of the state machine is any other state than the source state of the transition. While disabled, a transition will ignore any instances of its trigger event.
- A transition is enabled when the state machine enters the source state of the transition.
- A transition is triggered when it is enabled, and an instance of its triggering event occurs.
- A transition is fired (followed) when it is triggered and either has no guard condition or the condition evaluates to true.

An event can trigger more than one transition at a time, but only one of those transitions fires. If an event triggers more than one outbound transition from a given state, the order that the transitions are checked is undefined. In the case of nested composite states, if an event triggers transitions in more than one composite state, the transitions are checked from the innermost composite state to the outermost composite state.

Adding a transition to a state machine

Follow these steps to add a transition to a state machine using the business state machine editor:

1. Click a state on the canvas, which is the source state for transition.

   Alternative step: Move the cursor over the state (on the canvas) until the yellow “lollipop” appears above it.

2. Right-click the selected transition, and select **Add** → **Transition**.

   Alternative step: Hold the left mouse button, and drag the cursor onto the canvas. The cursor icon becomes a crossed-out circle.

3. Drag the cursor out over the canvas. When you hover over a valid target, the crossed-out circle disappears.

4. Click the state that is the target state for this transition. A dark grey arrow displays on the canvas linking the source and target states.

5. To change the visual appearance of transition on the canvas, right-click the transition, select **Layout**, and choose one of the following options:

   – **Rectilinear**: Choose this setting to have the transition drawn automatically with the bends at 90 degree angles.

   – **Bendpoint**: This option is the default setting and automatically chooses the shortest and most direct path. You can adjust this path manually by selecting the transition, clicking the tiny black box that displays at the midpoint of the transition, and then dragging it to a new location. Two new black boxes display at the mid-point of each half, and you can adjust these as well until the transition displays on the canvas as you want.

> **Note:** You see only one of these options at a time. When one option displays on the menu, then the other one is, by default, the active setting that is used for the layout.

Figure 4-80 shows a transition with rectilinear layout, between SimpleState1 and SimpleState2.

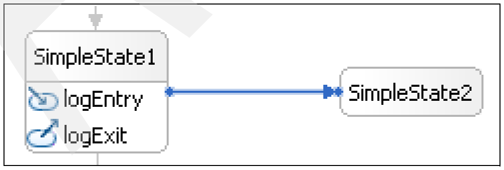


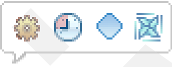
Figure 4-80 A sample transition between two states

Configuring transitions

You can add the following elements to a transition:

- *Operation*: An operation (call event) triggers the movement from one state to another.
- *Timeout*: A timeout imposes a duration or an expiration on a state so that it is not maintained indefinitely while waiting for an operation that might never occur. When the timeout expires, the transition is triggered.
- *Condition*: A condition guards the transition and allows movement to the next state only if the condition evaluates to true.
- *Action*: An action is an activity that is executed on a transition.

To add an operation or timeout, condition, action to a transition using the business state machine editor, follow these steps:

1. Select and hover over the transition on the canvas.
2. In the action bar  click any one of the icons with following options:
   - Add an Operation
   - Add a Timeout
   - Add a Condition
   - Add an Action

   Alternative step: Right-click the selected transition, and elect **Add** → ***<option>***.
3. Configure the operation/timeout or condition or action as necessary in the properties area of the business state machine editor.

Figure 4-81 shows a transition from SimpleState1 to SimpleState2, configured with an event (someOperation), a condition (checkInput), and an action (doThisAction). In this case when someOperation is invoked, the state machine moves from SimpleState1 to SimpleState2 if the checkInput evaluates to true. During this transition, the state machine performs doThisAction.

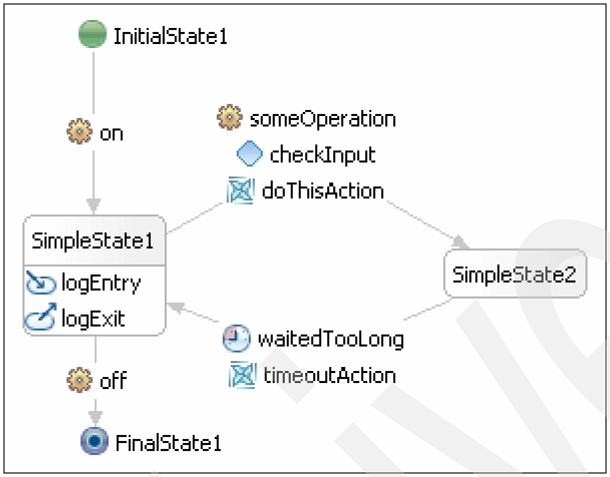


Figure 4-81 Transition configuration

4.7.7 Vending machine sample

This sample demonstrates how a business state machine can be used to simulate a vending machine as shown in Figure 4-82.

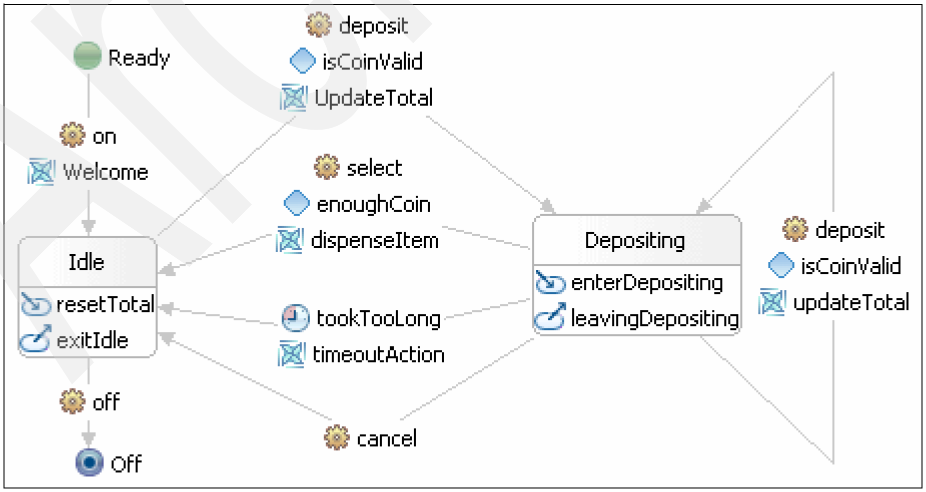


Figure 4-82 Business state machine implementation for a vending machine

When the state machine starts execution at the initial state, *Ready*, the first operation is *on*, and it makes the vending machine operational by displaying a welcome message that shows the available items list.

The state machine then enters the *Idle* state where it waits for an event to happen. There are two operations to which this state can react:

- The *Off* operation, which shuts down the state machine and moves to the final state, *Off*.
- The *deposit* event that signals the arrival of a coin. This event moves the state machine to the *Depositing* state while checking to make sure that it is a real coin and calculating its value.

The primary purpose of the *Depositing* state is to keep track of how much money has been deposited. There are several transitions out of the state, and one that cycles back into it. If the user enters another coin, then a transition checks the validity of the coin, updates the total, and returns the state machine to the *Depositing* state. If the user makes a selection, then another transition confirms that the total amount of money is sufficient to dispense the item, and moves the state machine to the *Idle* state after dispensing the item. Another transition has a timeout that determines if a transaction has taken too long to complete, and moves the state machine into the *Idle* state. The last transition is followed when the user cancels the transaction.

You can access this sample, with detailed instructions to build and test it, from WebSphere Integration Developer help menu. Select **Help** → **Samples / Tutorials (WebSphere Integration Developer)**.

4.8 Business rules

Business rules are service components that declare policy or conditions that must be satisfied within a business process. A business rule is a representation of how business policies or practices apply to a business activity. A rule can enforce business policy, establish common guidelines within an organization, make a decision, or infer new data from existing data.

Business rules externalize business policies, conditions, and values that are used to affect the operation of a business process. These policies, conditions, and values might change over time. They should not be embedded in code, which makes them difficult to change. Business rules make business process applications more flexible to respond quickly to changing business conditions.

IBM WebSphere Integration Developer provides graphical programming environment tools to develop business rules. IBM WebSphere Process Server includes the business rules manager, a Web-based runtime tool for business analysts to update business rules as business needs dictate, without affecting other components or SCA services.

Business rules are combination of conditions and the actions to be performed when a specific condition is met. There are two types of implementations for business rules:

- A *rule set* typically consists of a number of if-then rules, where the action is performed when the condition evaluates to true.
- A *decision table* captures simple rule logic in a table format where in the action to be performed is decided by more than one condition.

Rule sets and decision tables do not exist in isolation as a components for module assembly. They must be contained within a rule group. It is the rule group that exposes the interface and determines which set of rules are executed.

4.8.1 Cashback business rule sample

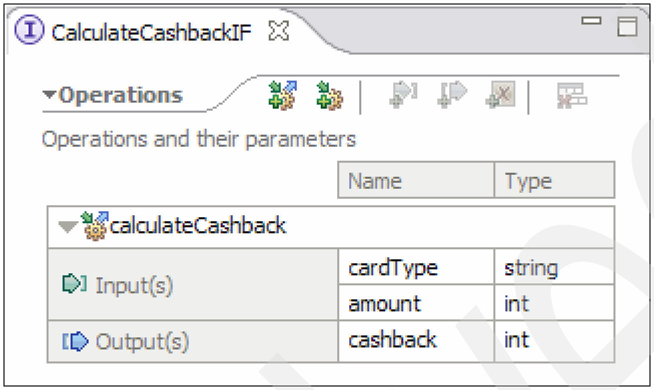
Consider the following example of using business rules in a “cash back” credit card offer. A major credit card company offers cash back when a credit card holder uses the card for purchases. This cash back offers varies depending upon the type of card and the season. Now let us look at the following business policies and offers for the cash back:

- A flat percentage on the amount spent from October to January:
  - 3% for Silver card customers
  - 5% for Gold card customers
- Varying percentages from February to September to offer incentives for larger purchases:
  - Silver
    - 3% if the amount spent <= 5000
    - 10% if the amount spent > 5000
  - Gold
    - 5% if the amount spent <=5000
    - 12% if the amount spent > 5000

Business leaders of the credit card company change these offers occasionally to compete with other credit card companies. Using business rules allows them to make these changes easily.

The first rule is implemented using a rule set. The second rule is implemented with a decision table. The criteria selection (the season) is configured in a rule group which holds both of these rule set and decision table.

Figure 4-83 shows the rule group that implements the interface.



	Name	Type
calculateCashback		
Input(s)	cardType	string
	amount	int
Output(s)	cashback	int

Figure 4-83 Interface of a rule group

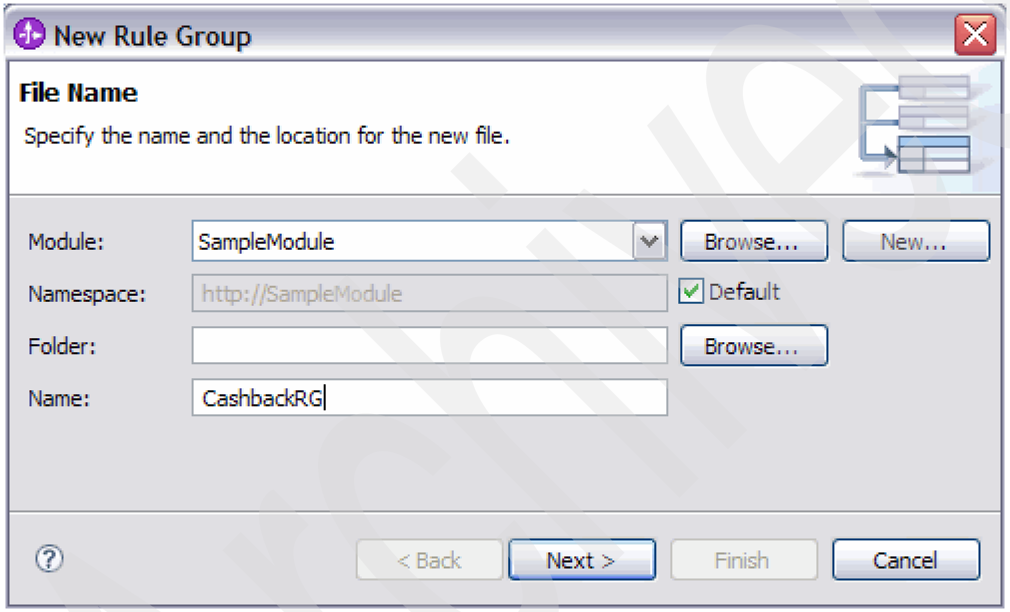
4.8.2 Rule groups

A *rule group* is a type of component implementation that logically groups rule sets and decision tables. Similar to other component types, a rule group implements one or more interfaces. The selection of which business rule to invoke is based on date criteria that is configured in the rule group.

Creating a rule group

(This example assumes that a module and an interface, CalculateCashbackIF, have been created as shown in Figure 4-83.) Follow these steps to create a new rule group:

1. Select **File** → **New** → **Rule Group**.
2. In the New Rule Group window (shown in Figure 4-84):
   a. Browse to an existing module (or click **New** to create one).
   b. Specify a folder (optional).
   c. Specify a name for the new rule group (for example, CashbackRG).
   d. Click **Next**.



New Rule Group

File Name

Specify the name and the location for the new file.

Module: SampleModule

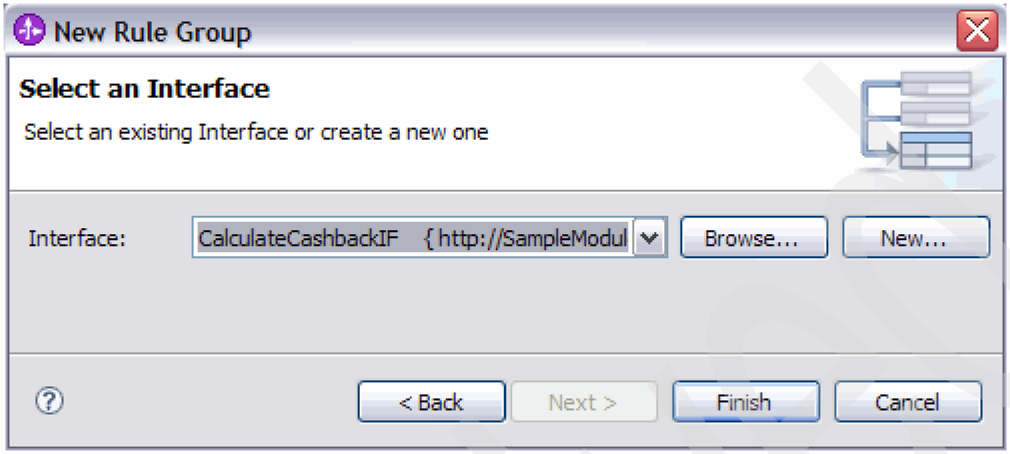
Namespace: http://SampleModule ☑ Default

Folder:

Name: CashbackRG

Figure 4-84 New Rule Group wizard

3. In the Select an Interface window (Figure 4-85), follow these steps:
   a. Browse to an existing interface or click **New** to create one.
   b. Click **Finish**.


New Rule Group

Select an Interface

Select an existing Interface or create a new one

Interface: CalculateCashbackIF {http://SampleModul

Figure 4-85 Selecting an interface from New Rule Group wizard

> **Note:** At this point, you might see an error in the Problems view indicating that the rule set or decision table association to an operation is missing. This error is corrected after you configure the rule group, which can be done only after creating the rule set or decision table.
>
> The definition of the calculateCashback operation is missing the rule destination.

Configuring rule group logic

A rule group is exposed to the caller as an interface. Rule sets or decision tables provide the implementation for this interface. For each operation of the associated interface, you can have different rule sets or decision tables defined.

Whenever a rule group is invoked, it selects a *rule logic* using the selection criteria and date range entries. Selection criteria is either a current date, an XPath expression that gets a date from a business object, or a Java snippet that returns a date.

A date range entry specifies a range of dates, during which a rule logic (such as a rule set or a decision table) is applicable. If the date returned from the selection criteria is within the date range, then the corresponding rule logic entry is used. The date range entries are optional. In fact, if no date range entry is specified or if no date range matches, the default rule logic is invoked.

Note: This section assumes the rule set and decision table have been created. We discuss how these are built in 4.8.3, “Rule sets” on page 244 and 4.8.4, “Decision tables” on page 252. We use these artifacts when configuring the rule group.

Note: Remember that the first rule is implemented as a rule set, and the second rule, as a decision table.

To configure the rule group to implement the business rules follow these steps:

1. Select the operation (calculateCashback) that will be associated with the rules in the rule group editor. You might need to expand the interface to select the operation that is contained within it (Figure 4-86).

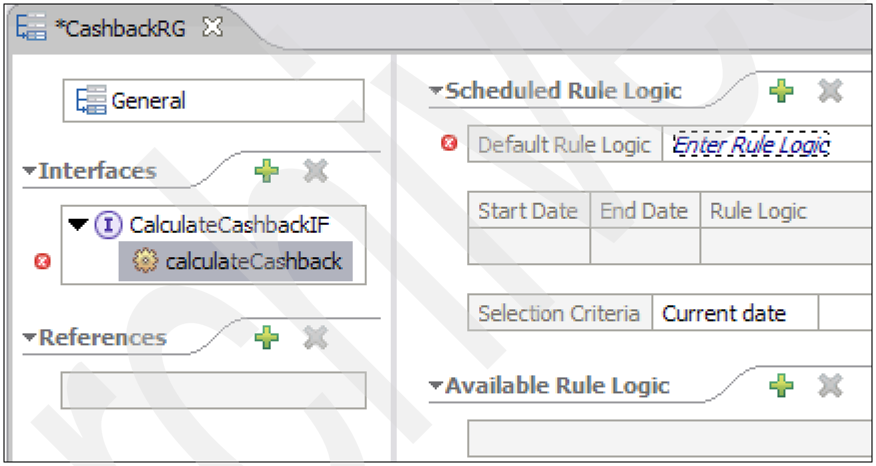
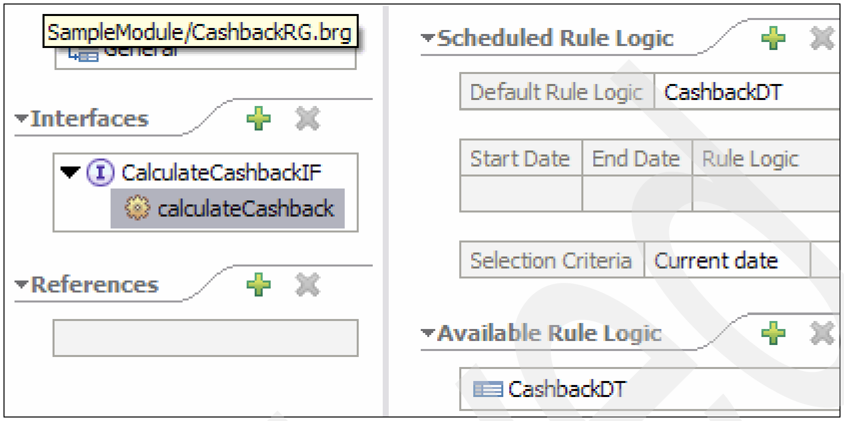


Figure 4-86 CashbackRG configuration in rule group editor

2. Click **Enter Rule Logic** on the Default Rule Logic row under the Scheduled Rule Logic section and select the existing decision table, CashbackDT. Make this as the default rule logic as shown in Figure 4-87.



SampleModule/CashbackRG.brg

Interfaces

CalculateCashbackIF
calculateCashback

References

Scheduled Rule Logic

Default Rule Logic: CashbackDT

Start Date	End Date	Rule Logic

Selection Criteria: Current date

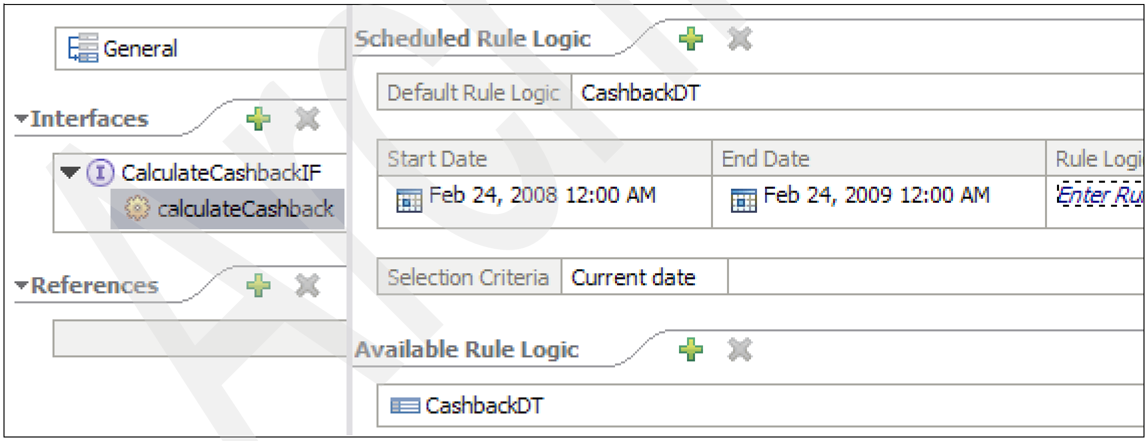
Available Rule Logic

CashbackDT

Figure 4-87 Using decision table as a default rule logic

3. As per the scenario, rule 1 (CashbackRS rule set) is used from October to January.

   Click the Add Date Selection Entry (✚) icon to add a selection entry (a row that shows rule logic with start and end dates) as shown in Figure 4-88.



General

Interfaces

CalculateCashbackIF
calculateCashback

References

Scheduled Rule Logic

Default Rule Logic: CashbackDT

Start Date	End Date	Rule Logi
Feb 24, 2008 12:00 AM	Feb 24, 2009 12:00 AM	Enter Ru

Selection Criteria: Current date

Available Rule Logic

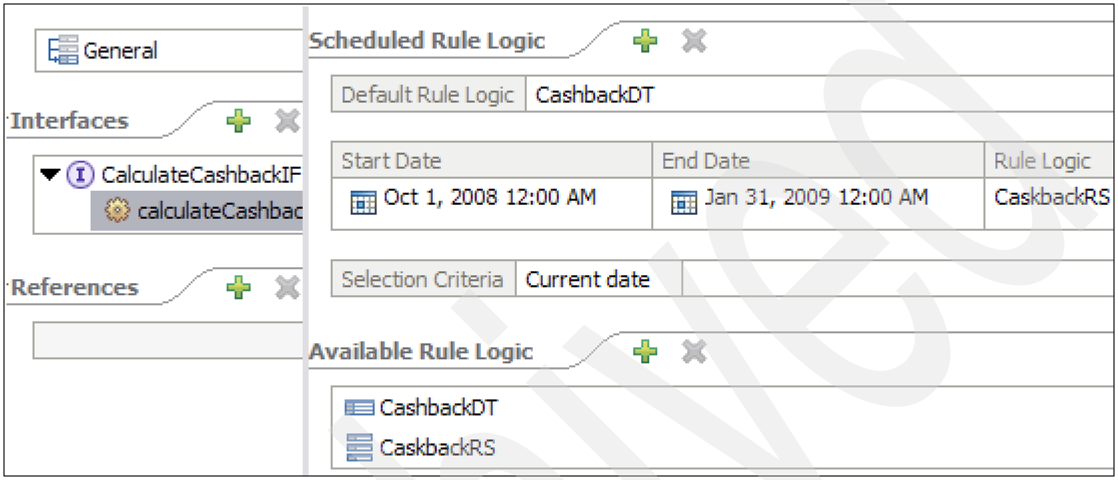
CashbackDT

Figure 4-88 Adding a date selection entry in rule group

4. To complete the rule group configuration, which looks similar to the one shown in Figure 4-89, follow these steps:

   a. Click the calendar icon (▦) and select the appropriate dates under Start Date and End Date columns.

   b. Click **Enter Rule Logic** to select the existing rule set, CashbackRS, in the Rule Logic column.



General

Interfaces

CalculateCashbackIF
calculateCashbac

References

Scheduled Rule Logic

Default Rule Logic	CashbackDT

Start Date	End Date	Rule Logic
Oct 1, 2008 12:00 AM	Jan 31, 2009 12:00 AM	CaskbackRS

Selection Criteria	Current date

Available Rule Logic

- CashbackDT
- CaskbackRS

Figure 4-89 Adding rule sets selection criteria in a rule group

4.8.3 Rule sets

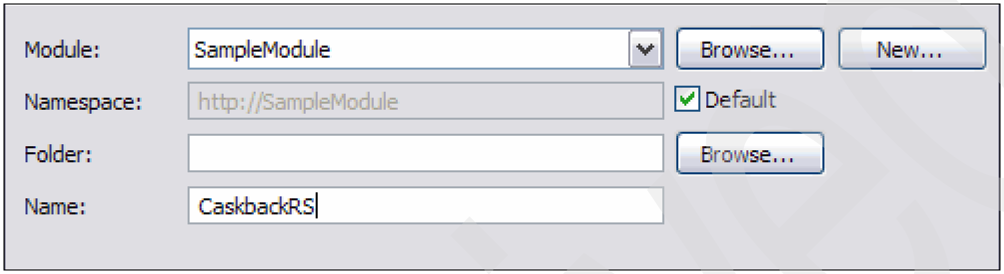
A *rule set* is a group of if-then statements or rules. The action specified by the “then” clause is performed when the condition specified in the “if” clause evaluates to true. Rule sets are best suited for those business rules that have very few condition clauses.

If the condition is met, then the action is performed, which can result in more than one action being performed by the rule set. The order of rule processing is determined by the order of the rule statements in the if-then rule set. Therefore, when you modify or add a rule, you need to make sure that it is in the correct sequence.

Creating a rule set

Follow these steps to create a new rule set:

1. Select **File** → **New** → **Rule Set**.
2. In the New Rule Set window (shown in Figure 4-90):
   a. Browse to an existing module (or click **New** to create one).
   b. Specify a folder (optional).
   c. Specify a name for the new rule set (for example, CashbackRS).
   d. Click **Next**.



Module: SampleModule

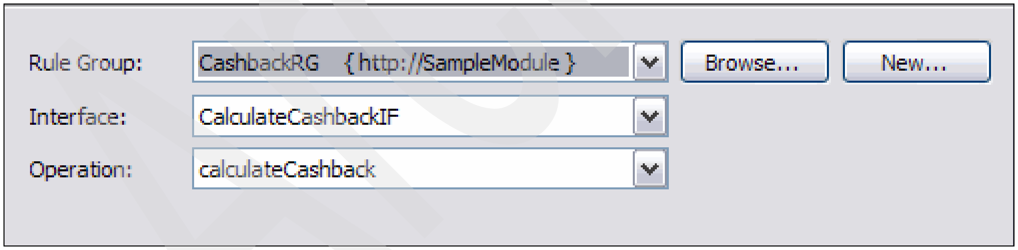
Namespace: http://SampleModule ☑ Default

Folder:

Name: CaskbackRS

Figure 4-90 New Rule Set wizard

3. In the Interface and Operation window (Figure 4-91):
   a. Browse to an existing rule group, or click **New** to create a new one.
   b. If necessary, use the drop-down lists to select a different interface and operation.
   c. Click **Finish**.



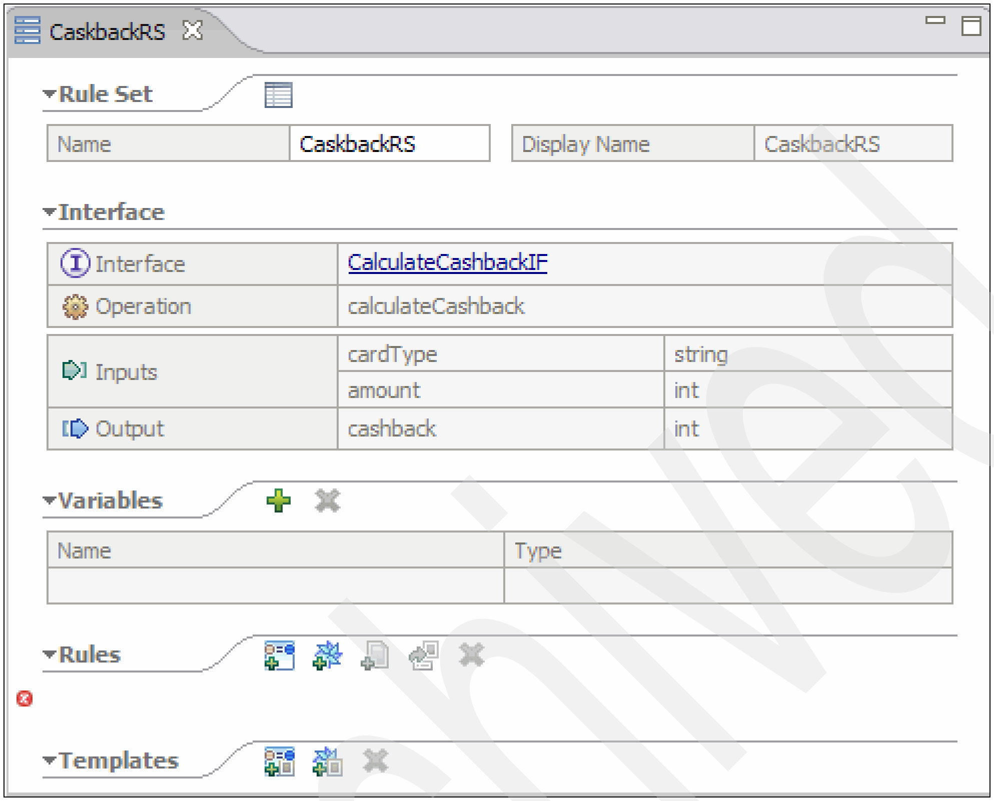
Rule Group: CashbackRG { http://SampleModule }

Interface: CalculateCashbackIF

Operation: calculateCashback

Figure 4-91 Selecting a rule group

The created rule set opens in the rule set editor as shown in Figure 4-92.



Name	CaskbackRS	Display Name	CaskbackRS

Interface	CalculateCashbackIF	
Operation	calculateCashback	
Inputs	cardType	string
	amount	int
Output	cashback	int

Name	Type

Figure 4-92 A rule set in a rule set editor

Rule types

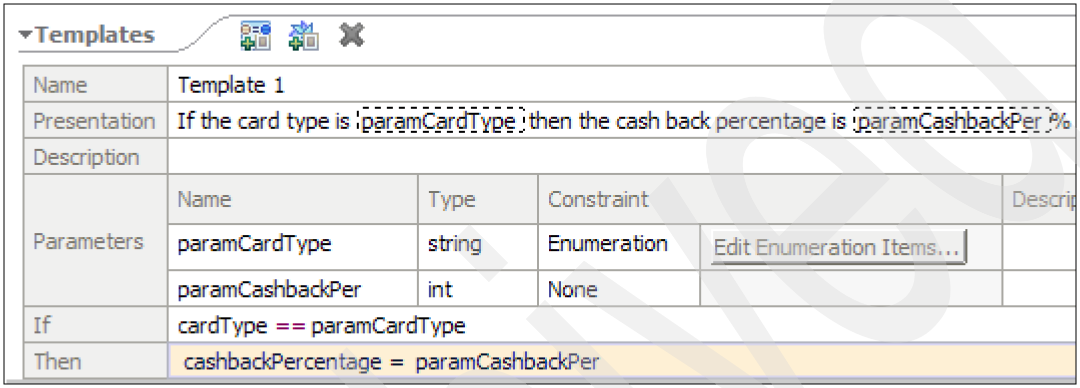
A rule set can have two types of rules:

- An *if-then* rule determines what action is performed based on the condition of the incoming message.
- An action rule determines what action is performed regardless of the incoming message. This rule does not have any conditions, so it always performs the specified action.

Templates

A rule template is a pattern for creating similar looking rules. Use a rule set template to define the implementation and parameters for an if-then or action rule. Then, this template can be used to create new instances of the same rule using different parameters. The parameters in a rule instance at run time can be modified using the Business Rules Manager (as discussed in 4.8.5, “Business rules manager” on page 261).

There are two ways to create a template. You can create a rule and then convert it to a rule template, or you can create a rule template directly. The presentation field is the text that describes the rule template to an administrator or business analyst using the business rules manager, and it specifies the parameters that the rule template accepts. The parameters are inputs to the template and are listed under the parameters section.



Templates

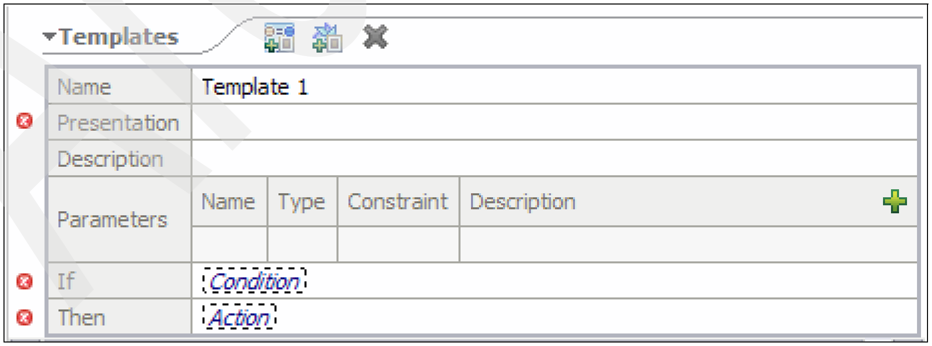
Name	Template 1				
Presentation	If the card type is paramCardType then the cash back percentage is paramCashbackPer %				
Description					
Parameters	Name	Type	Constraint		Descrip
	paramCardType	string	Enumeration	Edit Enumeration Items...	
	paramCashbackPer	int	None		
If	cardType == paramCardType				
Then	cashbackPercentage = paramCashbackPer				

Figure 4-93   A rule template

Creating a rule set template

Follow these steps to create a template for either a if-then or an action rule:

1. Click either the Add If Then Template ( ) or the Add Action Template ( ) icon under the Templates section in the rule set editor. A new template is created as shown in Figure 4-94.



Templates

Name	Template 1			
Presentation				
Description				
Parameters	Name	Type	Constraint	Description
If	Condition			
Then	Action			

Figure 4-94   A template in rule set editor

2. Create parameters in the Parameters row by following these steps:

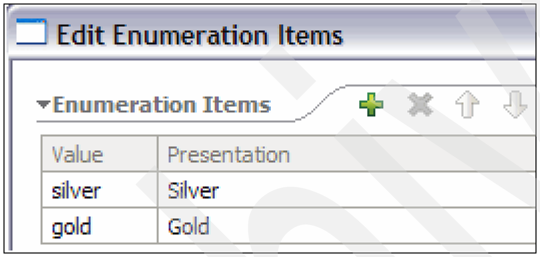
   a. Click the Add Template Parameter (✚) icon in the Parameters row. A new parameter is entered in the Parameters row.

   b. Click the parameter name, **Param1**, to rename it.

   c. Click **Select Type** in the Type column and select an appropriate data type from the list.

   d. To add a restriction on the parameter, click **None** under the Constraint column to choose either **Range** or **Enumeration**.

      - If Range is selected, provide an expression by clicking **Enter Expression**.
      - If Enumeration is selected, specify values by clicking **Edit Enumeration Items** and adding values (Figure 4-95).



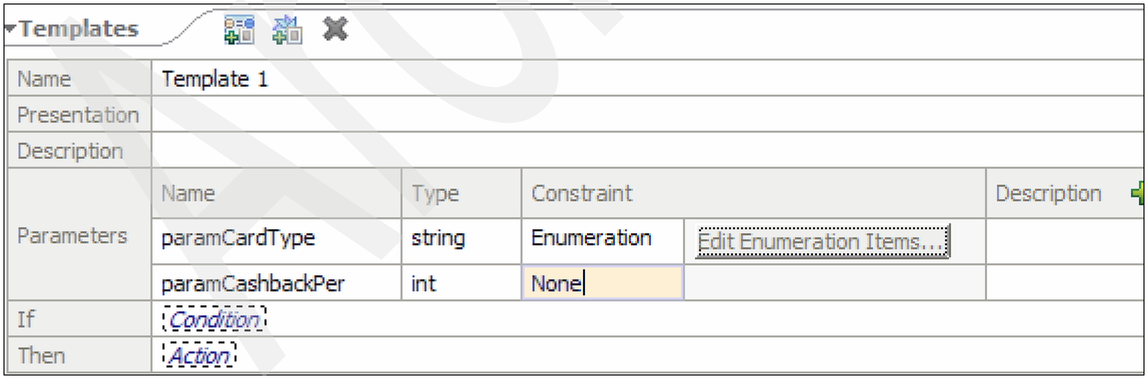
Edit Enumeration Items

Enumeration Items

Value	Presentation
silver	Silver
gold	Gold

Figure 4-95 Edit Enumeration Items window

Figure 4-96 shows the parameters that are created to implement the example scenario.



Templates

Name	Template 1				
Presentation					
Description					
Parameters	Name	Type	Constraint		Description
	paramCardType	string	Enumeration	Edit Enumeration Items...	
	paramCashbackPer	int	None		
If	Condition				
Then	Action				

Figure 4-96 Template parameters view

3. Type a sentence to describe the rule in the Presentation row.

   Note that when you enter text in this field, you can click the text and an icon, › displays below it. Click the icon to launch a list of parameters from which you can select. The parameter is inserted into the Presentation field, allowing users of the template to make changes to the description.

4. Click **Condition** in the If row and use the choices in the window that displays to compose the condition. Figure 4-97 shows a sample of this pop-up window.

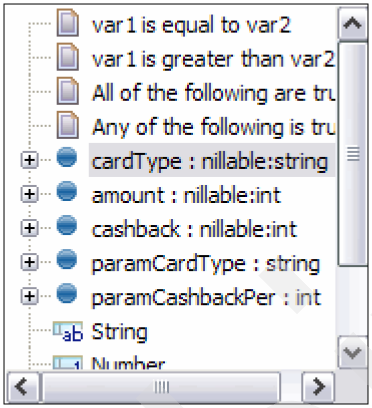


Figure 4-97 Sample pop-up window to compose a template condition

5. Click **Action** in the Then row and use the choices in the window that displays to compose the action. Figure 4-98 shows a sample pop-up window.

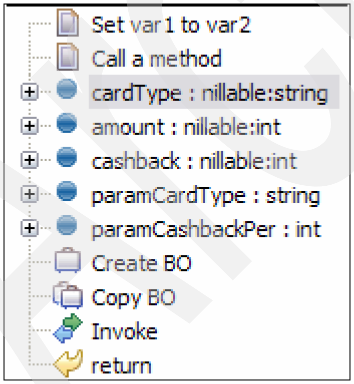


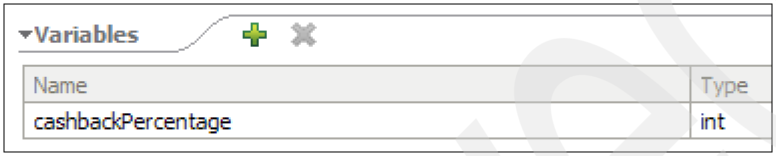
Figure 4-98 Sample pop-up window to compose a template action

Variables

Variables within a rule set have scope across all the rules that are contained within the rule set. Usually, you use a variable for the duration of the rules to hold temporary values. This variable can be changed by an individual rule during processing and is used to calculate the final response returned to the caller.

Variables can be primitive or business object types. When variables are defined, they have no value and are not initialized. If you try to use them without initializing them, you get a runtime exception. Therefore, you must first assign a new business object instance to a business object variable before using it.

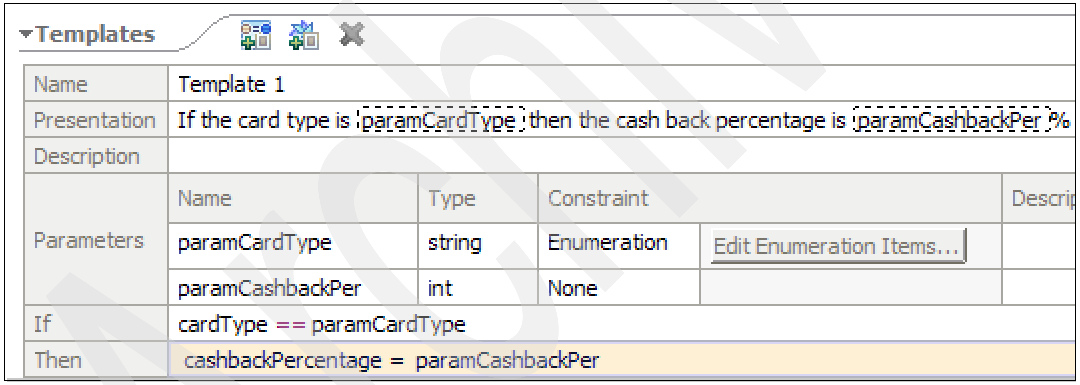
You create variables by clicking the plus icon in the Variables section of a rule set editor. Figure 4-99 shows the variable that is created to implement the example scenario.



Name	Type
cashbackPercentage	int

Figure 4-99 Variables in a rule set

Figure 4-100 shows the template is created to implement the example scenario.



Name	Template 1				
Presentation	If the card type is paramCardType then the cash back percentage is paramCashbackPer %				
Description					
Parameters	Name	Type	Constraint		Descrip
	paramCardType	string	Enumeration	Edit Enumeration Items...	
	paramCashbackPer	int	None		
If	cardType == paramCardType				
Then	cashbackPercentage = paramCashbackPer				

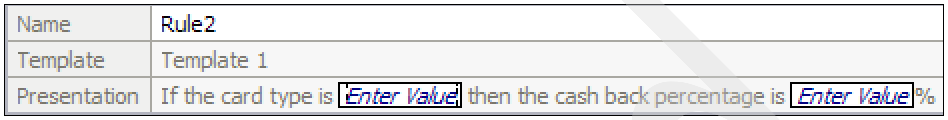
Figure 4-100 Template created for example scenario

Creating a new rule from a template

You can create rules from existing templates. In this way, you can create a similar rule without having to redefine the implementation and by making changes to the parameters within the constraints specified.

To use a template as the pattern to create a new rule in the rule set editor:

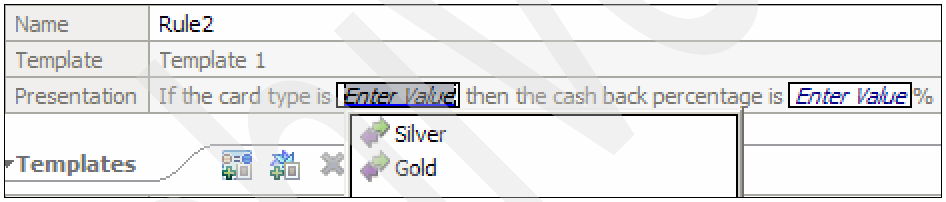
1. Click the Add Template Rule () icon under the Rules section in the rule set editor.
2. Click the appropriate template from the drop-down list that shows all the available templates. Figure 4-101 shows the new rule that gets created.



Name	Rule2
Template	Template 1
Presentation	If the card type is Enter Value then the cash back percentage is Enter Value %

Figure 4-101 New rule created from rule template

3. Define the parameters by clicking **Enter Value** in the Presentation row. If there is an enumeration constraint for this parameter, a menu displays showing the available options, as shown in Figure 4-102, which shows enumeration values Silver and Gold. Otherwise, you can enter a value directly.



Name	Rule2
Template	Template 1
Presentation	If the card type is Enter Value then the cash back percentage is Enter Value %

Figure 4-102 Enumeration values during rule creation from a template

Creating an if-then or an action rule

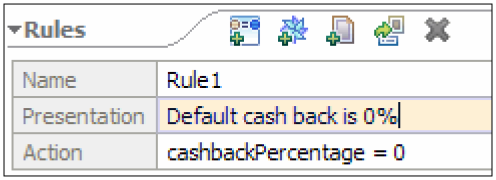
Follow these steps to create an if-then or an action rule without any template:

1. Click either the Add If-Then Rule icon () or the Add Action Rule () icon under the Rules section in the rule set editor.
2. Type a sentence that describes the rule in the Presentation row.
3. Click **Condition** in the If row and use the choices in the window that displays to compose the condition. *Skip this step for action rule*.
4. Click **Action** in the Then row and use the choices in the window that displays to compose the action.

Completing the rules creation for the example scenario

The following steps are required to complete the example scenario:

1. Create an action rule to initialize the variable, cashbackPercentage, as shown in Figure 4-103.

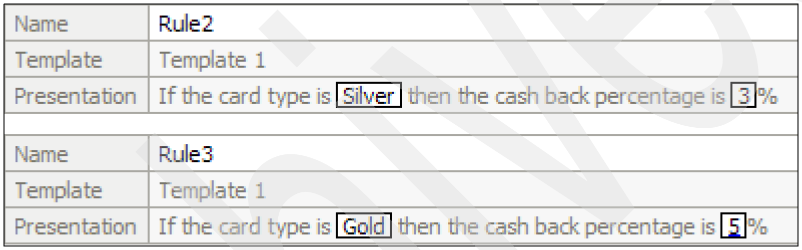


Rules

Name	Rule1
Presentation	Default cash back is 0%
Action	cashbackPercentage = 0

Figure 4-103 Action rule1 for scenario

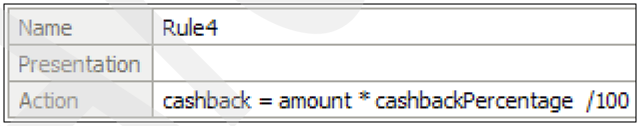
2. Create two if-then rules from the template. These rules need to look similar to those shown in Figure 4-104.



Name	Rule2
Template	Template 1
Presentation	If the card type is Silver then the cash back percentage is 3%
Name	Rule3
Template	Template 1
Presentation	If the card type is Gold then the cash back percentage is 5%

Figure 4-104 If-then rules for scenario

3. After executing the preceding rules, the cashbackPercentage variable contains the cash back percentage for a particular customer. Add an action rule that looks similar to that shown in Figure 4-105 to calculate the cash back value.



Name	Rule4
Presentation	
Action	cashback = amount * cashbackPercentage /100

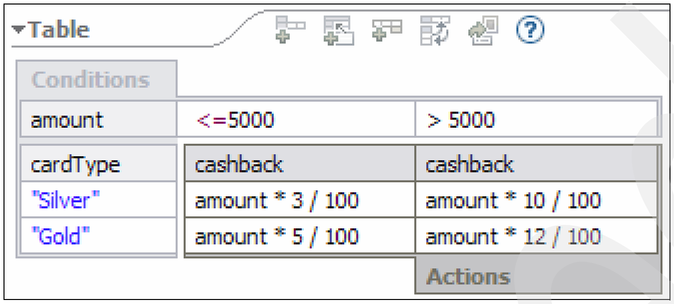
Figure 4-105 Action rule to compute cashback for the scenario

4.8.4 Decision tables

A decision table captures simple rule logic in a table format that consists of conditions, represented in the row and column headings, and actions, represented as the intersection points of the conditional cases in the table.

Decision tables are best suited for business rules that have multiple conditions. Adding another condition is done by simply adding another row or column.

The decision table shown in Figure 4-106 represents the decision table for example scenario.



Conditions

amount	<=5000	> 5000
cardType	cashback	cashback
"Silver"	amount * 3 / 100	amount * 10 / 100
"Gold"	amount * 5 / 100	amount * 12 / 100

Actions

Figure 4-106 Decision table for example scenario

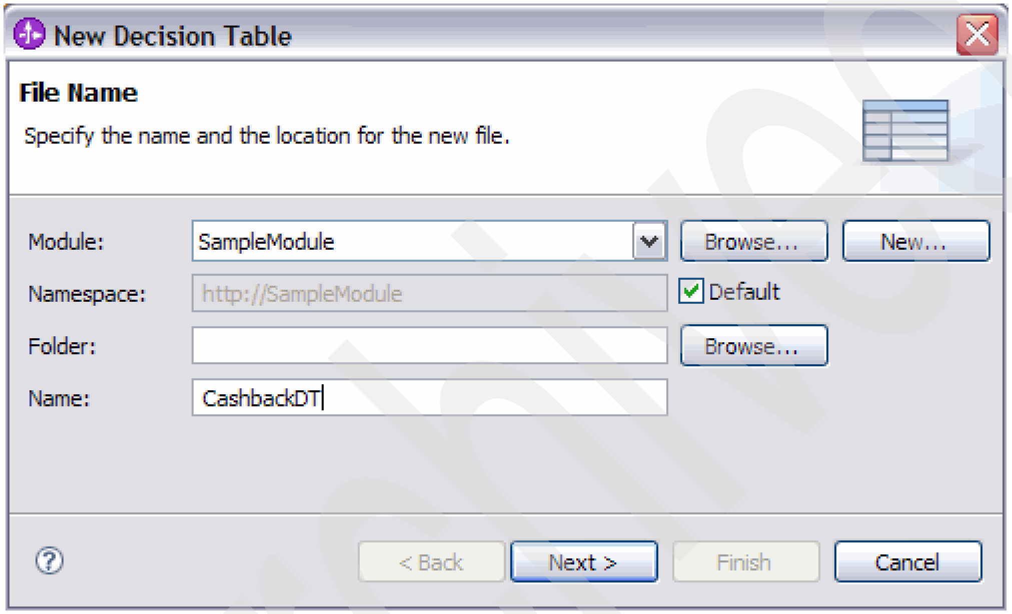
The condition states that if the cardType is Silver and the amount is less than or equal to 5000, then the cask back is 3% of the amount.

Creating a decision table

Follow these steps to create a new decision table:

1. Select **File** → **New** → **Decision Table**.
2. In the New Decision Table window (shown in Figure 4-107 on page 254):

   a. Browse to an existing module (or click **New** to create one).
   b. Specify a folder (optional).
   c. Specify a name for the new decision table (for example, CashbackDT).
   d. Click **Next**.


New Decision Table

File Name

Specify the name and the location for the new file.

Module: SampleModule

Namespace: http://SampleModule ☑ Default

Folder:

Name: CashbackDT

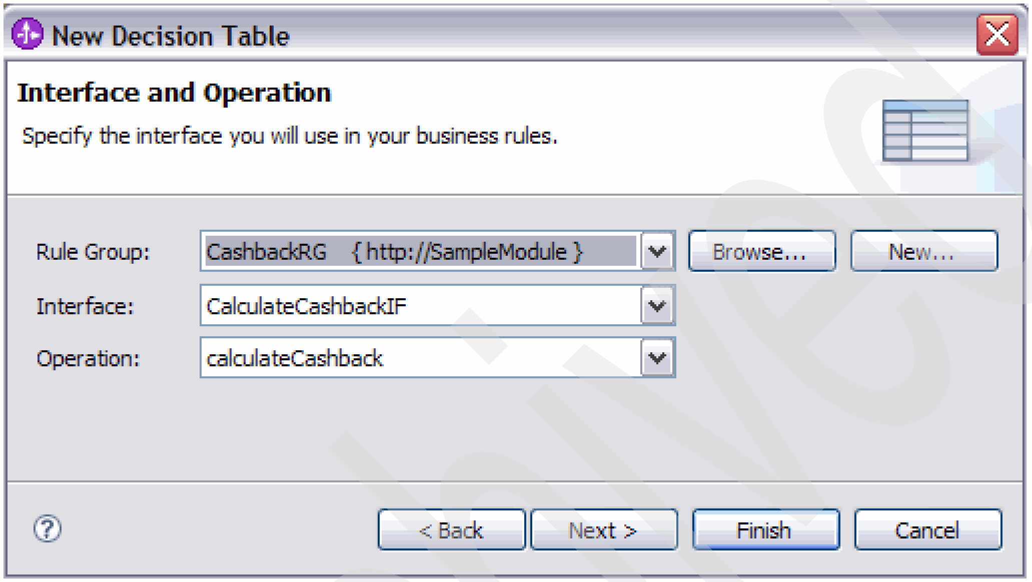
Figure 4-107 New Decision Table wizard

3. In the Interface and Operation window (Figure 4-108):

   a. Browse to an existing rule group, or click **New** to create a new one.

   b. If necessary, use the drop-down lists to select a different interface and operation.

   c. Click **Next**.



New Decision Table

Interface and Operation

Specify the interface you will use in your business rules.

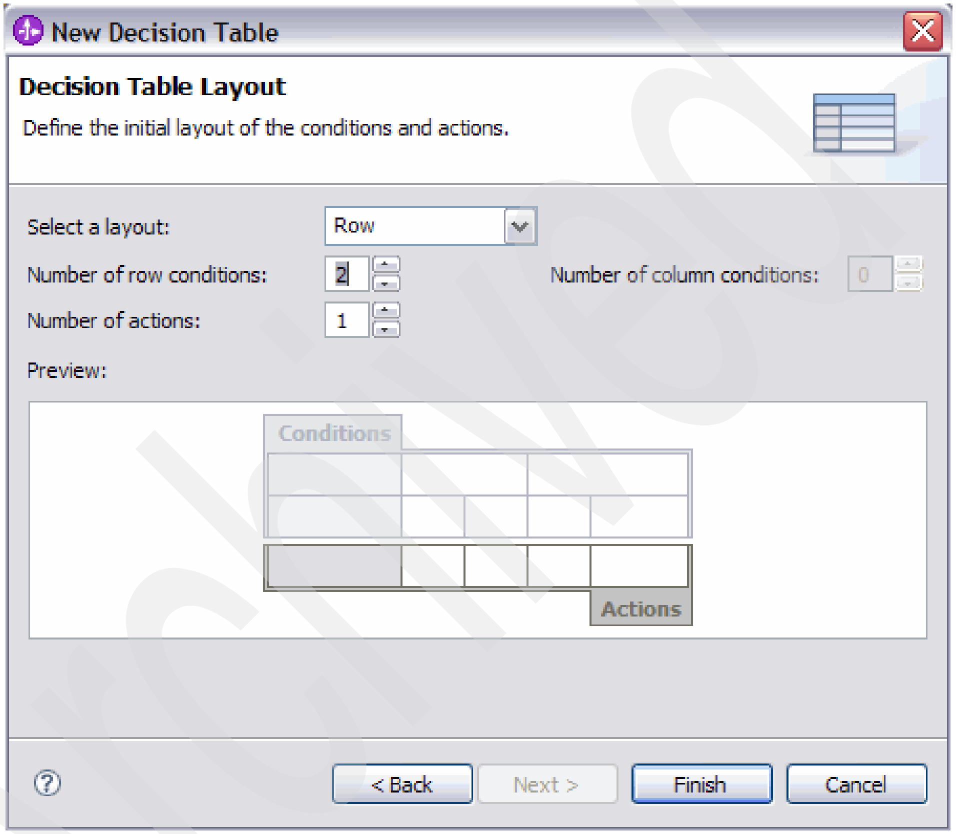
Rule Group: CashbackRG { http://SampleModule }

Interface: CalculateCashbackIF

Operation: calculateCashback

Figure 4-108 Create a new decision table

4. In the Decision Table Layout window (Figure 4-109):
   a. Set the Number of row conditions, 2 in this case.
   b. Optionally, you can change the layout. The Preview option allows you to see how the table will look.
   c. Click **Finish**.



New Decision Table

Decision Table Layout

Define the initial layout of the conditions and actions.

Select a layout: Row

Number of row conditions: 2

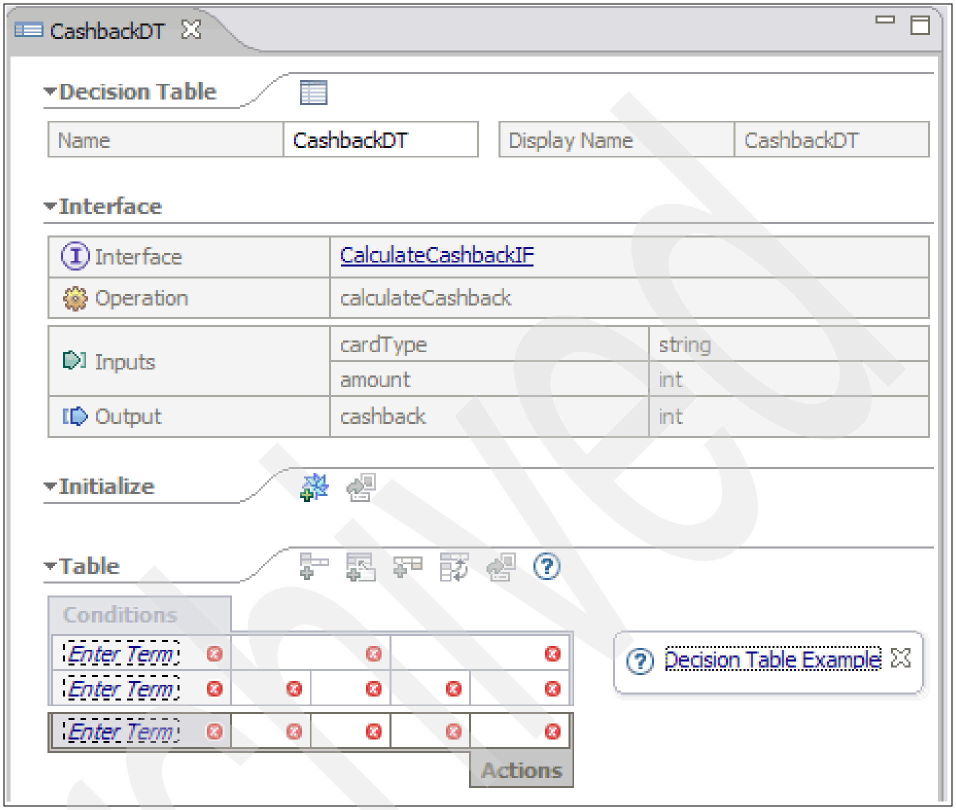
Number of column conditions: 0

Number of actions: 1

Preview:

Figure 4-109 Decision Table Layout

The created decision table opens in decision table editor as shown in Figure 4-110.



CashbackDT

Decision Table

Name	CashbackDT	Display Name	CashbackDT

Interface

Interface	CalculateCashbackIF	
Operation	calculateCashback	
Inputs	cardType	string
	amount	int
Output	cashback	int

Initialize

Table

Conditions

Enter Term

Enter Term

Enter Term

Actions

Figure 4-110 Decision Table in decision table editor

Configuring conditions and actions in a decision table

The following steps discuss the configuration of conditions and actions for the decision table to implement the one required for the example scenario. The example decision table created in the previous step looks similar to that shown in Figure 4-111. It has two conditions and one action for each of the conditions.

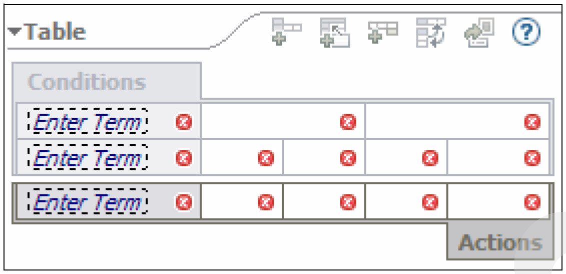


Figure 4-111 Decision table before configuration

To configure conditions and actions for the decision tree:

1. Click **Enter Term** in the Conditions column and select the a condition variable (cardType in this example) from the list. The table now looks similar to that shown in Figure 4-112.

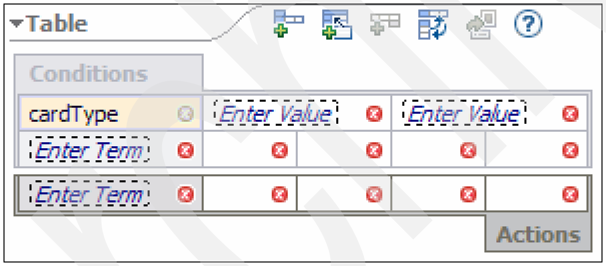
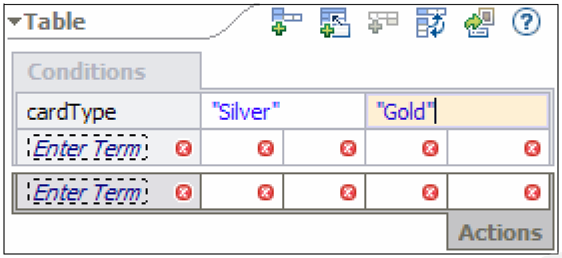


Figure 4-112 Enter the condition term

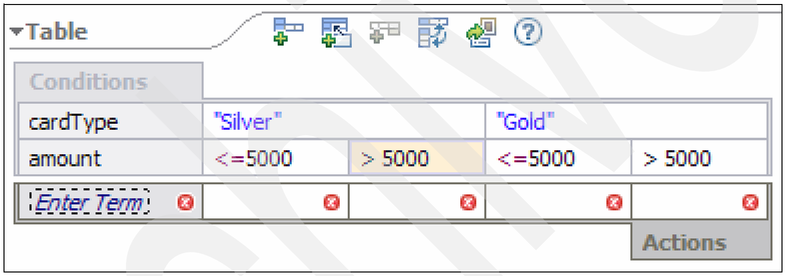
2. Specify a value for each column in the new condition row. In the example, click **Enter Value** on the cardType row and specify Silver as a string using the pop-up window. Specify Gold in the second column of the same row. Now the table looks similar to that shown in Figure 4-113.



cardType	"Silver"		"Gold"	
Enter Term				
Enter Term				

Figure 4-113 Enter the condition values

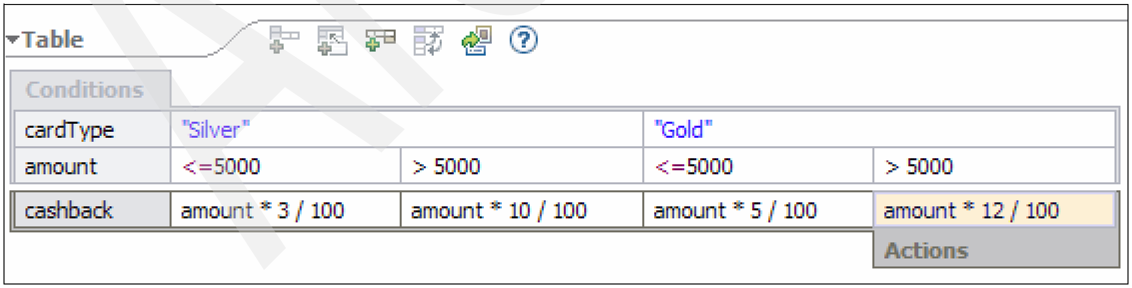
3. Configure the next condition variable, amount, and its conditions as shown in Figure 4-114.



cardType	"Silver"		"Gold"	
amount	<=5000	> 5000	<=5000	> 5000
Enter Term				

Figure 4-114 New conditions

4. Now, configure the corresponding actions in the last row as shown in Figure 4-115.



cardType	"Silver"		"Gold"	
amount	<=5000	> 5000	<=5000	> 5000
cashback	amount * 3 / 100	amount * 10 / 100	amount * 5 / 100	amount * 12 / 100

Figure 4-115 Configuring actions in decision table

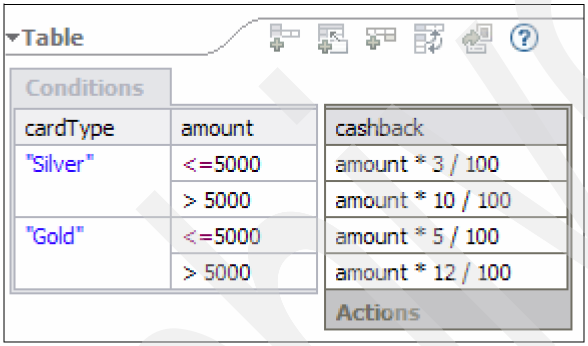
Changing the layout of decision table

Some decision tables are easier to understand if you change their orientation. By default, decision tables are created with row layout, where the conditions and actions appear as rows as shown in Figure 4-115 on page 259.

You can change this layout to column, so that the conditions and actions appear as columns. Follow these steps to change the layout from the decision table editor:

1. Click any cell in the condition row of an existing decision table.
2. Click the Changes Orientation of Condition icon (⇄).

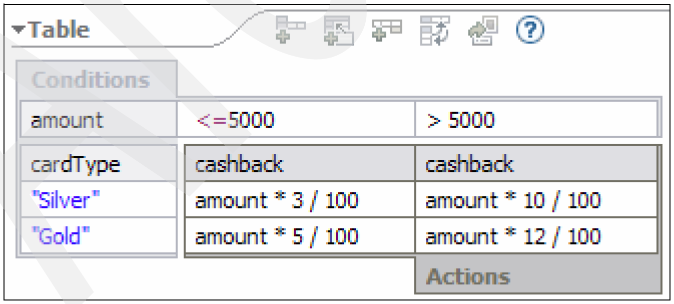
The orientation of the decision table changes so that the actions and conditions now display in columns instead of rows as shown in Figure 4-116.



cardType	amount	cashback
"Silver"	<=5000	amount * 3 / 100
	> 5000	amount * 10 / 100
"Gold"	<=5000	amount * 5 / 100
	> 5000	amount * 12 / 100

Figure 4-116 Column layout of a decision table

Now, repeat these steps to change the layout to row and column layout and the decision table looks similar to that shown in Figure 4-117.



amount	<=5000	> 5000
cardType	cashback	cashback
"Silver"	amount * 3 / 100	amount * 10 / 100
"Gold"	amount * 5 / 100	amount * 12 / 100

Figure 4-117 Row and column layout of a decision table

4.8.5 Business rules manager

The business rules manager is a Web-based console that allows the business analyst to view and modify business rules. The tool is a Web application that you can select to install during WebSphere Process Server profile creation or later after installing the server.

Templates are the patterns for rule sets and decision tables. They provide the mechanism for business rule runtime authoring in the business rules manager. Using a template, you can modify business rule values, create a new rule within a rule set, create a new condition or action within a decision table, and publish changes to business rule definitions at run time.

Launching business rules manager

To launch the business rules manager, follow these steps:

1. Open the business rules manager URL from a Web browser:

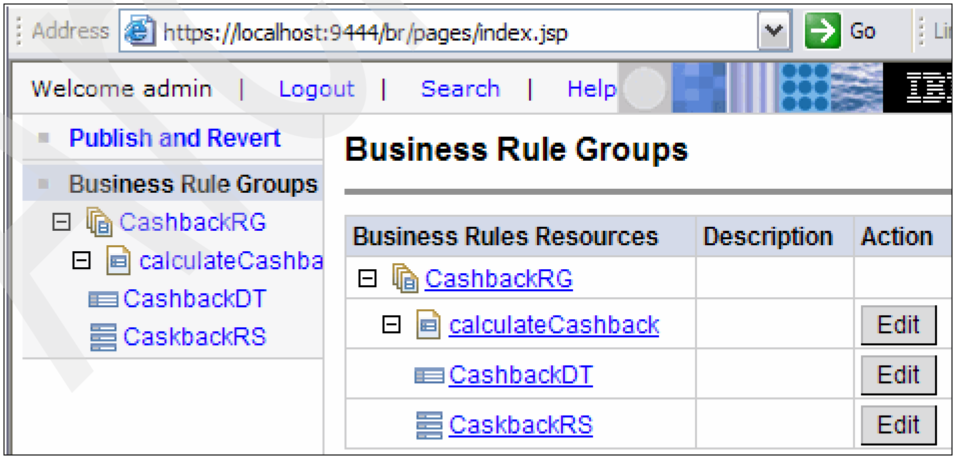
   ```
   http://<app_server_host>:<default_host_port_no>/context_root
   ```

   For example:

   ```
   http://localhost:9080/br
   ```

2. If security is enabled, you must enter a user ID and password, and then click **Login.**

Figure 4-118 shows the business rules manager page. It shows the installed rule group, CashbackRG, with a decision table, CashbackDT, and a rule set, CashbackRS.



Business Rule Groups

Business Rules Resources	Description	Action
⊟ CashbackRG		
⊟ calculateCashback		Edit
CashbackDT		Edit
CaskbackRS		Edit

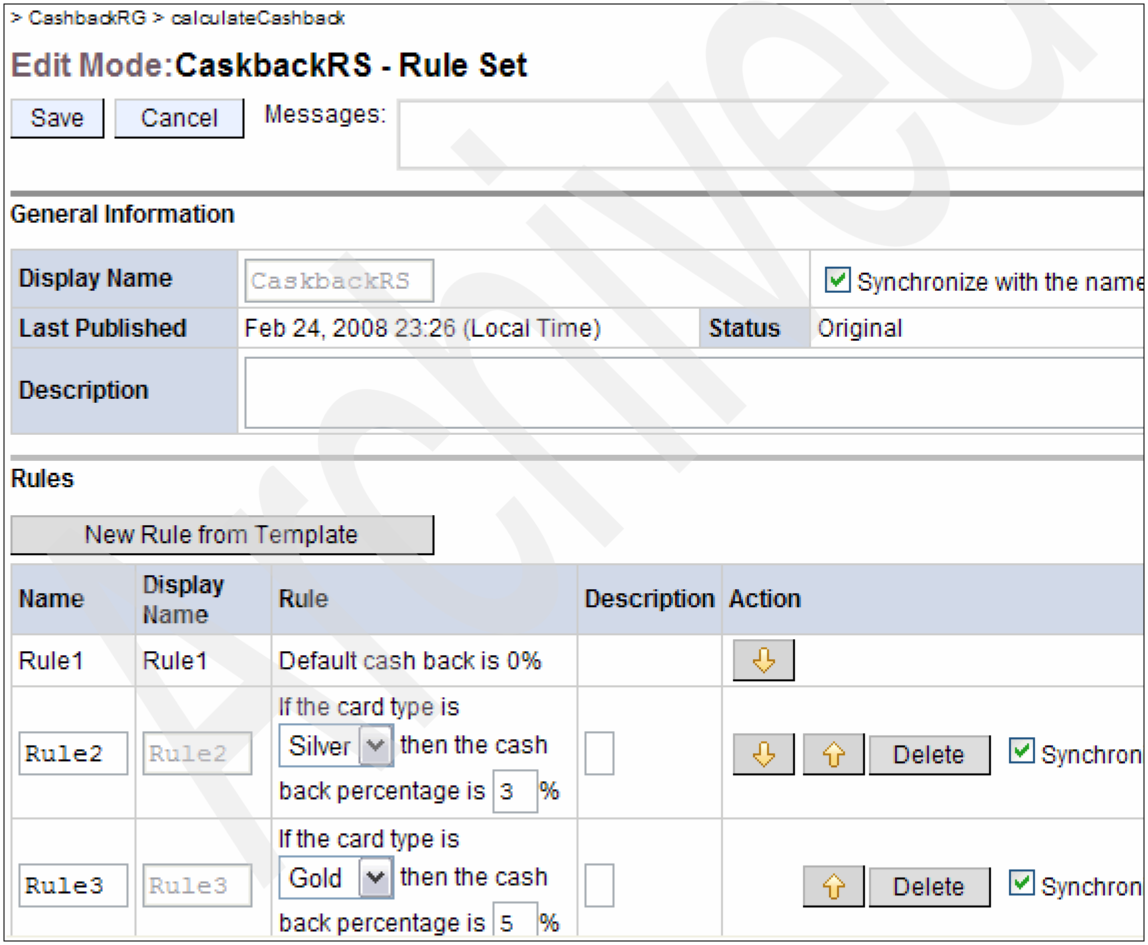
Figure 4-118 Business rules manager

Launching the business rules manager from WebSphere Integration Developer

Alternatively, you can perform the following steps to start the business rules manager if you are using a WebSphere Integration Developer test environment:

1. Switch to the Business Integration perspective and Servers view.
2. Select a server and start it.
3. Right-click the server and select **Launch** → **Business Rules Manager**.

Figure 4-119 shows a rule set, CashbackRS, in edit mode in the business rule manager and the options that are available to work with a rule set.



Edit Mode:CaskbackRS - Rule Set

General Information

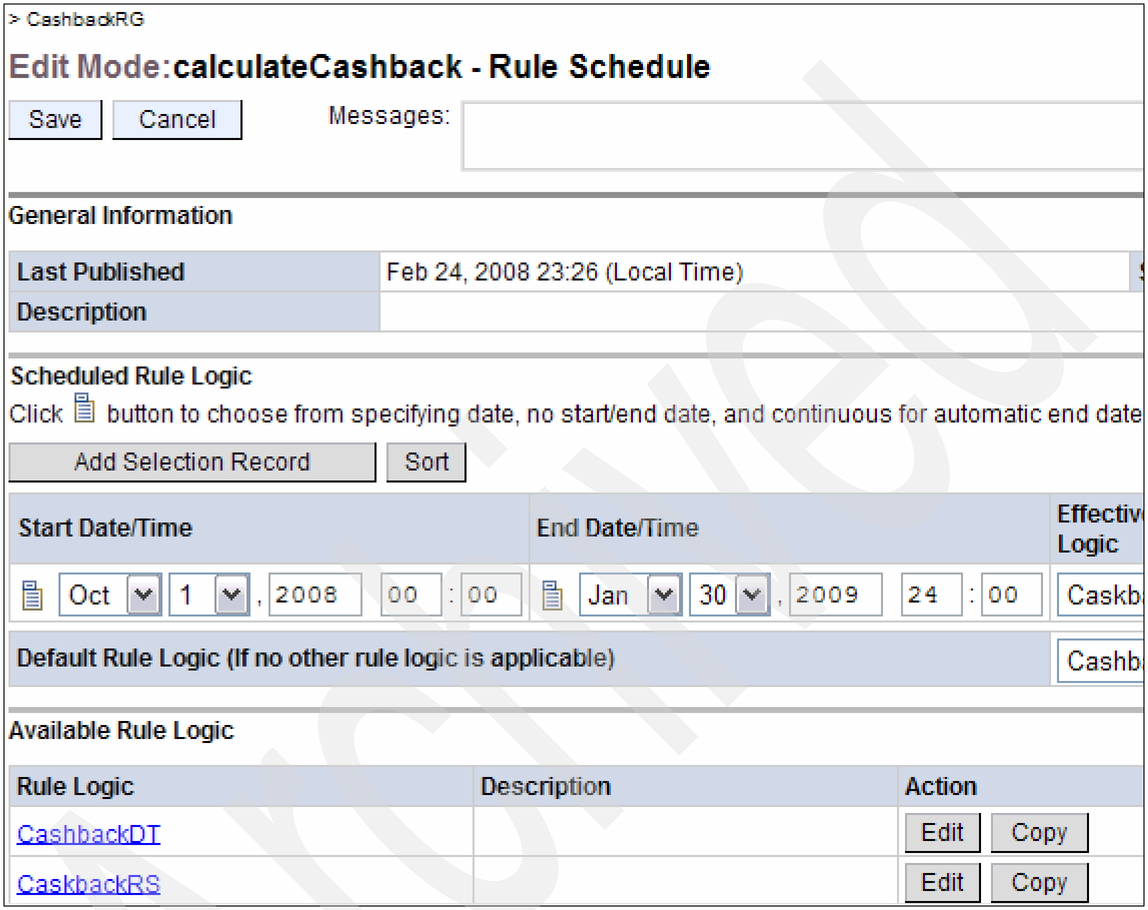
Display Name	CaskbackRS	☑ Synchronize with the name	
Last Published	Feb 24, 2008 23:26 (Local Time)	Status	Original
Description			

Rules

Name	Display Name	Rule	Description	Action
Rule1	Rule1	Default cash back is 0%		⇩
Rule2	Rule2	If the card type is Silver then the cash back percentage is 3 %	□	⇩ ⇧ Delete ☑ Synchron
Rule3	Rule3	If the card type is Gold then the cash back percentage is 5 %	□	⇧ Delete ☑ Synchron

Figure 4-119 A rule set in edit mode in the business rules manager

Figure 4-120 shows the scheduled rule logic for the CashbackRG in edit mode in the business rule manager and the options that are available to work with scheduling of rule logic.



> CashbackRG

Edit Mode:calculateCashback - Rule Schedule

General Information

Last Published	Feb 24, 2008 23:26 (Local Time)
Description	

Scheduled Rule Logic

Click button to choose from specifying date, no start/end date, and continuous for automatic end date

Start Date/Time	End Date/Time	Effectiv Logic
Oct 1 , 2008 00 : 00	Jan 30 , 2009 24 : 00	Caskb
Default Rule Logic (If no other rule logic is applicable)		Cashb

Available Rule Logic

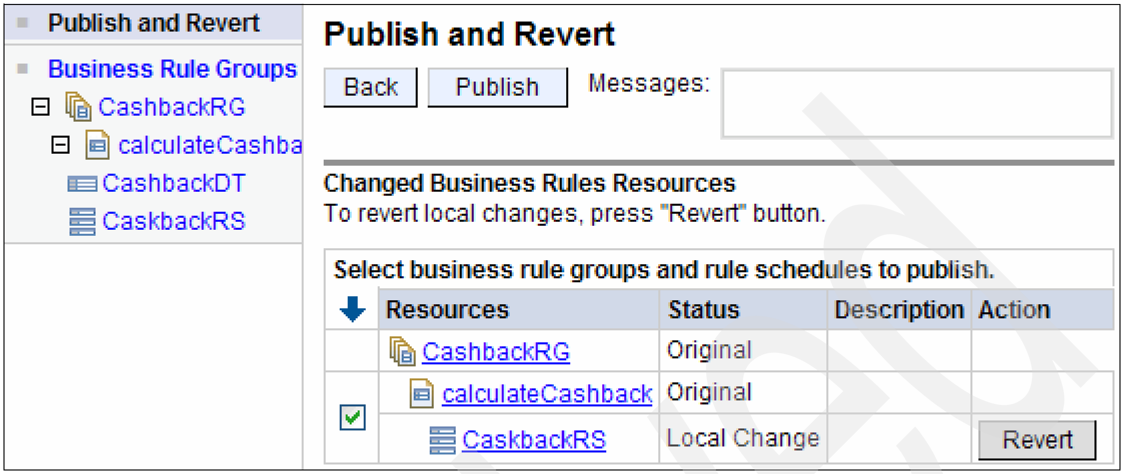
Rule Logic	Description	Action
CashbackDT		Edit Copy
CaskbackRS		Edit Copy

Figure 4-120 Scheduled rule logic of a rule group in business rule manager

Publishing and reverting business rules

When you change business rules in the business rules manager and save them, the changes are saved locally. In order to push the changes to the data repository that resides on the runtime server, you need to publish the changes. Alternatively, you can cancel the changes to a business rule that has been saved locally by reverting the rule to its original state.

Figure 4-121 shows the Publish and Revert page of the business rules manager after changing a rule set values of rule set, CashbackRS.



Publish and Revert

Business Rule Groups

- CashbackRG
  - calculateCashba
    - CashbackDT
    - CaskbackRS

Publish and Revert

Back | Publish | Messages:

Changed Business Rules Resources

To revert local changes, press "Revert" button.

Select business rule groups and rule schedules to publish.

⬇	Resources	Status	Description	Action
	CashbackRG	Original		
☑	calculateCashback	Original		
	CaskbackRS	Local Change		Revert

Figure 4-121 Publish and Revert page of business rules manager

4.8.6 New, enhanced business rules features with WebSphere Process Server V6.1

The new or enhanced business rules capabilities for business integration applications available with WebSphere Process Server V6.1 include:

- Custom business user clients can now administer business rules with a new business rules administration API as an alternative to using the business rules manager. The API provides the ability to create, retrieve, update, and delete rules.
- New custom properties can be assigned to business rule groups and accessed from the rule logic of rule sets and decision tables to provide these rules with access to environment information captured in the properties. The properties can also be used for searching on business rule groups through the business rules manager or custom administrative clients.

4.9 Selectors

The business integration applications that drive the business should change with the changing business needs. Some of those changes might require that certain applications or business processes return different results at different times. The selector component provides the framework for that flexibility without changing the design of the applications.

A *selector* is a routing component that determines dynamically which implementation to invoke at run time based on the defined set of criteria. Similar to a rule group, a selector has *date range entries*, *selection criteria*, and a default *destination*. When a selector is invoked, it selects a destination (that is the target component) using the selection criteria and date range entries. A destination can be any SCA component.

Selector components provide a single interface to a service that can change results based on certain selection criteria. This criteria evaluates to a specific date, based on which, the selector table determines which destination or target component processes the request. The service returns the processing result provided by a target component to the client. What the selector does is simply decouple the client application from a specific target implementation. So change of implementation does not require the client to change.

Selectors allow additional flexibility beyond business rules. The major difference between a selector and a rule group is that the destination of a selector can be any service component, while a destination in a rule group can be only a rule set or decision table. In other words, a selector can re-route a service call dynamically to any other component at run time.

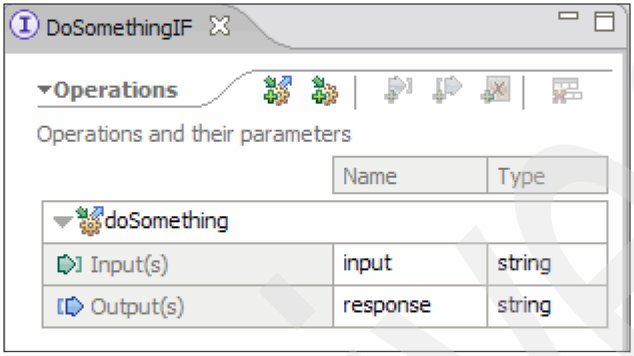
You can invoke selectors from any of your SCA components, because a selector uses the dynamic information to determine which component is selected and invoked to service the request. The set of destinations or target components is also configurable, allowing you to provide additional destinations at run time. A selector cannot be used in a mediation module. It can be deployed only to a WebSphere Process Server.

4.9.1 Creating a selector

Creating a selector is similar to creating any other component. As a best practice, design and create the interface of the selector component before proceeding with its creation. Often, you choose the interface based on the destination (that is the target component) implementation to which the selector routes.

Note: The interface operations of the selector must match the interface operations of the destinations exactly, including the parameters and their types.

Assume that a module named SelectorModule and an interface named DoSomethingIF have been created. The interface is shown in Figure 4-122.

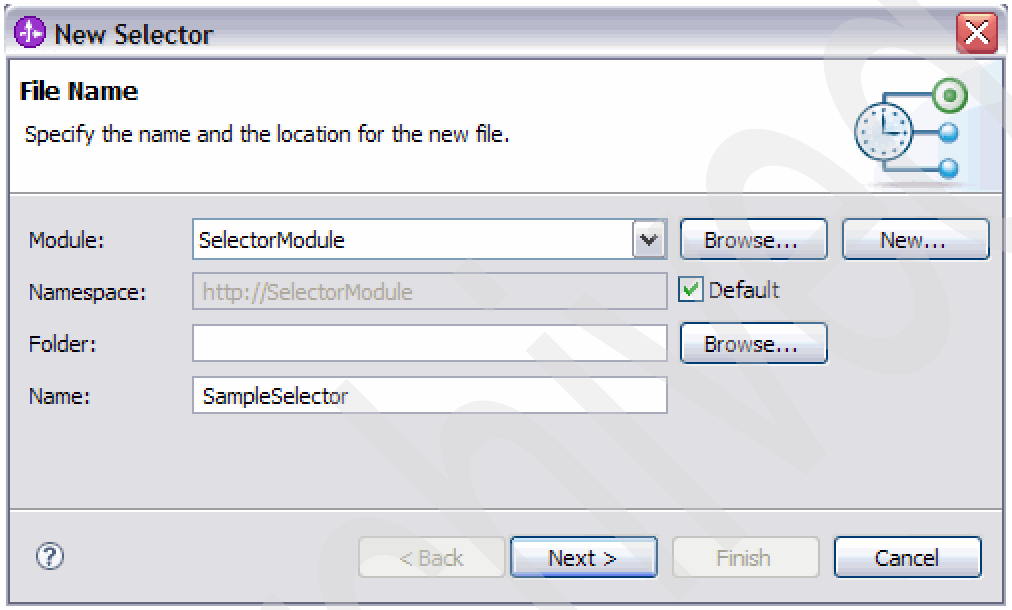


	Name	Type
doSomething		
Input(s)	input	string
Output(s)	response	string

Figure 4-122 DoSomethingIF for selector

Follow these steps to create a new selector:

1. Select **File** → **New** → **Selector**.
2. In the New Selector window (shown in Figure 4-123):
   a. Browse to an existing module (for example, SelectorModule).
   b. Specify a folder (optional).
   c. Specify a name for the new selector (for example, SampleSelector).
   d. Click **Next**.



New Selector

File Name

Specify the name and the location for the new file.

Module: SelectorModule

Namespace: http://SelectorModule ☑ Default

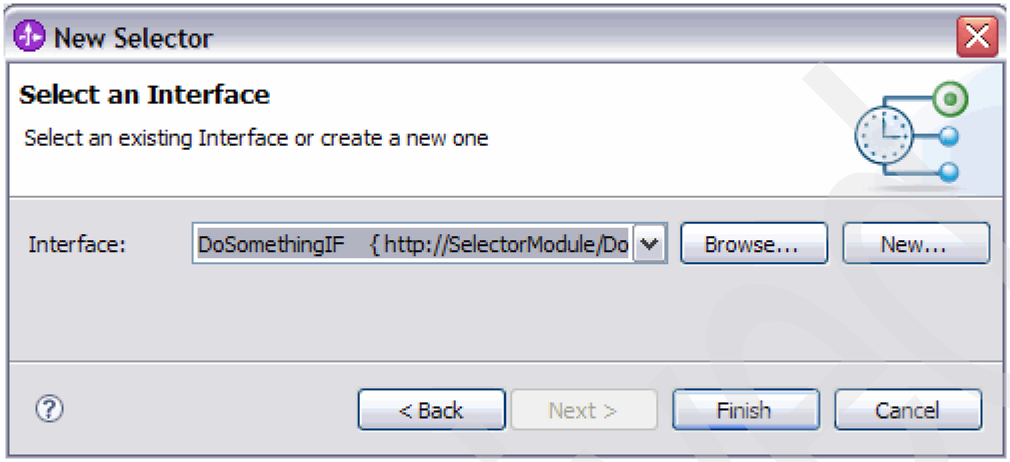
Folder:

Name: SampleSelector

Figure 4-123 New Selector wizard

3. In the Select an Interface window (Figure 4-124):

   a. Browse to an existing interface or click **New** to create one.
   b. Click **Finish**.



New Selector

Select an Interface

Select an existing Interface or create a new one

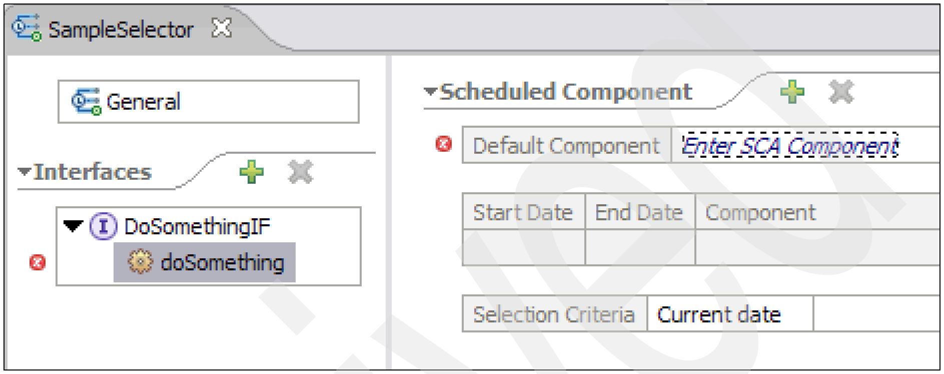
Interface: DoSomethingIF {http://SelectorModule/Do

Figure 4-124 Choosing an interface for a selector

> **Note:** At this point, you might see an error in the Problems view indicating that the target component association to an operation is missing. This error is corrected after you configure the selector.
>
> The definition of the doSomething operation is missing the target component destination.

4.9.2 Configuring the selector

You can use the selector editor, a graphical editor, to configure the selector. After creating the selector, it opens in the selector editor as shown in Figure 4-125. Click the doSomething operation to associate it with the components in the selector editor. You might need to expand DoSomethingIF interface and select the operation, doSomething, that is contained within it.



Default Component: Enter SCA Component

Start Date	End Date	Component

Selection Criteria: Current date

Figure 4-125 SampleSelector configuration in the selector editor

You typically follow these steps while configuring the selector:

1. Configure selection criteria.
2. Configure default destination or default component.
3. Add date selection entries and configure target components.

A selector is exposed to the caller as an interface. The actual implementation for this interface is provided by the target components that are configured as destinations in the selector.

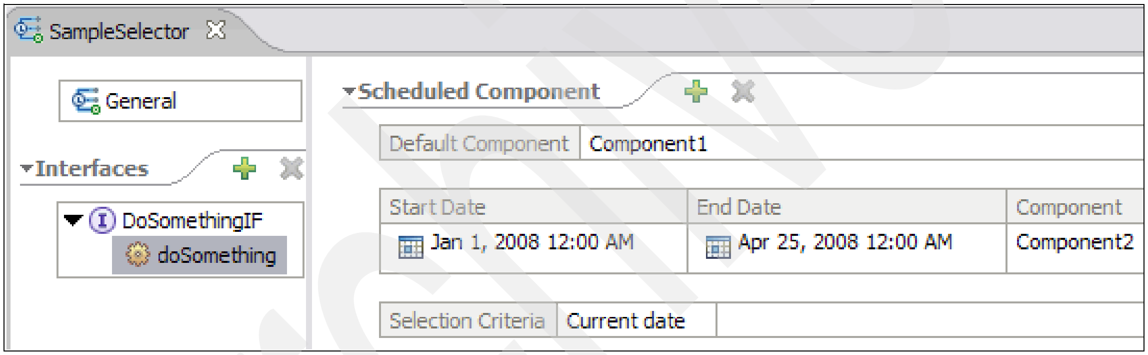
Whenever a selector is invoked, it selects a target component implementation using the selection criteria and date range entries. Selection criteria can be either a current date, an XPath expression that gets a date from a business object, or a Java snippet that returns a date. Click **Current date** link under Scheduled Component section of the selector editor to choose one of these options.

A date selection entry specifies a range of dates, during which a target component would be applicable. If the date that is returned from the selection criteria is within the date range, then the corresponding component entry is used. You can add date range entries by clicking the Add Date Selection Entry (✚) icon to add a selection entry (a row showing component with start and end dates).

The date selection entries are optional. In fact, if no date selection entry is specified or if no date range matches, the component that is configured as default component is invoked. Click **Enter SCA Component** next to Default Component label under the Scheduled Component section of the selector editor to choose a default target component. When you click this option, it displays a list of available components. The list includes the components in the same module and exports from other modules.

> **Note:** Keep in mind that if you do not specify a default destination and no date range entry matches the selection criterion, then an exception is thrown. Therefore, it is a good practice to always provide a default destination.

Figure 4-126 shows the configuration of a sample selector. The selector is configured to select Component2 if the current date is within the date range of Jan 1, 2008 to Apr 25, 2008. Otherwise it selects the default component, Component1.



Default Component: Component1

Start Date	End Date	Component
Jan 1, 2008 12:00 AM	Apr 25, 2008 12:00 AM	Component2

Selection Criteria: Current date

Figure 4-126 Configuration of SampleSelector

Using the WebSphere Process Server administrative console, you can alter existing destinations to replace an existing service implementation with another one or add new destinations of a selector and their corresponding date ranges at run time without redeploying the original application and without requiring a restart of the application or server.

4.9.3 Adding selectors to the assembly diagram

Similar to all SCA components, selectors can be added to the assembly diagram, allowing you to connect the selector to other parts of the module or connect it to a calling module through an export.

Figure 4-127 shows the assembly diagram of a sample selector module.

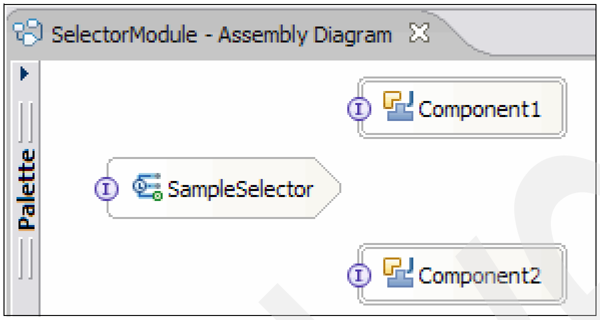


Figure 4-127 Assembly diagram with SampleSelector

Unlike other SCA components on the assembly diagram, you do not wire selectors to other components. In fact, you cannot start a wire from a selector because they make use of *dynamic wiring*. Dynamic wiring means the selector has predefined wires or target components which cannot be configured or bound to with a simple static wiring. This dynamic wiring is taken care of internally when you configure the selector as discussed previously. So, there is no need to wire the selector to any components because the destinations in the selector are not statically bound and because they might have destinations (exports from other modules) in other modules.

4.10 Interface maps

Business integration applications often contain components that were created for different applications. Typically, the interfaces of these components need a bridging component to make them interact. For example, two components can have methods that perform basically the same action but have different names, such as getData and getInformation.

An *interface map* is a bridge component between two SCA components that have interfaces with different method signatures, enabling them to communicate. An interface map maps the operations and parameters of these methods so that the differences are resolved and the two components can interact.

Interface mapping provides the context information that is required for the maintenance of relationships. There are two levels of interface mapping:

- Operation mappings

  In operation mappings, operations of the source interface are mapped to operations of the target interface (for example, a source operation called `getData()` to a target operation called `getInformation()`). An operation in one interface is bound to an operation in another interface. Operation mappings can have parameter mappings.

- Parameter mappings

  Parameter mappings are one level deeper than operation mappings. They map data from a source business object to a target business object. When a calling operation has different parameters than the receiving operation, an interface map reconciles the parameters.

  Parameter mapping is essential in the process of moving information between different applications and also between components within the WebSphere Process Server system.

  You can select the following types of parameter mappings:

  - *Map*: Reconciles parameters between business objects that have different fields (field names, number of fields, and so forth). Selecting this type of mapping allows you to use a business object map to define the parameter mappings. You can use an existing business object map or create a new one.
  - *Extract*: Extracts pertinent information from a complex parameter to reconcile it with output parameters.
  - *Move*: Makes a simple connection between source and target parameters of the same type.
  - *Custom* (Java): Calls Java code to perform the mapping.
  - *Assign*: Rather than mapping an input parameter to an output parameter, you can select the Assign transform option on the output parameter to assign a value to it.

Note: You can use interface maps only in modules for business services, not for mediation services.

4.10.1 Creating an interface map

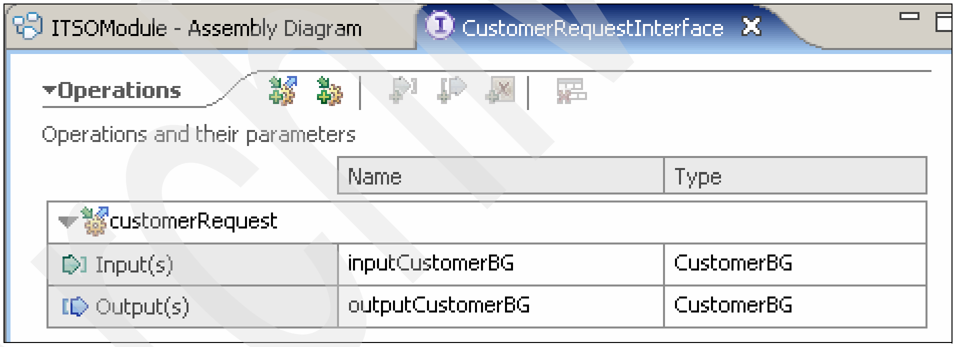
Working with interface maps involves the following steps:

1. Create the interface map artifact, which requires you to select the source and target interfaces.
2. Use the interface mapping editor to map the operations between the interfaces.
3. Use the interface mapping editor to map the parameters between the source and target operations.
4. Add the interface map to the assembly diagram.

Example

This example illustrates the use of an interface map that maps between two interfaces. It assumes the following:

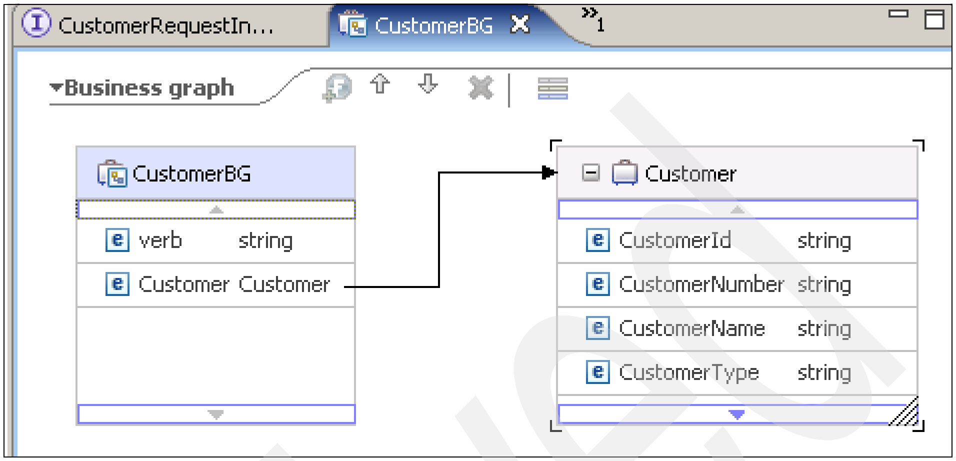
- A module called ITSOModule.
- The interface to the module is CustomerRequestInterface. It has one request-response operation called customerRequest, shown in Figure 4-128.


ITSOModule - Assembly Diagram
CustomerRequestInterface
Operations
Operations and their parameters

	Name	Type
customerRequest		
Input(s)	inputCustomerBG	CustomerBG
Output(s)	outputCustomerBG	CustomerBG

Figure 4-128 The customerRequestion operation

Both the input and output for this operation are of type CustomerBG, as shown in Figure 4-129.



CustomerBG

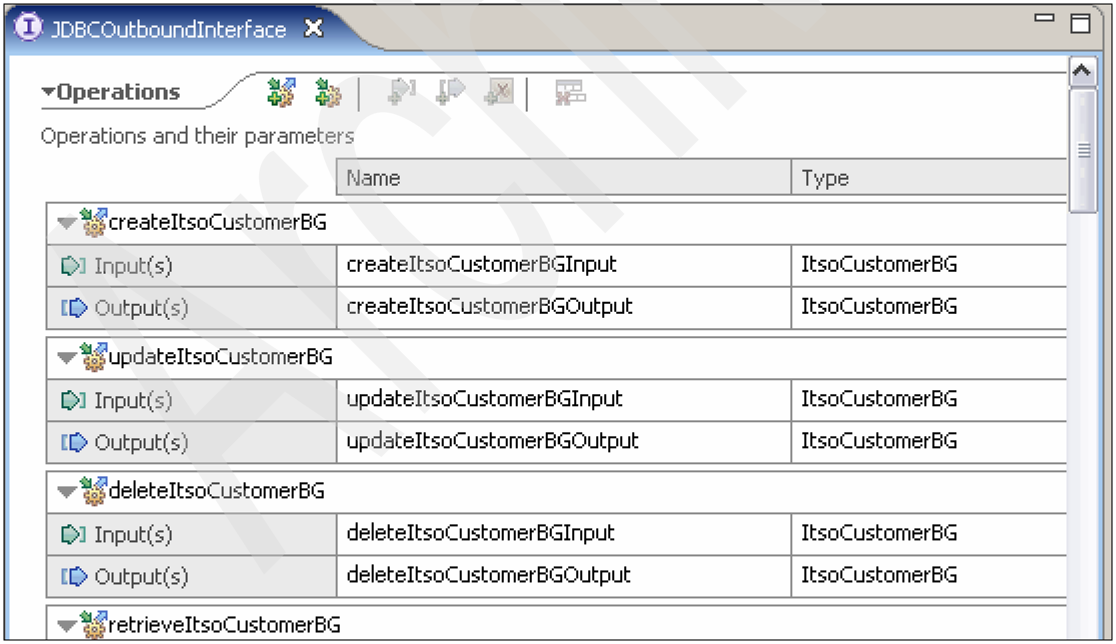
verb	string
Customer	Customer

Customer

CustomerId	string
CustomerNumber	string
CustomerName	string
CustomerType	string

Figure 4-129 customerBG business graph

- An import component represents a JDBC adapter has an interface called JDBCOutboundInterface, as shown in Figure 4-130.



Operations and their parameters

	Name	Type
createItsoCustomerBG		
Input(s)	createItsoCustomerBGInput	ItsoCustomerBG
Output(s)	createItsoCustomerBGOutput	ItsoCustomerBG
updateItsoCustomerBG		
Input(s)	updateItsoCustomerBGInput	ItsoCustomerBG
Output(s)	updateItsoCustomerBGOutput	ItsoCustomerBG
deleteItsoCustomerBG		
Input(s)	deleteItsoCustomerBGInput	ItsoCustomerBG
Output(s)	deleteItsoCustomerBGOutput	ItsoCustomerBG
retrieveItsoCustomerBG		

Figure 4-130 JDBCOutboundInterface

The input and output type for the operation used in the example, createItsoCustomerBG, is ItsoCustomerBG, as shown in Figure 4-131.

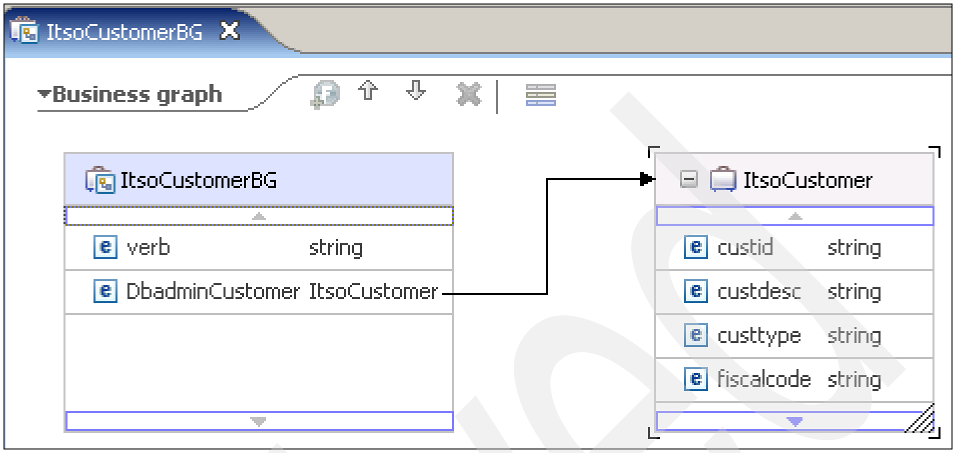


Figure 4-131 DBadminCustomerBG

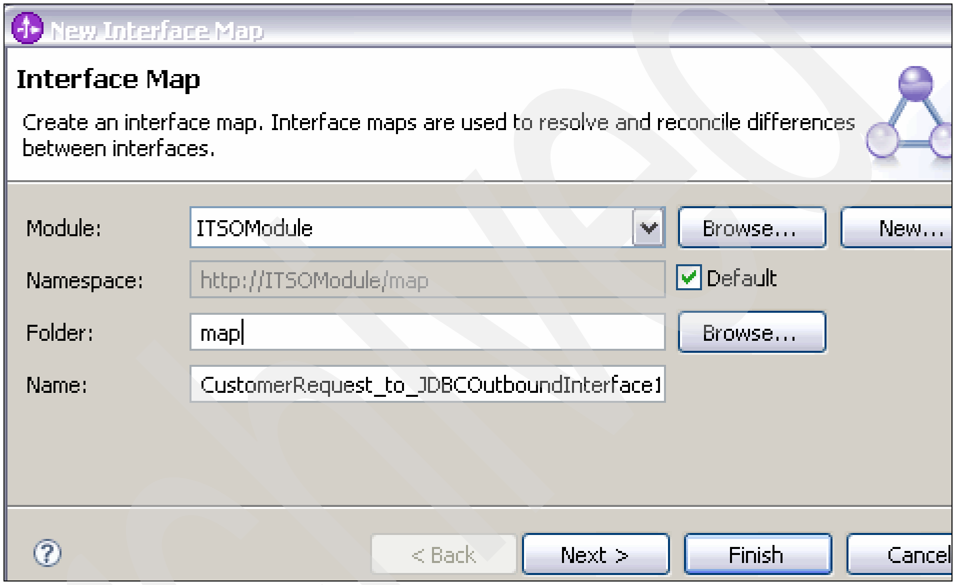
- CustomerRequest interface will be mapped to JDBCOutboundInterface.
- A Business Object Map, CustomerBG_to_ItsoCustomerBG. We use this map while mapping the parameters in step 5 under “Map the interface operations and parameters” on page 278.

  We discuss how to create business object maps in 3.7.2, “Creating a business object map” on page 81.

Create the interface map

To create a new interface map, follow these steps:

1. In the Business Integration view, expand the library or module and select the Mapping folder. Right-click and select **New** → **Interface Maps**. If this option is not available, select **New** → **Other** → **Business Integration** → **Interface Maps**.
2. In the New Interface Map dialog box (Figure 4-132), enter the name and location for the new map. Click **Next**.



New Interface Map

Interface Map

Create an interface map. Interface maps are used to resolve and reconcile differences between interfaces.

Module: ITSOModule

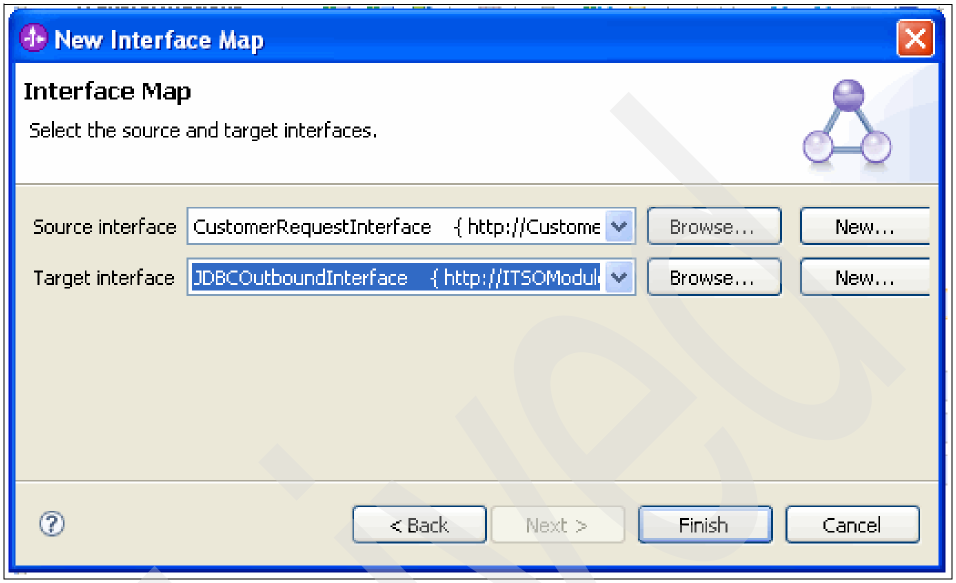
Namespace: http://ITSOModule/map ☑ Default

Folder: map

Name: CustomerRequest_to_JDBCOutboundInterface1

Figure 4-132 Create a New Interface Map wizard

3. Use the Browse button to find and select the source and target interfaces for the mapping, and click **Finish**.



New Interface Map

Interface Map

Select the source and target interfaces.

Source interface: CustomerRequestInterface {http://Custome

Target interface: JDBCOutboundInterface {http://ITSOModul

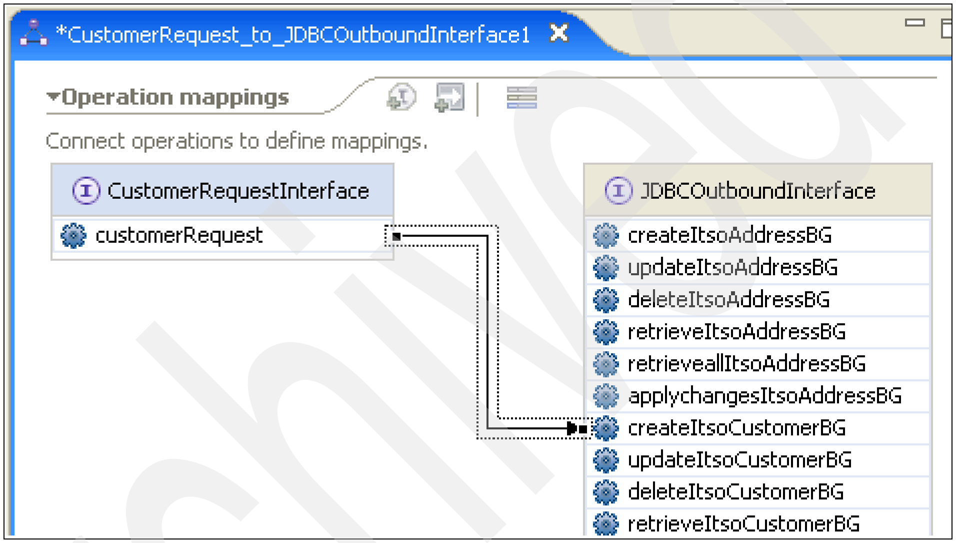
Figure 4-133 Select the source and target interfaces

The map is created and opens in the interface mapping editor.

Map the interface operations and parameters

The next step is to define the operation bindings and parameter transformations in the new interface map using the interface mapping editor. To do this, you connect operations in the source interface to operations in the target interface. Follow these steps:

1. In the Operation mappings section, select the source operation (customerRequest in Figure 4-134). As you hover the mouse over the operation an orange circle displays. Select the circle, and drag it to the target operation (createItsoCustomerBG operation).



*CustomerRequest_to_JDBCOutboundInterface1

Operation mappings

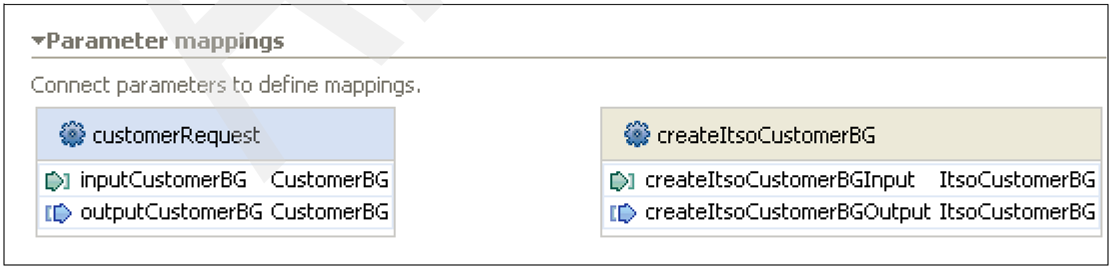
Connect operations to define mappings.

CustomerRequestInterface
- customerRequest

JDBCOutboundInterface
- createItsoAddressBG
- updateItsoAddressBG
- deleteItsoAddressBG
- retrieveItsoAddressBG
- retrieveallItsoAddressBG
- applychangesItsoAddressBG
- createItsoCustomerBG
- updateItsoCustomerBG
- deleteItsoCustomerBG
- retrieveItsoCustomerBG

Figure 4-134 Operation mappings wizard

2. Selecting the wire between the two operations opens a new section in the editor called *Parameter mappings*, which allows you to map parameters within the interface operations (Figure 4-135). You might have to scroll past all of the operations to view the Parameter mappings section.



Parameter mappings

Connect parameters to define mappings.

customerRequest	
inputCustomerBG	CustomerBG
outputCustomerBG	CustomerBG

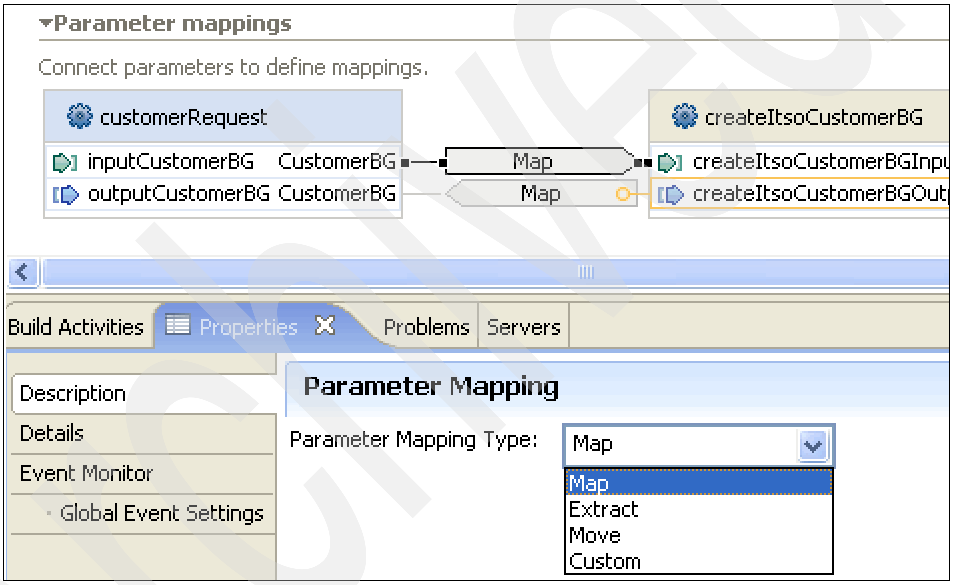
createItsoCustomerBG	
createItsoCustomerBGInput	ItsoCustomerBG
createItsoCustomerBGOutput	ItsoCustomerBG

Figure 4-135 Parameter mapping view

3. Using the same method as in the previous step, connect the source parameters to the target parameters.

> **Note:** You have to drag the connection for the output in the opposite direction (from createItsoCustomerBGOutput on the right to outputCustomerBG on the left).

4. By default, each connection has a Move parameter mapping type. To change the mapping type, (for example to *Map*) select a new type from the drop-down menu for Parameter Mapping Type in the Description tab of the Properties view. The options and tabs in the Properties view change to accommodate the mapping type that you select. See Figure 4-136.



Parameter mappings

Connect parameters to define mappings.

customerRequest			createItsoCustomerBG
inputCustomerBG	CustomerBG	Map	createItsoCustomerBGInpu
outputCustomerBG	CustomerBG	Map	createItsoCustomerBGOut

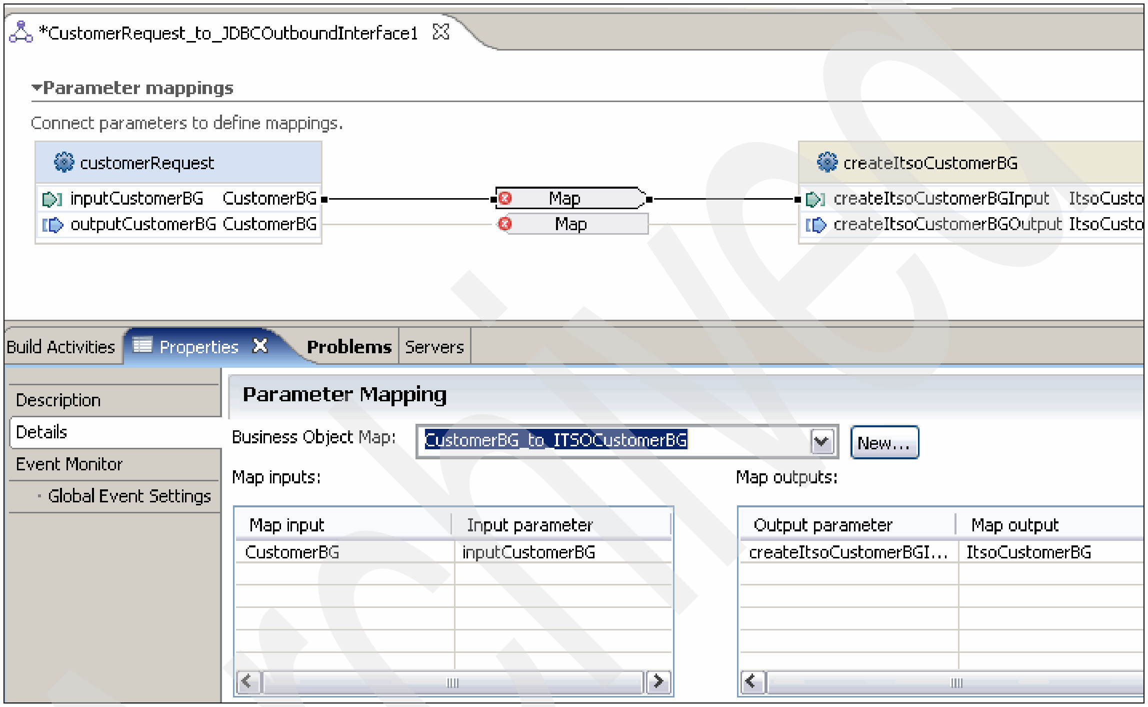
Parameter Mapping

Parameter Mapping Type: Map

Figure 4-136 Connect parameters to define mappings and Parameter Mapping

5. In the example, Map is selected as the mapping type for both connections. Select **Map** and go to the Details tab of the Properties view to define the map. You can select an existing map from a drop-down menu or to create a new map.

   In this example, we select an existing map, CustomerBG_to_ItsoCustomerBG map, from the Business Object Map drop-down list for the input mapping, as shown in Figure 4-137.



*CustomerRequest_to_JDBCOutboundInterface1

Parameter mappings

Connect parameters to define mappings.

customerRequest

inputCustomerBG	CustomerBG
outputCustomerBG	CustomerBG

createItsoCustomerBG

createItsoCustomerBGInput	ItsoCusto
createItsoCustomerBGOutput	ItsoCusto

Parameter Mapping

Business Object Map: CustomerBG_to_ITSOCustomerBG

Map inputs:

Map input	Input parameter
CustomerBG	inputCustomerBG

Map outputs:

Output parameter	Map output
createItsoCustomerBGI...	ItsoCustomerBG

Figure 4-137 Parameter mapping

6. Select the next Map connection for the output to select an existing map or to create a new map. In this example, we created a simple map by clicking **New** and defining new map connections between the two business graphs.

7. Save and close the completed interface map.

4.10.2 Adding the map to the assembly diagram

To use the interface map in this module:

1. Open the assembly diagram in the assembly editor by double-clicking **Assembly Diagram** for the module in the Business Integration view.
2. Drag the interface map from the Business Integration view to the assembly editor canvas as shown in Figure 4-138.

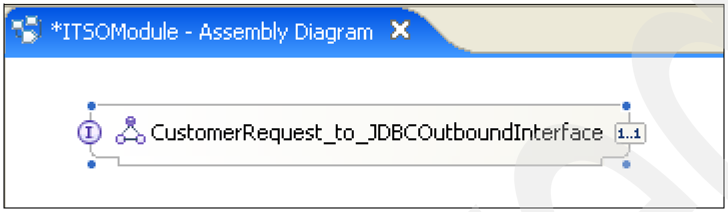

*ITSOModule - Assembly Diagram
I
CustomerRequest_to_JDBCOutboundInterface
1..1

Figure 4-138 CustomerRequest_to_JDBCOutboundInterface map in the Assembly diagram

3. Select the interface map in the diagram, right-click, and select **Generate Export → SCA Binding**.

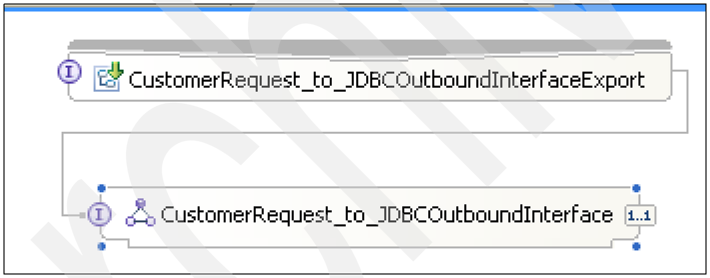
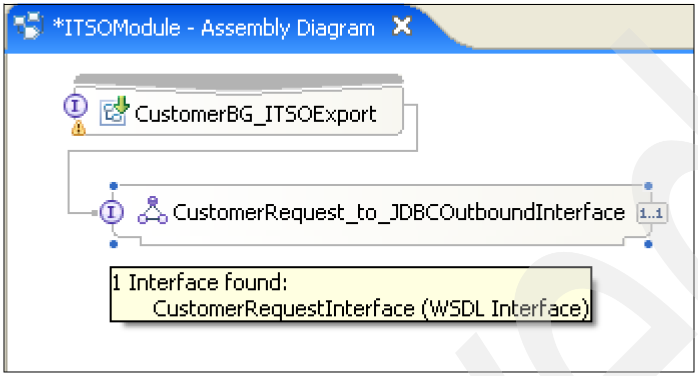

I
CustomerRequest_to_JDBCOutboundInterfaceExport
I
CustomerRequest_to_JDBCOutboundInterface
1..1

Figure 4-139 Generated SCA binding for CustomerRequest

Note that the name in this case is unusually long. For usability, you can change the name by simply typing over the Name field in the Description tab of the Properties view. We renamed the export component to *CustomerBG_ITSOExport* as shown in Figure 4-140.


*ITSOModule - Assembly Diagram

CustomerBG_ITSOExport

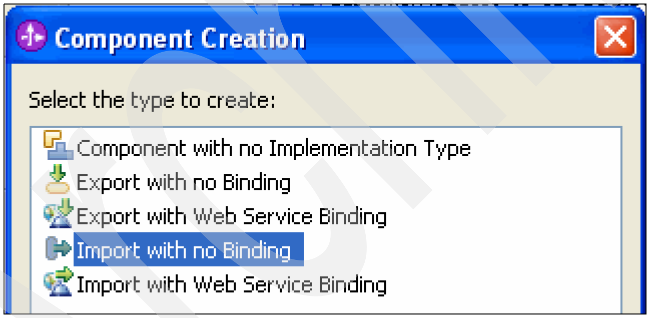
CustomerRequest_to_JDBCOutboundInterface

1 Interface found:
CustomerRequestInterface (WSDL Interface)

Figure 4-140 Rename the Export

4. Drag and drop the interface (JDBCOutbound) from the Business Integration view to the assembly editor. Then, follow these steps:

   a. Select **Import with no Binding** and click **OK**.


Component Creation

Select the type to create:

- Component with no Implementation Type
- Export with no Binding
- Export with Web Service Binding
- Import with no Binding
- Import with Web Service Binding

Figure 4-141 Generating an SCA Binding to Export component

b. Change the name of the of the JDBCOutboundInterfaceImport component to *JDBCOutboundInterface* (for usability) as shown in Figure 4-142.

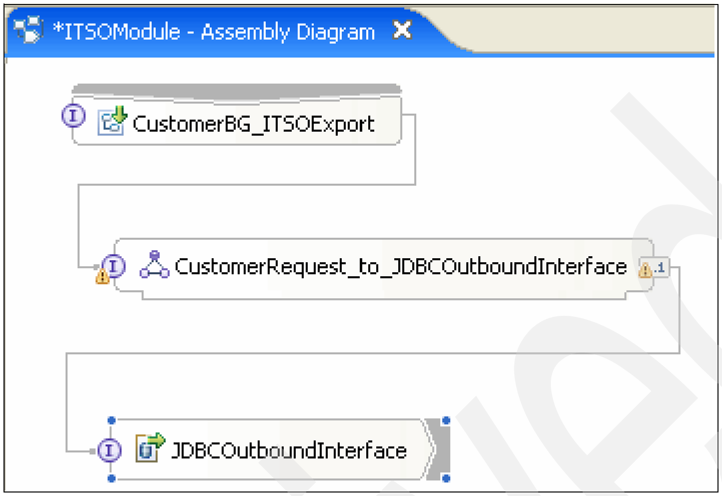

*ITSOModule - Assembly Diagram
CustomerBG_ITSOExport
CustomerRequest_to_JDBCOutboundInterface
JDBCOutboundInterface

Figure 4-142 Assembled ITSOModule with all the part

5. Wire the interface map to the import component.
6. Save and close the updated assembly diagram.

You can use the export for this module on the assembly diagram of another module as an import component.

5

Building mediations

WebSphere Enterprise Service Bus is a product that is designed to implement an enterprise service bus in a service-oriented architecture (SOA) solution. WebSphere Enterprise Service Bus functionality is also integrated into WebSphere Process Server. When we refer to *WebSphere Enterprise Service Bus* in this book, we are referring to either case; however, we implemented all of our samples with WebSphere Process Server.

This chapter discusses the architecture of WebSphere Enterprise Service Bus and how to build mediations for WebSphere Enterprise Service Bus using WebSphere Integration Developer. It includes the following topics:

- WebSphere Enterprise Service Bus architecture
- Service message objects
- Typical development flow
- Creating a mediation
- Service connection and invocation
- Transformation primitives
- Routing primitives
- Tracing primitives
- Error Handling primitives
- Custom Mediation primitive

> **Additional material:** We illustrate many of the primitives that we discuss in this chapter through samples that are included in the additional materials. For information about downloading the additional materials, see Appendix B, “Additional material” on page 511. The mediation examples are included in the MedationSamples.zip project interchange file.

5.1 WebSphere Enterprise Service Bus architecture

This section explores the structure of WebSphere Enterprise Service Bus by working through the different layers of the product architecture in a top-down manner.

5.1.1 Mediations, service consumers, and service providers

A service interaction in SOA defines both service consumers and service providers. The role of WebSphere Enterprise Service Bus is to intercept the requests of service consumers and fulfill additional tasks in mediations in order to support loose coupling. When the mediation completes, the relevant service providers are invoked.

The mediation tasks include:

- Centralizing the routing logic so that service providers can be exchanged transparently
- Performing tasks such as protocol translation and transport mapping
- Acting as a facade to provide different interfaces between service consumers and providers
- Adding logic to provide tasks such as logging

As shown in Figure 5-1, mediations customize the protocol and the details of a request and also modify the results of the reply.

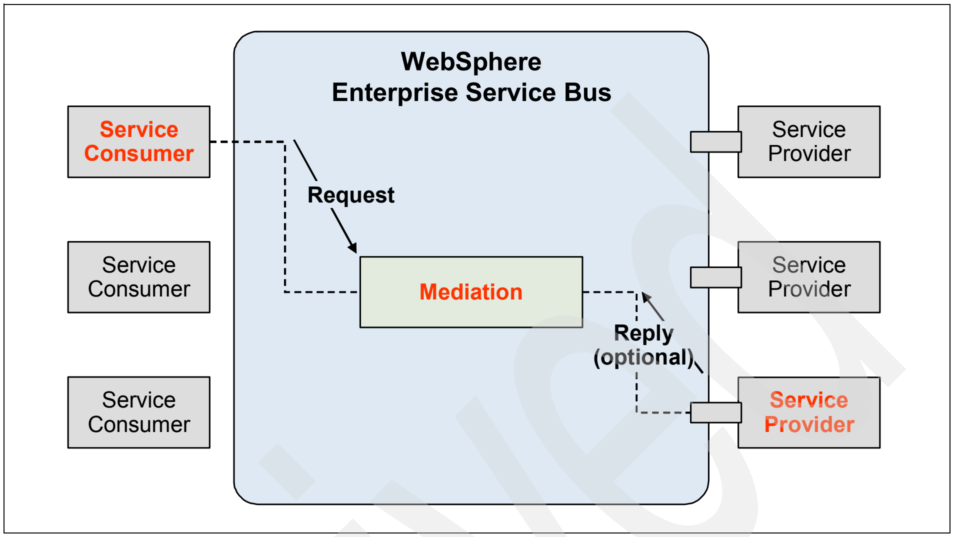


Figure 5-1 WebSphere Enterprise Service Bus and mediations

WebSphere Enterprise Service Bus can interconnect a variety of different service consumers and providers using standard protocols including:

- JMS
- SOAP over HTTP (for Web services)
- SOAP over JMS (for Web services)
- HTTP

For back-end applications (such as SAP), several IBM WebSphere Adapters are available.

WebSphere Enterprise Service Bus supports diverse messaging interaction models to meet your requirements, including the following models:

- One-way interactions
- Request-response
- Publish/subscribe

5.1.2 Mediation modules

The *mediation module* is a type of SCA component that can process or mediate service interactions. As illustrated in Figure 5-2, the mediation module is externalized or made available through an *export*, which specifies the interfaces that are exposed. These interfaces are defined in a WSDL document. *Stand-alone references* provide the externalized interface only for SCA clients. They do not define a WSDL document. Instead, they specify the interface declaration in Java (called a *reference*).

The mediation module typically invokes other service providers. These providers are declared with the creation of an *import*, which represents an external service to be invoked.

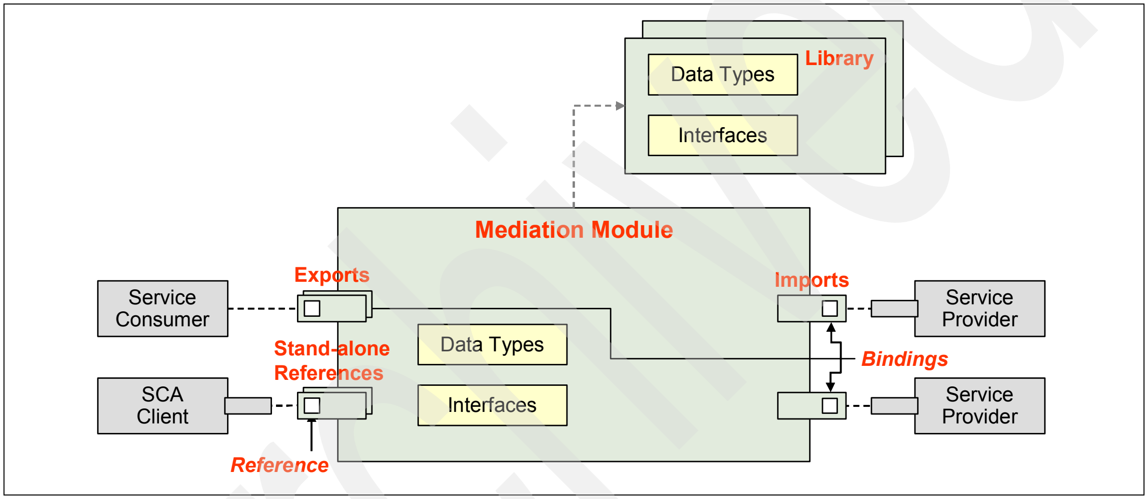


Figure 5-2 Mediation modules

For each export and import, you need to specify an interface. Each interface has multiple operations, which in turn can have multiple input and output parameters that are associated with either simple data types or business objects. A one-way operation has only input parameters.

Every export and import must be associated with a *binding*. A binding identifies a specific type of invocation for a service consumer or provider. WebSphere Enterprise Service Bus supports the same bindings that WebSphere Process Server supports (for more information, see 2.1.3, “Import and export bindings” on page 17).

Finally, data types (business objects) and interfaces can be defined on the module level, but they can also be defined and referenced in *libraries* in order to centralize them.

5.1.3 Mediation flow components

Inside a mediation module there can be one *mediation flow component*. Mediation flow components offer one or more *interfaces* and use one or more *partner references*. Both of these components are resolved and are assigned to exports or imports using wires, as shown in Figure 5-3.

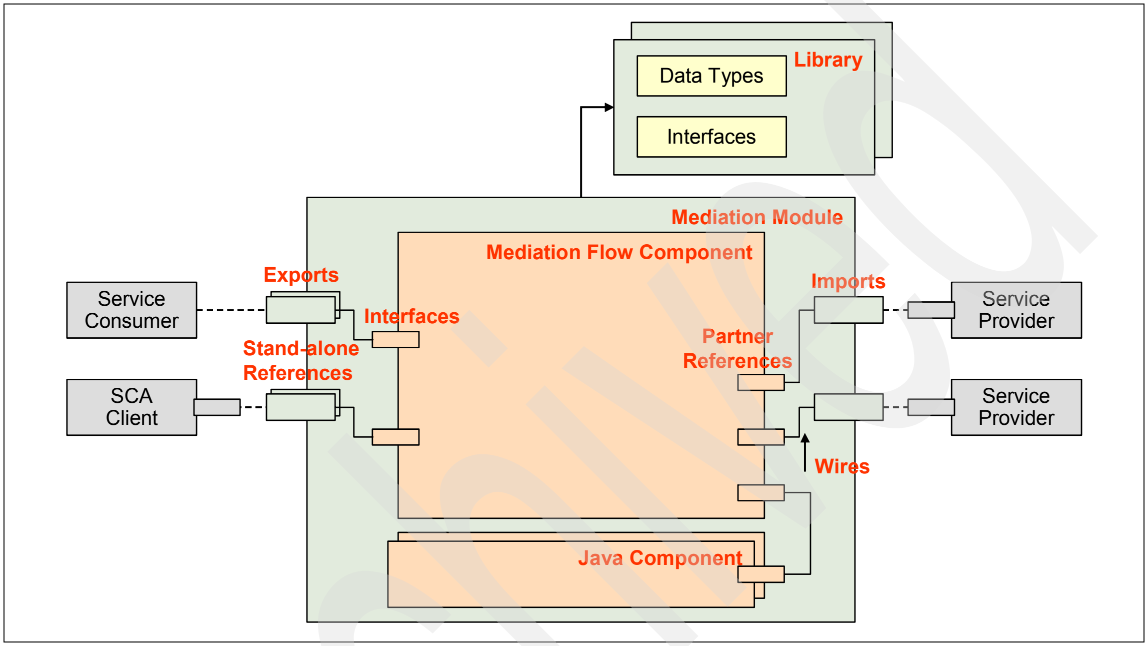


Figure 5-3 Mediation flow component

In addition to the mediation flow component, a mediation module can have one or more Java components that are created using custom mediation implementations.

Restriction: WebSphere Integration Developer does not stop you from creating more than one mediation flow component per mediation module; however, only one mediation flow is allowed. Therefore, there is a one-to-one relationship between a mediation module and a mediation flow component.

5.1.4 Mediation flows

Mediation flows (Figure 5-4) contain the high-level mediation logic. Thus, in a mediation flow, the different processing steps of a request are declared in a graphical way. In WebSphere Enterprise Service Bus, the processing of requests is separated from processing of responses. Therefore, we distinguish between a *request flow* and a *response flow*. In both directions, you can apply logic or modifications.

> **Note:** You need to define mediation flows for *every operation* that gets exposed using an export of a mediation module. For those operations that do not need any additional functionality to the wrapped interface, you wire them from input to input response.

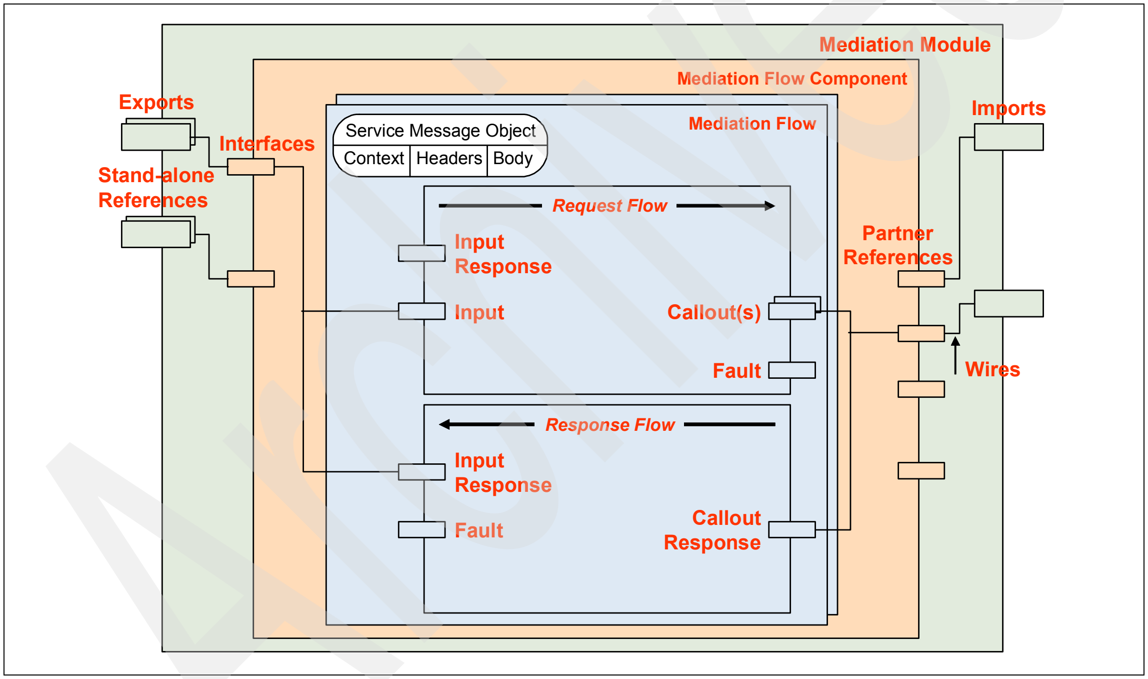


Figure 5-4 Mediation flows

Mediation flows consist of a sequence of processing steps that are executed when an input message is received. A *request flow* begins with a single *Input node* for the source operation and can have multiple *Callout nodes*. If a message is to be returned to the source directly after processing, it can be wired to an *Input response node* in the request flow. If fault messages are defined in the source operation, an *input fault* is also created.

A *response flow* begins with one or more *Callout response nodes* and ends with a single input response (and optionally a callout fault). Both a request flow and a response flow are associated with a mediation flow. The request flow can map data to a correlation context and the transient context.

In terms of the actual data, WebSphere Enterprise Service Bus introduces the service message object (SMO). SMO is a special kind of a service data object that represents the content of an application message as it passes through a mediation flow component. As well as the payload in the body, SMO contains context and header information that can be accessed and acted upon inside the mediation flows.

5.1.5 Mediation primitives

Mediation primitives (Figure 5-5) are the smallest building blocks in WebSphere Enterprise Service Bus. They are wired and configured inside mediation flows. With mediation primitives, you can change the format, content, or target of service requests, as well as log messages, perform database lookups, and so forth.

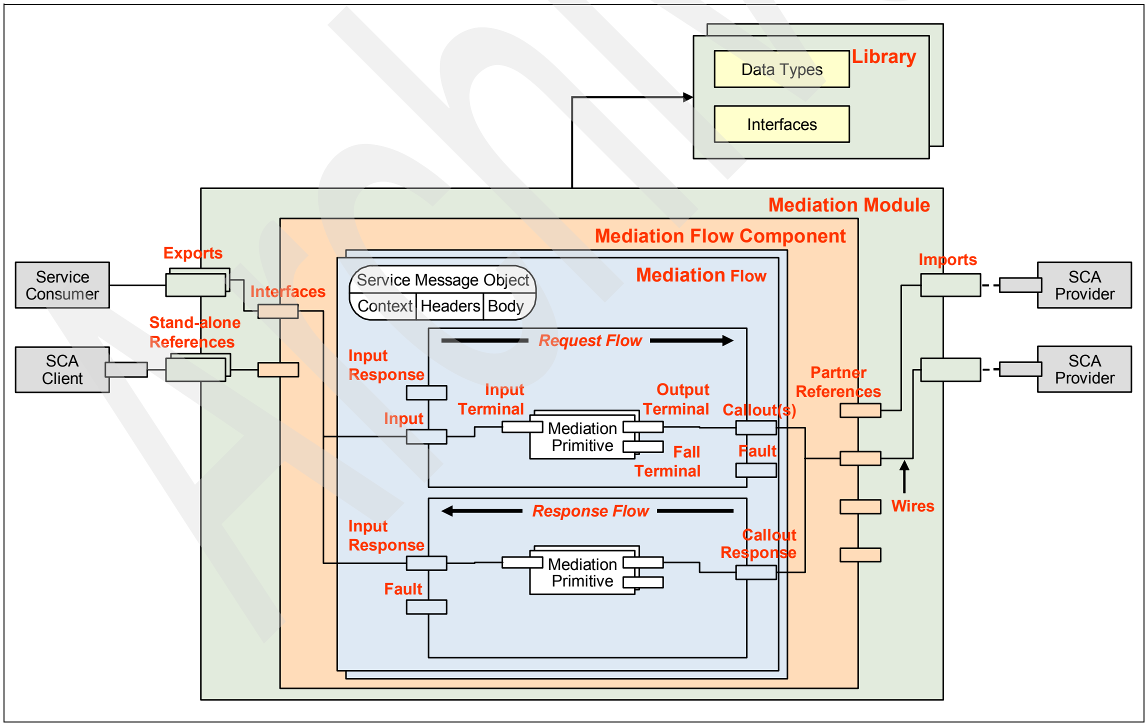


Figure 5-5 Mediation primitives (in the complete overview)

WebSphere Integration Developer and WebSphere Enterprise Service Bus V6.1 provides the following standard mediation primitives:

- The *Message Logger* primitive (V6.0) logs a copy of a message to a database for future retrieval or audit. For example, the integration developer can customize the primitive by naming the database.
- The *Database Lookup* primitive (V6.0) retrieves values from a database to add them to a message.
- The *Message Filter* primitive (V6.0) compares the content of a message to expressions that are configured by the developer and routes the message to the next primitive based on the result.
- The *XSL Transformation primitive* (V6.0) transforms messages according to transformations that are defined by an XSL style sheet.
- The *Fail* primitive (V6.0) throws an exception and terminates the path through the mediation flow.
- The *Stop* primitive (V6.0) silently terminates the path through the mediation flow.
- The *Business Object Map* primitive (V6.1) allows you to define message transformation using a business object map.
- The *Service Invoke* primitive (V6.1) calls a service from inside a mediation flow. If the service returns a fault, a retry option is available to call the same service or a different one.
- The *Fan Out* primitive (V6.1) is used to create different messages from a repeating element in the input message.
- The *Fan In* primitive (V6.1) is used to combine multiple messages that were created using Fan Out mediation.
- The *Endpoint Lookup* primitive (V6.0.2) searches for service information in a WebSphere Service Registry and Repository.
- The *Event Emitter* primitive (V6.0.2) emits events from inside a mediation flow.
- The *Message Element Setter* primitive (V6.0.2) sets the content of messages.
- The *Set Message Type* primitive (V6.1) allows the user to overlay message fields with more detailed structures.
- The *Custom Mediation primitive* allows the user to implement their own mediate method using Java. The Custom mediation, similar to other primitives, receives an SMO and returns an SMO. It can be used to perform tasks that cannot be performed using the other primitives.

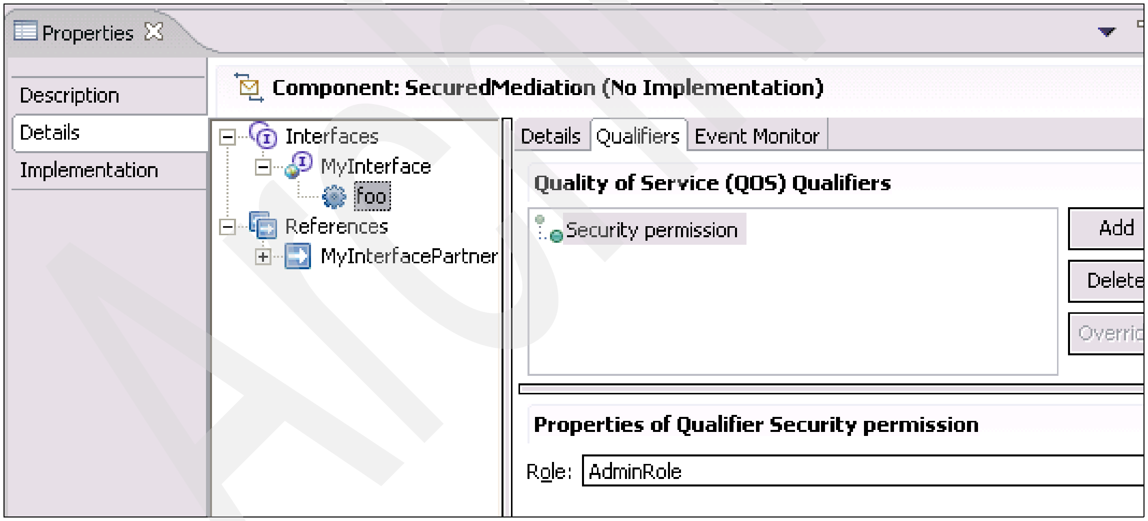
Mediation primitives use terminals for message flow. The type and number of terminals depends on the type of primitive. The three types of terminals are:

- *In terminal*: All primitives have an input terminal that can be wired to accept a message.
- *Out terminal*: Most primitives have one or more output terminals that can be wired to propagate a message (exceptions are the stop and the fail primitive).
- *Fail terminal*: If an exception occurs during the processing of an input message, then the Fail terminal propagates the original message, together with any exception information.

5.1.6 Security

In WebSphere Integration Developer, you specify *security attributes* for mediation flow components in the Properties view at the boundaries and the implementation of an component (Figure 5-6).

At the mediation flow component level, you can define the role a user must have in order to call the interface. At the interface level, you can define the permission for every operation.


Component: SecuredMediation (No Implementation)

Quality of Service (QOS) Qualifiers

Security permission

Properties of Qualifier Security permission

Role: AdminRole

Figure 5-6 Security permission qualifier on interfaces

5.2 Service message objects

Messages can come from a variety of sources, so the payload must be able to carry a number of different types of messages. Mediation primitives need to be able to operate on these messages, and SMOs represent the common representation that is needed.

The types of messages that are handled by WebSphere Enterprise Service Bus include:

- SDO data object
- SDO data graph
- SCA component invocation message (request, reply or exception)
- SOAP message
- JMS message
- HTTP message

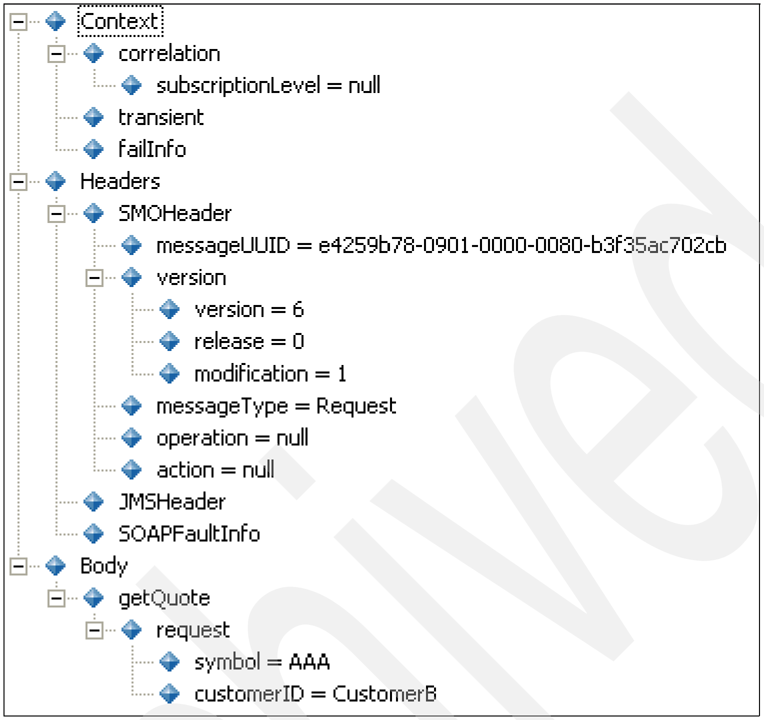
The SMO model is extensible so that it can support other message types in the future, such as COBOL structures. SMO extends SDO with additional information to support the needs of a messaging subsystem.

5.2.1 SMO structure

All SMOs have the same basic structure, which is defined by an XML schema. An SMO has three major sections:

- The *body* contains the application data (payload) of the message, particularly the input or output values of an operation.
- The *headers* contain the information relevant to the protocol that is used to send the message.
- The *context* covers the data specific to the logic of a flow or failure information.

Figure 5-7 shows a sample SMO when calling the stock quote sample that is provided with WebSphere Enterprise Service Bus.



```
Context
    correlation
        subscriptionLevel = null
    transient
    failInfo
Headers
    SMOHeader
        messageUUID = e4259b78-0901-0000-0080-b3f35ac702cb
        version
            version = 6
            release = 0
            modification = 1
        messageType = Request
        operation = null
        action = null
    JMSHeader
    SOAPFaultInfo
Body
    getQuote
        request
            symbol = AAA
            customerID = CustomerB
```

Figure 5-7 Sample SMO

Data section

The data that is carried in the SMO body is the operation that is defined by the interface specification and the inputs, outputs, and faults that are specified in the message parts set in the business object definition (Figure 5-8).

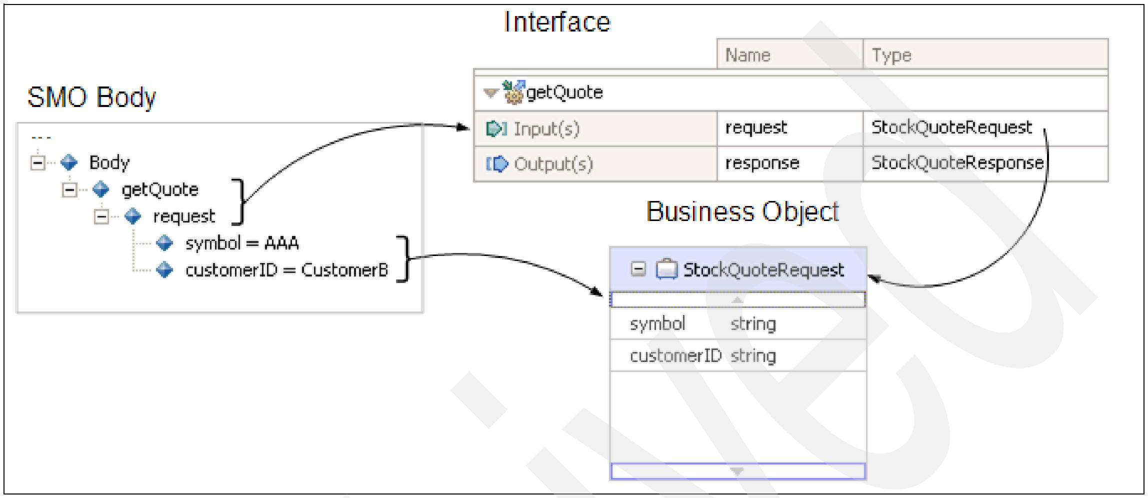


Figure 5-8 Content of the SMO body

Context section

The context includes the correlation, transient, and shared context information.

Correlation context is used to maintain data across a request/response flow, while *transient* context maintains data only in one direction. Both of these contexts are used to pass application data between mediation primitives.

Shared context is a thread-based storage area that you can use to aggregate data. There is one shared context per thread per flow. You usually have one thread per flow. So, one shared context is valid along the flow. However if you have a Service Invoke primitive, for example, that has an asynchronous invocation style with a callback outside a Fan Out and Fan In aggregation sequence, this situation creates a new thread, and the shared context will be empty after the invocation.

So, basically, there might be different copies of an SMO along a flow, each one having its correlation and transient contexts, while shared context will be unique for a given thread in a flow for all of the SMO in that thread.

Contexts are described as business objects containing XML schema that are described as data objects and that are specified on the mediation flow's input node properties. You need to design shared context objects that are suitable for all of the aggregation scenarios along the flow.

The context also includes the `failInfo`, which is added to the SMO when a Fail terminal flow is used. The information that is provided includes the `failureString` (nature of the failure), `origin` (primitive in which the failure occurred), `invocationPath` (the flow taken through the mediation), and `predecessor` (previous failure).

Context example

This section includes a general to-do list for working with contexts in a mediation flow. The example makes use of the Fan Out and Fan In primitives, which split and aggregate messages in a flow. The example follows these steps:

1. Design and define business objects for the contexts that you need.

   Correlation context business objects hold properties that you need to access along both the request and response flows. See Figure 5-9.

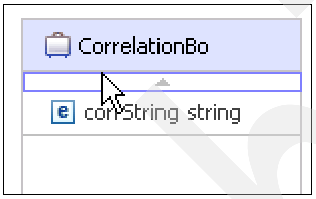


Figure 5-9 Correlation context example Business Object

   Transient business objects hold properties that you need to access along the *current* flow (might be request or response). See Figure 5-10:

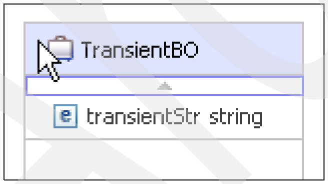
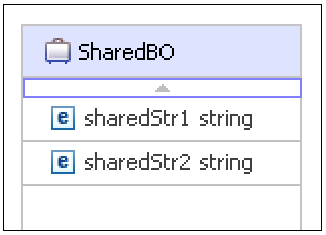


Figure 5-10 Transient context example Business Object

Shared business objects hold properties that you need for storing aggregated results from service invocations. The attributes needed in the object depend on how many services you invoke and what object they return. In this example (Figure 5-11), we invoke two services that each return one string.

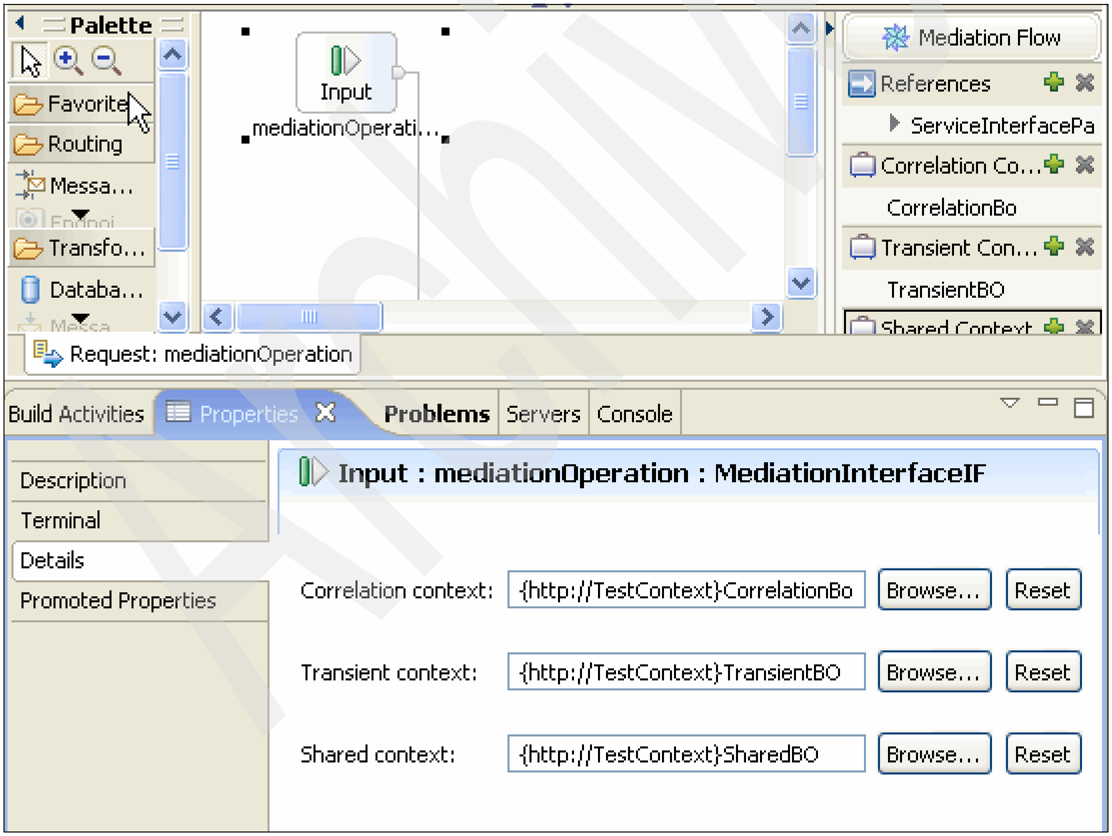

SharedBO

sharedStr1 string

sharedStr2 string

Figure 5-11 Shared context example Business Object

2. When developing the flow, you must define the context business objects in the detail properties of the Input node (Figure 5-12).



Input : mediationOperation : MediationInterfaceIF

Correlation context: {http://TestContext}CorrelationBo

Transient context: {http://TestContext}TransientBO

Shared context: {http://TestContext}SharedBO

Figure 5-12 Defining contexts at the mediation flow input node level

3. Then, move the appropriate content from the input message body elements to the correlation and transient context elements using a transformation primitive, such as an XSL Transformation primitive (Figure 5-13).

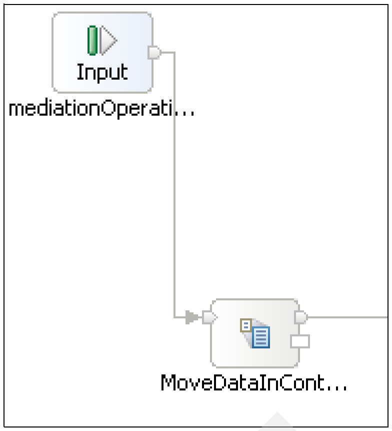
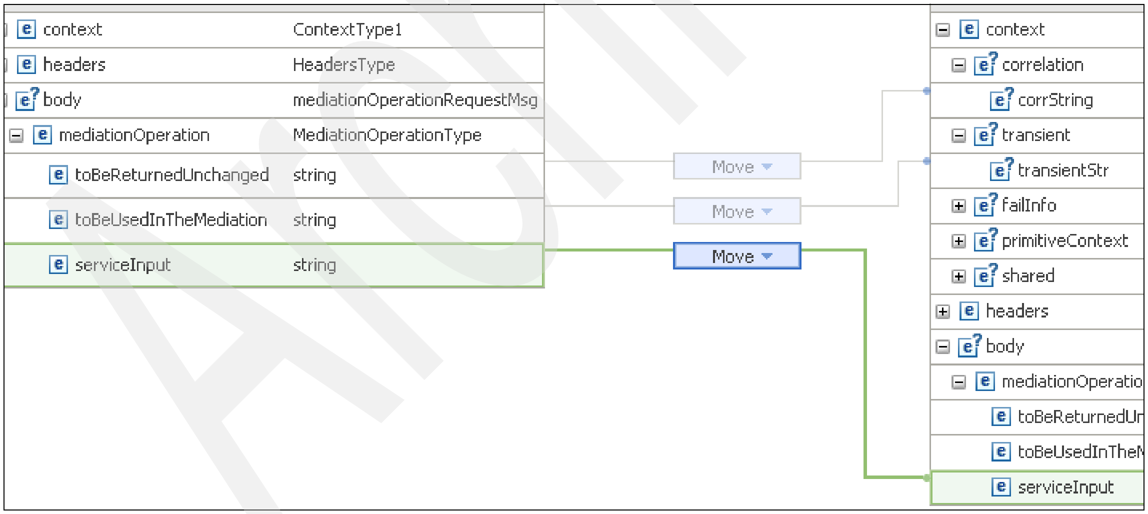


Figure 5-13 Moving body elements in the contexts

Figure 5-14 illustrates a possible implementation of such a transformation. Note how some body elements are moved to the contexts while others are carried on in the body, because they are used as service invocations inputs.



context	ContextType1
headers	HeadersType
body	mediationOperationRequestMsg
mediationOperation	MediationOperationType
toBeReturnedUnchanged	string
toBeUsedInTheMediation	string
serviceInput	string

Figure 5-14 Moving elements from the input body to the contexts

4. At a certain point in the mediation flow, you invoke your services. In this example, the flow is split by the Fan Out primitive, and the services are invoked with identical copies of the SMO message, with the shared context in common. See Figure 5-15.

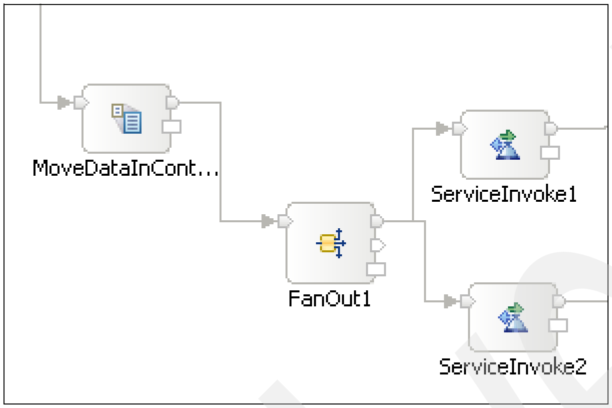


Figure 5-15 Flow split and service invocation

5. Next, use the Message Element Setter primitive or XSLT transformation primitive to aggregate the responses from the shared context of each service. In the example, the Aggregate1 and Aggregate2 XSLT Transformation primitives are placed between the Service Invoke and the Fan In that rejoins the flows (Figure 5-16).

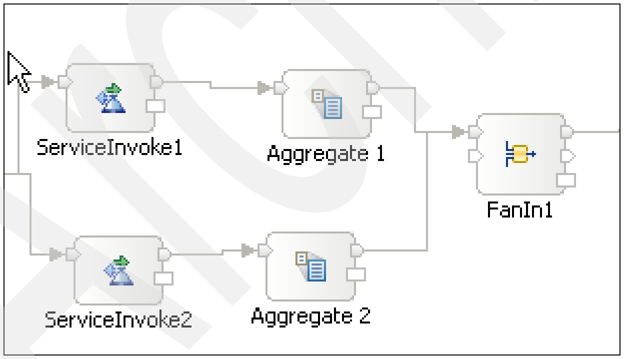


Figure 5-16 Aggregating service responses and merging flows

6. To implement Aggregate 1:

   a. Ensure that all contexts are mapped using the **Map source to target based on names and types** utility.

   b. Go into the Inline Maps implementation and remove the mapping for the element that you want to take from the message body (that is the service result). Figure 5-17 shows Aggregate1 for this example.

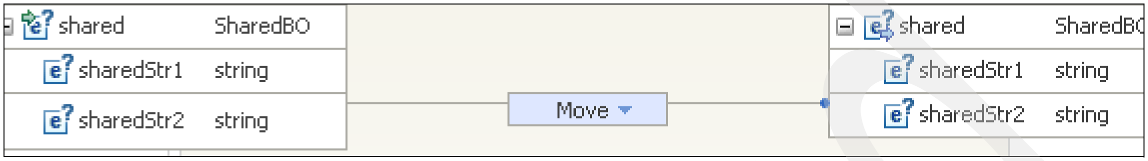


Figure 5-17 Shared context inline map must miss automatic mapping for what you carry from the body

   c. Map the body elements to the appropriate shared context elements as shown in Figure 5-18.

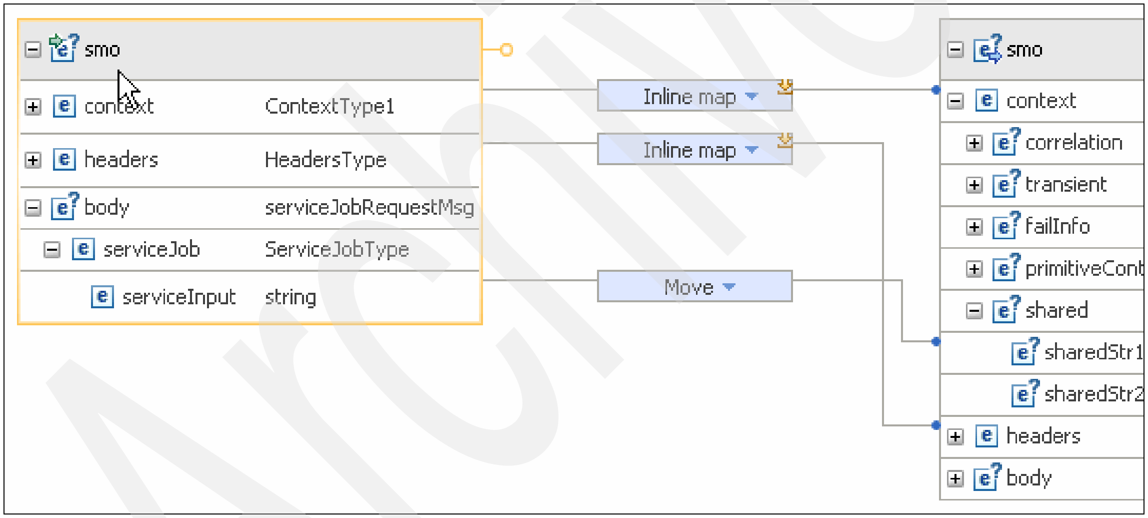


Figure 5-18 Moving service invocation response to the shared context

7. Use another transformation to get content from the transient context and to make it available to some other primitive (in this example, Custom Mediation primitive) as shown in Figure 5-19. This primitive can work with the service responses from the shared context.

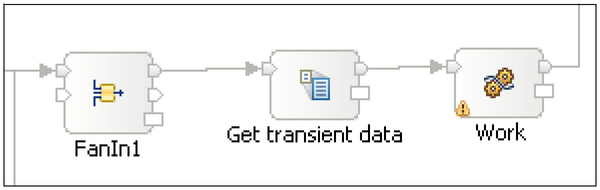
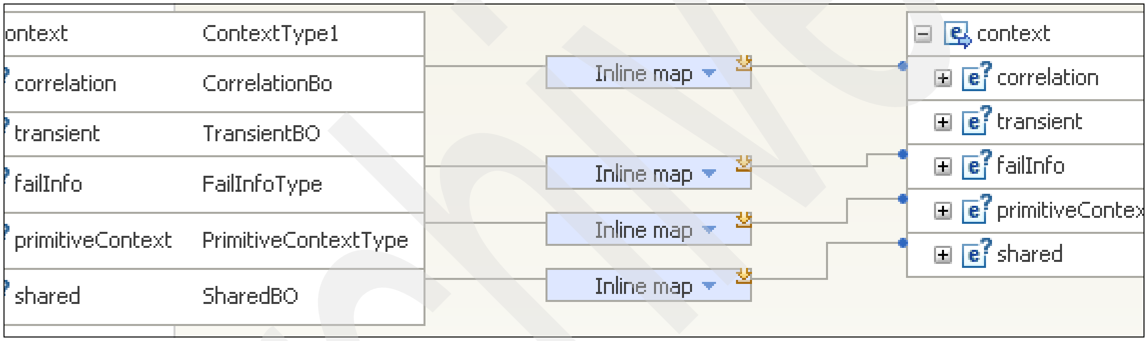


Figure 5-19 Getting transient data for further elaboration

a. Use the **Map source to target based on names and types** utility to create the initial mapping. Remove the mapping between the transient contexts (Figure 5-20).

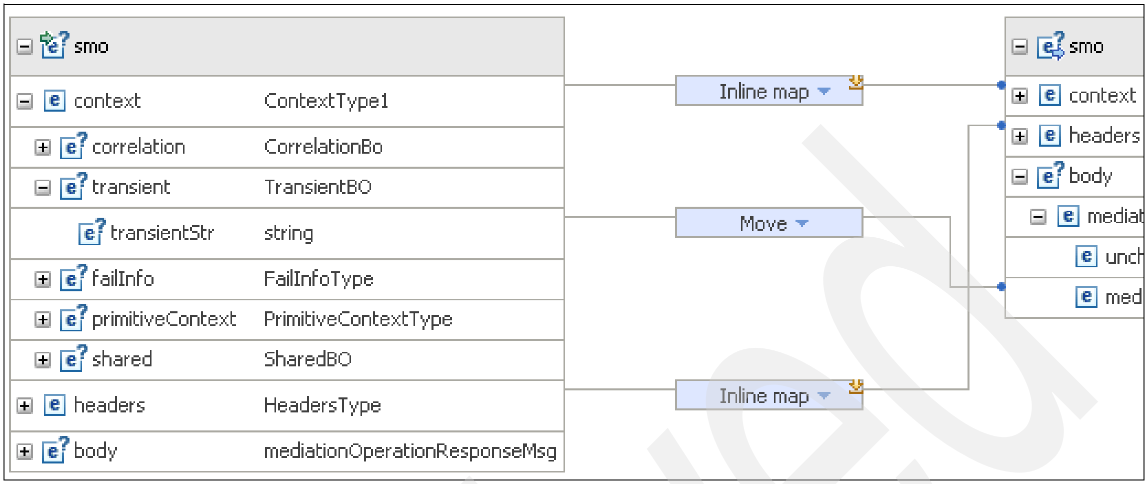


ontext	ContextType1
correlation	CorrelationBo
transient	TransientBO
failInfo	FailInfoType
primitiveContext	PrimitiveContextType
shared	SharedBO

context
- correlation
- transient
- failInfo
- primitiveContex
- shared

Figure 5-20 Removing transient automatic mapping from the context Inline Map

b. Map the transient content to the appropriate body element (Figure 5-21).



smo	
context	ContextType1
correlation	CorrelationBo
transient	TransientBO
transientStr	string
failInfo	FailInfoType
primitiveContext	PrimitiveContextType
shared	SharedBO
headers	HeadersType
body	mediationOperationResponseMsg

Figure 5-21 Moving transient context in the body

8. Retrieve the correlation context and put it in the appropriate output message body element using a last transformation (Figure 5-22).

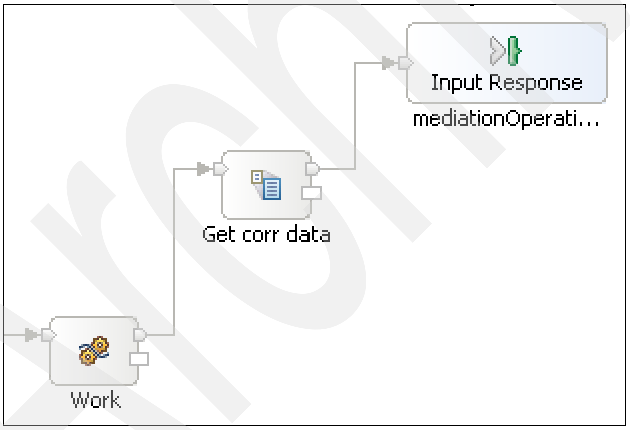
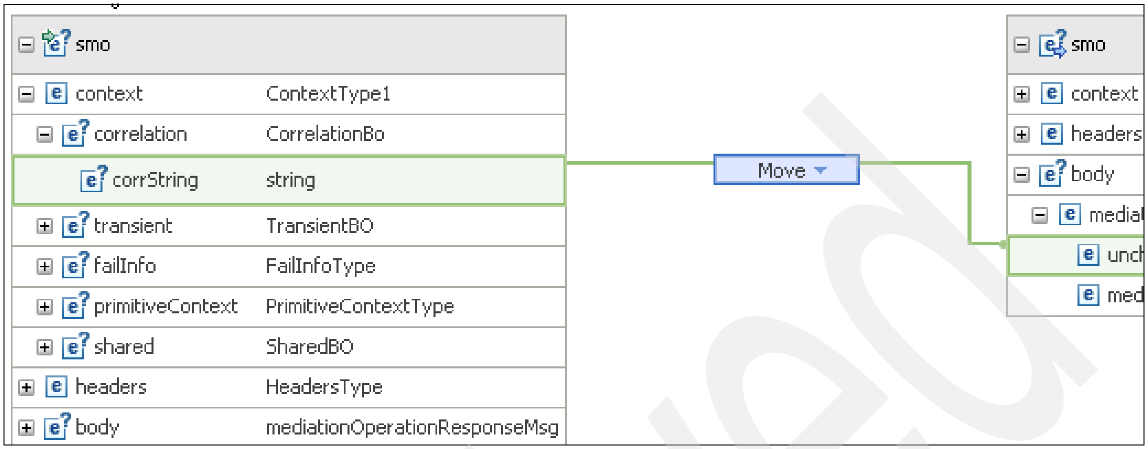


Figure 5-22 Getting back correlation context data in the mediation flow

9. Finally, move the correlation context in the appropriate element of the body (Figure 5-23).



smo	
context	ContextType1
correlation	CorrelationBo
corrString	string
transient	TransientBO
failInfo	FailInfoType
primitiveContext	PrimitiveContextType
shared	SharedBO
headers	HeadersType
body	mediationOperationResponseMsg

Figure 5-23 Getting correlation context back to the body

Header section

The *header section* of an SMO includes the following supplemental information:

- SMOHeader: Information about the message (message identifier, SMO version)
- JMSHeader: Used when there is a JMS import or export binding
- SOAPHeader: Used when there is a Web services import or export binding
- SOAPFaultInfo: Includes information about SOAP faults
- Properties[]: Arbitrary list of name value pairs (for example, JMS user properties)

5.2.2 SMO manipulation

During the execution of mediation flows, the active primitives can access and manipulate the SMO. There are three different ways to access SMOs:

- XPath V1.0 expressions

  The primary mechanism that is used by all primitives.

- XSL style sheets

  Used by the XSL Transformation primitive. The common method to modify the SMO type within a flow. You can also use an XSL style sheet to modify the SMO without changing the type (using XSLT function and logical processing with XSL choose statements).

- Java code

  Using the Custom Mediation primitive, you can access the SMO either using the generic DataObject APIs (commonj.sdo.DataObject, which is loosely typed) or the SMO APIs (com.ibm.websphere.sibx.smobo, strongly typed).

5.3 Typical development flow

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

In Chapter 3, “Basics of development” on page 29, we discussed the typical development flow for modules and mediation modules, which is as follows:

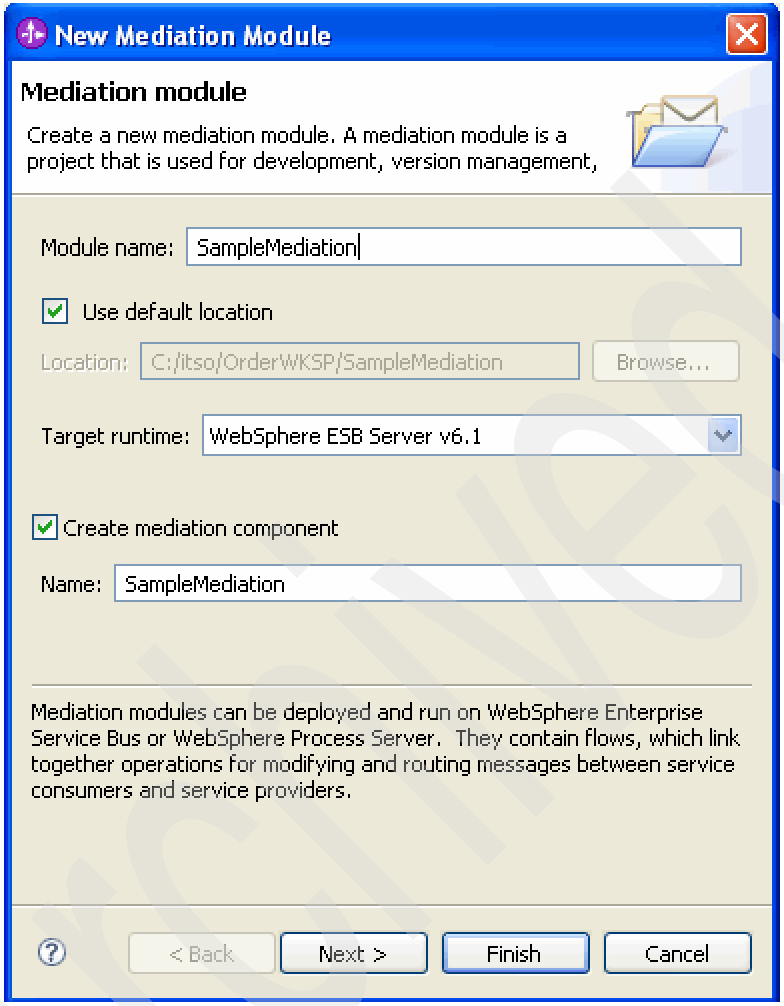
1. Start WebSphere Integration Developer and open a workspace (as described in 3.2.2, “Start WebSphere Integration Developer” on page 42).
2. Switch to the Business Integration perspective for development (as described in 3.2.3, “Using the Business Integration perspective” on page 43).
3. Create a library to store artifacts, such as business objects and interfaces, that are shared among multiple modules (as described in 3.3.1, “Libraries” on page 44).
4. Create a new mediation module (as described in 3.3.2, “Modules and mediation modules” on page 46).
5. Create the business objects to contain the application data, for example customer or order data (as described in 3.4, “Business objects” on page 53).
6. Create the interface and define the interface operations for each component. The interface determines what data can be passed from one component to another (as described in 3.5, “Interfaces” on page 63).
7. **Create and implement the service components**.
8. Build the module assembly by adding the service components, imports, and exports to the assembly diagram. Bind the imports and exports to a protocol (as described in 3.6, “Module assembly” on page 69).
9. Test the module in the integrated test environment (as described in 3.12, “Test tools” on page 105).
10. Deploy the module to WebSphere Process Server. This step is discussed in *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus, Part 3: Run time,* SG24-7643.
11. Share the tested module with others on the team by putting it in a repository (as described in 3.13, “Team development” on page 115).

This chapter focuses on step 7, the implementation of the service components for a mediation module.

5.4 Creating a mediation

The common steps for building a mediation module are as follows.

1. Open WebSphere Integration Developer and open the Business Integration perspective.
2. Select **File** → **New** → **Mediation Module** from the top menu bar.
3. In the first panel (Figure 5-24 on page 307):
   a. Enter a name for the module.
   b. Select the target run time. Mediation modules can run in WebSphere Enterprise Service Bus or WebSphere Process Server application servers.
   c. Click **Next**.



New Mediation Module

Mediation module

Create a new mediation module. A mediation module is a project that is used for development, version management,

Module name: SampleMediation

☑ Use default location

Location: C:/itso/OrderWKSP/SampleMediation

Target runtime: WebSphere ESB Server v6.1

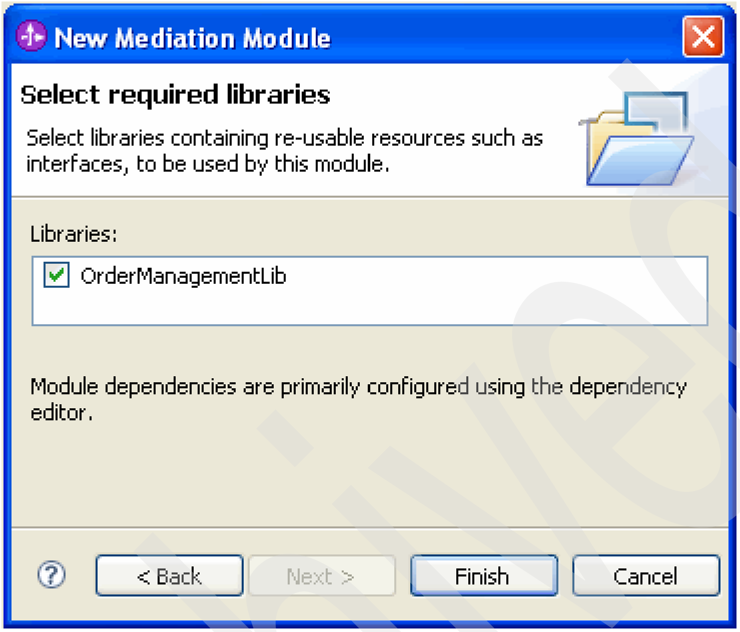
☑ Create mediation component

Name: SampleMediation

Mediation modules can be deployed and run on WebSphere Enterprise Service Bus or WebSphere Process Server. They contain flows, which link together operations for modifying and routing messages between service consumers and service providers.

Figure 5-24 New mediation module settings

4. Add any required libraries. For example, if you created a library to contain the business objects and interfaces that this module uses, select that library here. Click **Finish**. See Figure 5-25.



Select required libraries

Select libraries containing re-usable resources such as interfaces, to be used by this module.

Libraries:

☑ OrderManagementLib

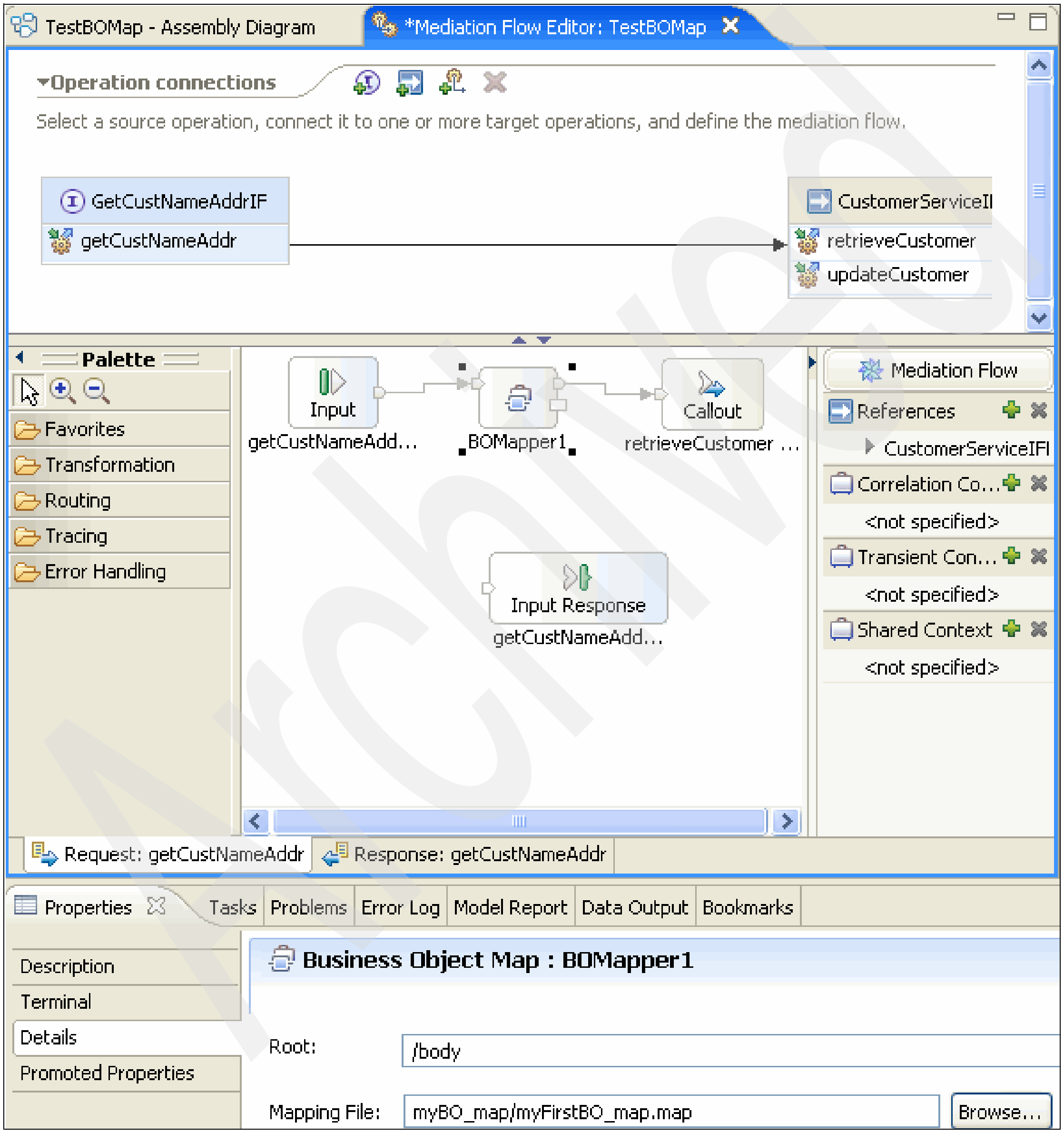
Module dependencies are primarily configured using the dependency editor.

Figure 5-25 Select required libraries

5. The new mediation module is created. You see the structure in the Business Integration view. The assembly diagram opens in the workspace and is populated with one component that represents the mediation flow.
6. Add interfaces for services that you want to import to the assembly diagram. These interfaces can come from any libraries that are specified as required during the module creation wizard, or you can create new interfaces. These interfaces are added to the diagram as import components.
7. Select the mediation flow component, and click **Regenerate Implementation** to open a Mediation Flow Editor.
8. Define the mediation logic using the Mediation Flow Editor.

5.4.1 Mediation flow editor

Generating an implementation of a mediation component opens the Mediation Flow Editor in WebSphere Integration Developer automatically, as shown in Figure 5-26.



Select a source operation, connect it to one or more target operations, and define the mediation flow.

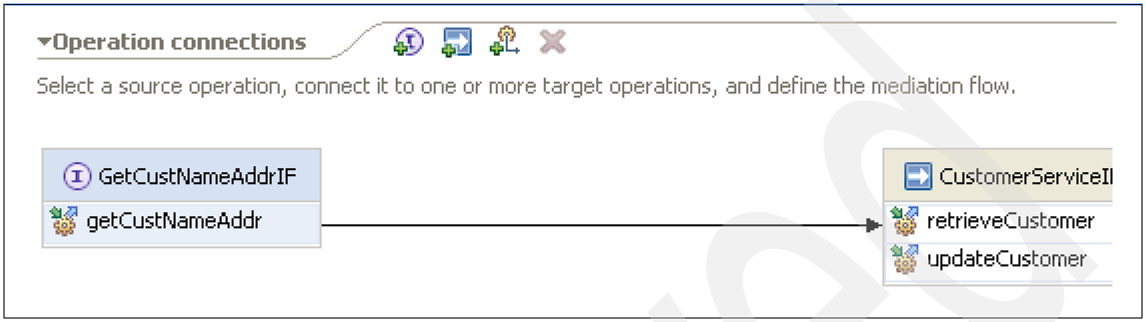
Root: /body

Mapping File: myBO_map/myFirstBO_map.map

Figure 5-26 Mediation flow editor

Operation connections area

The operations connections area (Figure 5-27) is where you wire interface operations (on the left) to reference operations (on the right). Each wire implies a separate mediation flow. You can have many interfaces and interface operations and many references and reference operations.



Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 5-27 Operation connections area

Mediation flow area

Clicking a wire between the interface and a reference in the operations connections area opens the mediation flow for that connection (Figure 5-28). Separate canvases represent the request and response flows, each with a tab at the bottom of the mediation flow area that allows you to switch between them. The Request flow canvas is populated automatically with Input, Callout, and Input Response nodes. The Response flow canvas contains Callout Response and Input Response nodes. You are not obliged to use them.

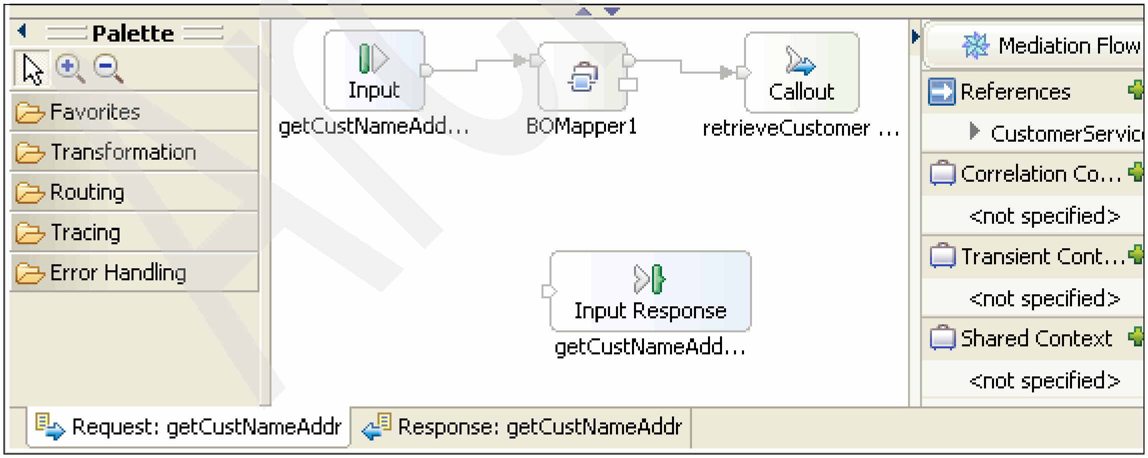


Figure 5-28 Mediation flow editor

To the left of the work area is a *Palette* that contains folders with mediation primitives grouped by their functionality. There are four groups of mediation primitives located in the Palette:

- Transformation
- Routing
- Tracing
- Error Handling

Clicking a folder opens a list of the primitives in that category.

To the right of Palette is a *canvas* that visually represents the mediation flow. The visual elements of the flow are called *mediation nodes* and *wires*. When you drop a primitive onto a canvas, it becomes a new node. You connect a node with another node or nodes by wires.

Mediation flows

Different patterns of mediation logic are encapsulated in mediation primitives. If there is no matching mediation primitive, you can employ a Custom Mediation primitive that allows you to write your own Java code. One more option is to import the third-party mediation primitives.

Figure 5-29 shows six nodes. Each node has at least one terminal that represents the node's ability to connect with other nodes.

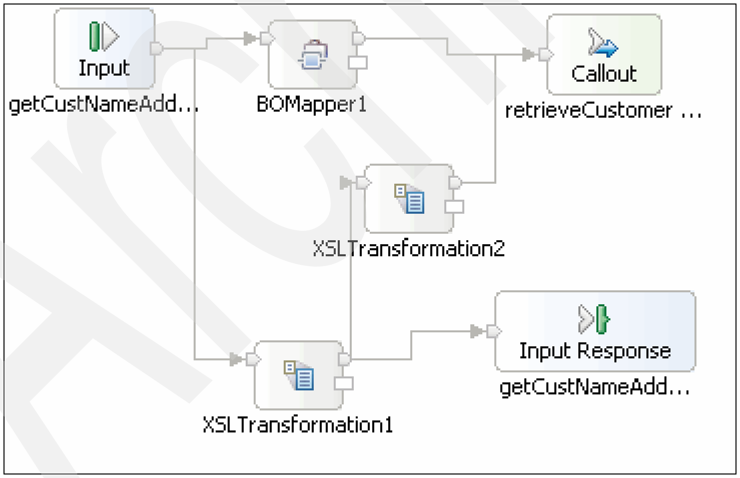


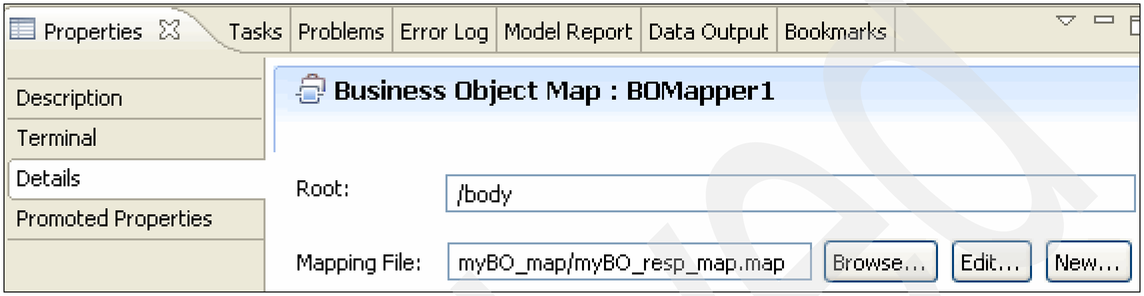
Figure 5-29 Wiring nodes

Wires connect terminal nodes and represent directions of the mediation data flow. Most often a primitive has an input, an output, and a fail terminal. There can be more than one wire coming out or going into a node.

The data type of terminals is NULL until wired. You can also assign fixed data types for terminals. Most Input and Output terminals can be wired only if their data types match.

Mediation nodes

To set or change a node's properties, select the node in the canvas and then go to the Details tab of the primitive's Properties view, as shown in Figure 5-30.



Business Object Map : BOMapper1

Root: /body

Mapping File: myBO_map/myBO_resp_map.map

Figure 5-30 Details tab of the Properties view

Mediation primitives often use XPath language expressions to extract or modify elements of XML structures. We do not discuss XPath here. However, some basic understanding of the language is required because some mediation samples do use simple XPath expressions.

5.5 Service connection and invocation

Service access is a basic function of a mediation:

- Service connection, in terms of protocol binding, import components, and export components is common to both business integration and mediation modules.
- Service invocation in a mediation module can be inline or through a Callout node.
  - When service invocation is inline, it contributes to the overall mediation flow. The inline service invocation is performed using the Service Invoke primitive, as discussed in 5.7.4, "Service Invoke primitive" on page 352.
  - When service invocation is through a Callout, it becomes the final target of the mediation module. This is the classic usage of service invocation, in which the mediation module mediates access to the service to provide message or protocol transformation, routing, and so forth. You can find examples of this invocation throughout the mediation examples.

The basic tasks to invoke a service through a Callout or a Service Invoke primitive include:

1. Ensure that you have the service interface in your module or in a library that is imported by the module.
2. Import the service with the appropriate binding in the assembly diagram. The binding protocol that you use depends on the requirements of the service. Follow these steps:
   a. Create the Import component in the assembly diagram.
   b. Add the service interface to the Import.
   c. Generate the binding for the Import component.
   d. Wire the mediation flow component to the import accepting the automatic reference creation.

The assembly diagram looks similar to that shown in Figure 5-31.

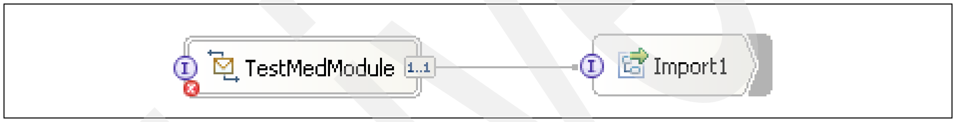


Figure 5-31 Mediation Module Assembly Diagram importing a service

3. Regenerate the implementation for the mediation flow component. Regeneration is the easiest way to have the implementation aligned with the interface and references of the components. The new interfaces and reference partners display in the Mediation Flow Editor (Figure 5-32).

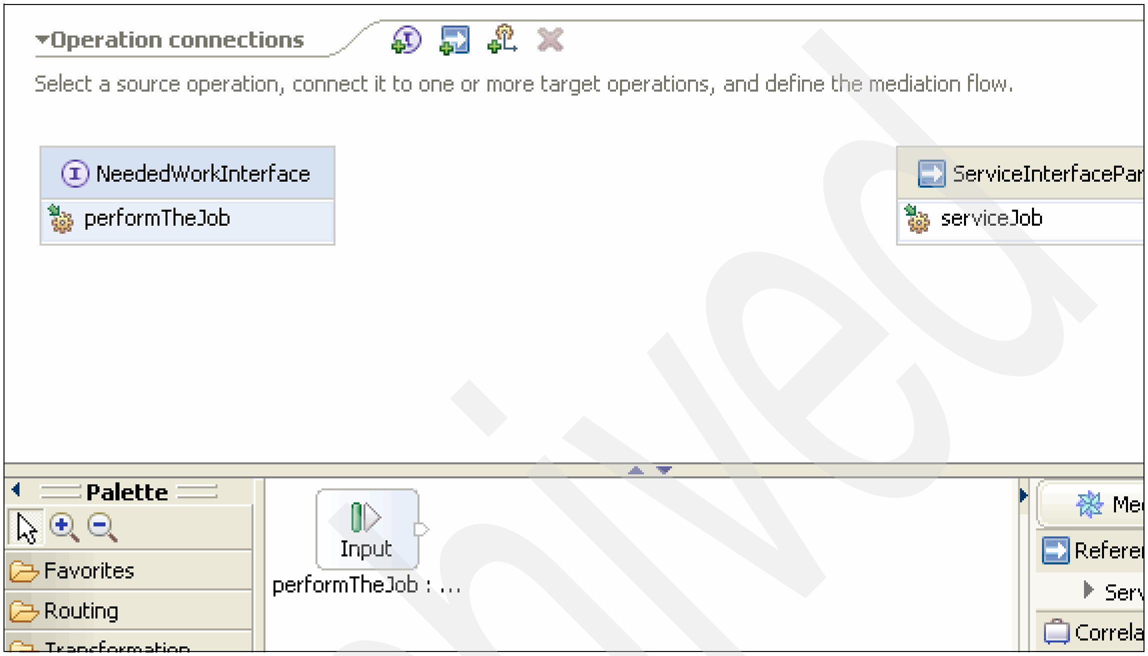
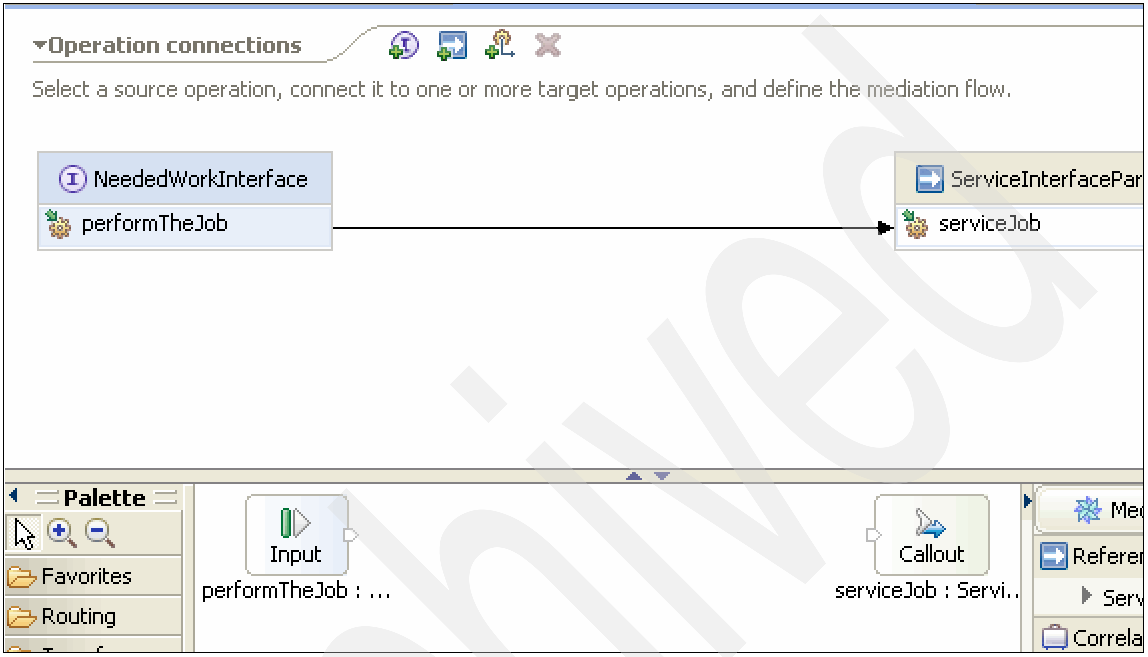


Figure 5-32 Mediation editor after implementation regeneration

At this stage you can wire the NeededWorkInterface performTheJob operation to ServiceInterfacePartner serviceJob to create a service callout, or you can leave these unwired and have the mediation flow invoke the ServiceInterfacePartner inline as part of the logic.

If you are mediating the message flow to the service (using a callout):

1. Wire the interface operation (performTheJob) to the Partner operation (serviceJob in the example). This creates a Request flow with a Callout and, if the operation is two-way, a Response flow. See Figure 5-33.

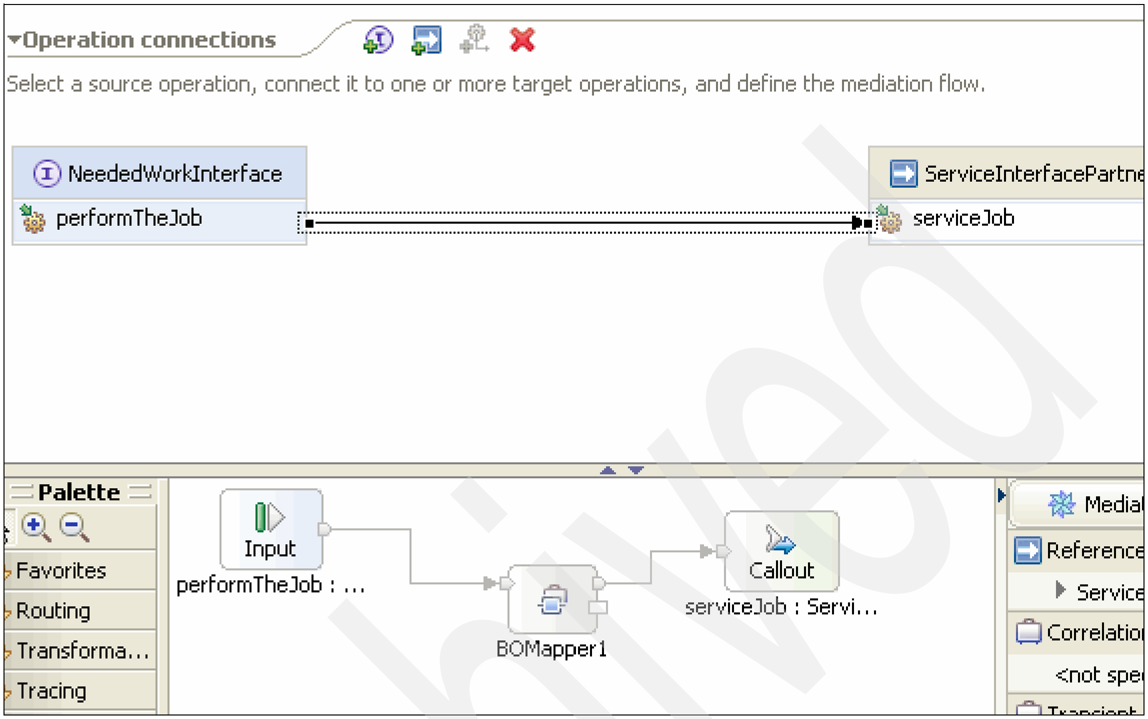


Operation connections

Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 5-33 Wiring operations for mediating access

2. Develop the mediation flow and then you are finished.



Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 5-34 Example request flow completed

To access the service as part of the mediation logic, leave the operations unwired. Use the Service Invoke primitive (selecting the partner that you want too invoke, such as ServiceInterfacePartner in the example) at the point that you need in the flow, as shown in Figure 5-35.

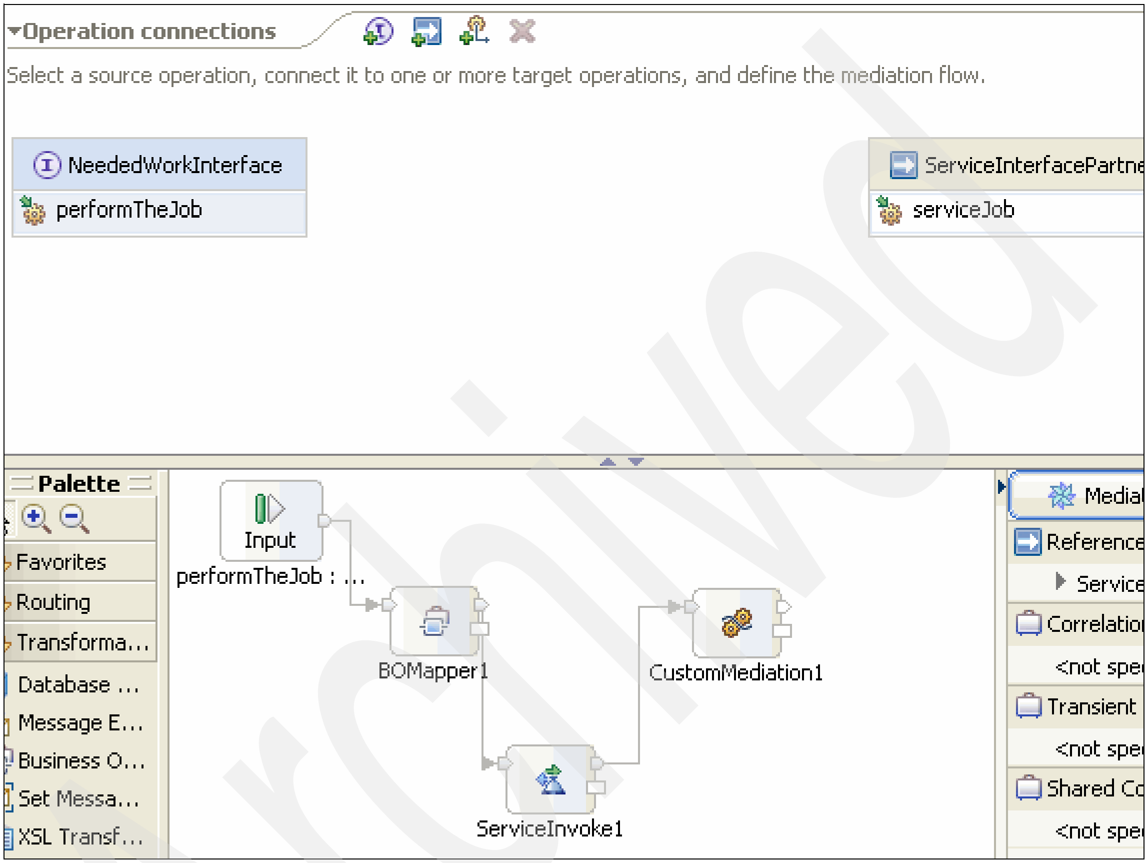

Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 5-35   Invoking a service as part of the mediation flow

5.6 Transformation primitives

The primitives that you find in the Mediation Flow Editor in the Transformation category are:

- XSL Transformation primitive
- Business Object Map primitive
- Database Lookup primitive
- Message Element Setter primitive
- Set Message Type primitive
- Custom Mediation primitive

5.6.1 XSL Transformation primitive

The XSL Transformation primitive performs Extensible Stylesheet Language (XSL) transformations on an XML serialization of the message. XSL transformation is very common in message flows. You can find many examples of using this primitive in the scenario that we describe in this book.

Properties

The properties to note in the Details tab of the Properties view are:

- Mapping file: Specifies the name of the XSL style sheet that the primitive uses.
- Root: An XPath 1.0 expression that specifies the root of the transformation. You can specify:
  - / refers to the complete SMO
  - /headers refers to the headers of the SMO
  - /context refers to the context of the SMO
  - /body refers to the body section of the SMO

  This property is used for both the input message and the transformed message.
- Validate input: If true, causes the input message to be validated before the mediation is performed.

Using the XSL Transformation primitive

The XSL Transformation primitive has one input terminal (In), one output terminal (Out), and one fail terminal (Fail). The service message object arrives at the In terminal and exits through the Out terminal.

As an example, consider the mediation flow shown in Figure 5-36. The message from the Input is of type updateCustomerRequestMsg.

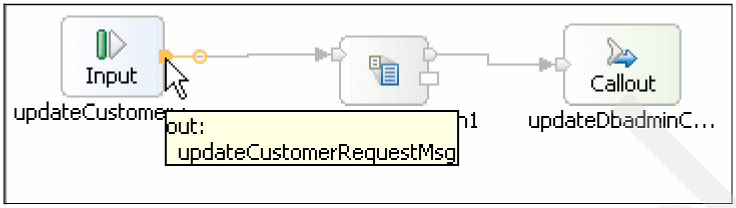


Figure 5-36 Wiring primitive terminals

The message that goes to Callout needs to be of type updateDbadminCustomerBGRequestMsg, as shown in Figure 5-37.

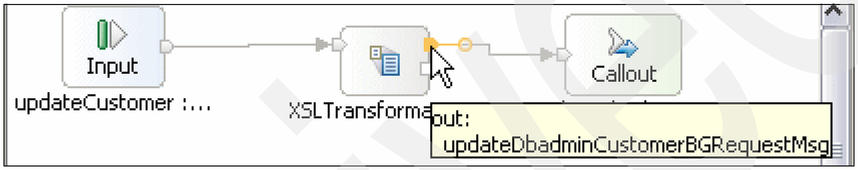
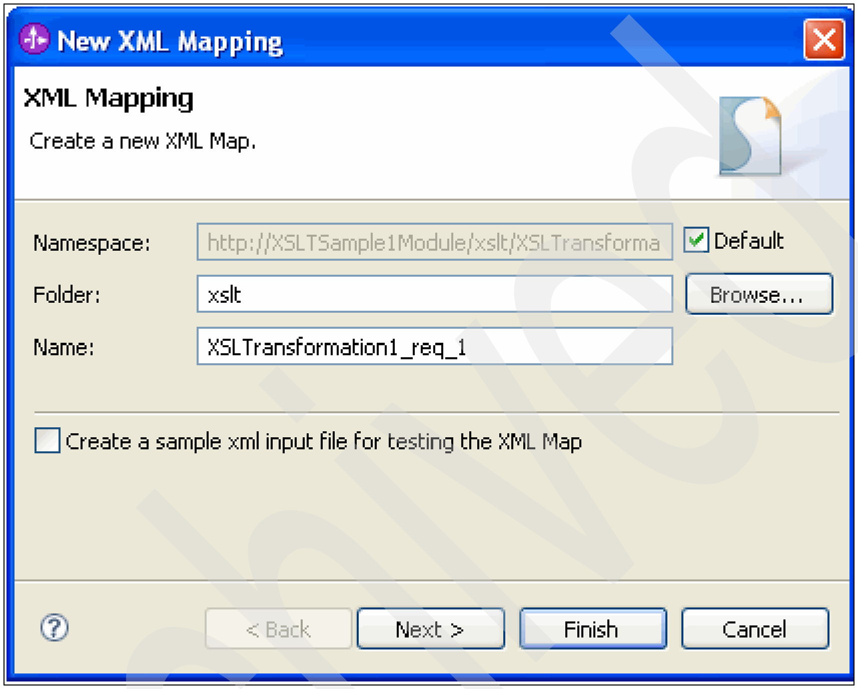


Figure 5-37 Wiring primitive terminals

The XSLT Transformation primitive is used to map the input message fields to the appropriate output message fields as follows:

1. Open the mapping for this node by double-clicking it in the canvas. The XML mapping wizard starts (Figure 5-38). Accept the defaults and click **Next**.



XML Mapping

Create a new XML Map.

Namespace: http://XSLTSample1Module/xslt/XSLTransforma ☑ Default

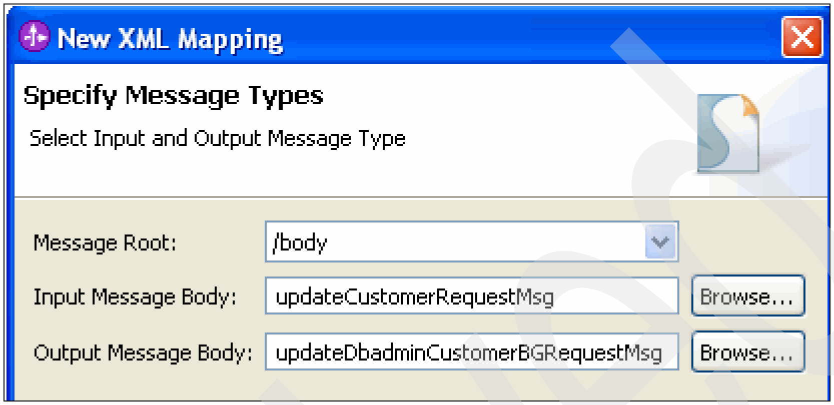
Folder: xslt

Name: XSLTransformation1_req_1

□ Create a sample xml input file for testing the XML Map

Figure 5-38 Create a new XML mapping

2. Specify the message root, the input body, and the output body (Figure 5-39). The input and output body fields default to the input and output message types if the XSLT transformation is wired. If it has not been wired, you need to select the messages that you will map. Click **Finish**.



Message Root: /body

Input Message Body: updateCustomerRequestMsg

Output Message Body: updateDbadminCustomerBGRequestMsg

Figure 5-39 Specify message types

3. The map opens with the input message fields on the left and the output on the right. Develop the XSL transformation graphically by mapping elements from the source SMO to the target SMO. Expand the source and target messages in order to see all individual components (Figure 5-40).

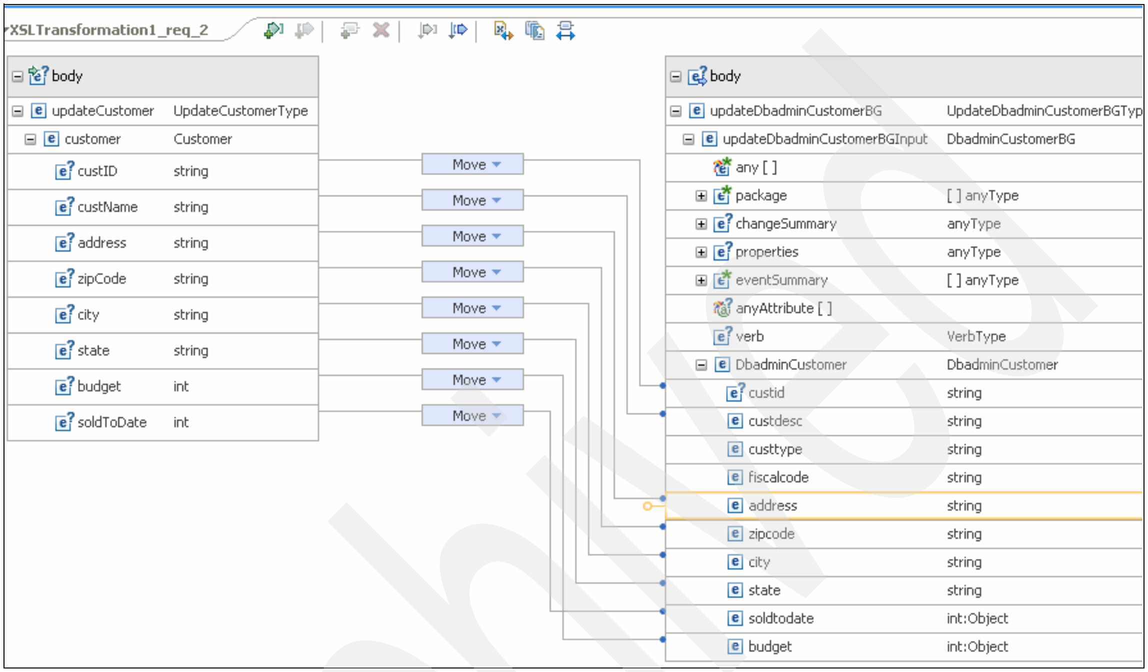


Figure 5-40 XML Mapping

4. Perform the mapping functions.
5. Save and close the mapping file.
6. Save and close the mediation flow editor.
7. Save the assembly diagram.

When a mapping is required in the request flow, a corresponding reverse mapping is often required in the response flow to map the response messages.

Mapping functions

To map an element, drag a line between the source and target elements. The mapping functions that are available for mapping simple message elements include:

- *Move*: Moves the value from the source to the target (the default).
- *Custom*: Uses an XPath expression or an XSL style sheet to define the mapping logic.
- *Normalize*: Normalizes the input string. For example, you can use this function to remove multiple occurrences of white space (such as space, tab, or return).
- *Substring*: Extracts elements of a source value based on an index and a delimiter.

If the mapping takes place between two complex types, the default operation is Move. Other mapping functions that are available for complex types are:

- *Custom*
- *Inline map*: Executes an embedded map within the same XSL style sheet.
- *Submap*: Calls an external XSL style sheet as part of the current transformation.

While the mapping operations require both the input message elements and the output message elements, you can also apply a transformation to the output message elements only.

Transforms

To transform an output message element, right-click it and select **Create Transform** as shown in Figure 5-41.

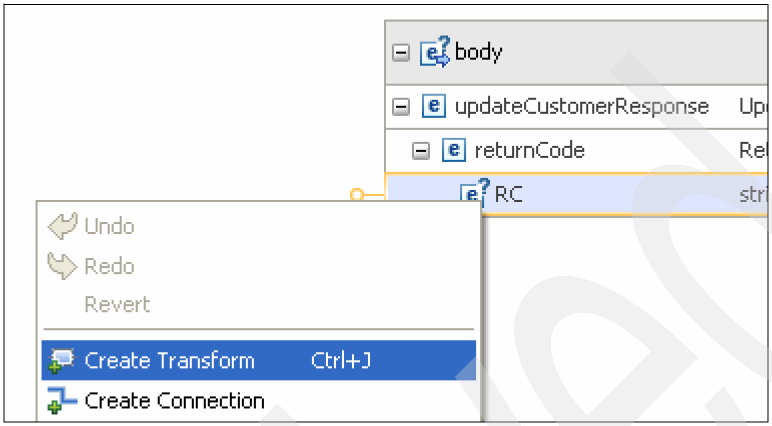


Figure 5-41 Create a transform action on an output message element

The default value is `Assign` for a base type (Figure 5-42) and `Input map` for a complex type. Custom operation is available for all cases.

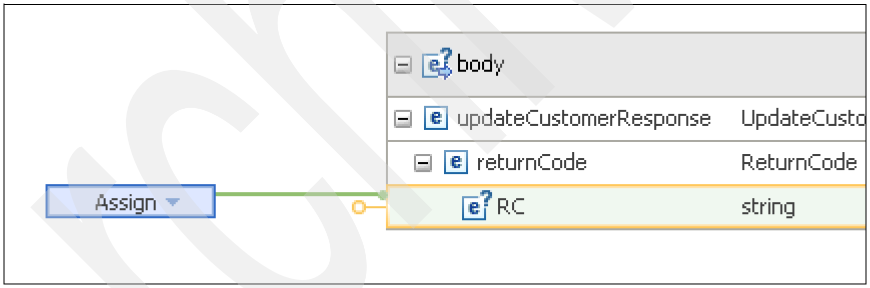
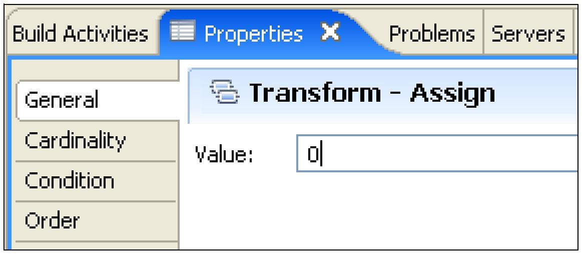


Figure 5-42 Assign transform

The Assign transform action allows you to designate a value for the message element. To supply the value, select **Assign**, go to the Properties view, and set the value as shown in Figure 5-43.



Value: 0

Figure 5-43 Assigning a value

Mapping utilities

Mapping utilities provide a shortcut for mapping similar input message elements to output message elements.

Select **Match Mapping** from the pop-up menu for the SMO or individual message elements, or select **Map source to target based on names and types** from the horizontal icon menu.

If matching names and types in the input and output messages are found, a default Inline map action is created (Figure 5-44). To see the content of the inline map, click the yellow arrow in the upper-right corner of the inline map rectangle.

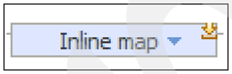


Figure 5-44 Inline map

5.6.2 Business Object Map primitive

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice. You can use *business object maps* to assign values to the target business objects based on the values in the source business objects.

The Business Object Map primitive is used to change the context, headers, or body of incoming or outgoing messages. Transformations are defined by a business object map. The Business Object Map primitive has one input terminal, one output terminal, and one fail terminal.

The Business Object Map primitive is located under Transformation folder of the Mediation Flow Editor.

The Business Object Mapping Editor provides following mapping actions:

- *Assign*: Assigns a constant value to the target element.
- *Custom*: Provides Java code to define mapping logic.
- *Custom Assign*: Provides Java code to assign a value.
- *Custom Callout*: Provides Java code to define mapping logic.
- *Extract*: Provides a convenient way to extract a substring from a string defining a delimiter and a substring index.
- *Join*: Provides a method to combine two or more elements.
- *Move*: Copies the source element value to the target element value.
- *Relationship*: Creates an association between data from two or more business objects. The source and target of a relationship transform must be complex types (business objects).
- *Relationship Lookup*: Maintains a mapping between the same data that is represented with different values in different business objects (for example, North Carolina <=> NC or euro <=> EUR).
- *Submap*: Maps complex types.

The Business Object Map primitive provides a similar, but enhanced, functionality to that of an XSL Transformation primitive.

The key difference is that the XSL Transformation primitive performs transformations in XML, using a style sheet, where the Business Object Map primitive performs transformations on business objects using Service Data Objects (SDO). If you have existing XSL style sheets, you might be able to reuse them with the XSL Transformation primitive. In addition, if you have existing business object maps, you might be able to reuse them with the Business Object Map primitive. Some types of transformation are easier to perform in XSL, while others are easier using a business object map.

5.6.3 Database Lookup primitive

The Database Lookup primitive modifies messages using information from a user-supplied database.

Additional materials: We include a sample mediation called *TestDBLookup* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

Properties

The properties to note in the Details tab of the Properties view are:

- *Data source name*: JNDI name of the data source defined on the WebSphere Enterprise Service Bus server.
- *Table name*: Name of the database table, including the schema name.
- *Key column name*: Name of the column in the table to be used for the look up.
- *Key path*: An XPath expression that returns a value from the message.
- *Validate input option*: If selected, enables input message validation to ensure it matches its type definition at run time.
- *Data elements table*: Each line defines the column from which the value is retrieved, the type of information it is, and the place in the message where the retrieved value is to be stored.

Using the Database Lookup primitive

The Database Lookup primitive has one input terminal (In), two output terminals (out and keyNotFound), and one fail terminal.

When a service message object arrives at the In terminal, one of the following actions is taken:

- If the key is found, the message, updated with database values that are associated with the key, is returned using the Out terminal.
- If the key is not found in the database, the message is unchanged and returned using the keyNotFound terminal.
- If an exception occurs both the unchanged message and exception are returned using the Fail terminal.

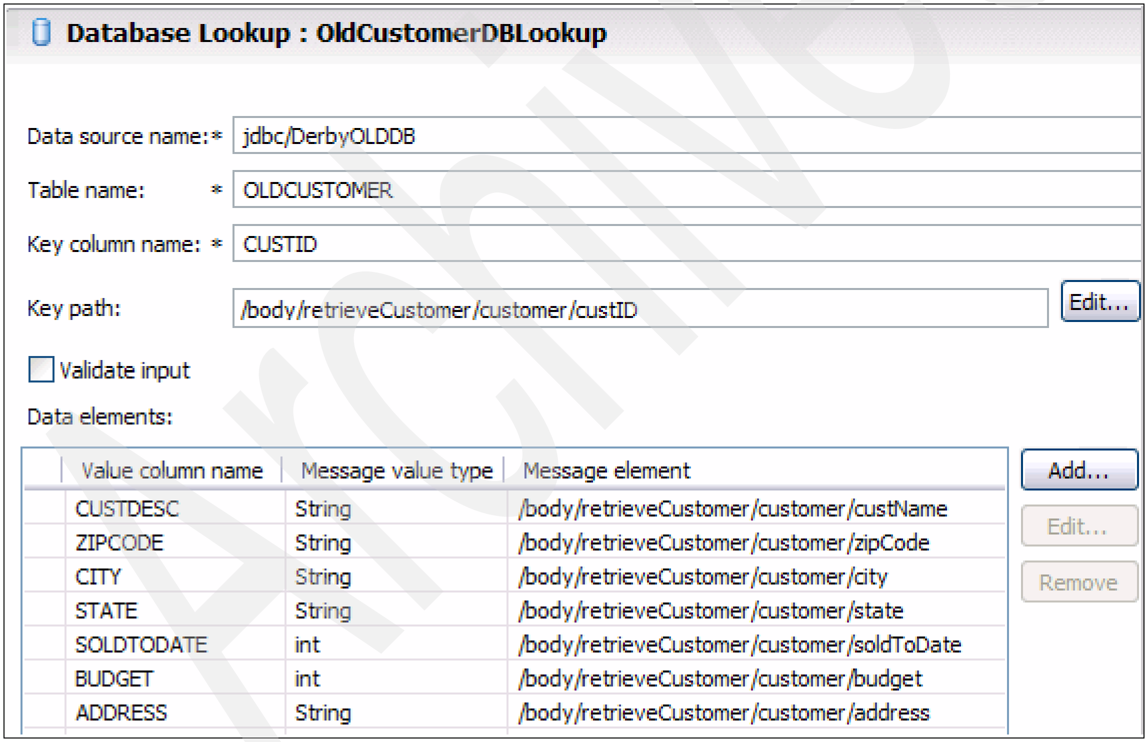
Note: The Database Lookup primitive is used commonly in conjunction with the Message Filter primitive. Using Database Lookup, a value can be obtained from a database and stored in the transient or correlation context. The Message Filter can then filter messages using this value by defining an XPath expression to the context.

The following example searches the OLDCUSTOMER table of the OLDDB database for customer information. If a database record is found that has a CUSTID value equal to the value of /body/retrieveCustomer/customer/custID in the input message, the message is updated.

The message fields to be updated are listed under the Message elements column of Data elements. *Value column name* is the name of the column in the database record. In other words, a database value from *Value column name* is copied into *Message element* of the message.

If a database record with a matching value is not found, the message exits the primitive unchanged.

Figure 5-45 shows the property values of the primitive. Data elements list column names of OLDCUSTOMER table, value type, and message elements to which the values need to be copied.



Database Lookup : OldCustomerDBLookup

Data source name:* jdbc/DerbyOLDDB

Table name: * OLDCUSTOMER

Key column name: * CUSTID

Key path: /body/retrieveCustomer/customer/custID

□ Validate input

Data elements:

Value column name	Message value type	Message element
CUSTDESC	String	/body/retrieveCustomer/customer/custName
ZIPCODE	String	/body/retrieveCustomer/customer/zipCode
CITY	String	/body/retrieveCustomer/customer/city
STATE	String	/body/retrieveCustomer/customer/state
SOLDTODATE	int	/body/retrieveCustomer/customer/soldToDate
BUDGET	int	/body/retrieveCustomer/customer/budget
ADDRESS	String	/body/retrieveCustomer/customer/address

Figure 5-45 Database Lookup properties

Figure 5-46 shows the Database Lookup primitive in the message flow. One of three actions is taken:

- If the matching record is found, the updated message exits the primitive through the Out terminal, then enters OldCustomerXSLT for further processing.
- If no matching record is found, the unchanged messages exits the primitive through the keyNotFound terminal, then enters WebServiceXSLT for further processing.
- If the lookup ended with failure, the unchanged message exits the primitive through the Fail terminal, then enters DBLookupFailure for further processing.

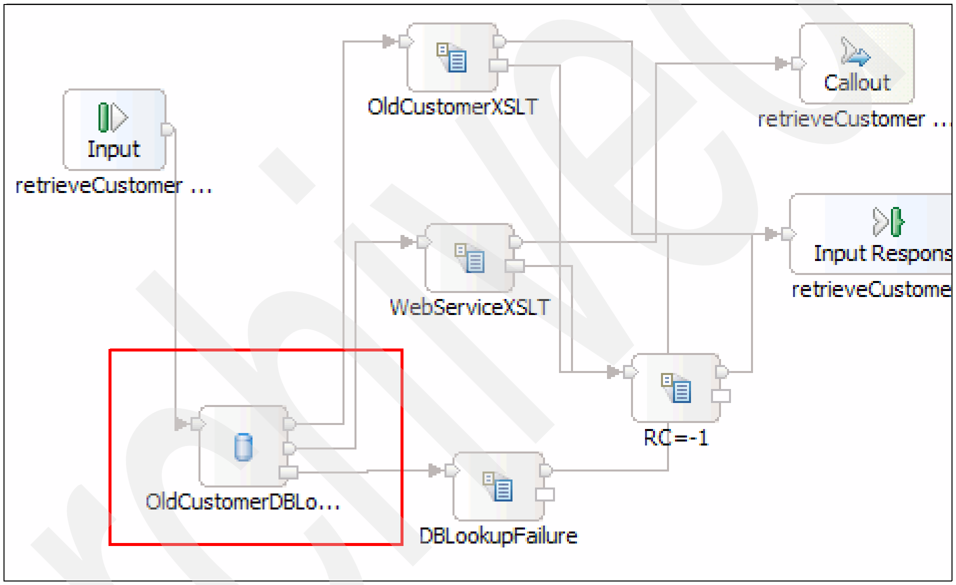


Figure 5-46 Database Lookup primitive in the message flow

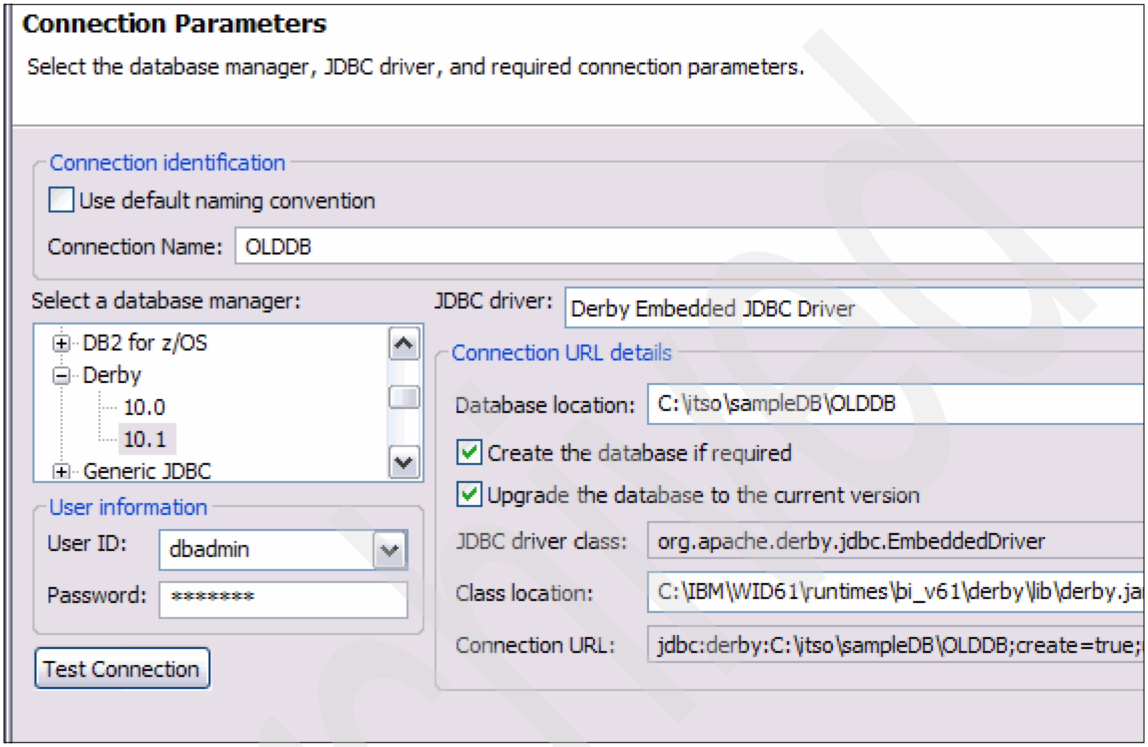
Accessing the database in the workspace

To access a database while you are building a message flow, you must define to the workspace.

To create a connection to the database:

1. Switch to Data perspective and create a connection to the database.
2. In the Database Explorer view, right-click **Connections** and select **New Connection**. If the database does not exist, you can create it using this wizard.

3. Select the database manager and version. The options on the panel change to those appropriate for the database type. Complete the required fields (Figure 5-47), including the location of the database, driver class location, and the user ID and password to access the database.



Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☐ Use default naming convention

Connection Name: OLDDB

Select a database manager:

- DB2 for z/OS
- Derby
  - 10.0
  - 10.1
- Generic JDBC

User information

User ID: dbadmin

Password: ********

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\itso\sampleDB\OLDDB

☑ Create the database if required

☑ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\IBM\WID61\runtimes\bi_v61\derby\lib\derby.ja

Connection URL: jdbc:derby:C:\itso\sampleDB\OLDDB;create=true;

Figure 5-47 New database connection properties

4. Click **Finish**.

You can execute SQL commands using the SQL editor to create tables, to populate the tables with data, and to perform other actions that you might need to take to prepare the database. To use the editor:

1. Click the Open SQL Editor icon.

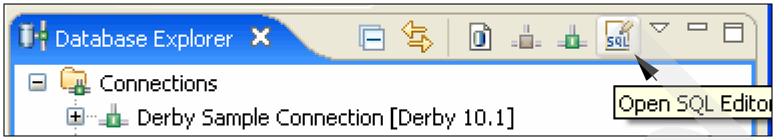
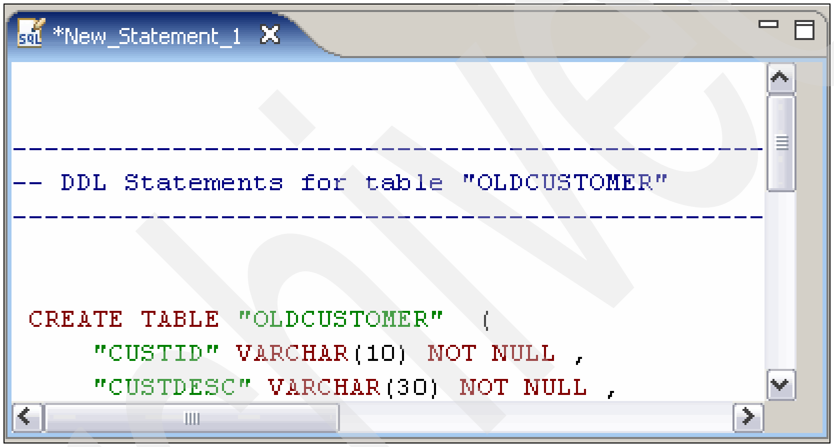


Figure 5-48 Open SQL Editor icon

A window opens where you can type or copy SQL statements (Figure 5-49).

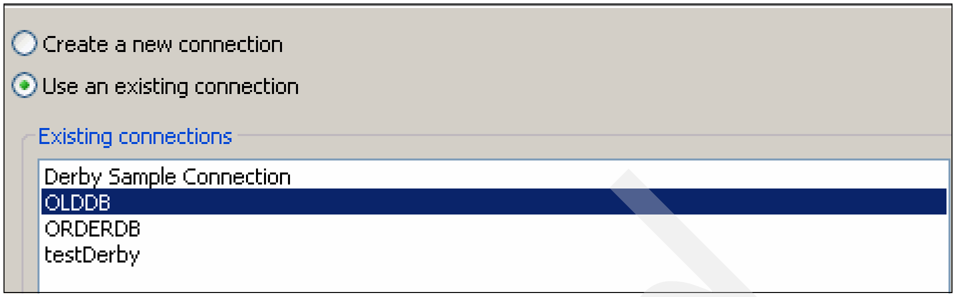


```
--------------------------------------------------------
-- DDL Statements for table "OLDCUSTOMER"
--------------------------------------------------------

 CREATE TABLE "OLDCUSTOMER"  (
     "CUSTID" VARCHAR(10) NOT NULL ,
     "CUSTDESC" VARCHAR(30) NOT NULL ,
```

Figure 5-49 Entering SQL statements

2. Save the statements by right-clicking in the window and selecting **Save**.
3. Run the SQL by right-clicking in the window and selecting **Run SQL**. Select the “Use an existing connection” option, and then select the connection. Click **Finish**.



○ Create a new connection

● Use an existing connection

Existing connections

Derby Sample Connection
OLDDB
ORDERDB
testDerby

Figure 5-50 Select the connection

4. You can view the data in a table by navigating to the table in the connection, as shown in Figure 5-51.

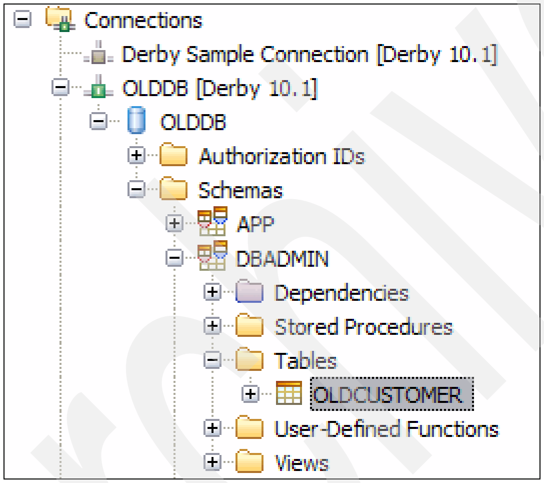
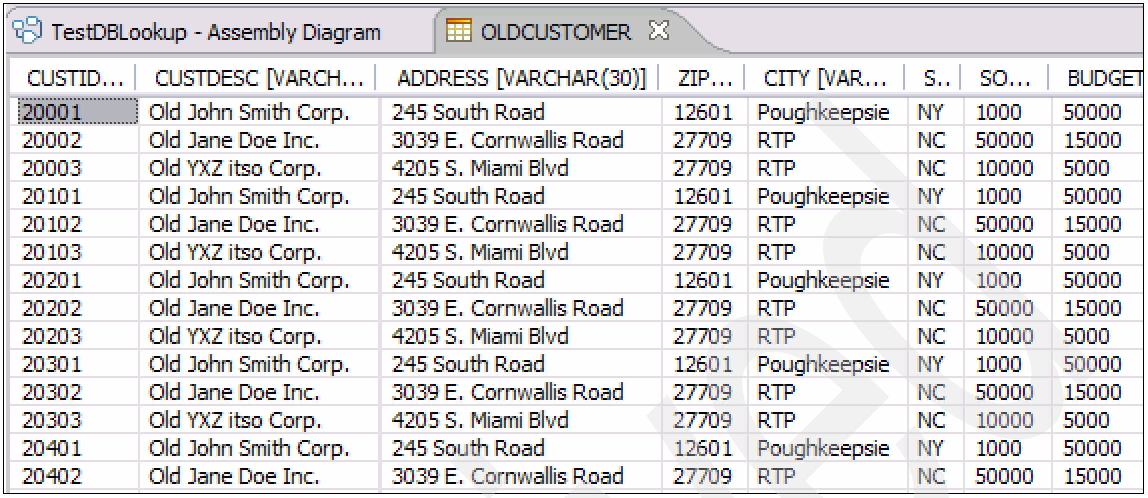


Figure 5-51 Database structure in the Connections directory

5. Right-click the table name and select **Data** → **Edit**. The table opens in a new view as shown in Figure 5-52.



CUSTID...	CUSTDESC [VARCH...	ADDRESS [VARCHAR(30)]	ZIP...	CITY [VAR...	S..	SO...	BUDGET
20001	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20002	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20003	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20101	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20102	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20103	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20201	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20202	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20203	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20301	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20302	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20303	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20401	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20402	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000

Figure 5-52 Viewing and editing database data

6. If you are working with Derby, remember to disconnect from the connection by selecting the connection, right-clicking, and then selecting **Disconnect**.

   The embedded version of Derby allows only one connection at a time to a database. When you have a connection open from the workspace, no other application can access the database.

5.6.4 Message Element Setter primitive

The Message Element Setter primitive provides a simple mechanism for setting the content of messages. It allows you to change, add, or delete message elements, but it does not allow you to change the type of the message.

> **Additional materials:** We include a sample mediation called *MessageElementSetterMediation* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

Properties

The message elements to be set are defined in the Details tab of the Properties view. Each message element in the table has the following parts defined:

- *Target*: An XPath 1.0 expression that describes the location of the element to set, create or delete. You can specify:

  – / to use the complete SMO
  – /headers to use the headers of the SMO
  – /context to use the context of the SMO
  – /body to use the body of the SMO

  If you want to set a target message element to a constant value, the target XPath expression must resolve to a single leaf element. If you want to copy from a source message element to a target message element, both the source and target must specify an XPath expression that resolves to a single message element (a single leaf or subtree).

  If you set multiple targets, all elements are effectively set simultaneously. Therefore, if you set element *X* from value 13 to value 14, and element *Y* to the value of element *X*, the mediation sets element *X* to value 14 and element *Y* to value 13.

  If you specify the same target element more than once, the last operation performed on the target element takes precedence.

- *Type*: The type of the element value.

  If you want to set the target to a constant value, the type must be a Java primitive or Java string.

  If you want to set the target to a value that is copied from somewhere in the input SMO, the Type property must be the keyword *copy*. When you copy a value from somewhere in the input SMO, the target type is assumed to be the same as the source type.

  If you want to copy a value from the input SMO to a new element instance, appended to a repeating element in the output, the Type property must be the keyword *append*. You can append only to a repeating element in the output, and the target type is assumed to be the same as the source type.

  If you want to delete an element instance, the Type property must be the keyword *delete*. You can delete only optional or repeating elements.

- *Value*: If the Type property is set to *copy* or *append*, the Value property should be an XPath 1.0 expression that identifies the source element. The copy and append operations always take their source value from the unmodified input SMO.

  If the Type property is set to *delete*, the Value property should not be set.

If the Type property is not set to *copy*, *append*, or *delete*, the Value and Type properties must be compatible. For example, if the Type is int, the Value could validly be 14, but not GoldAccount.

If the “Validate input” option is selected, the input message is validated before the mediation is performed.

Using the Message Element Setter primitive

The Message Element Setter primitive has one input terminal (In), one output terminal (Out), and one fail terminal.

The example shown in Figure 5-53 uses the Message Element Setter primitive to set a return value to be equal to the customer ID. The primitive is placed between the Callout Response and Input Response nodes of the response flow.

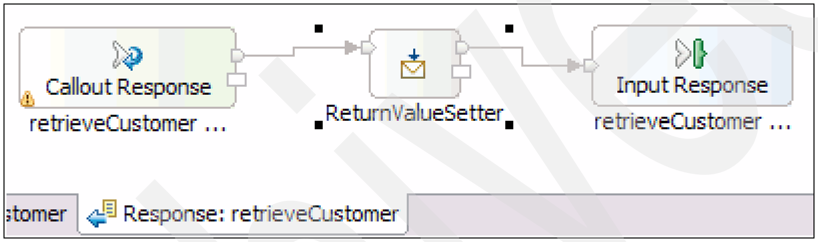
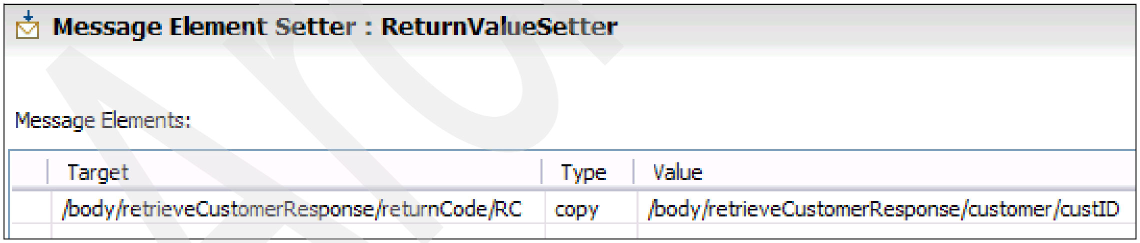


Figure 5-53 The Message Element Setter primitive in a mediation flow

Figure 5-54 shows the values that are used to define the message element to be changed.



Message Element Setter : ReturnValueSetter

Message Elements:

Target	Type	Value
/body/retrieveCustomerResponse/returnCode/RC	copy	/body/retrieveCustomerResponse/customer/custID

Figure 5-54 Message element definition

5.6.5 Set Message Type primitive

Additional materials: We include a sample mediation called *SetMessageTypeMediation* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

The Set Message Type primitive lets you do the equivalent of casting a generic data type to a more specific data type. 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.

Typically, a weakly-typed field is declared as having an XML schema type of anyType or anySimpleType or is an XML wildcard element (any). Less commonly used weak-typing constructs are abstract types and substitution groups.

The Set Message Type primitive is used to overlay message fields with more detailed structures. It can overlay both weakly-typed and strongly-typed fields, allowing you to have more control in manipulating message content.

There are three weakly-typed fields as of WebSphere Enterprise Service Bus V6.1:

- anyURI
- anySimpleType
- anyType

Notes:

- If no Set Message Type primitive is used in the mediation flow, the XSLT editor and other editors display the weakly-typed attribute. Custom nodes (that is, Custom Mediation primitive, custom XSL style sheet) are required to access the weakly-typed field.
- The Set Message Type primitive can also overlay a strongly-typed field with another strongly-typed attribute (if the new strongly-typed attribute is derived from the original one).

Properties

The properties to note are:

- *Message field refinements*: Specifies how a weakly-typed field needs to be cast to a more specific data type. It has two parameters.
  - Weakly typed field: Specify the XPath of the weakly-typed field that you want to refine.
  - Actual field type: Specify the data type that you want to use for the refinement.
- *Reset message type*: If true, causes the current primitive to forget Message field refinements information from previous Set Message Type primitives.
- *Validate*: If true, causes the Set Message Type primitive to perform runtime validation.

Using the Set Message Type primitive

The Set Message Type primitive has one input terminal (In), one output terminal (Out), and one fail terminal.

To use the Set Message Type primitive:

1. Create a business object of the weakly typed field (that is anyType). Add one field and change the type to a weak type.

   To change the type to anyType:

   a. Click the field’s type.
   b. In the window that opens, click **Browse**, and select **anyType**.
   c. Then, click **OK** to set the new data type.
2. Create an interface with this object and add the interface to the mediation flow component as an export.
3. Add a Set Message Type primitive to the canvas and wire the In and Out terminals. The primitive is located in the Transformation folder of the Palette.
4. Cast the message type to the actual business object. In the Properties view, select the weakly-typed field (the new business object that was created in step 1) and set the business object to which it needs to be cast.

For example, assume that we have a weakly-typed customerData field as shown in Figure 5-55.

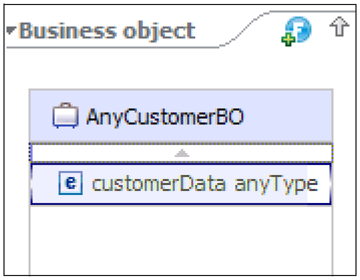
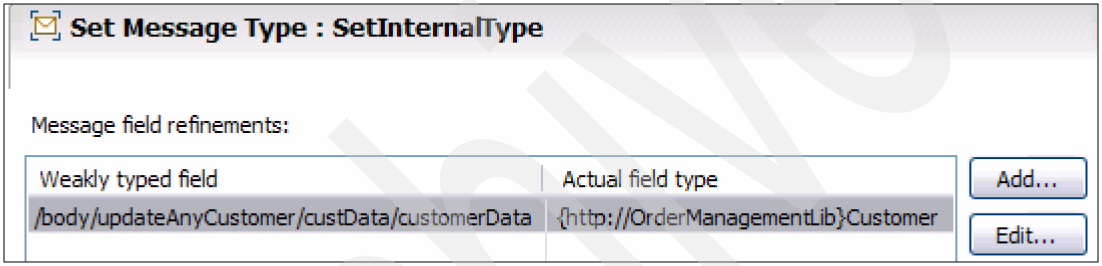


Figure 5-55 Weakly typed object

To cast this object to a complex Customer data type, set the primitive properties as shown in Figure 5-56.


Set Message Type : SetInternalType

Message field refinements:

Weakly typed field	Actual field type
/body/updateAnyCustomer/custData/customerData	{http://OrderManagementLib}Customer

Figure 5-56 Set Message Type properties

As an example of when you might use this primitive, assume that there are two different Web services to retrieve and update customer information. The Web services have dissimilar interfaces—different operation names (updateCustomer and updateExtCust) and different data types (Customer and ExternalCustomerInfoBO). A message is routed to a specific Web service depending on a value in the message. For example, if the customer ID starts with *1*, it is sent to CustomerService service. Otherwise, it is sent to ExternalCustomerService.

The input message is defined using a unifying interface AnyCustomerIF with updateAnyCustomer operation and AnyCustomerBO business object that has one weakly-typed field called customerData. This field is of anyType type.

This allows a caller to pass in any data structure. In the mediation module, the message is first cast to the Customer data type, and the custID value is checked. If the value starts with *1*, the message is sent to the CustomerService service. If not, the message is cast to ExternalCustomerInfoBO and sent to the ExternalCustomerService service.

Figure 5-57 illustrates the mediation flow.

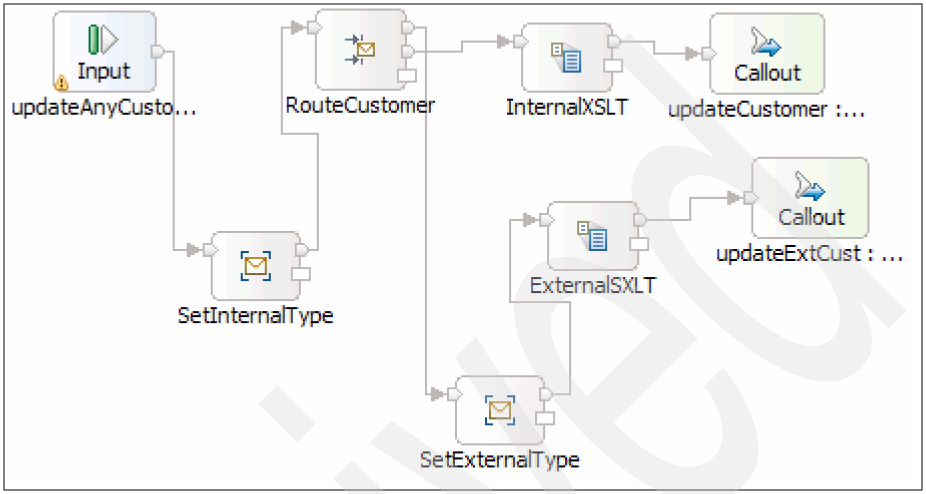


Figure 5-57 Set Message Type primitive in a mediation flow

5.7 Routing primitives

The primitives that you find in the Mediation Flow Editor in the Routing category include:

- Message Filter primitive
- Endpoint Lookup primitive
- Fan In primitive
- Fan Out primitive
- Service Invoke primitive

5.7.1 Message Filter primitive

The Message Filter primitive routes messages down different paths, based on the message content.

Additional materials: We include a sample mediation called *MessageFilterMediation* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

Properties

The properties to note are:

- *Filters*: A list of expressions, and associated terminal names that define the filtering performed by the primitive. The order is significant in the list of expressions. Expressions are evaluated in the order they appear in the table.
- *Pattern*: An XPath 1.0 expression against which the message is tested. The expression is evaluated starting from the XPath expression /, which refers to the complete SMO.
- *Terminal name*: The name of an output terminal. There is one terminal name for each pattern XPath expression. The terminal name must be a valid connection endpoint, and it must not be fail or default. The default value is empty, which is invalid.
- *Distribution mode*: Determines the behavior of the primitive when an inbound message matches multiple expressions. If the Distribution mode is set to *First*, the message is propagated to the first matching output terminal. If the Distribution mode is set to *All*, the message is propagated to all matching output terminals. If there is no matching output terminal, the default terminal is invoked.

Using the Message Filter primitive

By default, the Message Filter primitive has one input terminal (In), one output terminal (Default), and one fail terminal (Fail). You can add any number of output terminals and assign meaningful names to them for routing filtered messages.

To add a new output terminal, right-click the primitive icon and select **Add Output Terminal** from the pop-up menu as shown in Figure 5-58.

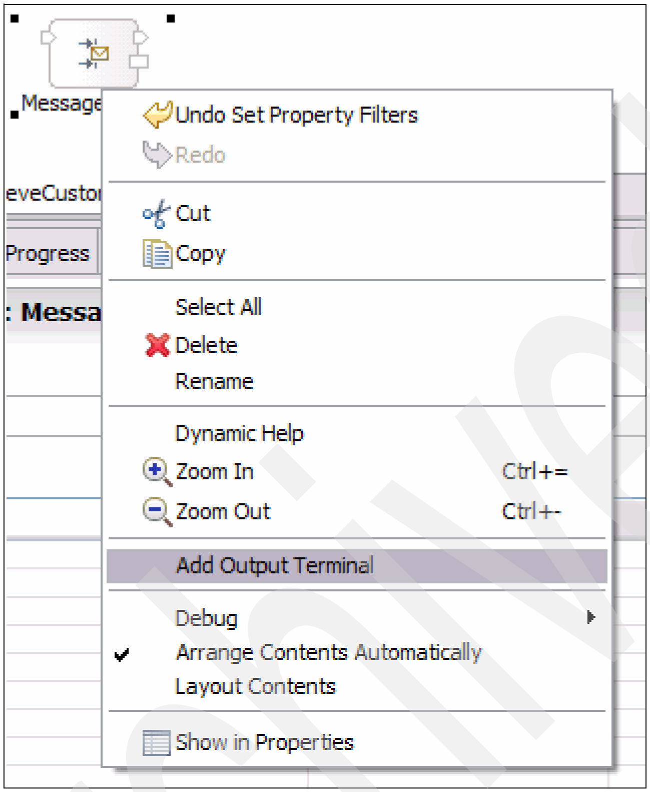
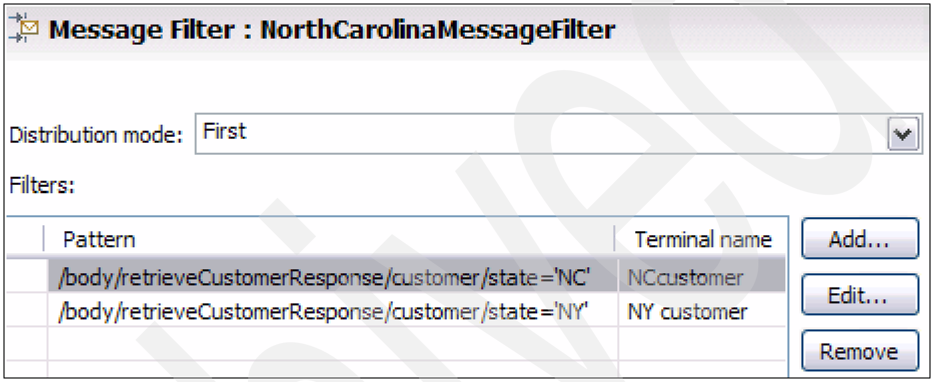


Figure 5-58 Add an output terminal

To filter out North Carolina (state = ‘NC’) and New York (state = ‘NY’) customers in the response flow of retrieveCustomer operation

1. Add two output terminals and call them `NCcustomer` and `NYcustomer`.
2. Then, create an XPath expression to detect an NC customer, and direct the message to leave the primitive through the NCcustomer terminal.
3. In the same way, direct the message to leave the primitive through the NYcustomer terminal if the customer is from NY.

   Figure 5-59 shows the details of this primitive.



Message Filter : NorthCarolinaMessageFilter

Distribution mode: First

Filters:

Pattern	Terminal name
/body/retrieveCustomerResponse/customer/state='NC'	NCcustomer
/body/retrieveCustomerResponse/customer/state='NY'	NY customer

Figure 5-59 Message Filter properties

4. Finally, connect the NCcustomer terminal to NorthCarolinaXSLT, the NYcustomer terminal to the Fail1 primitive, and the Default terminal to WebServiceResult, as shown in Figure 5-60.

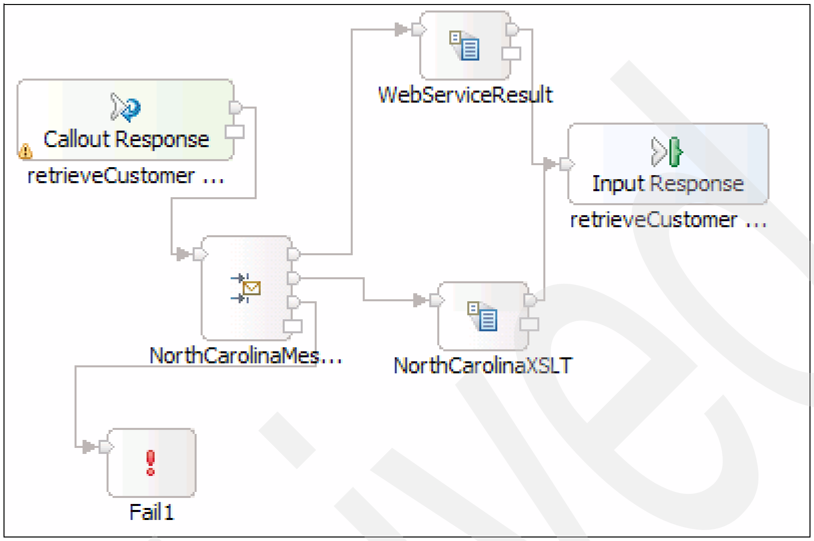


Figure 5-60 Message Filter primitive in the mediation flow

5.7.2 The Endpoint Lookup primitive

Additional materials: We include a sample mediation called *CustomerServiceEndpointMediation* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

The Endpoint Lookup primitive is used to route messages dynamically to appropriate service endpoints. The Endpoint Lookup primitive searches for service information in WebSphere Service Registry and Repository and, if found, updates the message header (/headers/SMOHeader/Target/address) with the appropriate dynamic endpoint information.

To use the Endpoint Lookup primitive, you need to add service endpoint information to the WebSphere Service Registry and Repository registry. If you want to extract service endpoint information about Web services, the WebSphere Service Registry and Repository registry must contain either the appropriate WSDL documents or SCA modules that contain exports using Web service bindings. If you want to extract service endpoint information about exports that use the default SCA binding, the WebSphere Service Registry and Repository registry must contain the appropriate SCA modules.

The Endpoint Lookup primitive lets you retrieve service endpoint information that relates to the following:

- Web services using SOAP/HTTP
- Web services using SOAP/JMS
- SCA module exports with Web service bindings, using SOAP/HTTP
- SCA module exports with Web service bindings, using SOAP/JMS
- SCA module exports with the default SCA binding

Properties

The properties to note are:

- *Name*: Search the registry for services that implement a particular portType, the name of which is specified by Name.
- *Namespace*: Search the registry for services that implement a particular portType, the namespace of which is specified by Namespace. The PortType Namespace can be specified in any valid namespace format (for example, URI or URN).
- *Version*: Search the registry for services that implement a particular portType, the version of which is specified by Version.
- *Registry Name*: Identifies the WebSphere Service Registry and Repository definition to be used by Endpoint Lookup primitive. A WebSphere Service Registry and Repository definition is created using the server administrative console. It provides connection information for a WebSphere Service Registry and Repository instance. At least one WebSphere Service Registry and Repository definition needs to exist on the server on which your mediation module is installed. If the Registry Name is absent, then the default WebSphere Service Registry and Repository definition is used.
- *Match Policy*: If the registry has more than one service that matches your query, the Match Policy determines how many service endpoints need to be added to the message.

Advanced properties

Advanced properties enable registry searches for services that are annotated with user-defined properties. The properties include:

- *Name*: The name of the user defined property.
- *Type*: The type of the user defined property. If the type is *String*, then what you specify as the Value is used as a literal in the search query. If the type is *XPath*, then what you specify as the Value must be an XPath expression. The XPath expression must resolve to a unique leaf node in the inbound SMO. The value of the leaf node is used in the search query.

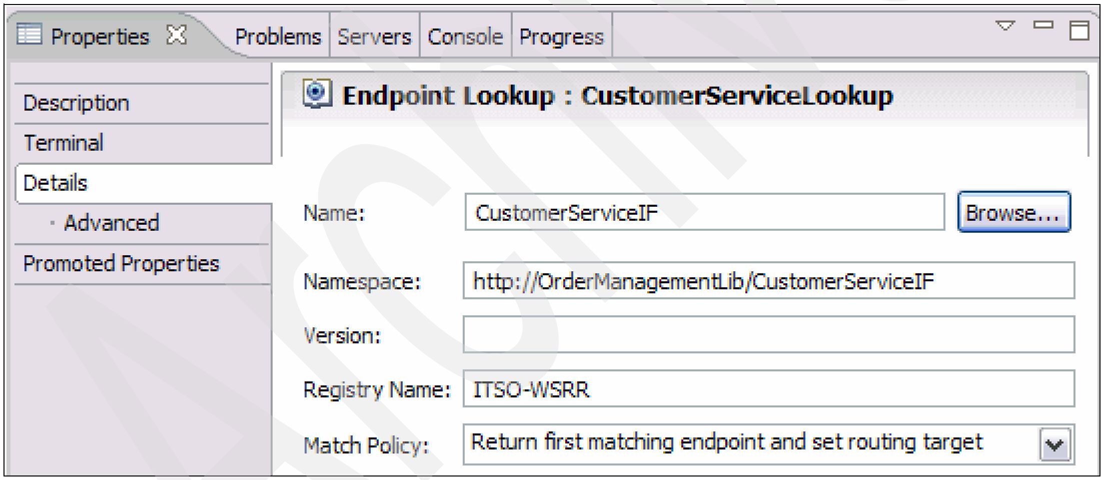
- *Value*: The value of the user-defined property. This value can be either a literal value or an XPath expression, depending upon the Type property.
- *Classification*: Search for objects that match a particular classification.

Using the Endpoint Lookup primitive

The Endpoint Lookup primitive has one input terminal (In), two output terminals (Out and noMatch), and a fail terminal. If a service endpoint is found, the updated message exits through the Out terminal. If no matching services are found, the message exits the node through the noMatch terminal.

This example assumes that a WebSphere Service Registry and Repository implementation called ITSO-WSRR is up and running. It contains a registered service that implements CustomerServiceIF with the http://OrderManagementLib/CustomerServiceIF namespace. Additionally, the service registry has a user-defined property called CustomerID that has a value of *1*.

Figure 5-61 shows how to set the properties to retrieve endpoint information from ITSO_WSRR for a given interface name and the namespace value.

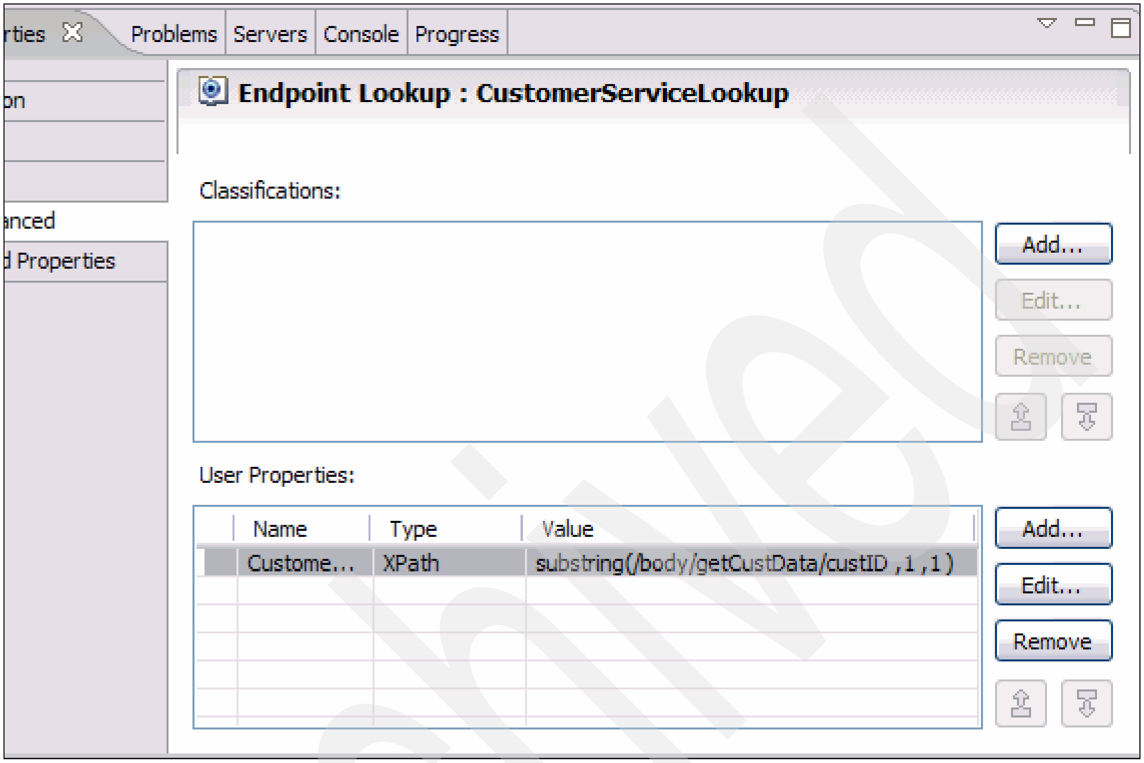


Endpoint Lookup : CustomerServiceLookup

Name: CustomerServiceIF
Namespace: http://OrderManagementLib/CustomerServiceIF
Version:
Registry Name: ITSO-WSRR
Match Policy: Return first matching endpoint and set routing target

Figure 5-61 Endpoint Lookup properties

To stipulate that this service is invoked only if the customer ID in the Input message starts with *1*, set the Advanced properties as in Figure 5-62.



Endpoint Lookup : CustomerServiceLookup

Classifications:

User Properties:

Name	Type	Value
Custome...	XPath	substring(/body/getCustData/custID ,1 ,1)

Figure 5-62 Endpoint Lookup advanced properties

If the matching service is found successfully, the updated message exits the primitive through the Out terminal to proceed with a Web service call as shown in Figure 5-63.

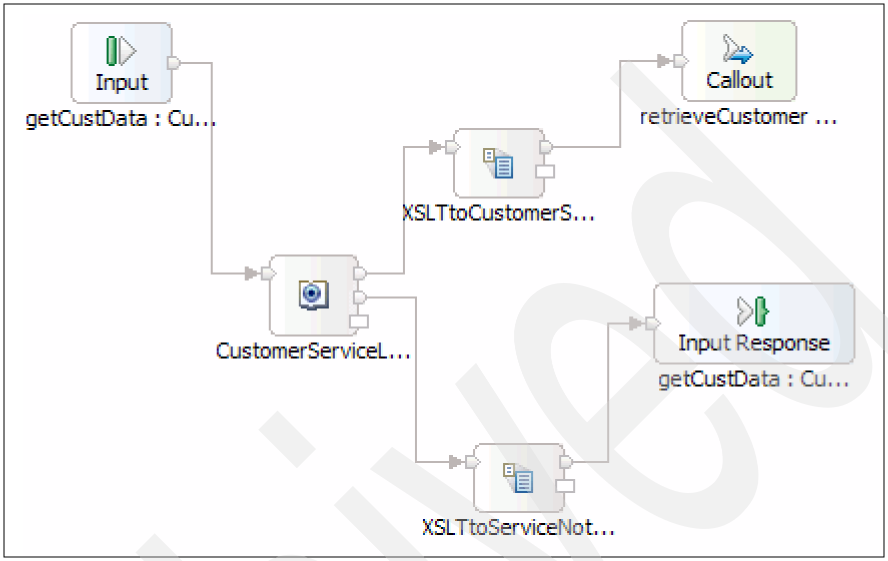
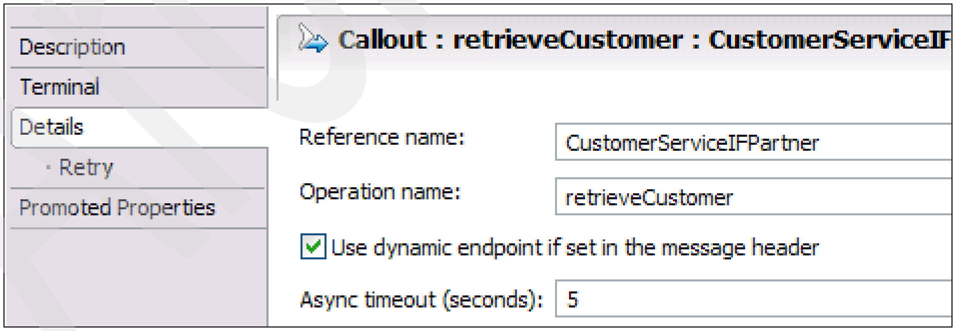


Figure 5-63 Endpoint Lookup in the mediation flow

To use the dynamic endpoint information, you need to enable the “Use dynamic endpoint if set in the message header” property for the Callout node as shown in Figure 5-64. Otherwise, the run time uses the default endpoint if there is one, or throws an error.



Callout : retrieveCustomer : CustomerServiceIF

Reference name: CustomerServiceIFPartner

Operation name: retrieveCustomer

☑ Use dynamic endpoint if set in the message header

Async timeout (seconds): 5

Figure 5-64 Set dynamic endpoint setting in the Callout

5.7.3 Fan Out and Fan In primitive

This section provides an overview of the Fan Out and Fan In primitives (new in WebSphere Enterprise Service Bus V6.1). These primitives let you split and aggregate messages and flows within mediations to provide parallel processing over the message.

Use of the Fan Out and Fan In primitives involves the following concepts:

- Context management
- Service inline invocation through the new Service Invoke primitive
- Transformation primitives

Fan Out and Fan In primitives are used for two main objectives:

- To split a message, perform some processing, and then combine (aggregate) the resulting messages, as illustrated in Figure 5-65. Both the Fan Out and Fan In primitives are required to perform this type of function.

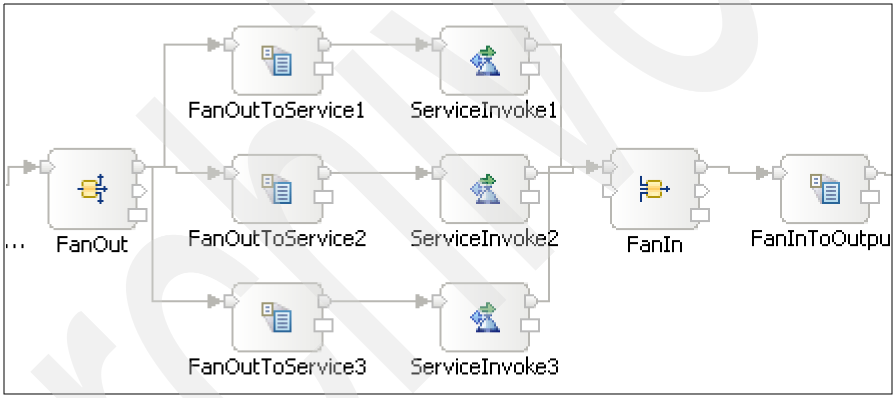


Figure 5-65 Fan Out and Fan In usage in a split and aggregate fashion

- To Broadcast messages one way to services using the Fan Out primitive, as illustrated in Figure 5-66.

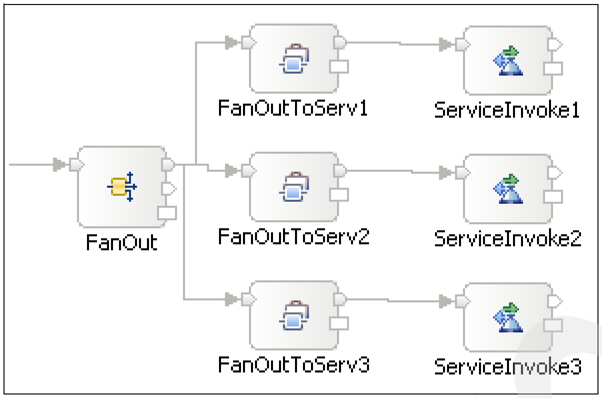


Figure 5-66 Broadcasting messaging through Fan Out

Transformation is likely to be involved in both cases, as follows:

- To map the input message to the format that the invoked service expects.
- In the split and aggregation scenario, to map the aggregated messages to the output format.
- To move elements in and out from the body to the correlation, transient, and shared context.

Also in both cases, you are also likely to invoke multiple services. These services are included in the assembly diagram (Figure 5-67).

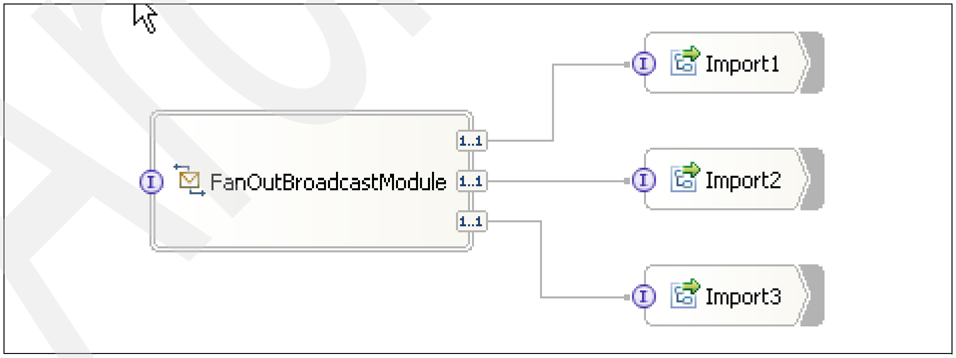


Figure 5-67 Aggregation and broadcast mediation flows wired with the used services

You do not need to connect the interface and partners in the mediation flow editor, as illustrated in Figure 5-68.

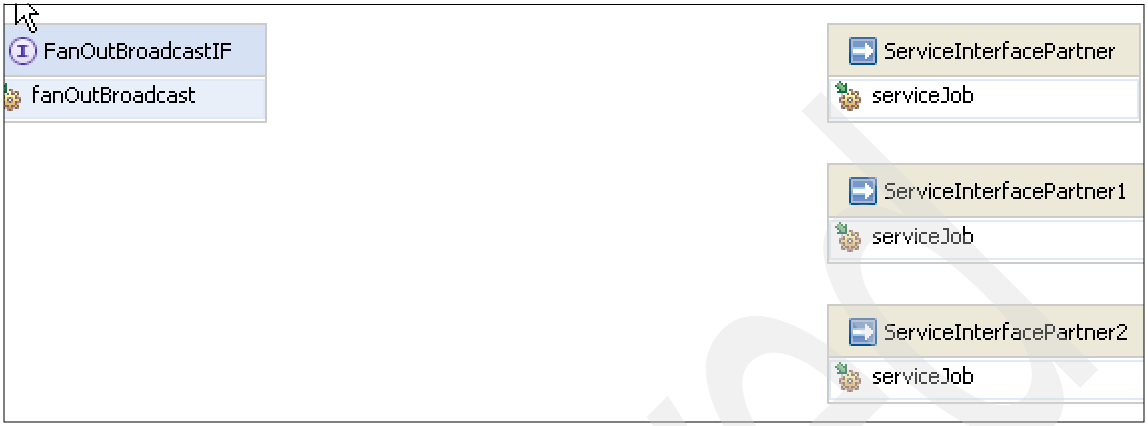


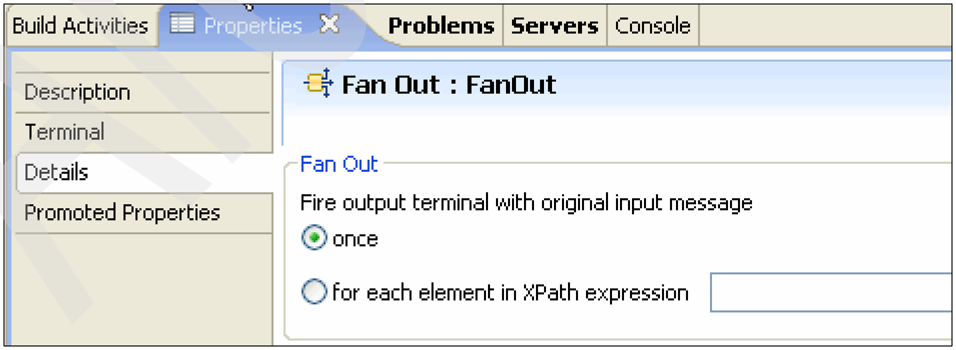
Figure 5-68 Mediation Flow Editor while using inline service invocations in Fan Out Fan In flows

Fan Out primitive

In the Fan Out primitive configuration, you must decide how many times you want the primitive to fire the input message.

If you specify *once*, Fan Out sends one identical copy of the same message along each of the output branches. All of the copies share a *shared context*, where you can store the service results after their execution.

Alternatively, you can send the same object multiple times along every output branch. In this case, you need to specify an XPath expression, and the message is fired as many times as the expression element occurrences.



Fan Out : FanOut

Fan Out

Fire output terminal with original input message

● once

○ for each element in XPath expression

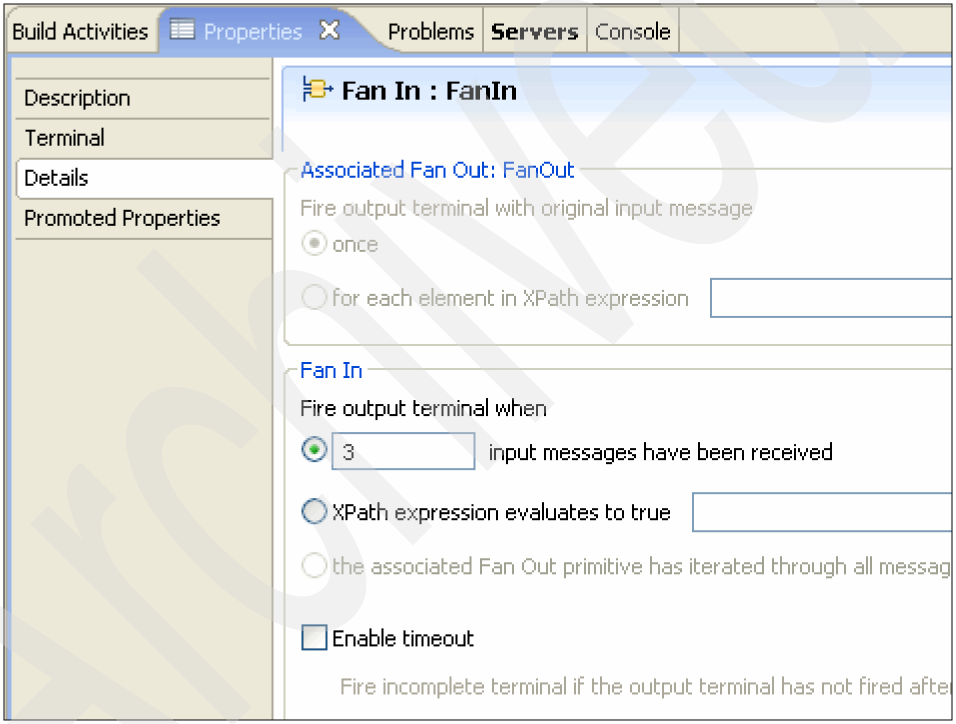
Figure 5-69 Possible configuration of the Fan Out primitive

Fan In primitive

The Fan In primitive is associated with a Fan Out primitive. You configure the condition that makes the primitive fire out the output message (Figure 5-70). The condition can be one of the following conditions:

- A certain number of messages has arrived to the Fan In over the incoming branches.
- An XPath expression that is compared to the incoming messages evaluates to true.

Optionally, you can specify a timeout that makes the primitive fire out the message when it expires.



Fan In : FanIn

Associated Fan Out: FanOut

Fire output terminal with original input message

● once

○ for each element in XPath expression

Fan In

Fire output terminal when

● 3 input messages have been received

○ XPath expression evaluates to true

○ the associated Fan Out primitive has iterated through all messag

□ Enable timeout

Fire incomplete terminal if the output terminal has not fired afte

Figure 5-70 Fan In configuration

> **Note:** When you insert a Fan In primitive and indicate the corresponding Fan Out primitive, both the primitive configurations show the corresponding primitive in read-only mode.

5.7.4 Service Invoke primitive

Additional materials: We include a sample mediation called *ServiceInvocationSample* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

The Service Invoke primitive lets you call a service from inside a mediation flow, rather than waiting until the end of the mediation flow and using the callout mechanism. The input message is used to call the service and, if the call is successful, the response is used to create the output message. If the call is unsuccessful you can retry the same service, or call another service.

Properties

The properties to note are:

- *Reference Name*: The name of the service reference to be called. The reference name is associated with a WSDL interface. Initially, the reference name is set through a WebSphere Integration Developer dialog box and cannot be changed afterwards. You have to create a new Service Invoke primitive to change the reference name.
- *Operation Name*: The name of the service operation to be called. The operation name is associated with a WSDL operation. Initially, the Operation Name is set through a WebSphere Integration Developer dialog box and cannot be changed afterwards. You have to create a new Service Invoke primitive to change the operation name.
- *Use Dynamic Endpoint if set in the message header*: Determines whether the SMO header field Target, if present, needs to be used to override the service endpoint specified by the reference operation. You can use the Endpoint Lookup primitive to set the Target field, or you can set the field yourself.
- *Async Timeout*: The time to wait for a response when a call is asynchronous with a deferred response. The Async Timeout property is not used for calls that are asynchronous with callback. If the Async Timeout is 0, there is no wait and the response is immediate. If the Async Timeout is -1, the wait is indefinite. When a timeout occurs the timeout terminal is fired. A timeout is treated as an unmodeled fault, with regard to retry.
- *Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback*: Set to true (select the check box) to force a service call to act in a synchronous manner. If true, an asynchronous call causes a deferred response, rather than a callback. Set this property to true if the whole mediation flow is to run in a single transaction. If you set this property to false and the primitive is involved in a Fan Out/Fan In operation,

the run time overrides the setting and forces the service call to act in a synchronous manner.

- *Retry On*: Determines whether, and how, fault responses cause a retry. The following values are valid:

  – Never
  – Any fault
  – Unmodeled fault
  – Modeled fault

  To enable retry, you must set the value of the Retry On property to Any fault, Unmodeled fault, or Modeled fault.

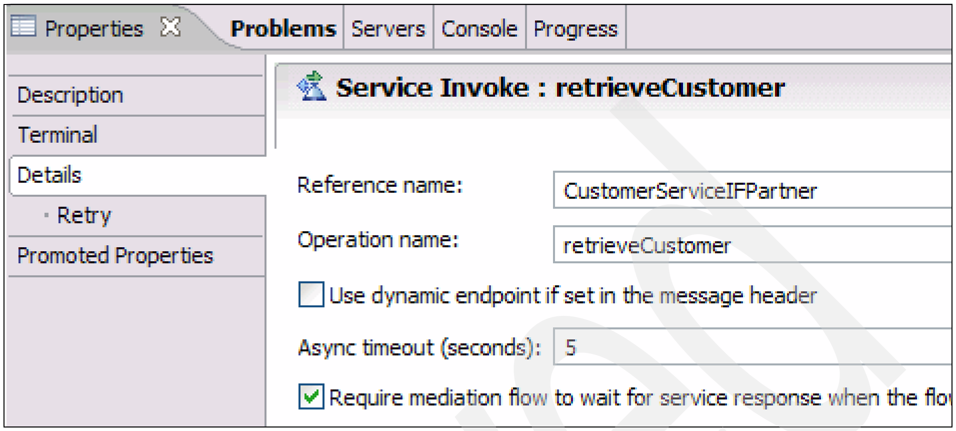
  Modeled faults are those that are explicitly listed in a WSDL file. Any other fault is an unmodeled fault. This property is only applicable to request-response operations.

- *Retry Count*: How many times a service call is retried before an output terminal is fired. The output terminal that is fired can be modeled fault, timeout, or fail. The value can be zero or a positive integer.
- *Retry Delay*: The delay (in seconds) between retry attempts. The value can be zero or a positive integer.
- *Try Alternate Endpoints*: Determines whether any alternate endpoints in the SMO need to be used on retries. Set to true (select the check box) to try to alternate endpoints.

Using the Service Invoke primitive

The Service Invoke primitive has one input terminal (In) and multiple output terminals. There is a fail terminal (Fail) for unmodeled faults, and one output terminal for each modeled fault. Modeled faults are those that are explicitly listed in a WSDL file. Any other fault is an unmodeled fault. In addition, there is an output terminal (Out) that maps to the WSDL response message and a timeout terminal (Timeout) that is used for some types of asynchronous calls.

Figure 5-71 shows how to set properties to call retrieveCustomer operation of CustomerServiceIF.



Service Invoke : retrieveCustomer

Reference name: CustomerServiceIFPartner

Operation name: retrieveCustomer

□ Use dynamic endpoint if set in the message header

Async timeout (seconds): 5

☑ Require mediation flow to wait for service response when the flo

Figure 5-71 Service Invoke properties

Figure 5-72 shows the retrieveCustomer Service Invoke primitive in the mediation flow. If the service invoke is successful, the updated message exits the primitive through the Out terminal and proceeds to CustomerInfoRetrieved transformation primitive. If the service call failed or timed out, the flow is routed to CustomerNotFound transformation.

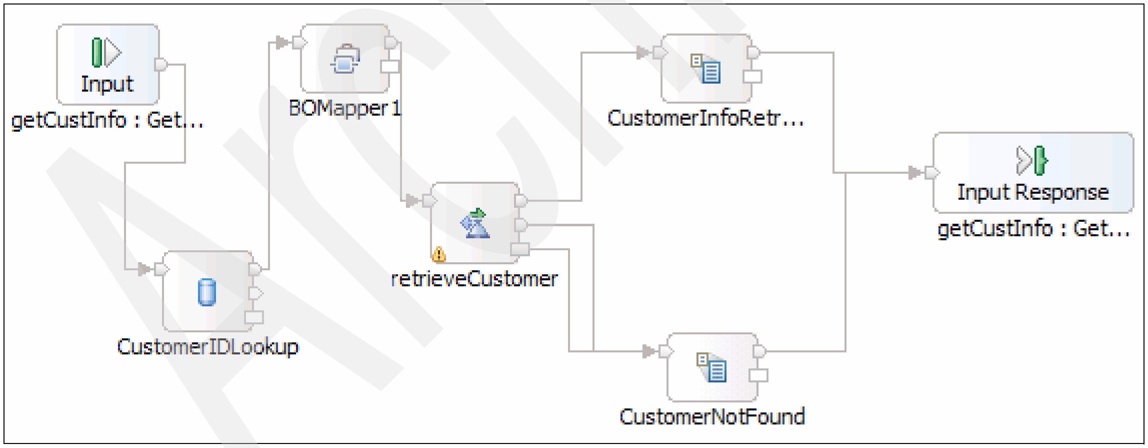


Figure 5-72 Service invoke primitive in a message flow

> **Note:** This type of mediation does not require a response flow.

5.8 Tracing primitives

The primitives in the Mediation Flow Editor in the Tracing category are:

- Message Logger primitive
- Event Emitter primitive

5.8.1 Message Logger primitive

The Message Logger primitive stores messages in a relational database during a mediation flow. The Message Logger primitive logs messages to a relational database using an IBM-defined database schema (table structure). It does not write to other storage mediums such as flat files.

Properties

The properties to note are:

- *Data source name*: The JNDI name of the data source that defines where the data will be logged. At V6.1, the default jdbc/mediation/messageLog maps to the Common database.
- *Root*: An XPath 1.0 expression representing the scope of the message to be logged. You can specify:
  - / refers to the complete SMO
  - /body refers to the body section of the SMO
  - /headers refers to the headers of the SMO
  - Your own XPath expression

  If you specify your own XPath expression, the part of the SMO you specify is processed. The message to be logged is converted to XML from the point specified by Root.
- *Transaction mode*: Defines whether to commit changes to the database, in the flow’s transaction or in a new transaction.

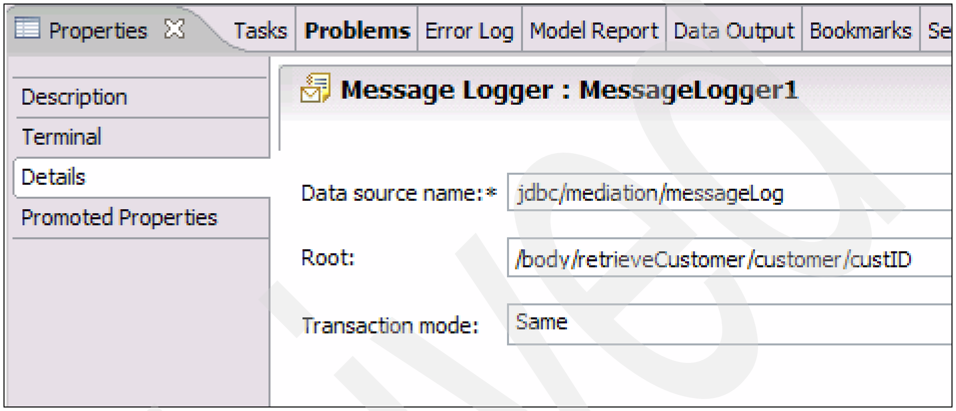
  If you specify `Same`, the message is logged in the flow’s transaction. By default, the flow is executed under a local transaction although the mediation component can be configured to run under a global transaction. If a global transaction is specified and a failure occurs in the flow then the global transaction, including the log operation, is rolled back.

  If you specify `New`, the message is logged in its own local transaction. In this case, if a failure occurs in the flow the message logging is not rolled back.

Using the Message Logger primitive

The Message Logger primitive has one input terminal (In), one output terminal (Out), and one Fail terminal.

Figure 5-73 shows how to set the primitive properties to log *custID* value from the Input message. Note that you do not have to type the *Data source name* value unless you want to change the default.



Message Logger : MessageLogger1

Data source name:* jdbc/mediation/messageLog

Root: /body/retrieveCustomer/customer/custID

Transaction mode: Same

Figure 5-73 Message Logger properties

The data source that defines that database must be defined to the application server. The data source definition must have the JNDI name defined to match the Data source name field specified here. The default definitions point to the Common database.

Figure 5-74 shows how Message Logger primitive is wired in the mediation flow.

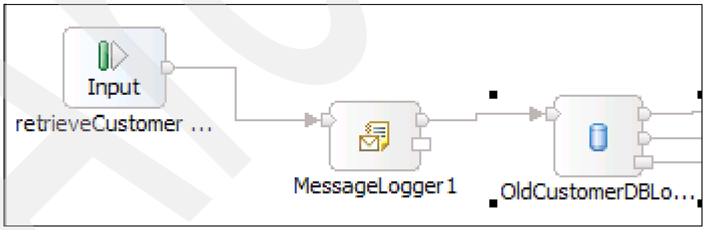
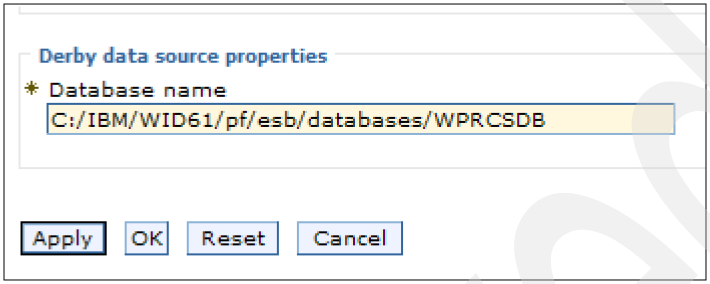


Figure 5-74 Message Logger primitive in a message flow

To view the Message Logger database, find the name and location of the database corresponding to the data source name jdbc/mediation/messageLog on the WebSphere Enterprise Service Bus server. To find the database:

1. Open the administrative console for WebSphere Enterprise Service Bus.
2. Select **Resources** → **JDBC** → **Data Sources**.

3. Select **All Scopes** as the scope.
4. Match the data source name specified in the Message Logger primitive to the JNDI name column.
5. Double-click the data source to open it, and find the database name in the data source properties (Figure 5-75).



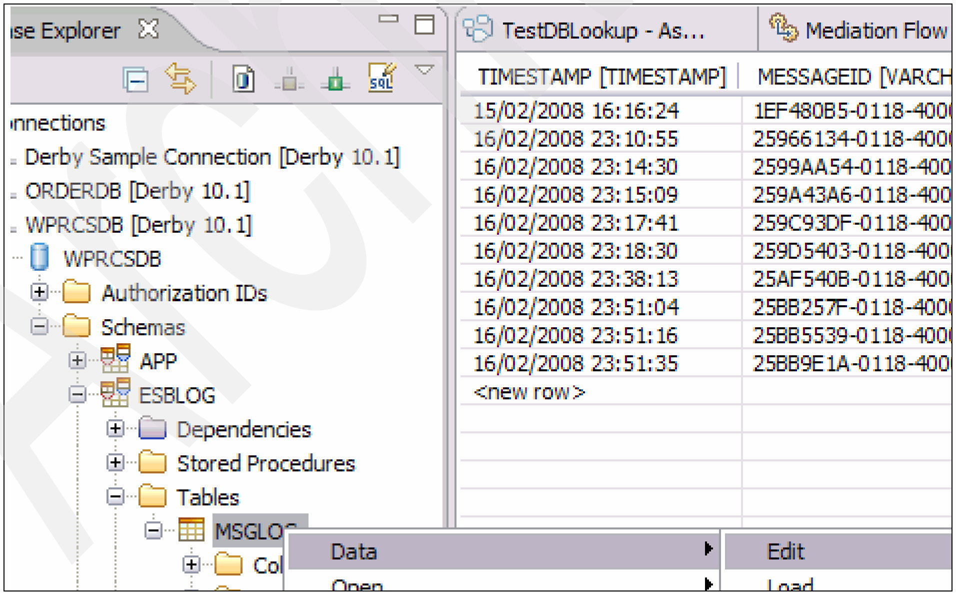
Derby data source properties

* Database name

C:/IBM/WID61/pf/esb/databases/WPRCSDB

Figure 5-75 Derby data source properties

You can use the Database Explorer view in the Data perspective to create a connection to the logging database and to view the logging records. By default, the records are in the MSGLOG table of the WPRCSDB database as shown in Figure 5-76.



TIMESTAMP [TIMESTAMP]	MESSAGEID [VARCH
15/02/2008 16:16:24	1EF480B5-0118-400
16/02/2008 23:10:55	25966134-0118-400
16/02/2008 23:14:30	2599AA54-0118-400
16/02/2008 23:15:09	259A43A6-0118-400
16/02/2008 23:17:41	259C93DF-0118-400
16/02/2008 23:18:30	259D5403-0118-400
16/02/2008 23:38:13	25AF540B-0118-400
16/02/2008 23:51:04	25BB257F-0118-400
16/02/2008 23:51:16	25BB5539-0118-400
16/02/2008 23:51:35	25BB9E1A-0118-400
<new row>	

Figure 5-76 Viewing data in a table

5.8.2 Event Emitter primitive

> **Additional materials:** We include a sample mediation called *EventEmitterMediation* in Appendix B, “Additional material” on page 511 to illustrate this primitive.

The Event Emitter primitive sends out business events during a mediation flow. The events are generated in the form of common base events (CBE) and are sent to a common event infrastructure (CEI) server.

Properties

The properties to note are:

- *Label*: Allows you to define a unique identifier for the event. The unique identifier maps to the extension name of the CBE. WebSphere Integration Developer provides a default label, but it is strongly recommended that you provide a more meaningful event label for your particular event type. Events are emitted to the CEI server, which can be accessed by many different event consumer applications. Therefore, event names need to be unique across the system in order to distinguish different event types. If two Event Emitter primitives generate exactly the same event type, it might be acceptable to have the same Label name.
- *Transaction mode*: Allows you to override the transaction mode set on the emitter. (An event source, such as an Event Emitter primitive, does not interact directly with the event server. Instead it interacts with an object called an *emitter*.) The transaction mode can be configured in the CEI infrastructure or overridden at the Event Emitter primitive level.

  If you specify the `Default` transaction mode, events are sent to the CEI server using the default setting in the CEI emitter.

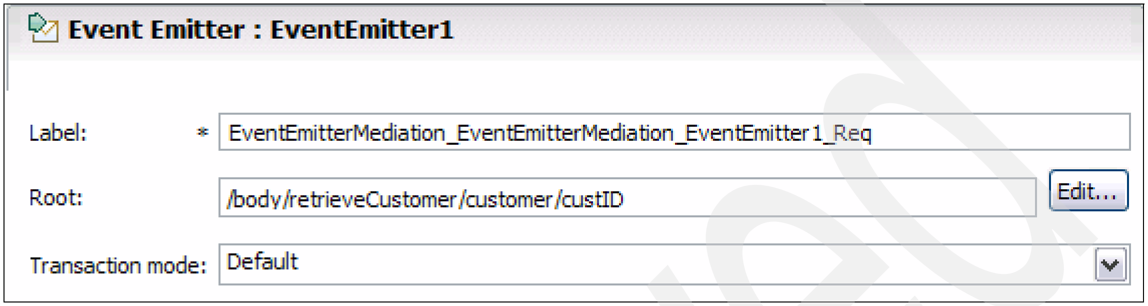
  If you specify the `Existing` transaction mode, events are sent to the CEI server in the flow’s transaction.

  If you specify the `New` transaction mode, events are sent to the CEI server outside the flow's transaction.
- *Root*: An XPath 1.0 expression representing the part of the message to be included in the CBE. You can specify:
  - /
  - /headers
  - /context
  - /body
  - Your own XPath expression

Using the Event Emitter primitive

The Event Emitter primitive has one input terminal, one output terminal, and a fail terminal.

Figure 5-77 shows the primitive properties for the case when custID of the Input message is used as event value.



Event Emitter : EventEmitter1

Label: * EventEmitterMediation_EventEmitterMediation_EventEmitter1_Req

Root: /body/retrieveCustomer/customer/custID

Transaction mode: Default

Figure 5-77 Event Emitter properties

In the sample mediation shown in Figure 5-78, the input message is routed to the retrieveCustomer service call if customer ID starts with *1*. Otherwise, a new event is emitted with customer ID as its content. The MessageFilter1 primitive is responsible for filtering, while EventEmitter1 emits the event.

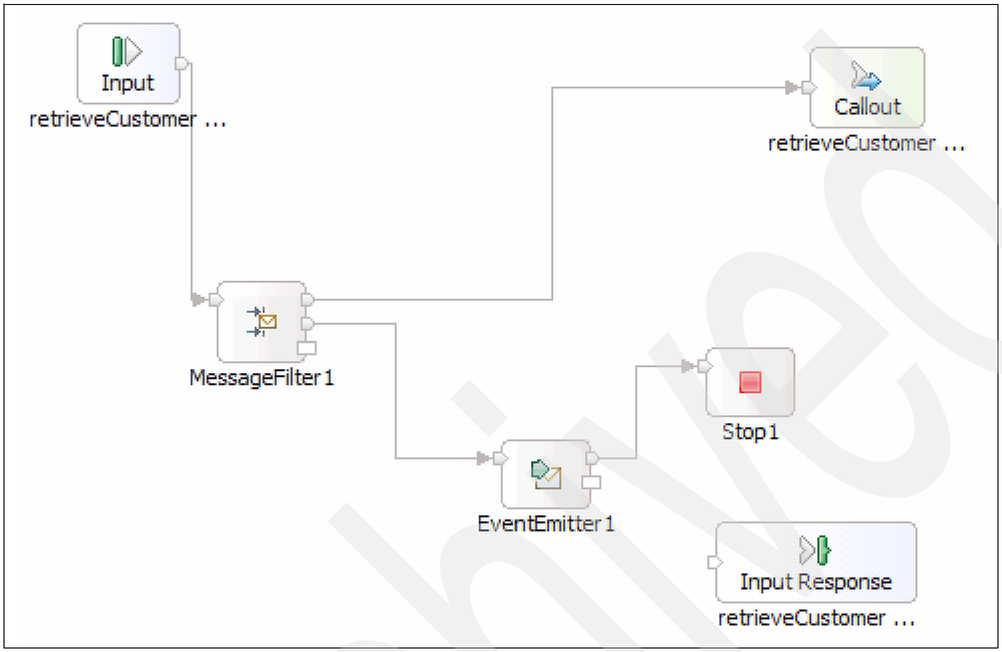


Figure 5-78 Event Emitter in a message flow

To see the events that are emitted, you need to start the Common Base Event Browser as shown in Figure 5-79.

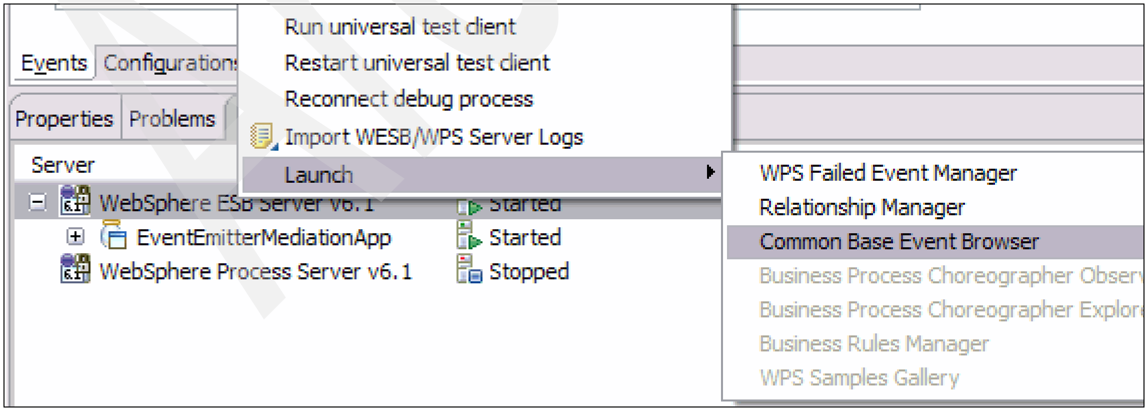

Events
Configuration
Properties
Problems
Server
WebSphere ESB Server v6.1
Started
EventEmitterMediationApp
Started
WebSphere Process Server v6.1
Stopped
Run universal test client
Restart universal test client
Reconnect debug process
Import WESB/WPS Server Logs
Launch
WPS Failed Event Manager
Relationship Manager
Common Base Event Browser
Business Process Choreographer Observ
Business Process Choreographer Explor
Business Rules Manager
WPS Samples Gallery

Figure 5-79 Starting the Common Base Event Browser

5.9 Error Handling primitives

The primitives in the Mediation Flow Editor in the Error Handling category are:

- Fail primitive
- Stop primitive

5.9.1 Fail primitive

The Fail primitive stops the mediation flow and raises an exception. Existing transactions are rolled back and the module throws a FailFlowException.

The Fail mediation has the *Error message* property. This property is an optional, user-supplied, error message. The Error message value is added to the FailFlowException that is generated by the Fail primitive.

The Fail primitive has only one in terminal.

.

> **Note:** If a transaction is in progress, when the mediation flow is stopped by a fail primitive, the transaction is rolled back, and the FailFlowException is stored in the transient context.

5.9.2 Stop primitive

The Stop primitive stops a particular path in the flow.

The primitive has no properties. The Stop primitive has only one in terminal.

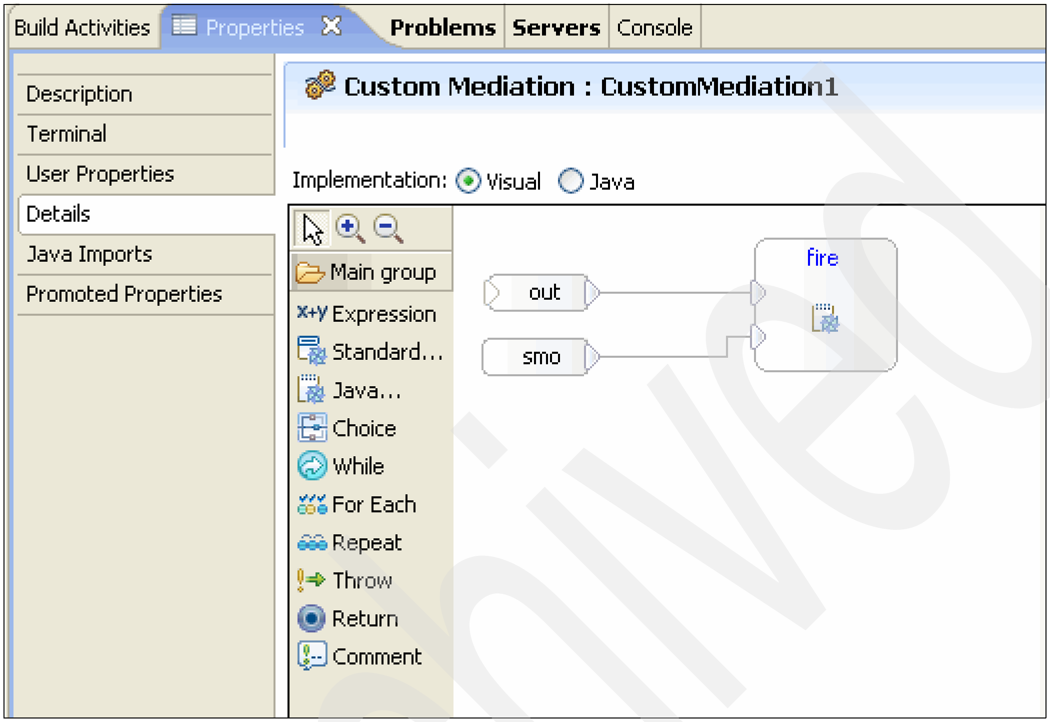
You can use the Stop primitive to consume exceptions silently. If another primitive’s Fail terminal is wired to a Stop primitive, then any messages that go to the Fail terminal are consumed. This has the same effect as an empty catch block in Java.

5.10 Custom Mediation primitive

A Custom Mediation primitive lets you do whatever you do not find covered in other default Transformation primitives. The Custom Mediation primitive receives an input SMO, manipulates it according to your definition, and fires it out.

The Custom Mediation primitive consists of Java code. You can enter the Java code in directly or use the Visual editing tool to create it.

Figure 5-80 shows an example of using the Visual editor to create the code. It shows the features that are available (for example, expression builders, iterative constructs, choices, and so forth).



Implementation: ● Visual ○ Java

Figure 5-80   Custom mediation visual editor

You can add an extra terminal to a Custom Mediation primitive by right-clicking the Custom Mediation primitive and selecting either **Add Input Terminal** or **Add Output Terminal**. The name that you give to a terminal is used in code generation and, therefore, must be a valid Java identifier.

You can define your own properties in a Custom Mediation primitive by going to the User Properties tab of the Properties view. You can add, edit, and remove user properties.

6

Mediation examples

This chapter includes a series of mediation examples. These examples illustrate the use of some of the mediation primitives. It also provides information about how to build and test mediations.

> **Additional material:** The examples that we discuss in this chapter (and more) are included in the additional materials for this book. For information about downloading the additional materials, see Appendix B, “Additional material” on page 511. The mediation examples are included in the MedationSamples.zip project interchange file. Be sure to import all the projects in this project interchange file.

This chapter includes the following topics:

- Database Lookup example
- Message Element Setter example
- Set Message Type example
- Message Filter example
- Endpoint Lookup example
- The Event Emitter primitive

6.1 Database Lookup example

Additional material: This example is included in the TestDbLookup and DBMSServiceMediation projects in the additional materials.

This example illustrates the use of a Database Lookup primitive and the JDBC adapter in a mediation flow. In this example, we assume that customer information is stored in multiple places. Some records are in a local database called OLDDB. Others can be retrieved through the CustomerService Web service. The Web service is emulated by the DBMSServiceMediationApp of our OrderManagement application.

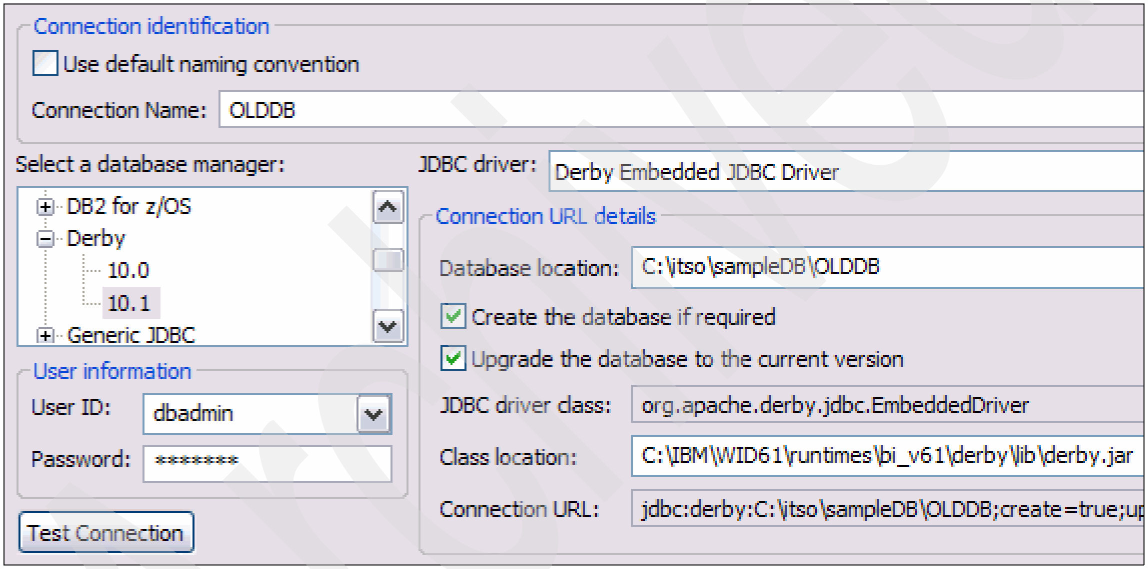
The instructions in this section illustrate how this mediation and its database were created.

6.1.1 Create the database and connection

This example assumes that the Derby database has been created. The next step defines a connection to the database from the WebSphere Integration Developer workspace.

Switch to the Data perspective and create a connection to the database:

1. In the Database Explorer view, right-click **Connections** and select **New Connection**.
2. Select **Derby 10.1** as the database manager and complete the required fields as shown in Figure 6-1.
3. Click **Finish**.



Connection identification

☐ Use default naming convention

Connection Name: OLDDB

Select a database manager:

- DB2 for z/OS
- Derby
  - 10.0
  - 10.1
- Generic JDBC

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\itso\sampleDB\OLDDB

☑ Create the database if required

☑ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\IBM\WID61\runtimes\bi_v61\derby\lib\derby.jar

Connection URL: jdbc:derby:C:\itso\sampleDB\OLDDB;create=true;up

User information

User ID: dbadmin

Password: ********

Figure 6-1 New database connection properties

Create the table

To create the table, follow these steps:

1. Click the Open SQL Editor icon (Figure 6-2). to open a New_Statement_1 tab, where you can type or paste SQL statements.

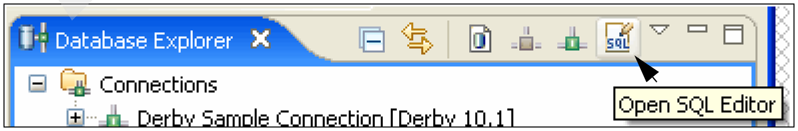


Figure 6-2 Open SQL Editor icon

2. Copy the contents of OLD_CUSTOMER_TABLE.ddl.
3. Right-click anywhere in the SQL editor panel and select **Run SQL** from the pop-up menu.
4. In the Connection Selection panel, select **Use an existing connection** and select the **OLDDB Connection**.
5. Click **Finish**. The table is created in the database, and the Connection structure reflects this (Figure 6-3).

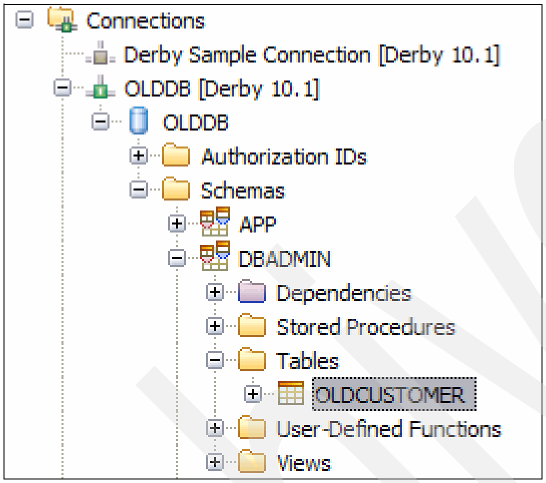


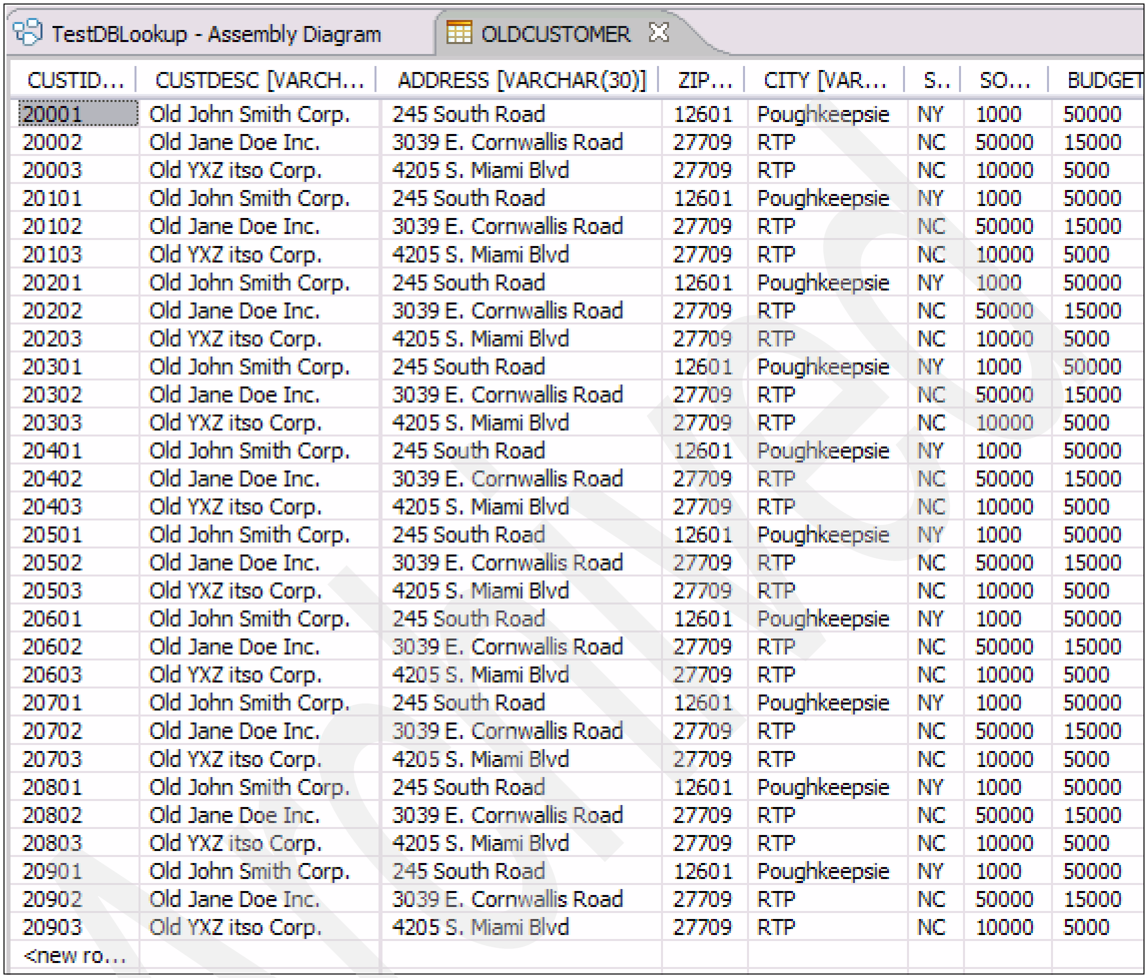
Figure 6-3 New connection to OLDDB

Load the data

To load the data, follow these steps:

1. Right-click the OLDCUSTOMER table, and select **Data** → **Load** from the pop-up menu.
2. On the Load Data panel select the OldCustomer.data file as the input file.

3. Click **Finish** to load the data. The customer data displays as shown in Figure 6-4.



CUSTID...	CUSTDESC [VARCH...	ADDRESS [VARCHAR(30)]	ZIP...	CITY [VAR...	S..	SO...	BUDGET
20001	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20002	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20003	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20101	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20102	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20103	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20201	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20202	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20203	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20301	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20302	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20303	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20401	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20402	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20403	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20501	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20502	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20503	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20601	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20602	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20603	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20701	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20702	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20703	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20801	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20802	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20803	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
20901	Old John Smith Corp.	245 South Road	12601	Poughkeepsie	NY	1000	50000
20902	Old Jane Doe Inc.	3039 E. Cornwallis Road	27709	RTP	NC	50000	15000
20903	Old YXZ itso Corp.	4205 S. Miami Blvd	27709	RTP	NC	10000	5000
<new ro...							

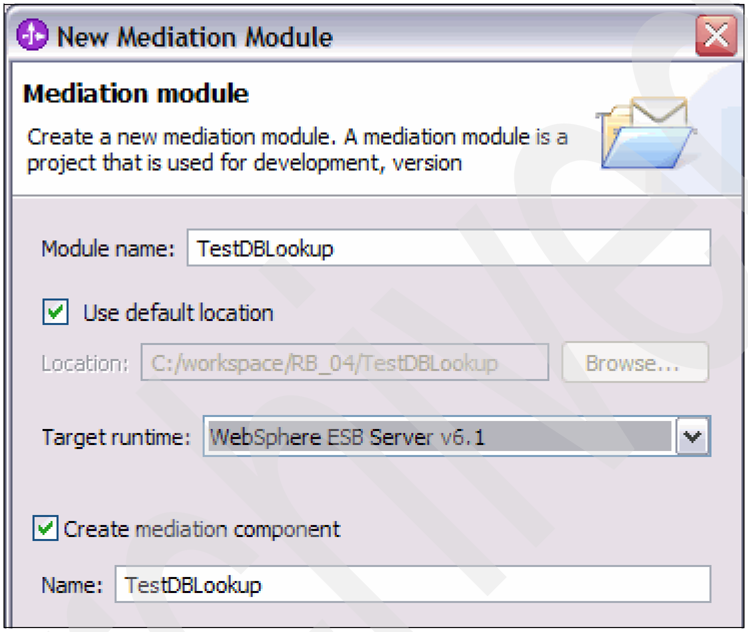
Figure 6-4 Customer data

4. Disconnect from the OLDDB database.

6.1.2 Create the mediation module

To create the mediation module, follow these steps:

1. Select **File** → **New** → **Mediation Module**. Name the new module TestDBLookup.
2. Set the target run time to **WebSphere ESB Server v6.1** as shown in Figure 6-5.



Mediation module

Create a new mediation module. A mediation module is a project that is used for development, version

Module name: TestDBLookup

☑ Use default location

Location: C:/workspace/RB_04/TestDBLookup

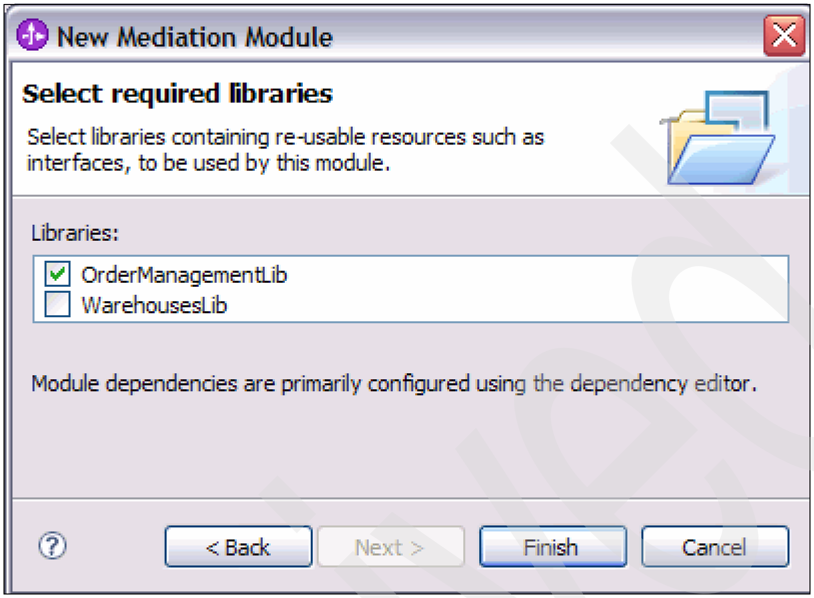
Target runtime: WebSphere ESB Server v6.1

☑ Create mediation component

Name: TestDBLookup

Figure 6-5 Create the TestDBLookup mediation module

3. Click **Next** and add **OrderManagementLib** as a required library (Figure 6-6).



Select required libraries

Select libraries containing re-usable resources such as interfaces, to be used by this module.

Libraries:

☑ OrderManagementLib
□ WarehousesLib

Module dependencies are primarily configured using the dependency editor.

Figure 6-6 Adding OrderManagementLib as a dependency

4. Click **Finish**.

6.1.3 Complete the assembly diagram

Initially, the assembly diagram contains one component, TestDBLookup. We need to add the import component for the CustomerService Web service.

1. In the Business Integration view, select **CustomerServiceIFExport1_CustomerServiceIFHttpPort** Web service port in OrderManagementLib (as shown in Figure 6-7).

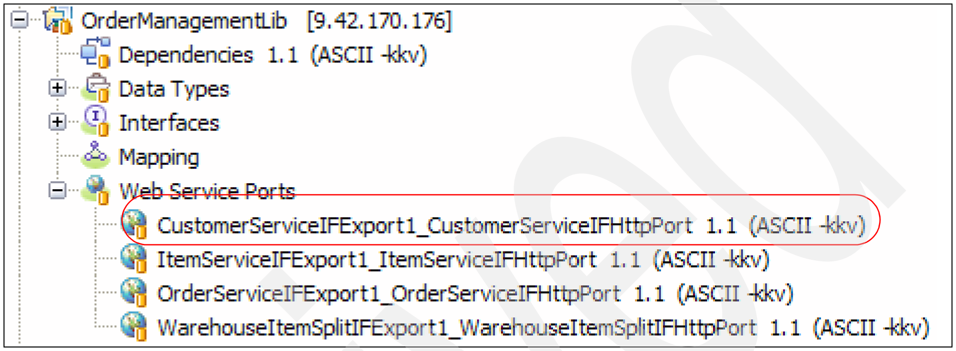

OrderManagementLib [9.42.170.176]
Dependencies 1.1 (ASCII -kkv)
Data Types
Interfaces
Mapping
Web Service Ports
CustomerServiceIFExport1_CustomerServiceIFHttpPort 1.1 (ASCII -kkv)
ItemServiceIFExport1_ItemServiceIFHttpPort 1.1 (ASCII -kkv)
OrderServiceIFExport1_OrderServiceIFHttpPort 1.1 (ASCII -kkv)
WarehouseItemSplitIFExport1_WarehouseItemSplitIFHttpPort 1.1 (ASCII -kkv)

Figure 6-7 Web service interface

2. Drag and drop it into the assembly diagram canvas. Choose **Import with Web Service Binding** in the Component Creation window and click **OK**. Name the import component `CustomerServiceIFImport`.
3. Wire TestDBLookup to CustomerServiceIFImport as shown in Figure 6-8.

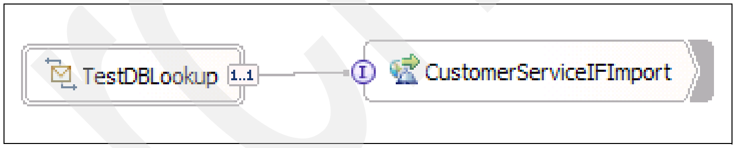
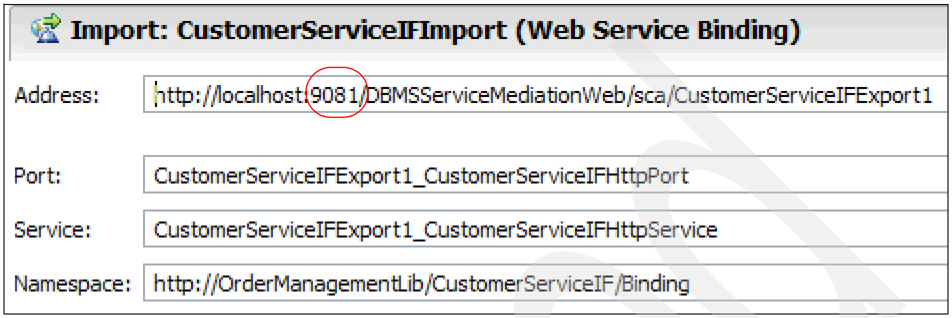


Figure 6-8 Assembly diagram

4. Select CustomerServiceIFImport and switch to its Binding tab in the Properties view. Make sure that the port number is set to the port that is used by the server on which you plan to test. Our test server uses port 9081. So, we change 9080 to 9081, as shown in Figure 6-9.


Import: CustomerServiceIFImport (Web Service Binding)

Address: http://localhost:9081/DBMSServiceMediationWeb/sca/CustomerServiceIFExport1

Port: CustomerServiceIFExport1_CustomerServiceIFHttpPort

Service: CustomerServiceIFExport1_CustomerServiceIFHttpService

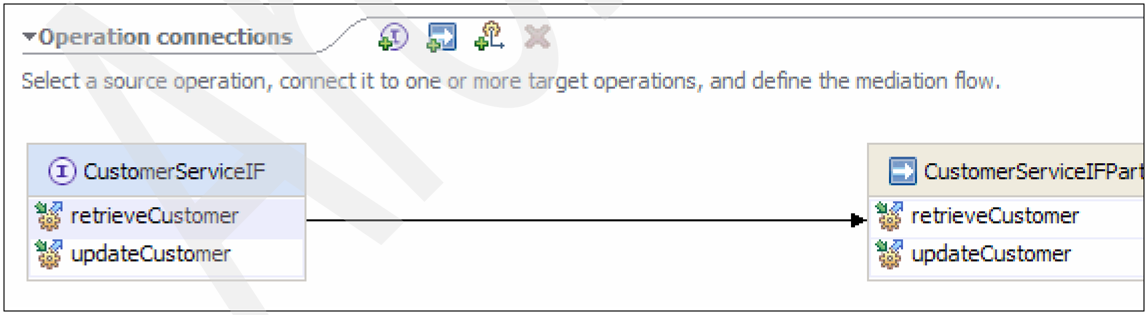
Namespace: http://OrderManagementLib/CustomerServiceIF/Binding

Figure 6-9 Set the port number

5. Add the CustomerServiceIF interface to the TestDBLookup component.
6. Right-click **TestDBLookup** and choose **Regenerate Implementation** from the pop-up menu. Answer *Yes* to replace the existing implementation to open a Mediation Flow Editor window.

6.1.4 Wire the operation connections

Wire the retrieveCustomer operation of CustomerServiceIF to the retrieveCustomer operation of CustomerServiceIFPartner. Select the wire that connects the two to open the mediation flow for that connection (Figure 6-10).


Operation connections

Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 6-10 Wire the operation connections

6.1.5 Build the mediation flow

Figure 6-11 shows the mediation flow that we build for this example.

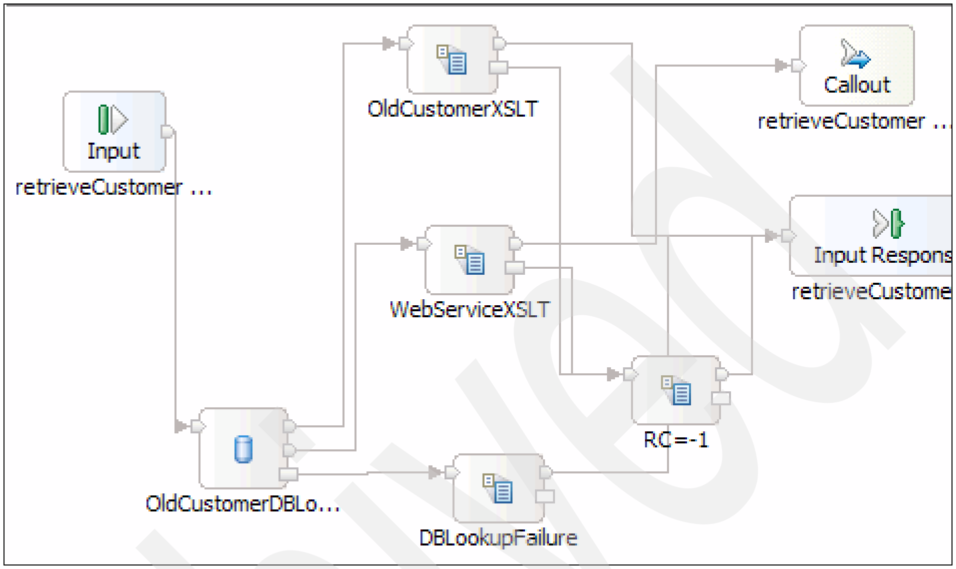


Figure 6-11 Database Lookup primitive in the message flow

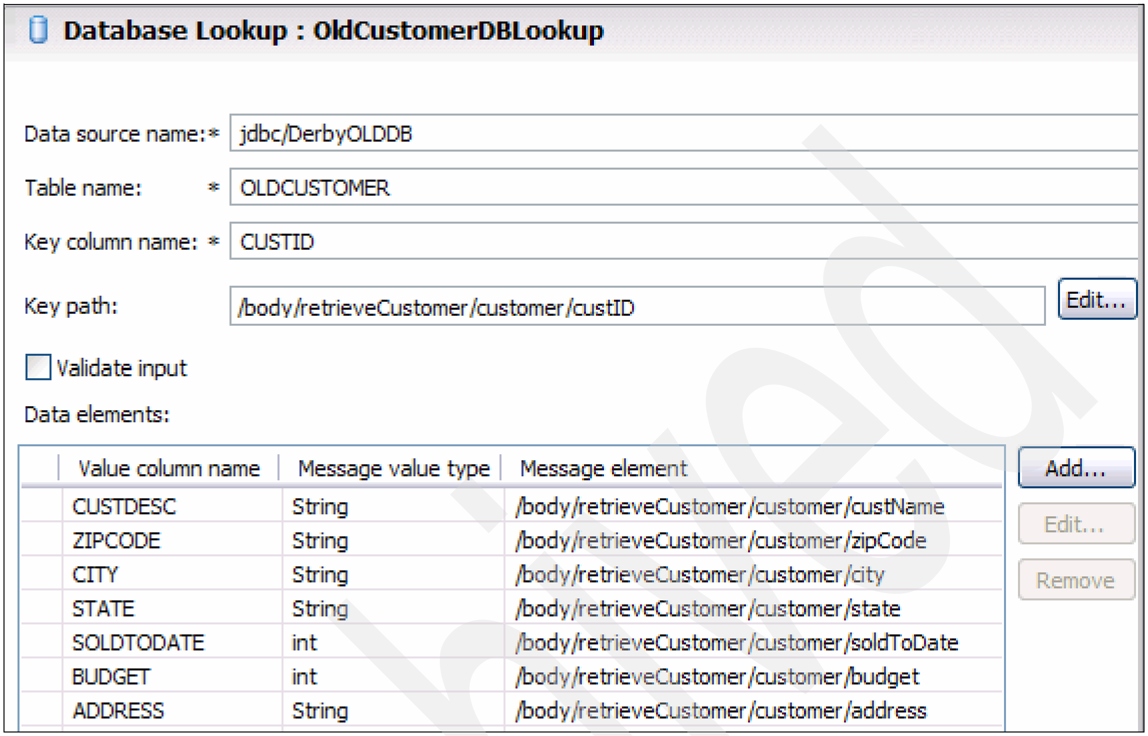
The mediation attempts to find customer data based on a key value (customer ID) in a local database. If the database lookup is successful, the information is retrieved from the local database and the result is returned. If the database lookup does not find a customer record, a Web service call retrieves the record. If an error occurs during the customer information retrieval, the return code is set to -1 and the flow is exited.

Database Lookup

The Database Lookup primitive is the first primitive that you add to the flow. To add and configure this primitive, follow these steps:

1. Add a Database Lookup primitive to the mediation flow canvas. The primitive is located under Transformation folder of Palette.
2. Rename the mediation to OldCustomerDBLookup and wire it into the flow. Connecting the wire to the in terminal sets the type of the input message for the primitive.

3. In the Details tab of Properties view set the values as shown in Figure 6-12.



Database Lookup : OldCustomerDBLookup

Data source name:* jdbc/DerbyOLDDB

Table name: * OLDCUSTOMER

Key column name: * CUSTID

Key path: /body/retrieveCustomer/customer/custID

□ Validate input

Data elements:

Value column name	Message value type	Message element
CUSTDESC	String	/body/retrieveCustomer/customer/custName
ZIPCODE	String	/body/retrieveCustomer/customer/zipCode
CITY	String	/body/retrieveCustomer/customer/city
STATE	String	/body/retrieveCustomer/customer/state
SOLDTODATE	int	/body/retrieveCustomer/customer/soldToDate
BUDGET	int	/body/retrieveCustomer/customer/budget
ADDRESS	String	/body/retrieveCustomer/customer/address

Figure 6-12 Database Lookup primitive details

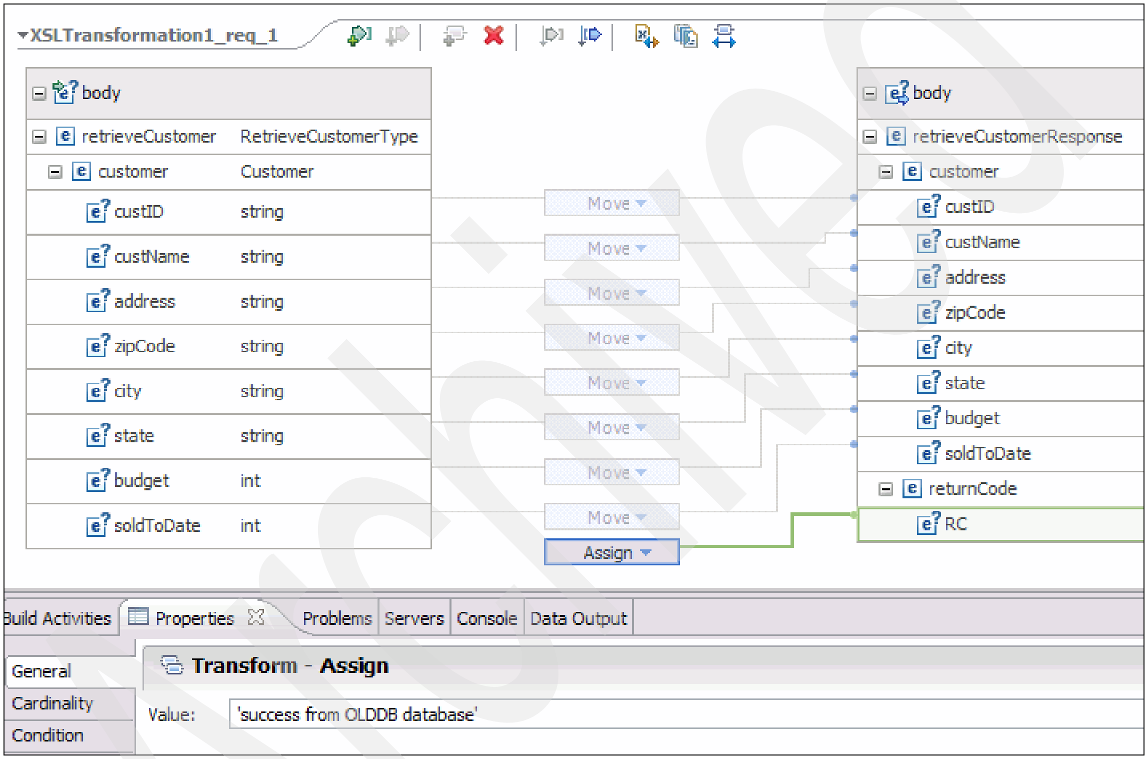
XSL Transformation primitives

Four XSL Transformation primitives are used to transform the new message to the required format, depending on the path the message takes:

1. Drag and drop an XSL Transformation primitive to the canvas, and rename it to `OldCustomerXSLT`.
2. Add another XSL Transformation primitive to the canvas, and rename it to `WebServicesXSLT`.
3. Add another XSL Transformation primitive to the canvas, and rename it to `RC=-1`.
4. Add another XSL Transformation primitive to the canvas, and rename it to `DBLookupFailure`.
5. Wire the out terminal of OldCustomerDBLookup to the in terminal of OldCustomerXSLT, the keyNotFound terminal to WebServiceXSLT, and the fail terminal to DBLookupFailure.

6. Wire the out terminal of the OldCustomerXSLT to Input Response node, and wire the fail terminal to RC-1.
7. Wire the out terminal of the WebServiceXSLT to the Callout node, and wire the fail terminal to RC-1.
8. Wire the out terminal of RC=-1 to the Input Response node.
9. Create the mappings for each XSL Transformation primitive.

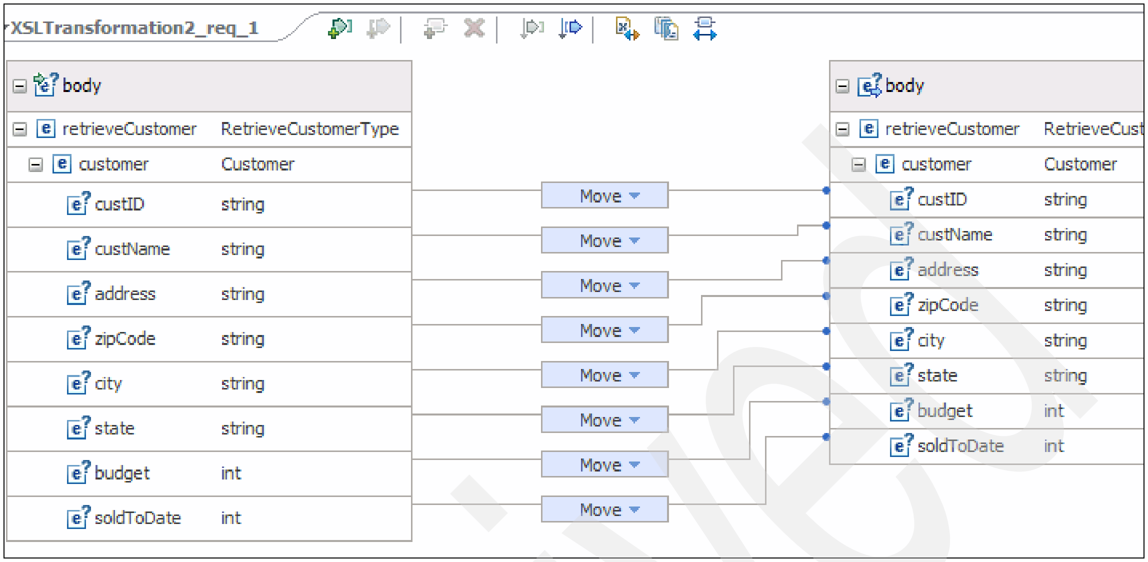
Figure 6-13 shows the mapping for OldCustomerXSLT.



Value: 'success from OLDDB database'

Figure 6-13 OldCustomerXSLT mapping

Figure 6-14 shows the mapping for WebServiceXSLT.

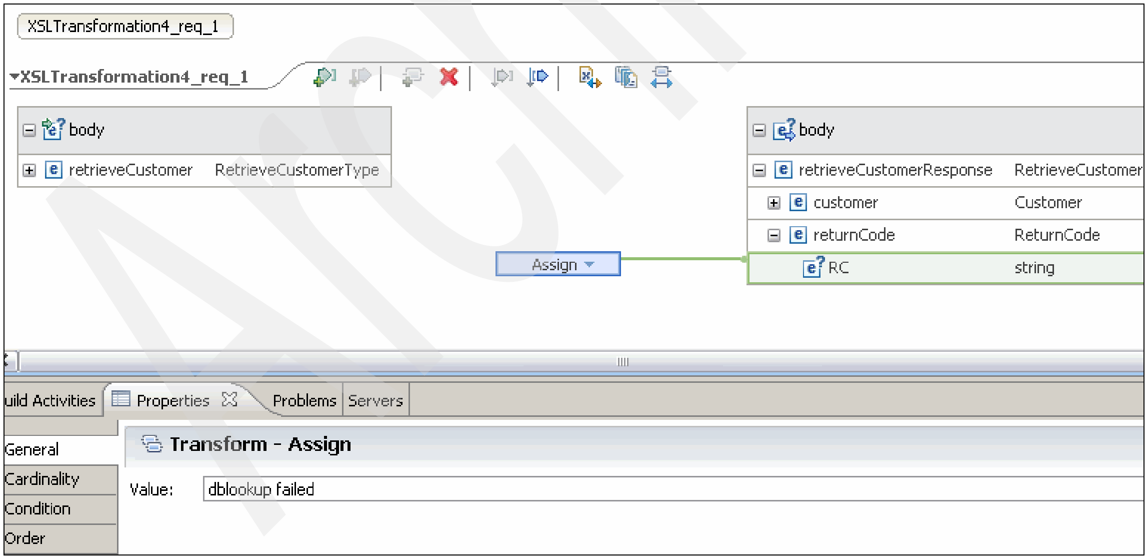

XSLTransformation2_req_1

body	
retrieveCustomer	RetrieveCustomerType
customer	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int

body	
retrieveCustomer	RetrieveCust
customer	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int

Figure 6-14 WebServiceXSLT mapping

Figure 6-15 shows the mapping for DBLookupFailure.


XSLTransformation4_req_1

XSLTransformation4_req_1

body	
retrieveCustomer	RetrieveCustomerType

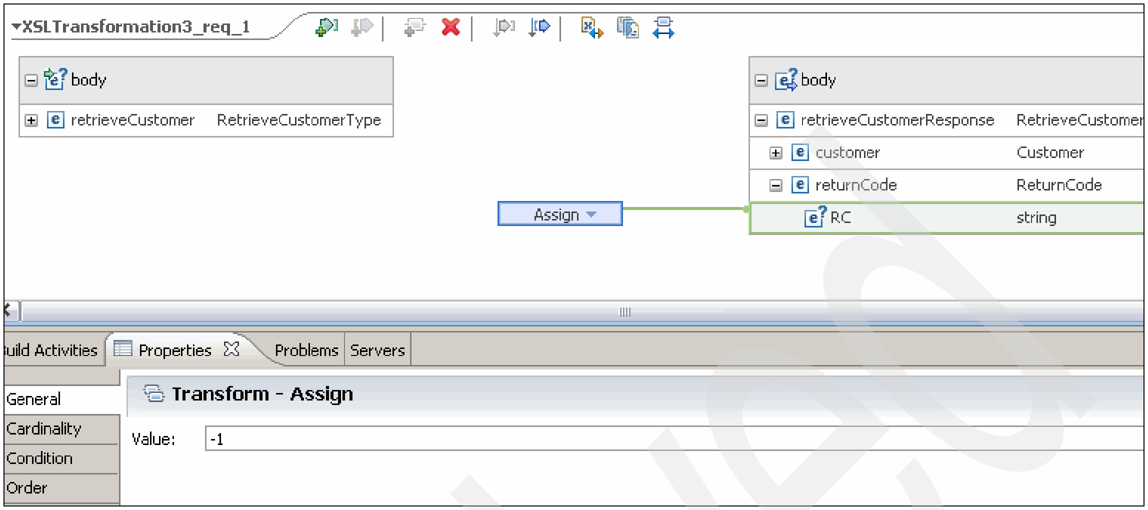
body	
retrieveCustomerResponse	RetrieveCustomer
customer	Customer
returnCode	ReturnCode
RC	string

Transform - Assign

Value: dblookup failed

Figure 6-15 DBLookupFailure mapping

Figure 6-16 shows the mapping for RC=-1.



Transform - Assign

Value: -1

Figure 6-16 Assign the return code value

6.1.6 Response flow

Build the response flow using the following steps:

1. Click the Response tab to open the response flow.
2. Add an XSL Transformation primitive. Name the new primitive `WebServiceResult`, and wire it between the Callout Response node and the Input Response node. See Figure 6-17.

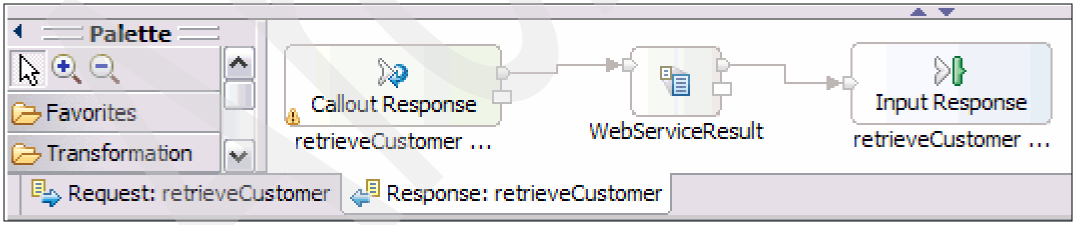
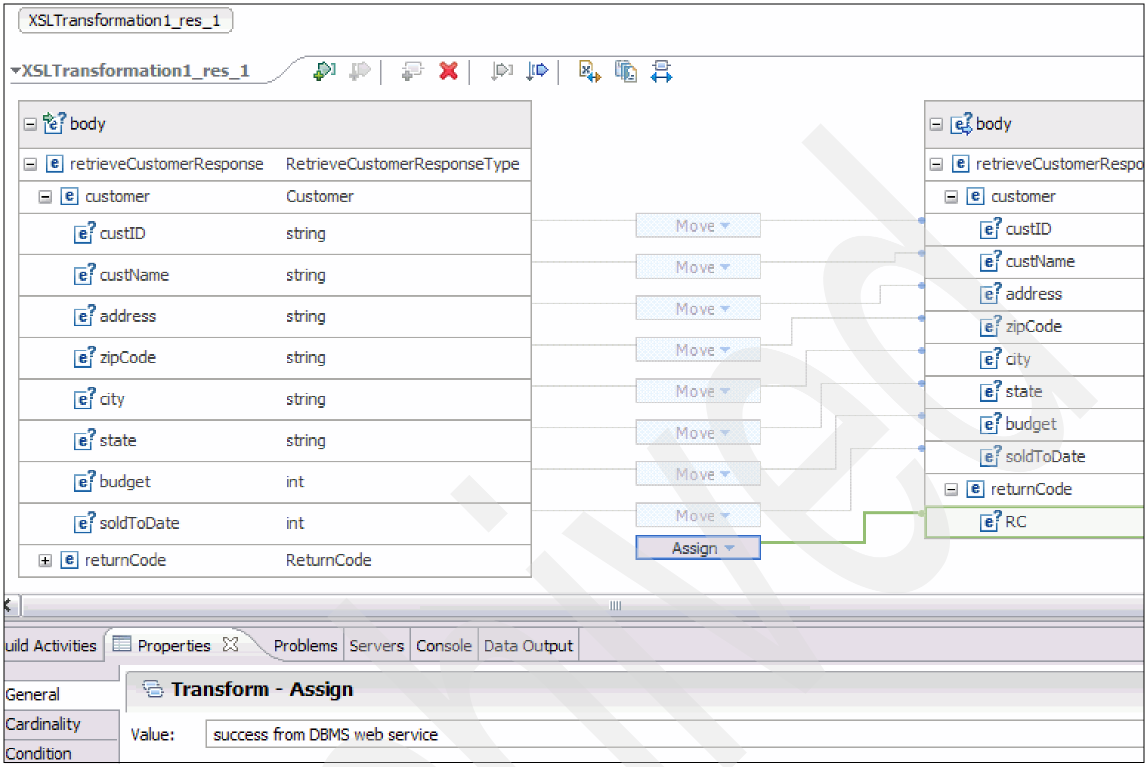


Figure 6-17 Response Flow

3. Complete the mapping for WebServiceResult as shown in Figure 6-18.



Transform - Assign

Value: success from DBMS web service

Figure 6-18 WebServiceResult mapping

4. Save the mediation flow component and the module.

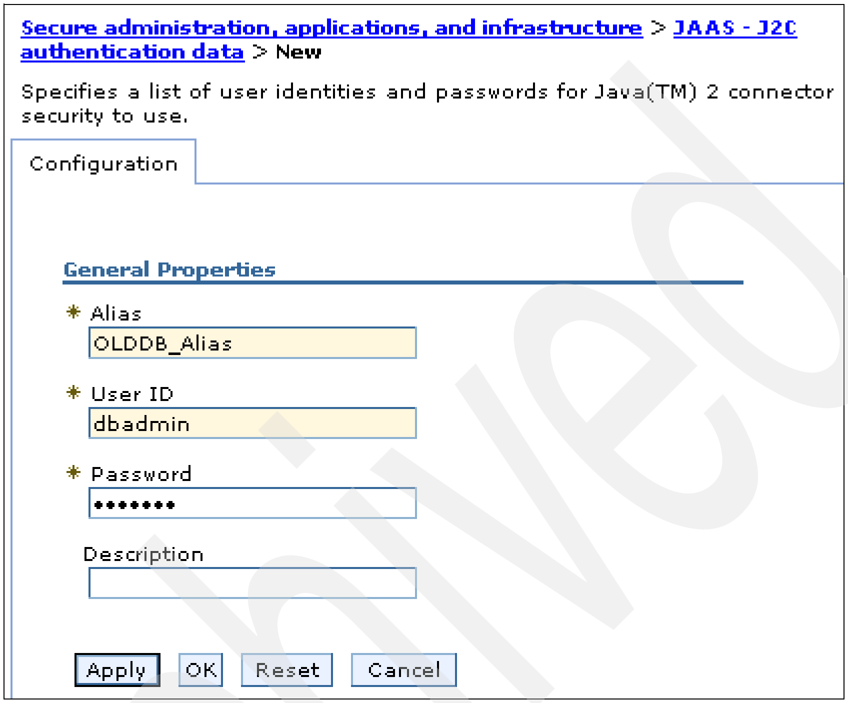
6.1.7 Preparing the run time

The properties of the Database Lookup primitive specify a data source reference to access the database. Before deploying the mediation module, you need to create a corresponding data source and a corresponding J2C authentication alias in the runtime environment.

In this case, you need to create a data source that specifies a JNDI name to match the data source name that is specified in the primitive (jdbc/DerbyOLDDB). Follow these steps:

1. In the administrative console, select **Security** → **Secure administration, applications and infrastructure** → **Java Authentication and Authorization Service** → **J2C authentication data**.

2. Click **New** to create a new authentication alias. Populate it with the user ID and password for the database (`dbadmin` for both, in this case, as shown in Figure 6-19).


Secure administration, applications, and infrastructure > JAAS - J2C authentication data > New

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

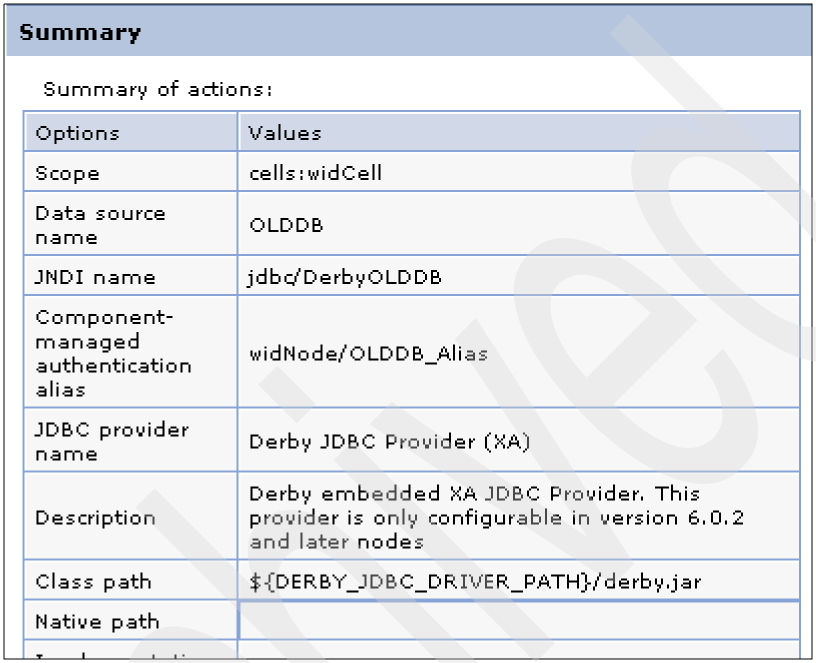
General Properties

* Alias: OLDDB_Alias
* User ID: dbadmin
* Password: •••••••

Description:

Figure 6-19 J2C authentication alias for the database

3. In the administrative console, select **Resources** → **JDBC** → **Data sources**.
4. Change the scope to the cell scope and click **New** to create a new data source. Follow the wizard using the information shown in the summary in Figure 6-20.



Summary

Summary of actions:

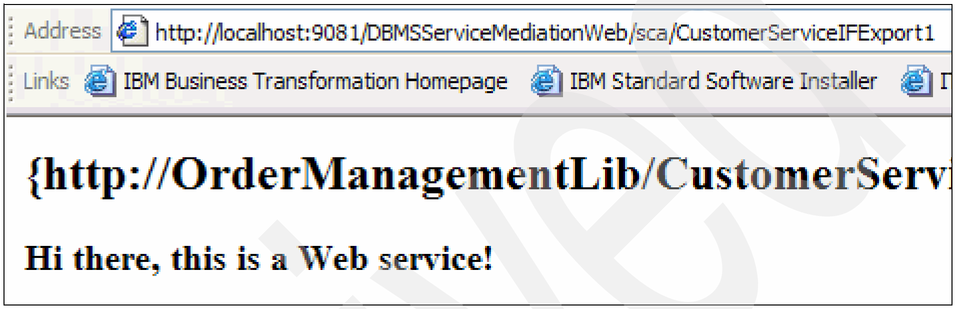
Options	Values
Scope	cells:widCell
Data source name	OLDDB
JNDI name	jdbc/DerbyOLDDB
Component-managed authentication alias	widNode/OLDDB_Alias
JDBC provider name	Derby JDBC Provider (XA)
Description	Derby embedded XA JDBC Provider. This provider is only configurable in version 6.0.2 and later nodes
Class path	${DERBY_JDBC_DRIVER_PATH}/derby.jar
Native path	

Figure 6-20 New data source

6.1.8 Test the module

Now test the module using the Integration Test Client following these steps:

1. Deploy both the TestDBLookup and the DBMSServiceMediation modules to the server and start them.
2. Use a Web browser to make sure the Web service is accessible using the URL that is specified in the binding tab for the CustomerServiceIFImport interface, as shown in Figure 6-21.


Address
Links
IBM Business Transformation Homepage
IBM Standard Software Installer

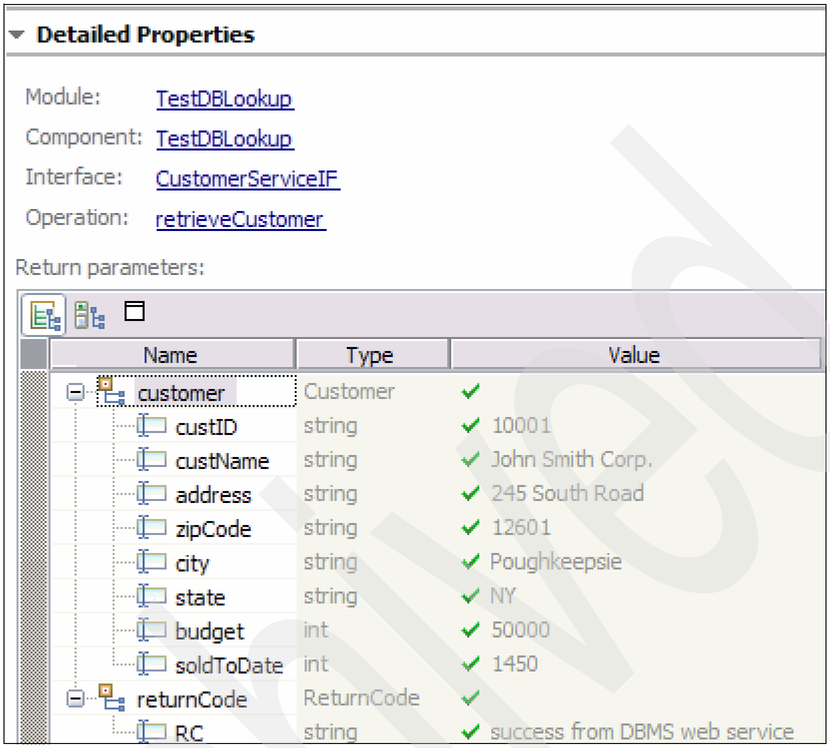
http://localhost:9081/DBMSServiceMediationWeb/sca/CustomerServiceIFExport1

{http://OrderManagementLib/CustomerServ

Hi there, this is a Web service!

Figure 6-21 Testing the Web service from a browser

3. Run the test using `10001` as the customer ID. Figure 6-22 shows the results.



Detailed Properties

Module: TestDBLookup

Component: TestDBLookup

Interface: CustomerServiceIF

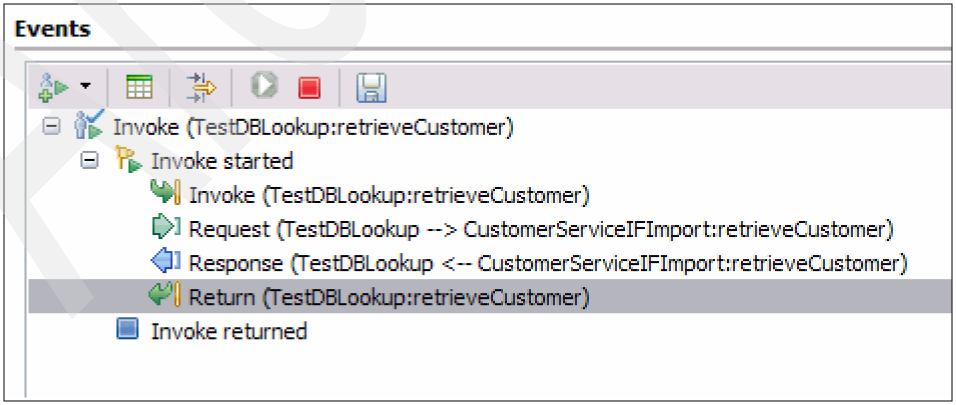
Operation: retrieveCustomer

Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10001
custName	string	✓ John Smith Corp.
address	string	✓ 245 South Road
zipCode	string	✓ 12601
city	string	✓ Poughkeepsie
state	string	✓ NY
budget	int	✓ 50000
soldToDate	int	✓ 1450
returnCode	ReturnCode	✓
RC	string	✓ success from DBMS web service

Figure 6-22 Test results

This value causes a keyNotFound condition and results in a call to the Web service, as shown in Figure 6-23.

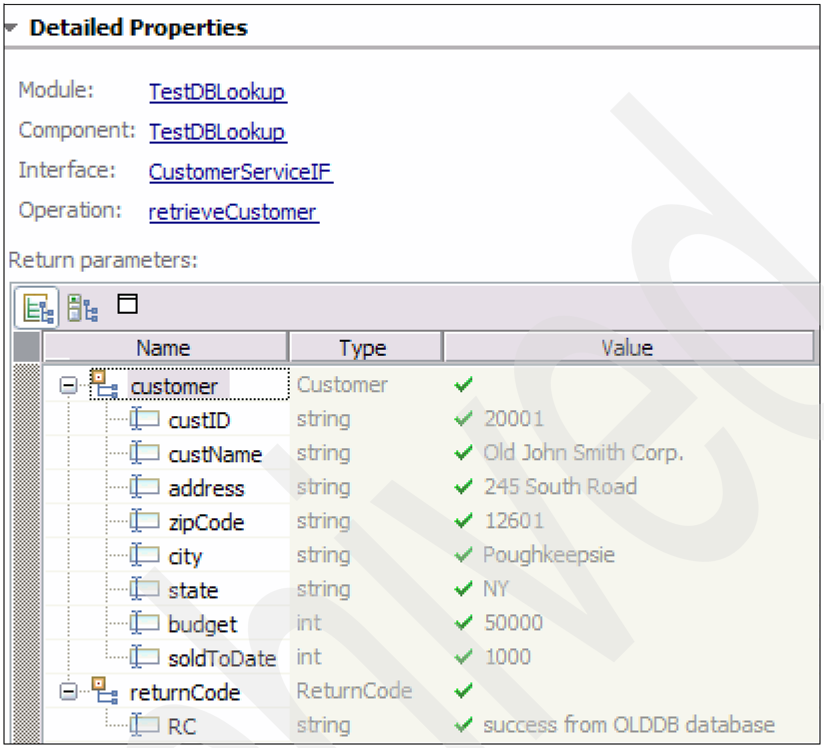


Events

- Invoke (TestDBLookup:retrieveCustomer)
  - Invoke started
    - Invoke (TestDBLookup:retrieveCustomer)
    - Request (TestDBLookup --> CustomerServiceIFImport:retrieveCustomer)
    - Response (TestDBLookup <-- CustomerServiceIFImport:retrieveCustomer)
    - Return (TestDBLookup:retrieveCustomer)
  - Invoke returned

Figure 6-23 List of events, including the Web service call

4. Repeat the test using `20001` as the customer ID. Figure 6-24 shows the results of the test.



Detailed Properties

Module: TestDBLookup

Component: TestDBLookup

Interface: CustomerServiceIF

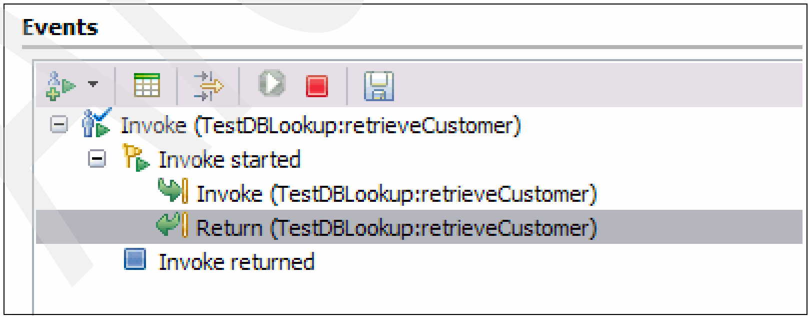
Operation: retrieveCustomer

Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 20001
custName	string	✓ Old John Smith Corp.
address	string	✓ 245 South Road
zipCode	string	✓ 12601
city	string	✓ Poughkeepsie
state	string	✓ NY
budget	int	✓ 50000
soldToDate	int	✓ 1000
returnCode	ReturnCode	✓
RC	string	✓ success from OLDDB database

Figure 6-24 Testing DatabaseLookupSample1Module

This time the customer is found in the OLDDB database. Note that the sequence of calls differs from the first test (Figure 6-25).

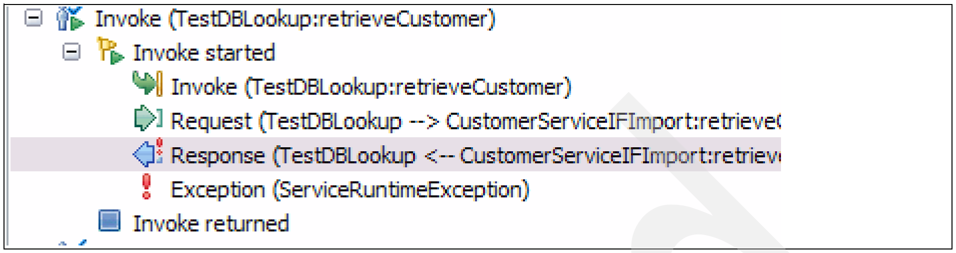


Events

- Invoke (TestDBLookup:retrieveCustomer)
  - Invoke started
    - Invoke (TestDBLookup:retrieveCustomer)
    - Return (TestDBLookup:retrieveCustomer)
  - Invoke returned

Figure 6-25 List of events with no Web service call

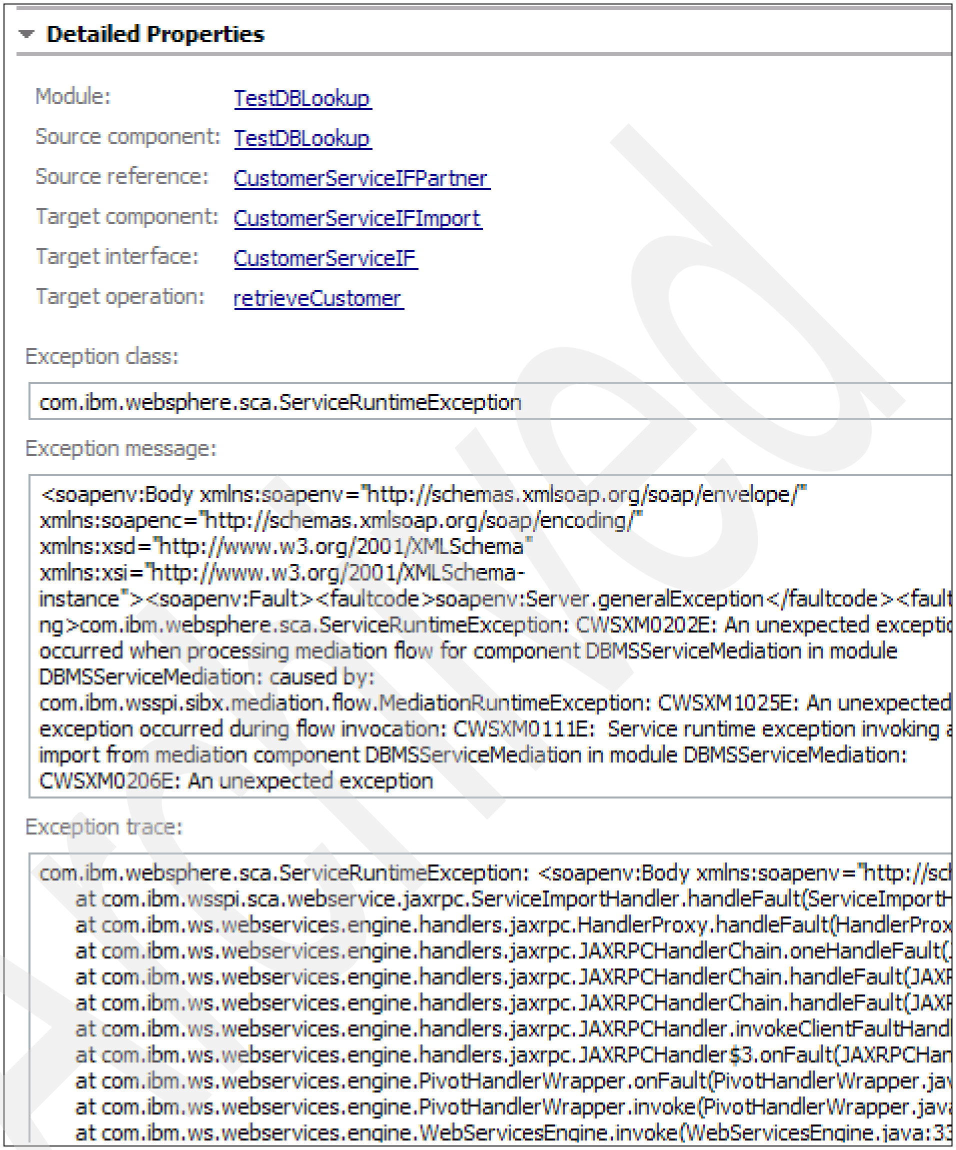
5. Repeat the test a third time using 30001 as a customer ID. This time, you see an exception thrown by the Web service, as shown in Figure 6-26.



Invoke (TestDBLookup:retrieveCustomer)
Invoke started
Invoke (TestDBLookup:retrieveCustomer)
Request (TestDBLookup --> CustomerServiceIFImport:retrieve(
Response (TestDBLookup <-- CustomerServiceIFImport:retriev(
Exception (ServiceRuntimeException)
Invoke returned

Figure 6-26 List of events, including an exception thrown by the Web service

The exception displays in the component test window (Figure 6-27).



Module: TestDBLookup

Source component: TestDBLookup

Source reference: CustomerServiceIFPartner

Target component: CustomerServiceIFImport

Target interface: CustomerServiceIF

Target operation: retrieveCustomer

Exception class:

com.ibm.websphere.sca.ServiceRuntimeException

Exception message:

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><soapenv:Fault><faultcode>soapenv:Server.generalException</faultcode><fault
ng>com.ibm.websphere.sca.ServiceRuntimeException: CWSXM0202E: An unexpected exceptio
occurred when processing mediation flow for component DBMSServiceMediation in module
DBMSServiceMediation: caused by:
com.ibm.wsspi.sibx.mediation.flow.MediationRuntimeException: CWSXM1025E: An unexpected
exception occurred during flow invocation: CWSXM0111E:  Service runtime exception invoking a
import from mediation component DBMSServiceMediation in module DBMSServiceMediation:
CWSXM0206E: An unexpected exception
```

Exception trace:

```
com.ibm.websphere.sca.ServiceRuntimeException: <soapenv:Body xmlns:soapenv="http://sc
    at com.ibm.wsspi.sca.webservice.jaxrpc.ServiceImportHandler.handleFault(ServiceImportH
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.HandlerProxy.handleFault(HandlerProx
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.JAXRPCHandlerChain.oneHandleFault(
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.JAXRPCHandlerChain.handleFault(JAXR
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.JAXRPCHandlerChain.handleFault(JAXR
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.JAXRPCHandler.invokeClientFaultHand
    at com.ibm.ws.webservices.engine.handlers.jaxrpc.JAXRPCHandler$3.onFault(JAXRPCHan
    at com.ibm.ws.webservices.engine.PivotHandlerWrapper.onFault(PivotHandlerWrapper.jav
    at com.ibm.ws.webservices.engine.PivotHandlerWrapper.invoke(PivotHandlerWrapper.java
    at com.ibm.ws.webservices.engine.WebServicesEngine.invoke(WebServicesEngine.java:33
```

Figure 6-27 Exception data

6. When the testing is complete remove the projects from the server.

6.2 Message Element Setter example

Additional material: This example is included in the MessageElementSetterMediation project in the additional materials.

This mediation illustrates the use of the Message Element Setter primitive.

6.2.1 Build the assembly diagram

To build the assembly diagram, follow these steps:

1. Create the MessageElementSetterMediaiton mediation module, and add OrderManagementLib to its dependencies.
2. Import CustomerServiceIFExport1_CustomerServiceIFHttpService.
3. Add CustomerServiceIF interface, and wire MessageElementSetterMediation to CustomerServiceIFImport1 as shown in Figure 6-28.

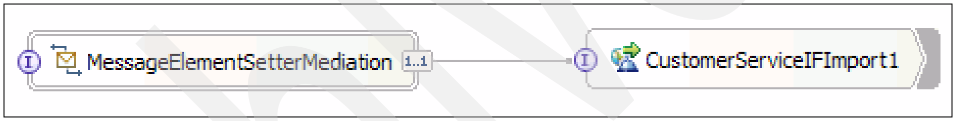
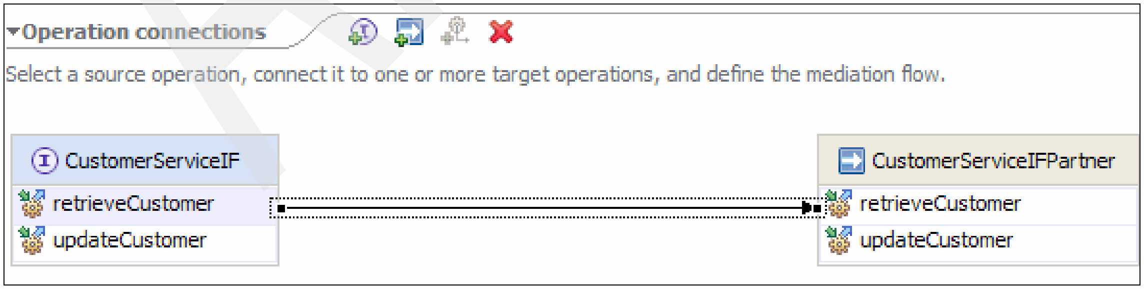


Figure 6-28 MessageElementSetterMediation

4. Right-click **MessageElementSetterMediation**, and select **Regenerate Implementation** from the pop-up menu to open a Mediation Flow Editor.

6.2.2 Operation connections

In the Operation connections section, wire the retrieveCustomer operations (Figure 6-29). Select the connection to open the mediation flow.



Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 6-29 Operation connections

6.2.3 Build the request flow

To use the Message Element Setter primitive:

1. Locate the primitive in the palette under the Transformation folder.
2. Drop it on the canvas, and name it *CustomerIdSetter*. Wire the in and out terminals for the primitive. See Figure 6-30.

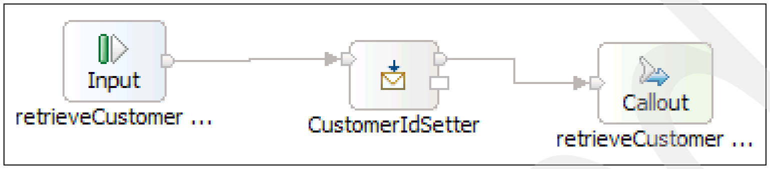
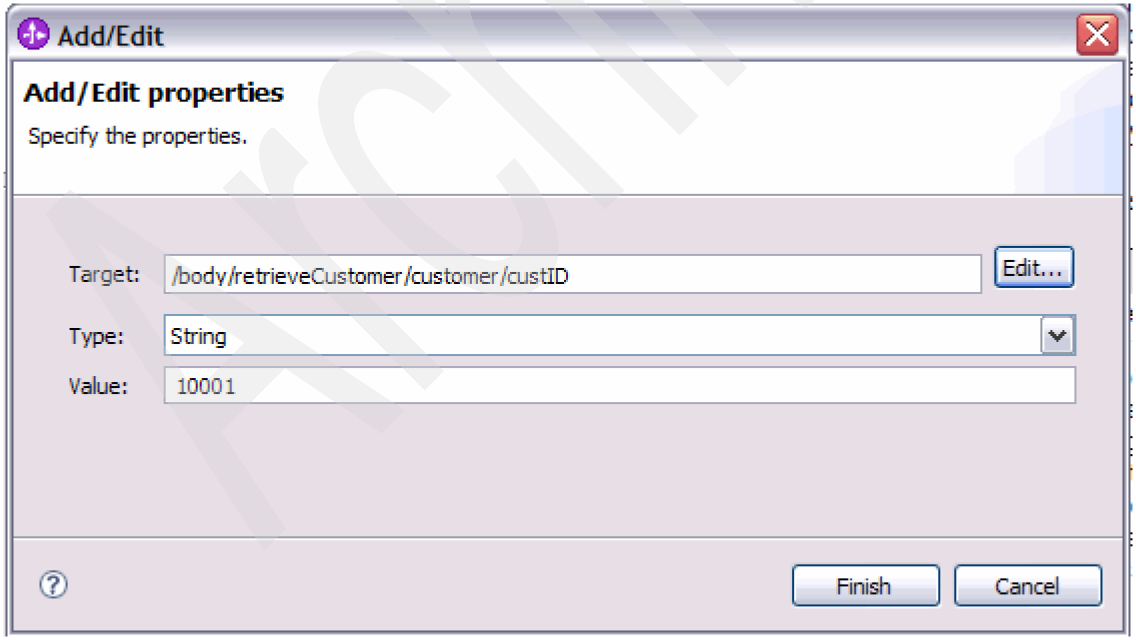


Figure 6-30 Message Element Setter in the mediation flow

3. Select the primitive and open the Details tab in the Properties view.
4. Click **Add** to define a message element.

   You can enter the Target, Type, and Value fields, or you can click **Edit** to open the XPath expression builder. The options shown in Figure 6-31 assign a string value of `10001` to the /body/retrieveCustomer/customer/custID element. Click **Finish**.



Add/Edit properties

Specify the properties.

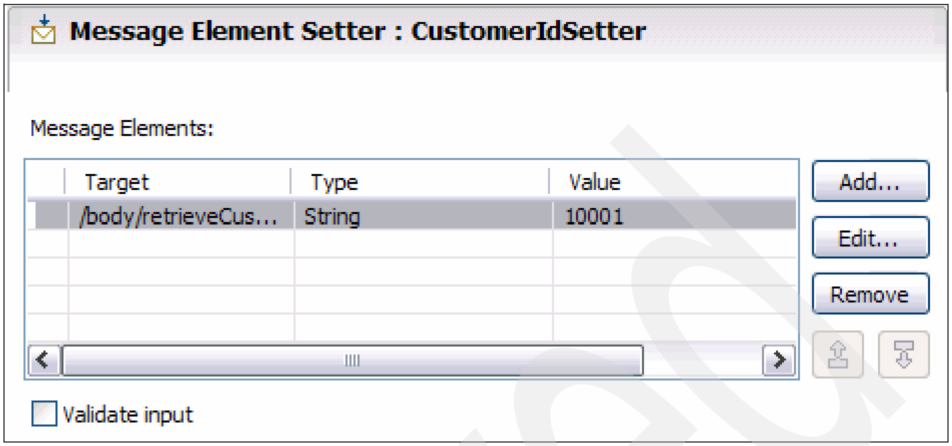
Target: /body/retrieveCustomer/customer/custID

Type: String

Value: 10001

Figure 6-31 Message Element Setter properties

5. Repeat the process for each target element that you want to set (Figure 6-32).



Message Element Setter : CustomerIdSetter

Message Elements:

Target	Type	Value
/body/retrieveCus...	String	10001

□ Validate input

Figure 6-32 Elements to be set

6.2.4 Build the response flow

To build the response flow, follow these steps:

1. Switch to the Response:retrieveCustomer tab.
2. Add a new Message Element Setter primitive to the canvas. Name it `ReturnValueSetter`.
3. Wire the new primitive between the Callout Response and Input Response nodes, as shown in Figure 6-33.

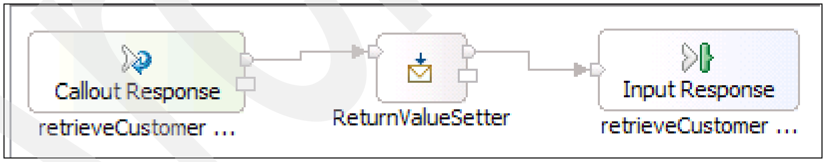
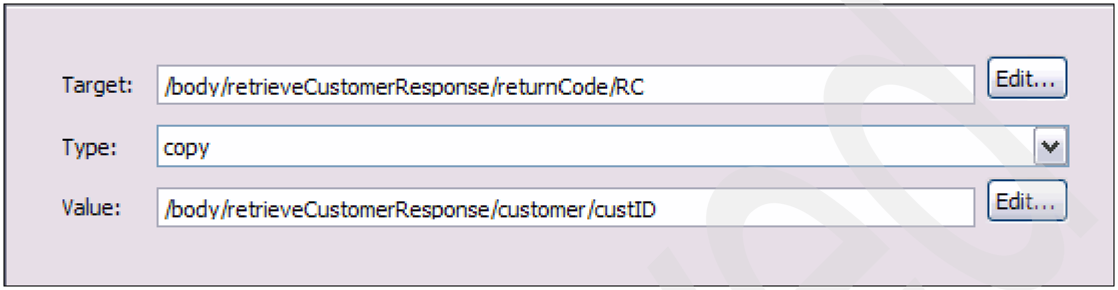


Figure 6-33 Response flow

4. Switch to the Details tab of the ReturnValueSetter Properties. Then:
   a. Set Target to /body/retrieveCustomerResponse/returnCode/RC.
   b. Select **copy** for the Type field.
   c. Set /body/retrieveCustomerResponse/customer/custID for the Value field.
   d. Click **Finish**.



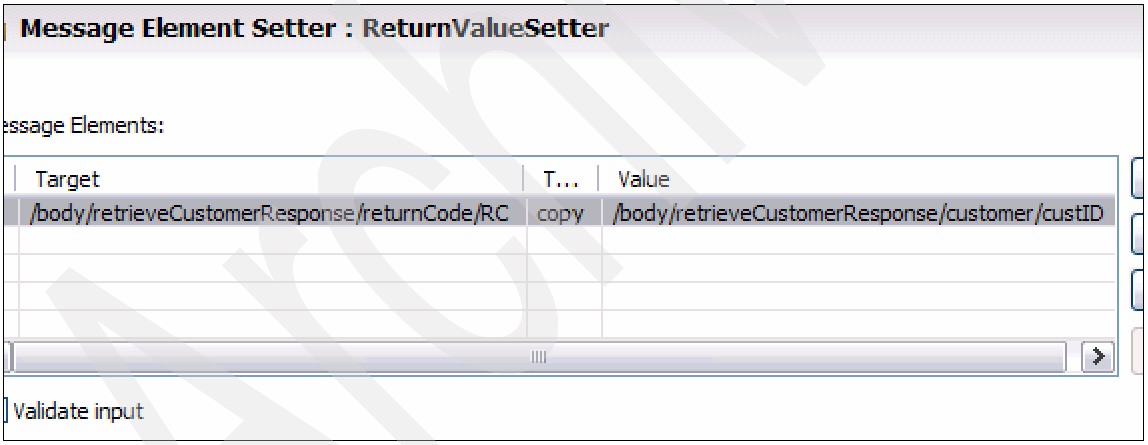
Target: /body/retrieveCustomerResponse/returnCode/RC

Type: copy

Value: /body/retrieveCustomerResponse/customer/custID

Figure 6-34 Message Element Setter properties

The properties that you set display as shown in Figure 6-35.



Message Element Setter : ReturnValueSetter

essage Elements:

Target	T...	Value
/body/retrieveCustomerResponse/returnCode/RC	copy	/body/retrieveCustomerResponse/customer/custID

Validate input

Figure 6-35 Message Element Setter properties

5. Save all the changes.

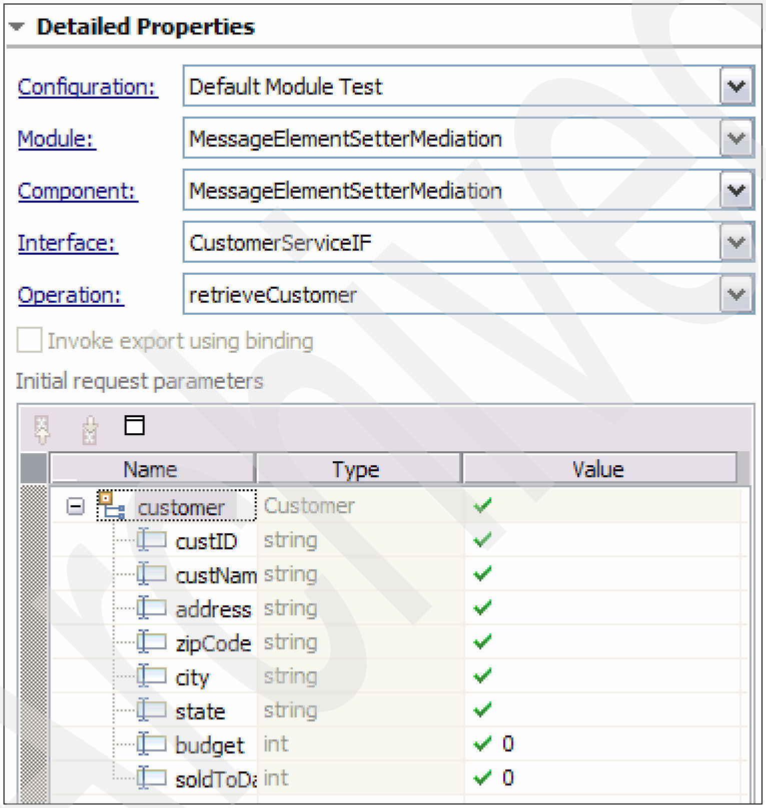
6.2.5 Test the module

To test the module:

1. Switch to assembly diagram. Right-click the diagram and select **Test Module** to open the Detailed Properties window (Figure 6-36).

   You do not need to set any values in this window. The Set Message Element primitive sets these values in the request flow.

2. Click **Continue**.



Detailed Properties

Configuration: Default Module Test

Module: MessageElementSetterMediation

Component: MessageElementSetterMediation

Interface: CustomerServiceIF

Operation: retrieveCustomer

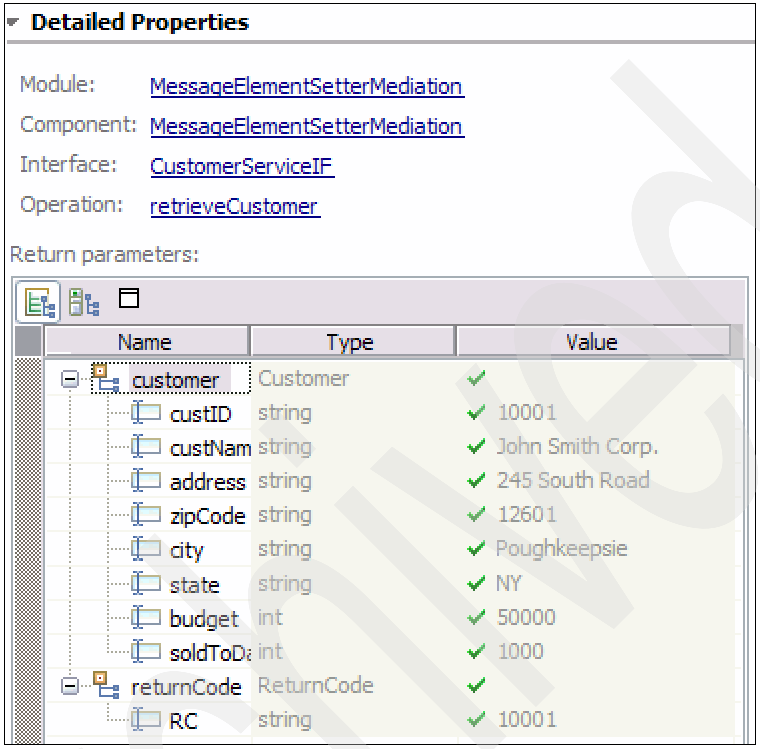
□ Invoke export using binding

Initial request parameters

Name	Type	Value
customer	Customer	✓
custID	string	✓
custNam	string	✓
address	string	✓
zipCode	string	✓
city	string	✓
state	string	✓
budget	int	✓ 0
soldToD	int	✓ 0

Figure 6-36 Test properties

3. During the test run, the mediation flow retrieves the record for customer ID 10001, and then sets RC to the `custID`, which in our case is `10001` as shown in Figure 6-37.



Detailed Properties

Module: MessageElementSetterMediation

Component: MessageElementSetterMediation

Interface: CustomerServiceIF

Operation: retrieveCustomer

Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10001
custNam	string	✓ John Smith Corp.
address	string	✓ 245 South Road
zipCode	string	✓ 12601
city	string	✓ Poughkeepsie
state	string	✓ NY
budget	int	✓ 50000
soldToDa	int	✓ 1000
returnCode	ReturnCode	✓
RC	string	✓ 10001

Figure 6-37 Test results

4. When testing is complete, remove the projects from the server.

6.3 Set Message Type example

> **Additional material:** This example is included in the SetMessageTypeMediation, ExternalCustomerLib, and ExternalCustomerInfo projects in the additional materials.

This example uses two Web services that can update customer information:

- *CustomerServiceIF* is an internal service that retrieves and updates customer information for customer IDs that start with *1*. This Web service is an existing service in OrderManagementLib.
- *ExternalCustomerIF* is an external service that processes all other customer IDs. This Web service and its interface are built in this example.

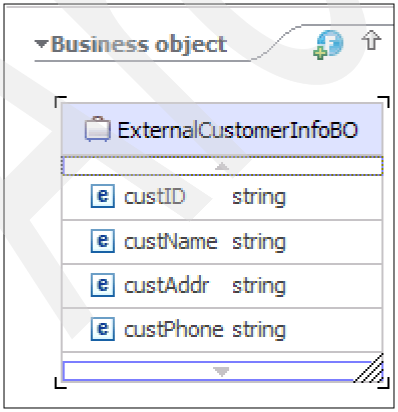
The Set Message Type mediation is used to cast the input object to the object type that is required by the service that is called.

6.3.1 Build the ExternalCustomerLib library

The ExternalCustomerLib library contains the Web service business objects and interfaces so that the WSDL files can be shared easily with clients.

To build the ExternalCustomerLib library:

1. Build a new library and name it `ExternalCustomerLib`.
2. Create the ExternalCustomerInfoBO business object in the library with the fields shown in Figure 6-38.

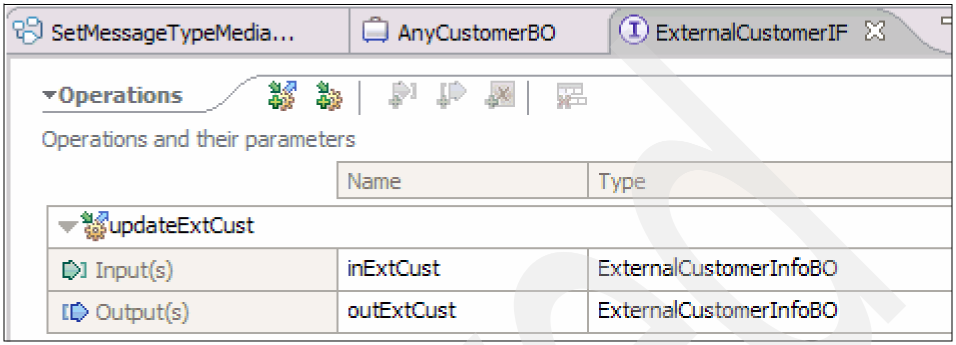


ExternalCustomerInfoBO

custID	string
custName	string
custAddr	string
custPhone	string

Figure 6-38 ExternalCustomerInfoBO business object

3. Build the ExternalCustomerIF interface with one updateExtCust operation that takes the inExtCust object for input and the outExtCust for output. Both objects are of type ExternalCustomerInfoBO. The interface is shown in Figure 6-39.



	Name	Type
updateExtCust		
Input(s)	inExtCust	ExternalCustomerInfoBO
Output(s)	outExtCust	ExternalCustomerInfoBO

Figure 6-39 ExternalCustomerIF interface

4. Save and close the library.

6.3.2 Build the ExternalCustomerInfo Web service

Now, you can create an ExternalCustomerInfo Web service. This Web service is actually a dummy mediation module that acts as the Web service and simply returns the input data.

To build the ExternalCustomerInfo Web service:

1. Select **File** → **New** → **Mediation Module** to build the new module.
   a. Name it `ExternalCustomerInfo`.
   b. Select **WebSphere ESB Server v6.1** for the Target run time.
   c. Add ExternalCustomerInfoLib as a required library.
   d. Click **Finish** to open the assembly diagram with the ExternalCustomerInfo component.
2. Add ExternalCustomerIF to the ExternalCustomerInfo component.
3. Select **Regenerate Implementation** from the component's pop-up menu to open the Mediation Flow Editor.

Build the request flow

In the mediation flow editor:

1. Drop an XSL Transformation primitive to the canvas and name it `XSLTexternalCust`.
2. Wire the Input node to the XSLTexternalCust input terminal, and write the Input Response node to the XSLTexternalCust output terminal as shown in Figure 6-40.

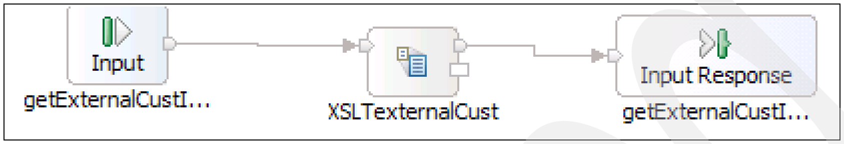


Figure 6-40 Mediation flow

3. Double-click the XSLTexternalCust to create the mapping. Move each field on the left to the corresponding field on the right to effectively configure the module to return the input data. See Figure 6-41.

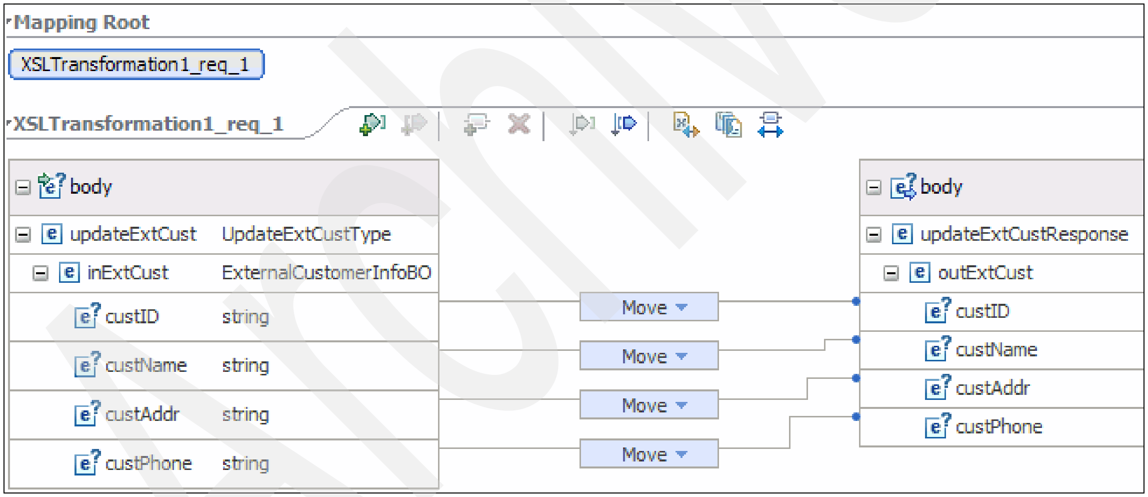


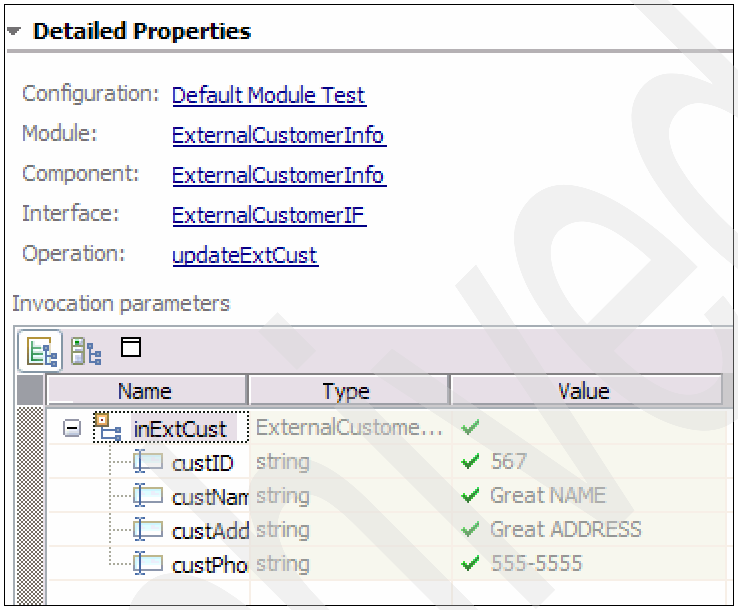
Figure 6-41 XSLTexternalCust mapping

4. Save the changes and close all open windows.

Test the module

Test the module to ensure that it returns the input values properly. Follow these steps:

1. Right-click in the assembly diagram and select **Test Module**.
2. Enter values for the test data (examples shown in Figure 6-42).



Detailed Properties

Configuration: Default Module Test

Module: ExternalCustomerInfo

Component: ExternalCustomerInfo

Interface: ExternalCustomerIF

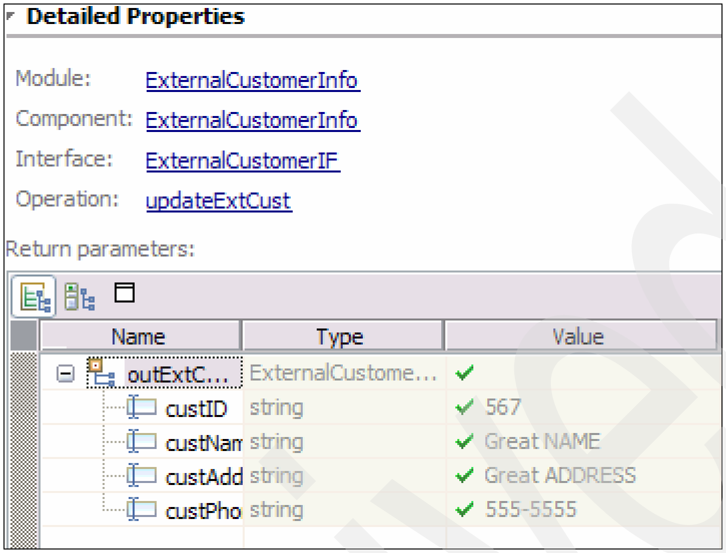
Operation: updateExtCust

Invocation parameters

Name	Type	Value
inExtCust	ExternalCustome...	✓
custID	string	✓ 567
custNam	string	✓ Great NAME
custAdd	string	✓ Great ADDRESS
custPho	string	✓ 555-5555

Figure 6-42 ExternalCustomerInfo Input

3. The results of the test should be the same data that you entered (as shown in Figure 6-43).



Detailed Properties

Module: ExternalCustomerInfo

Component: ExternalCustomerInfo

Interface: ExternalCustomerIF

Operation: updateExtCust

Return parameters:

Name	Type	Value
outExtC...	ExternalCustome...	✓
custID	string	✓ 567
custNam	string	✓ Great NAME
custAdd	string	✓ Great ADDRESS
custPho	string	✓ 555-5555

Figure 6-43 ExternalCustomerInfo Output

Export the module as a Web service

Export the module as a Web service:

1. Right-click the ExternalCustomerInfo component in the assembly diagram and select **Generate Export** → **Web Service Binding** from the pop-up menu to add the export to the assembly diagram, as shown in Figure 6-44.

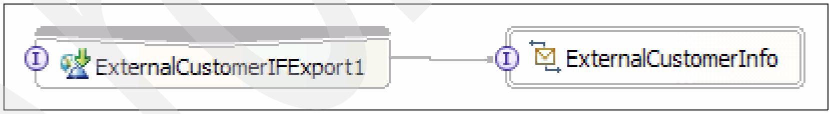
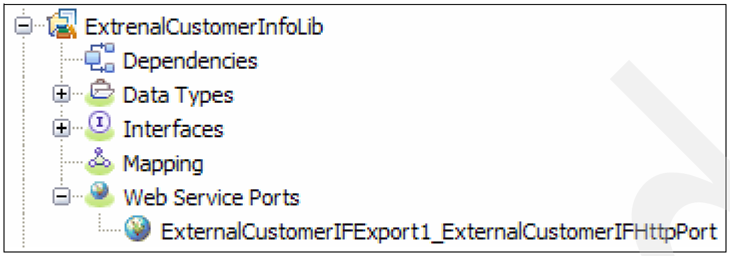


Figure 6-44 Add an export with a Web service binding

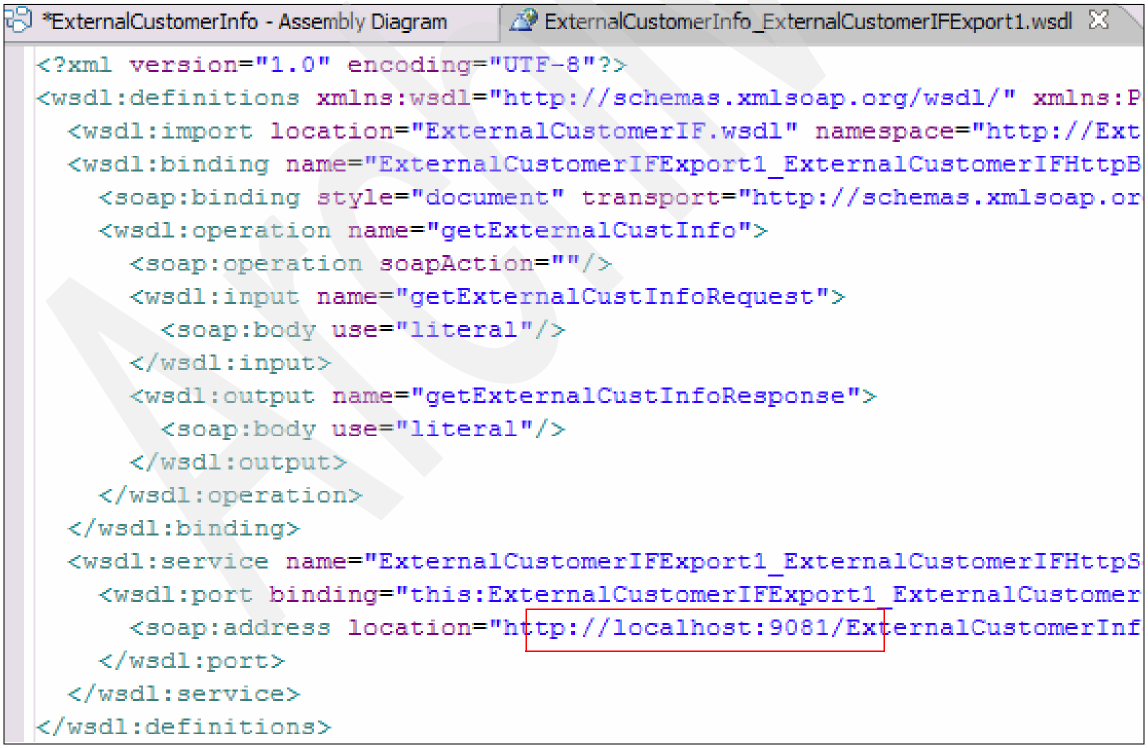
Notice, that an ExternalCustomerIFExport1_ExternalCustomerIFHttpPort is created and placed under Web Service Ports of ExternalCustomerLib, as shown in Figure 6-45.



ExtrenalCustomerInfoLib
- Dependencies
- Data Types
- Interfaces
- Mapping
- Web Service Ports
  - ExternalCustomerIFExport1_ExternalCustomerIFHttpPort

Figure 6-45 New Web service port definition

2. Double-click **ExternalCustomerIFExport1_ExternalCustomerIFHttpPort** to open a plain text WSDL file.
3. Note that the port number is 9080. Change this value to the port number for your test server. In this example, change it to 9081, as shown in Figure 6-46. Then, save the file.

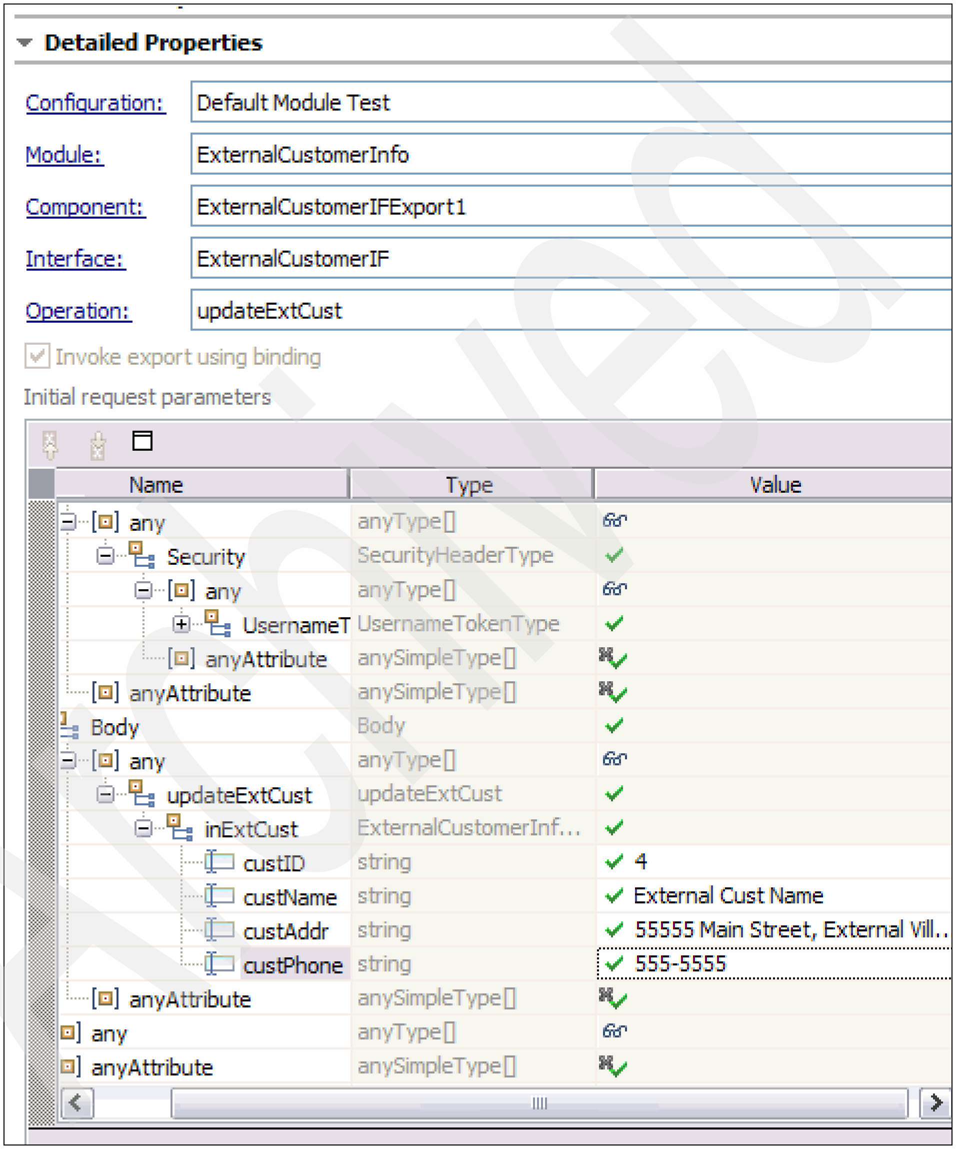


*ExternalCustomerInfo - Assembly Diagram
ExternalCustomerInfo_ExternalCustomerIFExport1.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:P
  <wsdl:import location="ExternalCustomerIF.wsdl" namespace="http://Ext
  <wsdl:binding name="ExternalCustomerIFExport1_ExternalCustomerIFHttpB
    <soap:binding style="document" transport="http://schemas.xmlsoap.or
    <wsdl:operation name="getExternalCustInfo">
      <soap:operation soapAction=""/>
      <wsdl:input name="getExternalCustInfoRequest">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getExternalCustInfoResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ExternalCustomerIFExport1_ExternalCustomerIFHttpS
    <wsdl:port binding="this:ExternalCustomerIFExport1_ExternalCustomer
      <soap:address location="http://localhost:9081/ExternalCustomerInf
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 6-46 Change the port in the WSDL

4. To test the export, right-click **ExternalCustomerIFExport1** and select **Test Component** from the pop-up menu.
5. Enter any custID, custName, custAddr, and custPhone as shown in Figure 6-47.



Detailed Properties

Configuration: Default Module Test

Module: ExternalCustomerInfo

Component: ExternalCustomerIFExport1

Interface: ExternalCustomerIF

Operation: updateExtCust

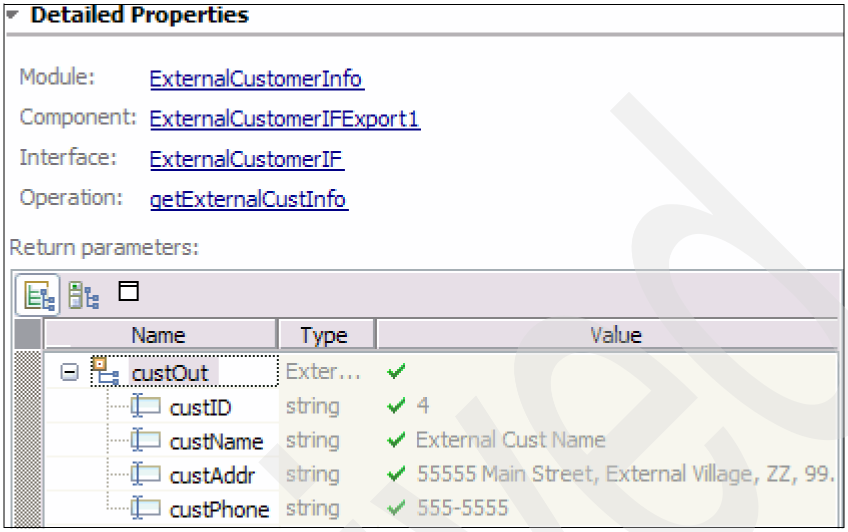
☑ Invoke export using binding

Initial request parameters

Name	Type	Value
any	anyType[]	
Security	SecurityHeaderType	✓
any	anyType[]	
UsernameT	UsernameTokenType	✓
anyAttribute	anySimpleType[]	✓
anyAttribute	anySimpleType[]	✓
Body	Body	✓
any	anyType[]	
updateExtCust	updateExtCust	✓
inExtCust	ExternalCustomerInf...	✓
custID	string	✓ 4
custName	string	✓ External Cust Name
custAddr	string	✓ 55555 Main Street, External Vill..
custPhone	string	✓ 555-5555
anyAttribute	anySimpleType[]	✓
any	anyType[]	
anyAttribute	anySimpleType[]	✓

Figure 6-47 Test the export

6. Click **Continue** and review the output (Figure 6-48) to ensure that the Web service export works as expected.



Detailed Properties

Module: ExternalCustomerInfo

Component: ExternalCustomerIFExport1

Interface: ExternalCustomerIF

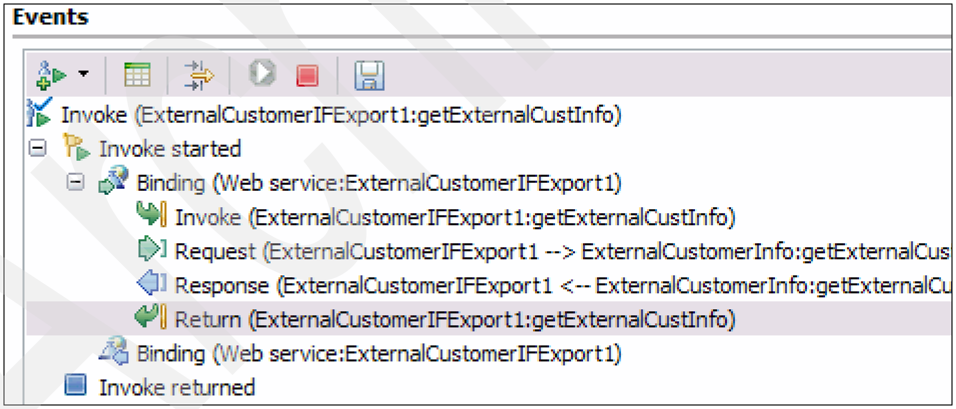
Operation: getExternalCustInfo

Return parameters:

Name	Type	Value
custOut	Exter...	✓
custID	string	✓ 4
custName	string	✓ External Cust Name
custAddr	string	✓ 55555 Main Street, External Village, ZZ, 99.
custPhone	string	✓ 555-5555

Figure 6-48 Export test results

Figure 6-49 shows the sequence of events.



Events

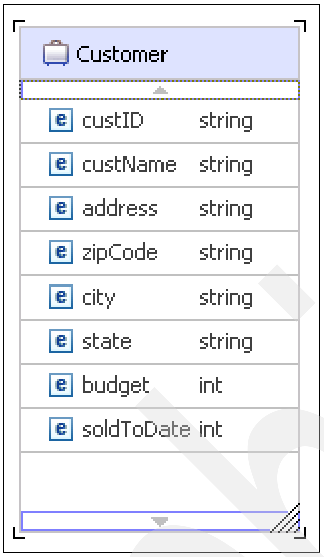
```
Invoke (ExternalCustomerIFExport1:getExternalCustInfo)
⊟ Invoke started
   ⊟ Binding (Web service:ExternalCustomerIFExport1)
        Invoke (ExternalCustomerIFExport1:getExternalCustInfo)
        Request (ExternalCustomerIFExport1 --> ExternalCustomerInfo:getExternalCus
        Response (ExternalCustomerIFExport1 <-- ExternalCustomerInfo:getExternalCu
        Return (ExternalCustomerIFExport1:getExternalCustInfo)
     Binding (Web service:ExternalCustomerIFExport1)
  Invoke returned
```

Figure 6-49 Test sequence

6.3.3 Build SetMessageTypeMediation module

The mediation flow sends a message to one of two Web services.

- *CustomerService*: An Internal Customer Service that you can use to retrieve and update customer information for a customer with customer ID starting with *1*. The interface, CustomerServiceIF, requires a *Customer object* (Figure 6-50) to be passed to updateCustomer operation.



Customer

custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int

Figure 6-50 Customer business object

- *ExternalCustomerInfo*: For all other customer ID’s, you have to call an external customer service that has a different interface.

  The interface, ExternalCustomerIF, has one operation called *updateExtCust*. This operation takes a inExtCust object for input and a outExtCust for output. Both objects are of type ExternalCustomerInfoBO, as shown in Figure 6-51.

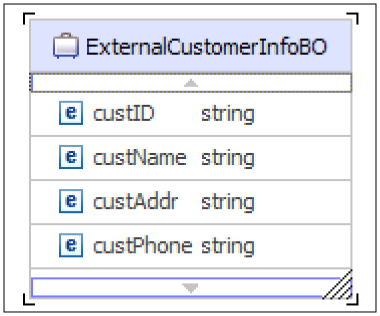


Figure 6-51 ExternalCustomerInfoBO

To start building the mediation:

1. Create a new mediation module called `SetMessageTypeMediation`.
2. Add both OrderManagementLib and ExternalCustomerInfoLib as required libraries.

6.3.4 Create an anyType business object and the interface

The next step is to build a business object and interface that will work with both Web services. Follow these steps:

1. Create a business object AnyCustomerBO with one weakly-typed field, customerData, of anyType.

   To change the type to anyType, click the field’s type to open a window that displays data types. Click **Browse** to look for and choose **anyType**. Click **OK**.

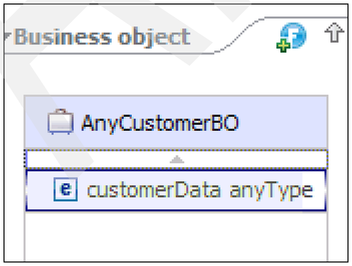
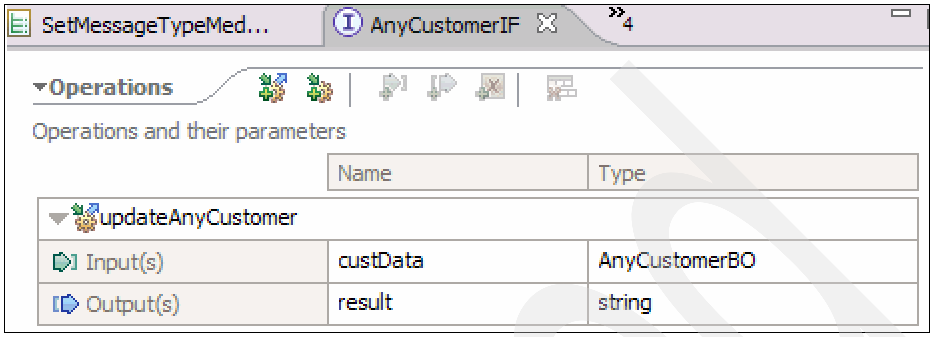


Figure 6-52 Weakly-typed message field

2. Build the interface to use the new business object. Figure 6-53 shows the AnyCustomerIF interface, which has one operation, updateAnyCustomer. The input parameter of this operation, custData, is of type AnyCustomerBO.


SetMessageTypeMed... | AnyCustomerIF | »4

Operations

Operations and their parameters

	Name	Type
updateAnyCustomer		
Input(s)	custData	AnyCustomerBO
Output(s)	result	string

Figure 6-53 Interface

3. Locate the Web service ports in the Business Integration view. Select the port for each of the following Web services and drag and drop it to the assembly diagram:

   – CustomerServiceIFExport1_CustomerServiceIFHttpPort
   – ExternalCustomerIFExport1_ExternalCustomerIFHttpService

   Leave the default names for imports.

4. Wire both imports to the SetMessageTypeMediation component.

5. Add AnyCustomerIF to the SetMessageTypeMediation component.

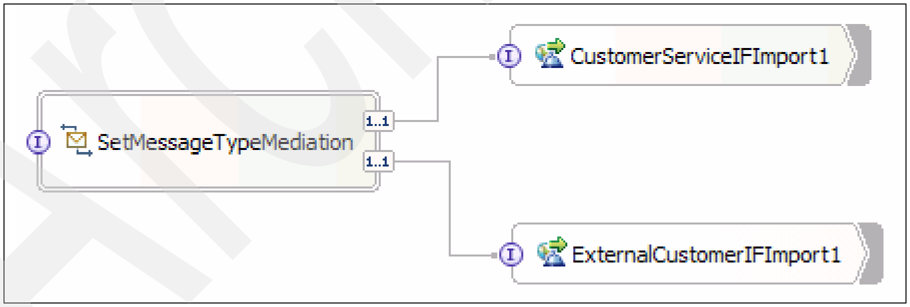
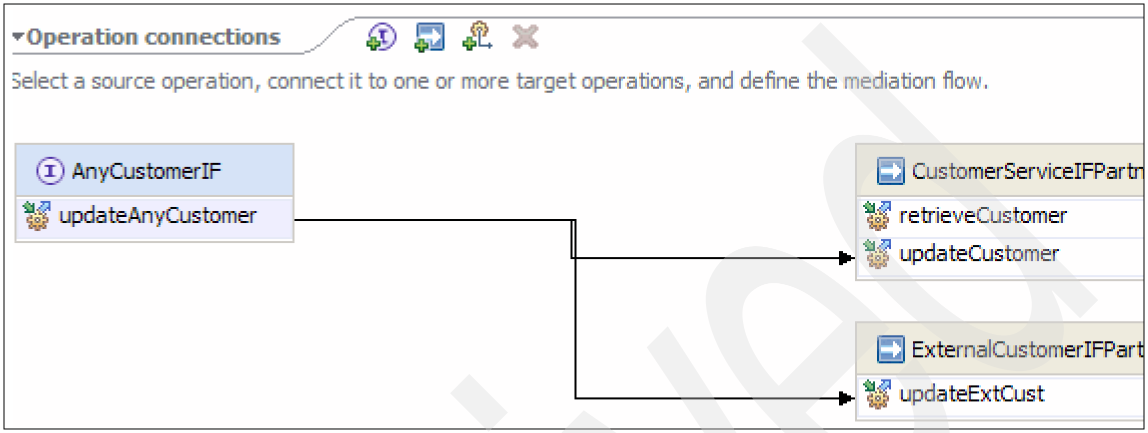


Figure 6-54 Assembly diagram

6. Regenerate the implementation. The Mediation Flow Editor opens.

6.3.5 Wire the connections

The operation connections for this module are wired as shown in Figure 6-55. Select one of the connections to open the mediation flow.



Operation connections

Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 6-55 Operation connections

6.3.6 Build the request flow

Figure 6-56 on page 403 shows the mediation flow. Add the following primitives to the mediation flow, name them, and wire them as shown in Figure 6-56.

- Two Set Message Type primitives called SetInternalType and SetExternalType.
- One Message Filter primitive called RouteCustomer. Add one output terminal to this primitive called internalCustomer. The default terminal is wired to SetExternalType. The internalCustomer terminal is wired to InternalXSLT.
- Two XSL Transformation primitives called InternalXSLT and ExternalXSLT.

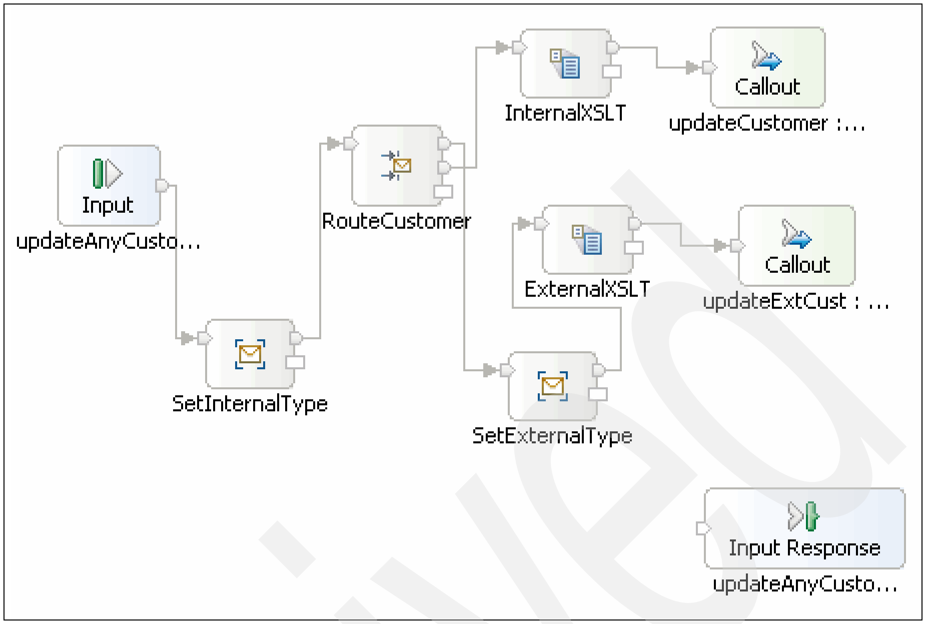
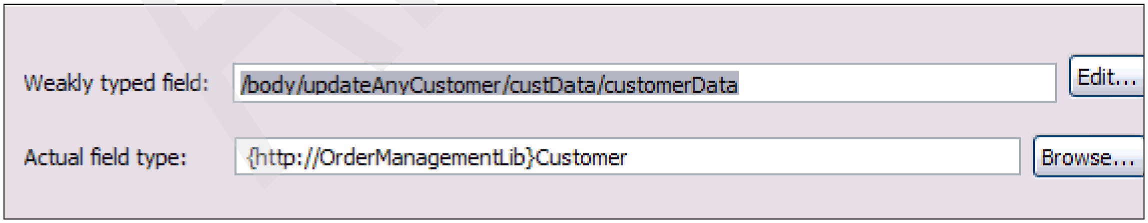


Figure 6-56 Set Message Type primitive in the mediation flow

SetInternalType Message Type primitive

This primitive casts AnyCustomerBO to Customer. To configure this primitive:

1. Select the primitive in the mediation flow and set the new message type:
   a. In the Details tab of the Properties view, click **Add** to set a new message type of the custData field.
   b. Click **Edit** by the Weakly typed field to open an XPath Expression Builder.
   c. Select **/body/updateAnyCustomer/custData/customerData** and click **Finish**.
   d. Click **Browse** for the Actual field type. Choose **Customer**.



Weakly typed field: /body/updateAnyCustomer/custData/customerData

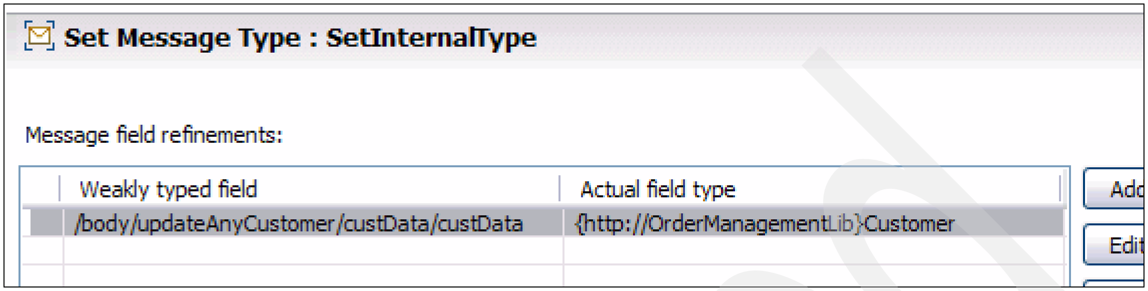
Actual field type: {http://OrderManagementLib}Customer

Figure 6-57 Set Message Type properties

2. Click **Finish**.

Now, you see the following weakly-typed field is treated as a strongly-typed Customer field in the following mediation flow (as shown in Figure 6-58):

/body/updateAnyCustomer/custData/customerData



Set Message Type : SetInternalType

Message field refinements:

Weakly typed field	Actual field type
/body/updateAnyCustomer/custData/custData	{http://OrderManagementLib}Customer

Figure 6-58 Set Message Type properties

RouteCustomer Message Filter primitive

The Message Filter primitive is used to route the message to the next primitive in the flow. The primitive has two out terminals. The default terminal is wired to SetExternalType. The internalCustomer terminal is wired to InternalXSLT.

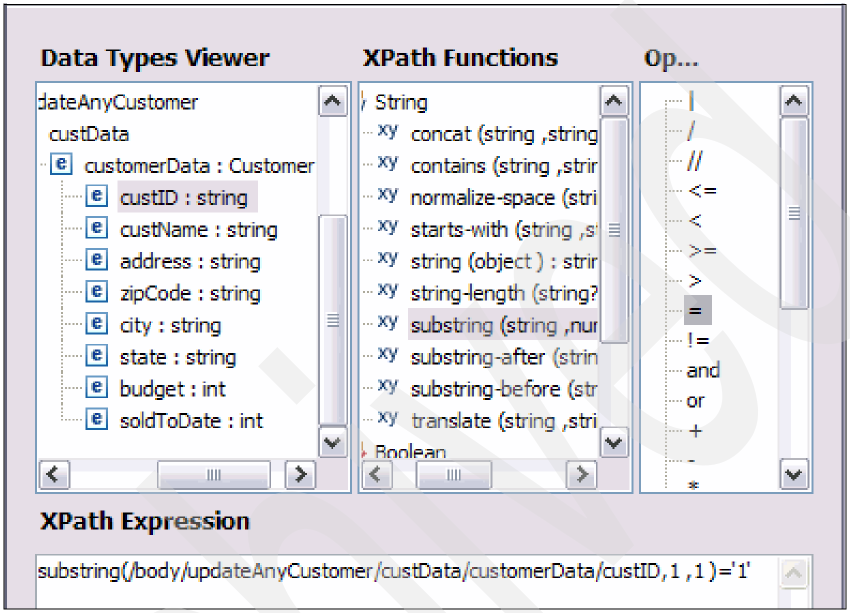
Using this filter:

- If the customer ID starts with *1*, the message is routed to InternalXSLT and eventually results in a call to the updateCustomer operation.
- If it does not start with a *1*, then the message is routed to ExternalXSLT and eventually results in a message to the updateExtCust operation.

To configure the RouteCustomer primitive, perform the following steps

1. Select the primitive in the mediation flow and open the Details tab of the Properties view.

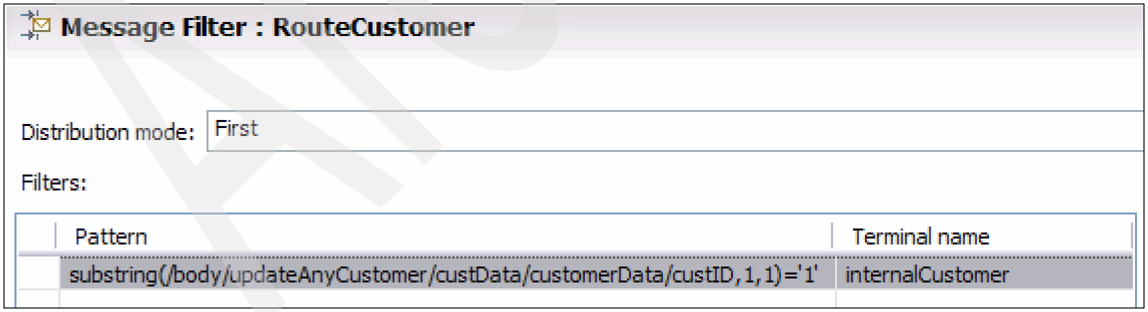
2. Click **Add** to create a the new filtering condition. This starts a wizard where you can enter the pattern. You can enter the pattern, or you can click **Edit** to the right of the Pattern field to use the xPath Expression Builder to build the pattern, as shown in Figure 6-59. Make sure that you leave no trailing spaces in the expression.



substring(/body/updateAnyCustomer/custData/customerData/custID,1 ,1)='1'

Figure 6-59 XPath Expression Builder

3. Select the terminal name, and click **Finish** to complete the wizard. Figure 6-60 shows the results.



Message Filter : RouteCustomer

Distribution mode: First

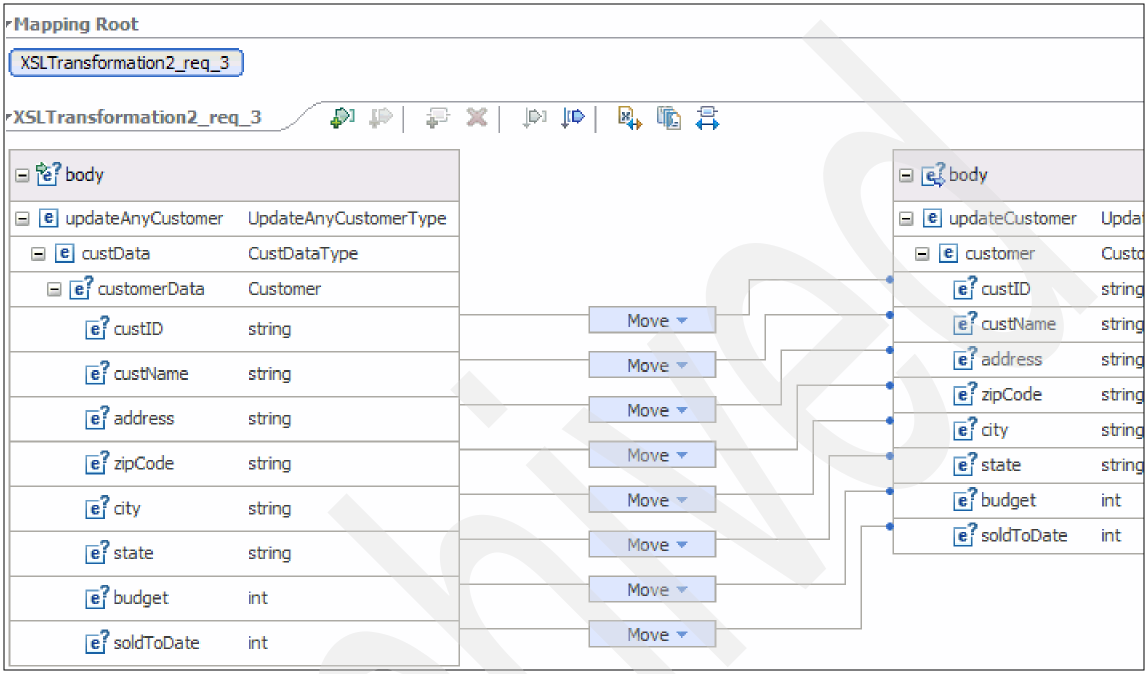
Filters:

Pattern	Terminal name
substring(/body/updateAnyCustomer/custData/customerData/custID,1,1)='1'	internalCustomer

Figure 6-60 Filtering condition

InternalXSLT XSL Transformation primitive

This primitive moves the values from the customerData typed as Customer. Open the map for the InternalXSLT primitive and add the mappings shown in Figure 6-61.



Mapping Root

XSLTransformation2_req_3

XSLTransformation2_req_3

body	
updateAnyCustomer	UpdateAnyCustomerType
custData	CustDataType
customerData	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int

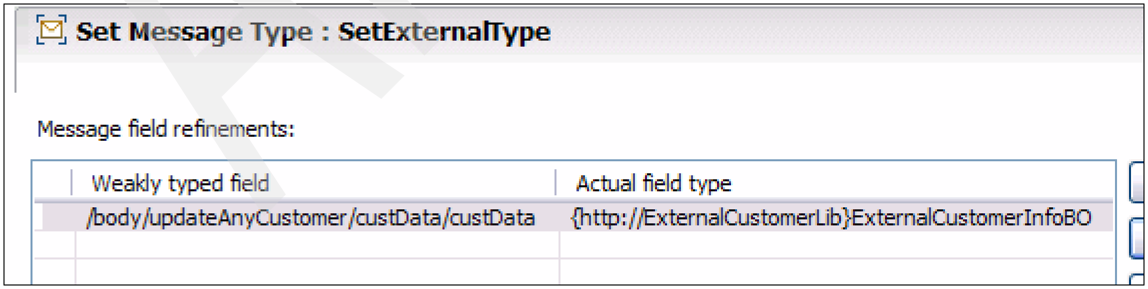
Move
Move
Move
Move
Move
Move
Move
Move

body	
updateCustomer	Upda
customer	Custo
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int

Figure 6-61 InternalXSLT mapping

SetExternalType

Update the properties for the second Set Message Type primitive called SetExternalType as shown in Figure 6-62. This casts AnyCustomerBO to ExternalCustomerInfoBO.



Set Message Type : SetExternalType

Message field refinements:

Weakly typed field	Actual field type
/body/updateAnyCustomer/custData/custData	{http://ExternalCustomerLib}ExternalCustomerInfoBO

Figure 6-62 Set Message Type properties

ExternalXSLT XSL Transformation primitive

Open the mapping for ExternalXSLT and create the mappings shown in Figure 6-63.

Note that body/updateAnyCustomer/custData/custData is now viewed as ExternalCustomerInfoBO type.

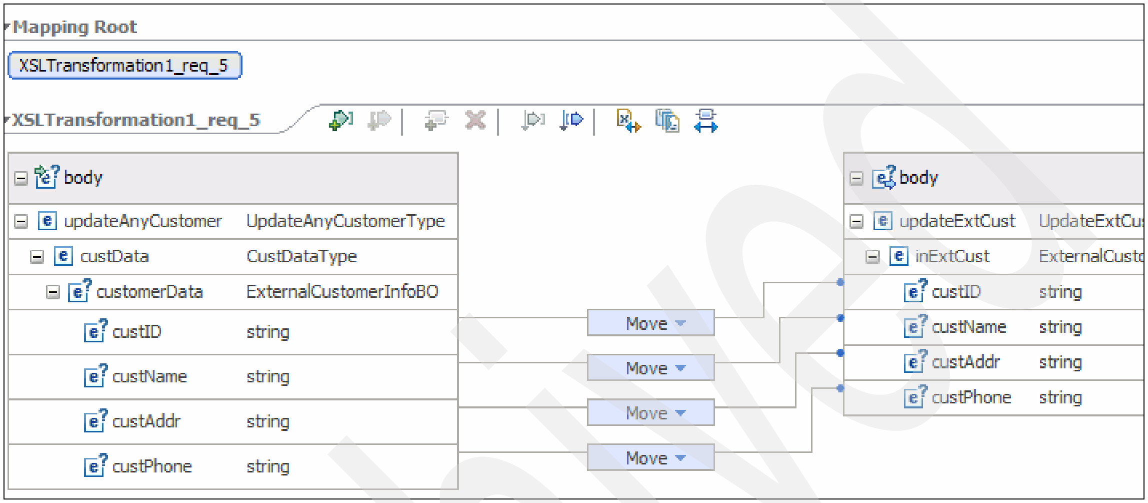


Figure 6-63 ExternalXSLT mapping

6.3.7 Response flow

To complete the response flow:

1. Open the response flow.
2. Add two XSL Transformation primitives called InternalCustResp (for the internal customers) and ExternalCustResp as shown in Figure 6-64.

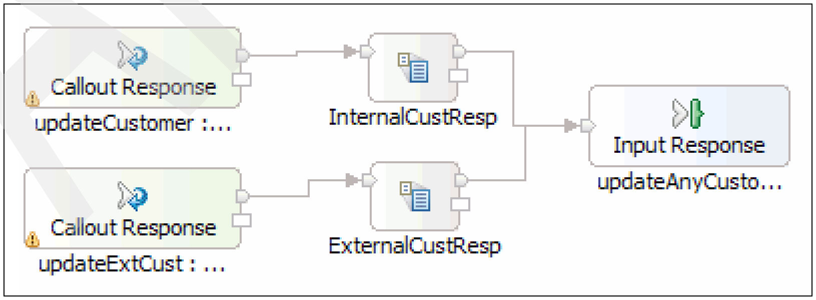


Figure 6-64 Response flow

3. Double-click the ExternalCustResp and use the Concat action to add all fields on the left to the result field on the right, as shown in Figure 6-65.

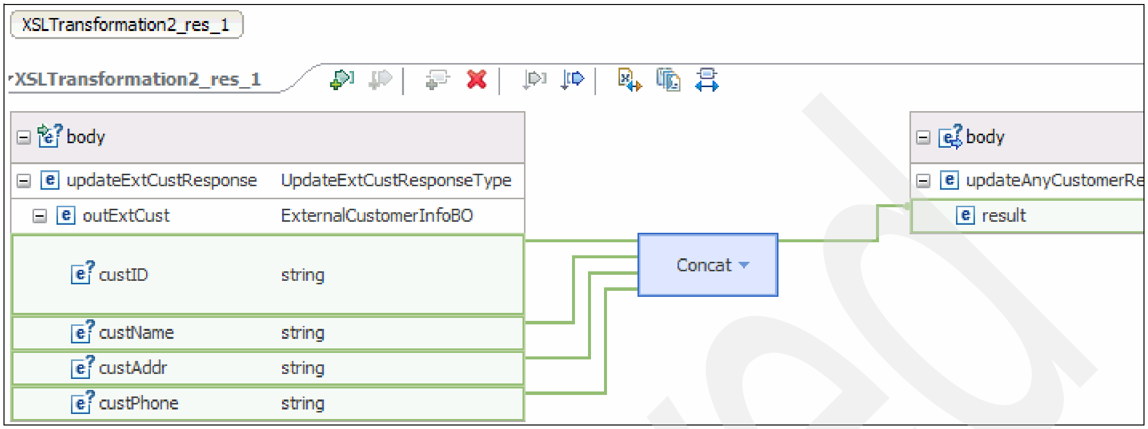
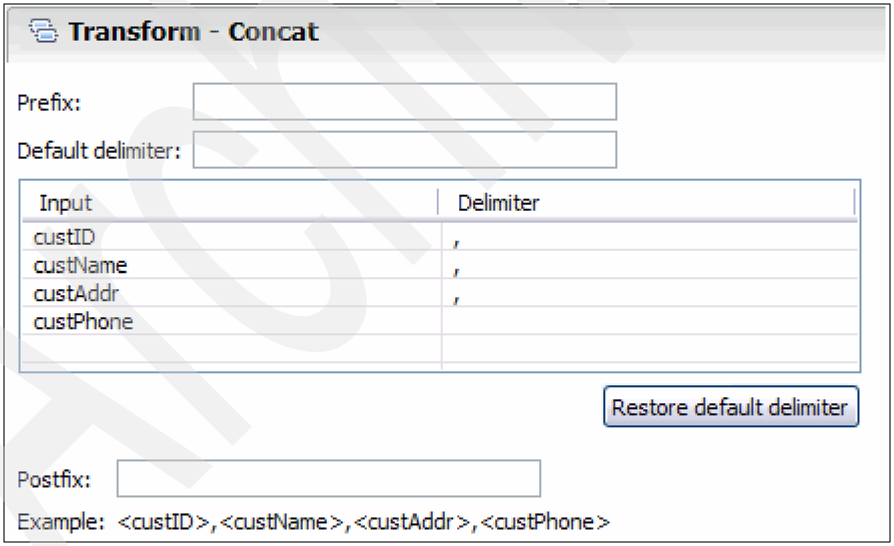


Figure 6-65 ExternalCustResp mapping

4. Select the Concat action. In the Properties view, add a delimiter to this operation to better format the result, as shown in Figure 6-66.



Transform - Concat

Prefix:

Default delimiter:

Input	Delimiter
custID	,
custName	,
custAddr	,
custPhone	

Postfix:

Example: <custID>,<custName>,<custAddr>,<custPhone>

Figure 6-66 Add a delimiter for the Concat action

5. Double-click the InternalCustResp primitive to open the mapping. Because updateCustomer operation of the internal CustomerService returns only one value, we move it the *result*, as shown in Figure 6-67.

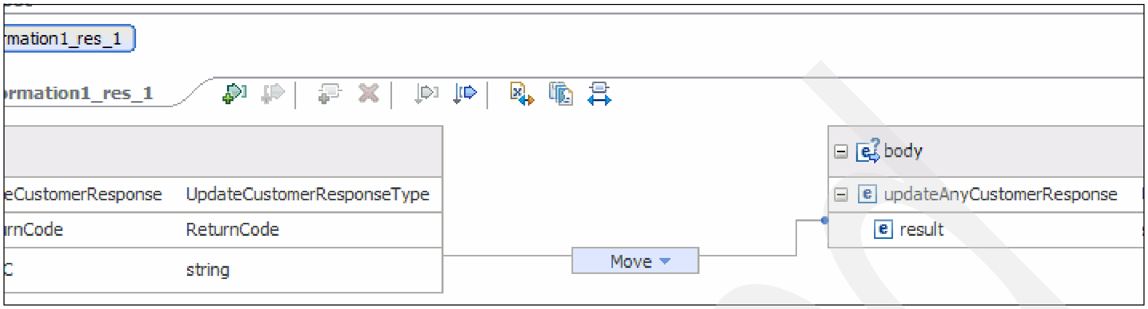


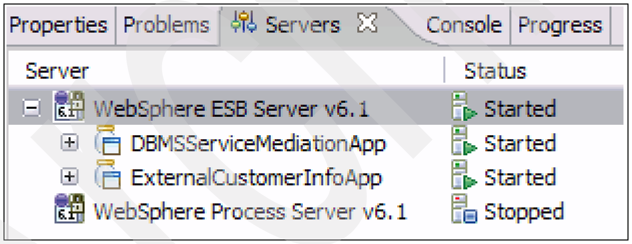
Figure 6-67 InternalCustResp mapping

6. Save and close all open windows.

6.3.8 Test the flow

To test the flow, follow these steps:

1. Launch the internal, DBMSServiceMediationApp, and external, ExternalCustomerInfoApp, Web services. See Figure 6-68.

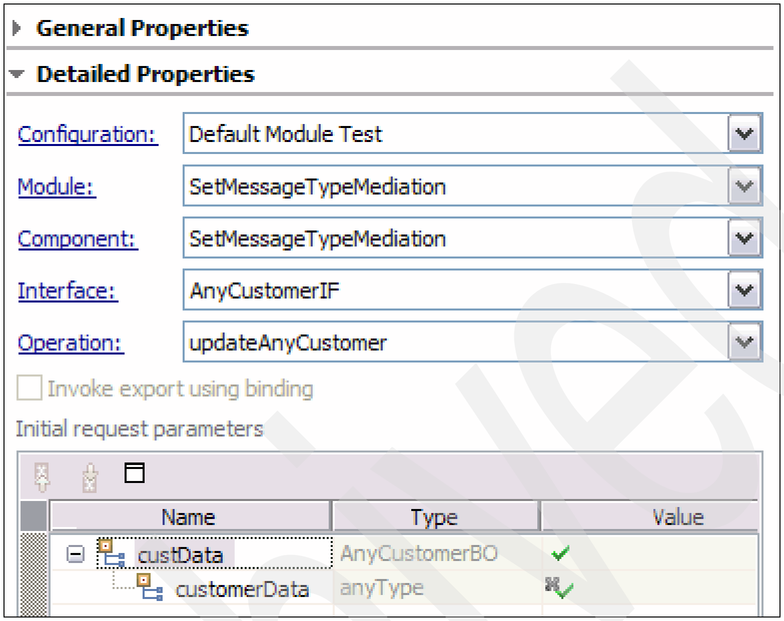


Server	Status
WebSphere ESB Server v6.1	Started
DBMSServiceMediationApp	Started
ExternalCustomerInfoApp	Started
WebSphere Process Server v6.1	Stopped

Figure 6-68 Start the server and applications

2. Click anywhere in the white space of the assembly diagram and select **Test Module** from the pop-up menu.

3. Make sure that you are testing SetMessageTypeMediation module and component, AnyCustomerIF interface, updateAnyCustomer operation, as shown in Figure 6-69.



Configuration: Default Module Test

Module: SetMessageTypeMediation

Component: SetMessageTypeMediation

Interface: AnyCustomerIF

Operation: updateAnyCustomer

□ Invoke export using binding

Initial request parameters

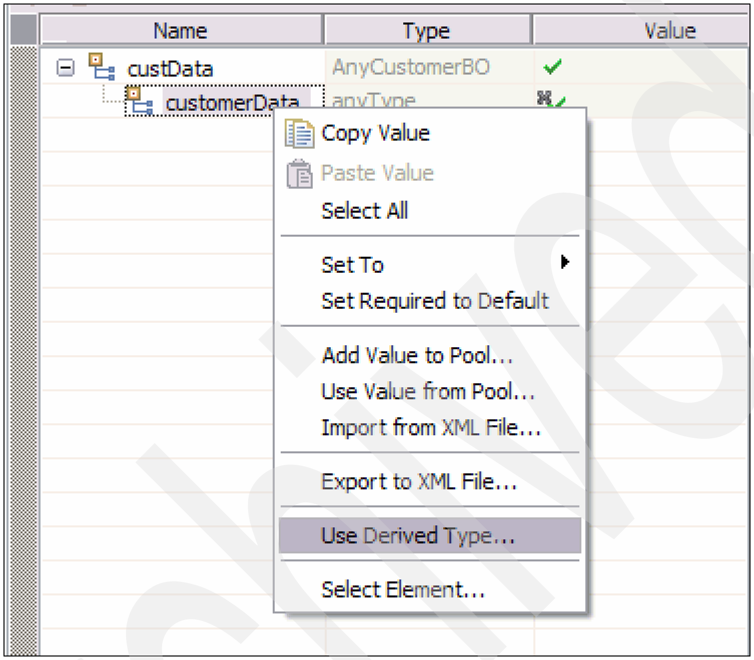
Name	Type	Value
custData	AnyCustomerBO	✓
customerData	anyType	✓

Figure 6-69 Test properties

4. Notice that customerData field is of type anyType. To define the type for this field:

   a. Select the field, right-click, and choose **Use Derived Type** from the pop-up menu, as shown in Figure 6-70.

   b. Select **ExternalCustomerInfoBo** from the Data Type Selection window.

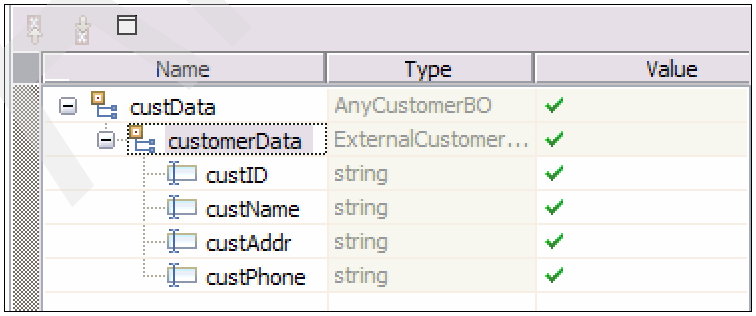


Name	Type	Value
custData	AnyCustomerBO	✓
customerData	anyType	✓

Figure 6-70 Use Derived Type

The field type changes; however, the field stays locked with an X sign.

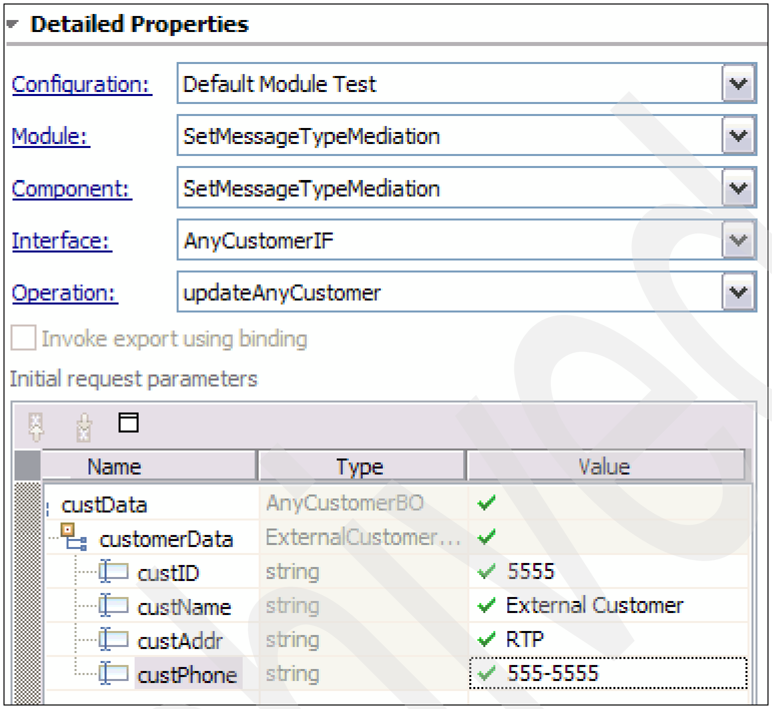
   c. Right-click the field again, and choose **Set To** → **Default** to make all fields available for setting test values (as shown in Figure 6-71).



Name	Type	Value
custData	AnyCustomerBO	✓
customerData	ExternalCustomer...	✓
custID	string	✓
custName	string	✓
custAddr	string	✓
custPhone	string	✓

Figure 6-71 New type for customerData

5. Enter the values of your choice, but make sure that custID does not start with *1* (see Figure 6-72). Then, click **Continue**.



Detailed Properties

Configuration: Default Module Test
Module: SetMessageTypeMediation
Component: SetMessageTypeMediation
Interface: AnyCustomerIF
Operation: updateAnyCustomer

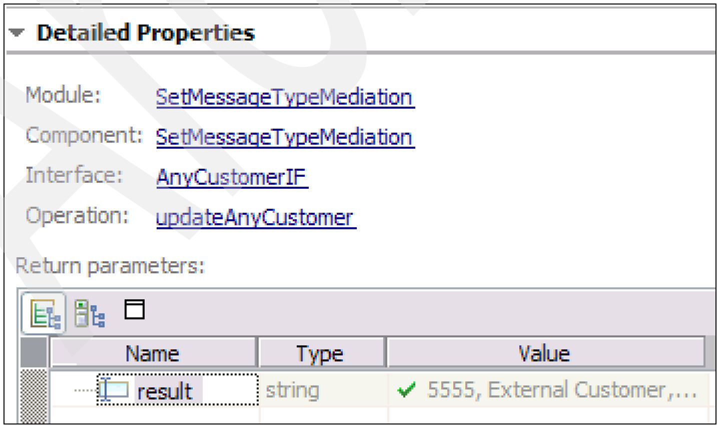
□ Invoke export using binding

Initial request parameters

Name	Type	Value
custData	AnyCustomerBO	✓
customerData	ExternalCustomer…	✓
custID	string	✓ 5555
custName	string	✓ External Customer
custAddr	string	✓ RTP
custPhone	string	✓ 555-5555

Figure 6-72 Enter the test values

You should see results similar to that shown in Figure 6-73.



Detailed Properties

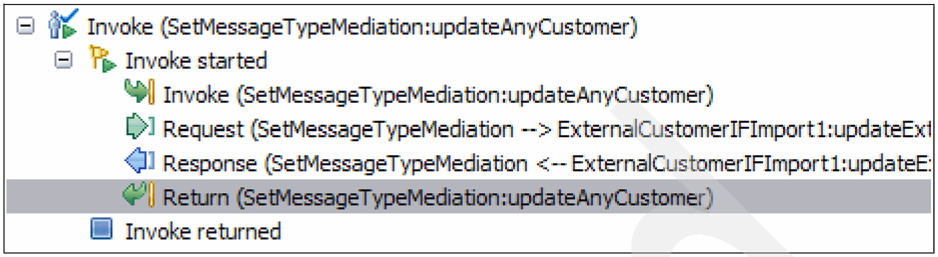
Module: SetMessageTypeMediation
Component: SetMessageTypeMediation
Interface: AnyCustomerIF
Operation: updateAnyCustomer

Return parameters:

Name	Type	Value
result	string	✓ 5555, External Customer,…

Figure 6-73 Test results

6. Check the sequence of components that are called to make sure that the ExternalCustomerInfo Web service is invoked as shown in Figure 6-74.

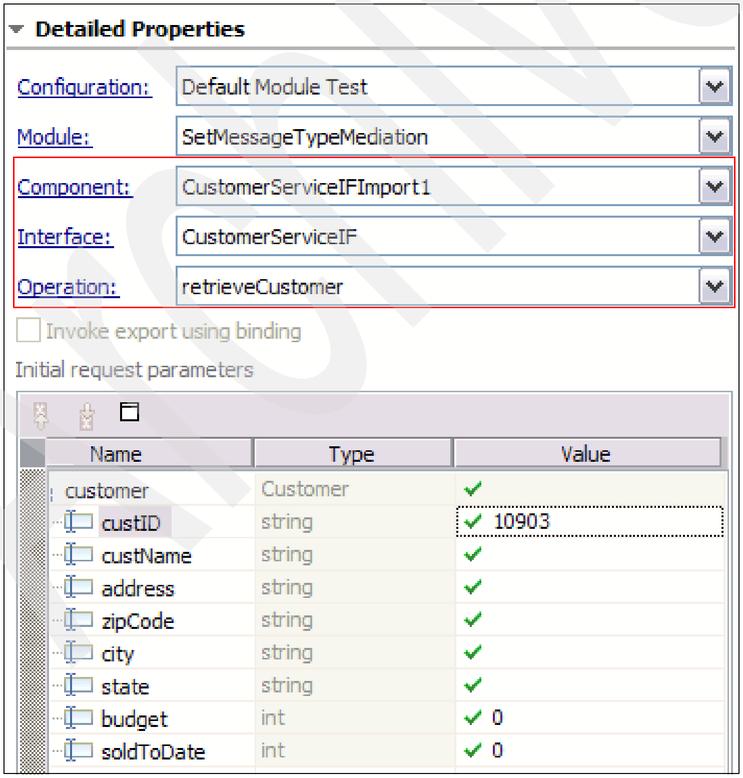


```
Invoke (SetMessageTypeMediation:updateAnyCustomer)
  Invoke started
    Invoke (SetMessageTypeMediation:updateAnyCustomer)
    Request (SetMessageTypeMediation --> ExternalCustomerIFImport1:updateExt
    Response (SetMessageTypeMediation <-- ExternalCustomerIFImport1:updateE:
    Return (SetMessageTypeMediation:updateAnyCustomer)
  Invoke returned
```

Figure 6-74 Test component sequence

7. Now, test the CustomerService. Because you will update the customer data, it might be worth knowing what that data is before the update.

To retrieve customer data change the test Component to CustomerServiceIFImport1, Interface to CustomerServiceIF, Operation to retrieveCustomer. Enter a valid customer ID, such as 10903.



Detailed Properties

Configuration: Default Module Test

Module: SetMessageTypeMediation

Component: CustomerServiceIFImport1

Interface: CustomerServiceIF

Operation: retrieveCustomer

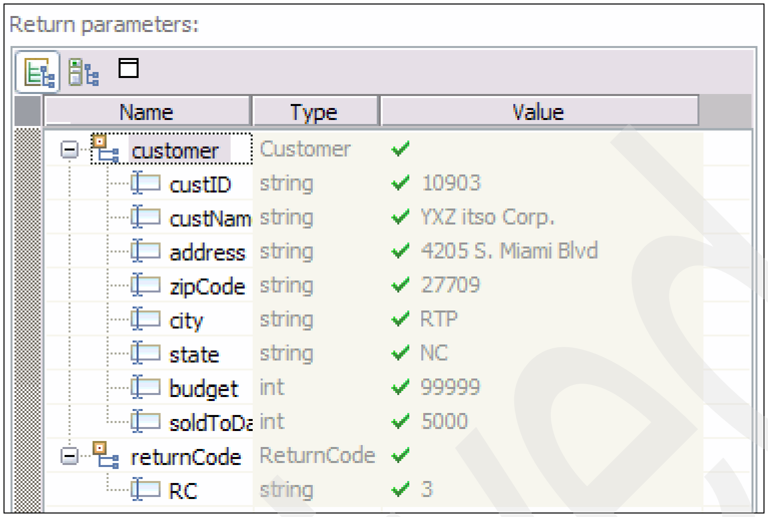
□ Invoke export using binding

Initial request parameters

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10903
custName	string	✓
address	string	✓
zipCode	string	✓
city	string	✓
state	string	✓
budget	int	✓ 0
soldToDate	int	✓ 0

Figure 6-75 Test CustomerServiceIFImport1

8. Click **Continue** and check the results. Figure 6-76 shows the results.



Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10903
custNam	string	✓ YXZ itso Corp.
address	string	✓ 4205 S. Miami Blvd
zipCode	string	✓ 27709
city	string	✓ RTP
state	string	✓ NC
budget	int	✓ 99999
soldToDa	int	✓ 5000
returnCode	ReturnCode	✓
RC	string	✓ 3

Figure 6-76 Test results

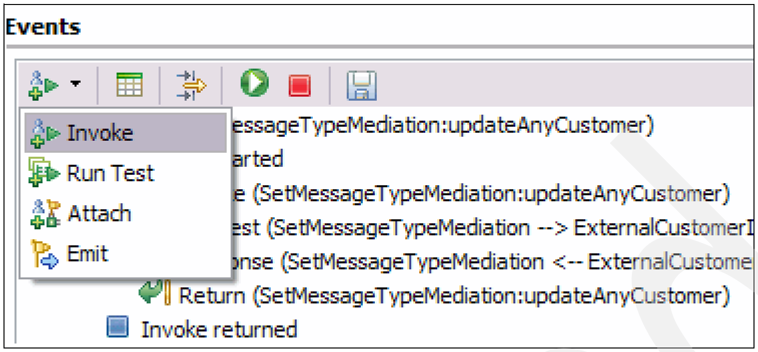
9. Note these values so that the test can be run with a customer ID of 10903.

Note: Normally, the easiest way to update the values is to save the values for custID=10903 in the following manner:

1. Select the customer object under the Name column, right-click, and choose **Copy Value** from the pop-up menu.
2. Then, when you have invoked the new test and changed the customerData object to Customer type, use the **Paste Value** option to fill in the data.

However, there seems to be a bug for this method when the type is anyType. So we entered the values manually.

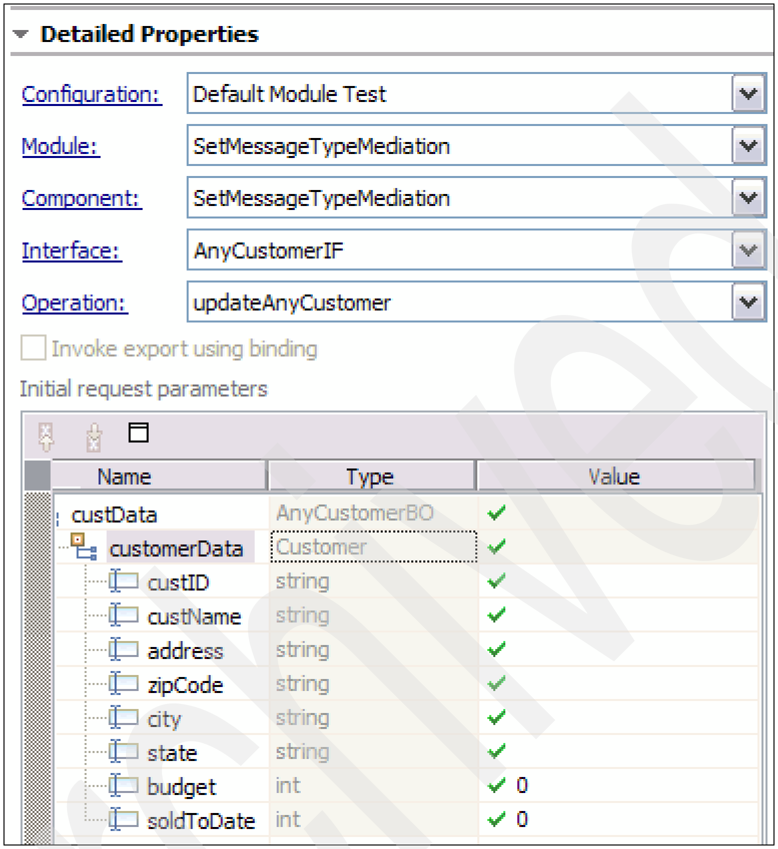
3. Now, click **Invoke** again (as shown in Figure 6-77).



```
essageTypeMediation:updateAnyCustomer)
arted
e (SetMessageTypeMediation:updateAnyCustomer)
est (SetMessageTypeMediation --> ExternalCustomerI
nse (SetMessageTypeMediation <-- ExternalCustome
    Return (SetMessageTypeMediation:updateAnyCustomer)
  Invoke returned
```

Figure 6-77 Invoke the test

4. Select the SetMessageTypeMediation component to test. Set the customerData object type to `Customer` as shown in Figure 6-78.



Detailed Properties

Configuration: Default Module Test

Module: SetMessageTypeMediation

Component: SetMessageTypeMediation

Interface: AnyCustomerIF

Operation: updateAnyCustomer

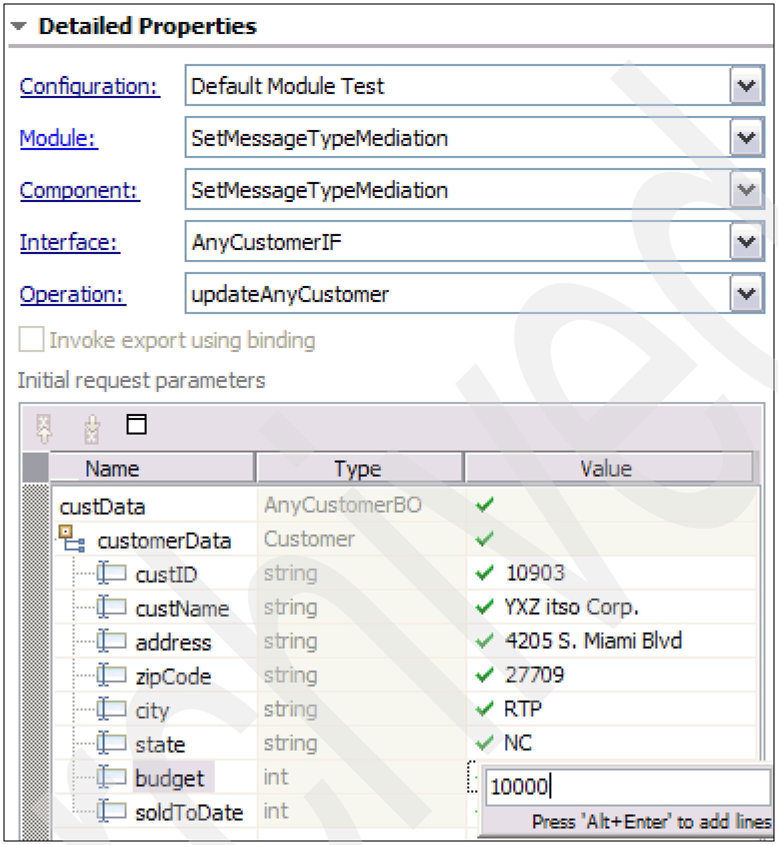
□ Invoke export using binding

Initial request parameters

Name	Type	Value
custData	AnyCustomerBO	✓
customerData	Customer	✓
custID	string	✓
custName	string	✓
address	string	✓
zipCode	string	✓
city	string	✓
state	string	✓
budget	int	✓ 0
soldToDate	int	✓ 0

Figure 6-78 New test settings

5. Enter the new test data values based on those that you noted previously. Change the budget value from 99999 to 10000 by clicking the **99999** value in the **budget** field. Enter a new value of 10000 as shown in Figure 6-79.



Detailed Properties

Configuration: Default Module Test

Module: SetMessageTypeMediation

Component: SetMessageTypeMediation

Interface: AnyCustomerIF

Operation: updateAnyCustomer

□ Invoke export using binding

Initial request parameters

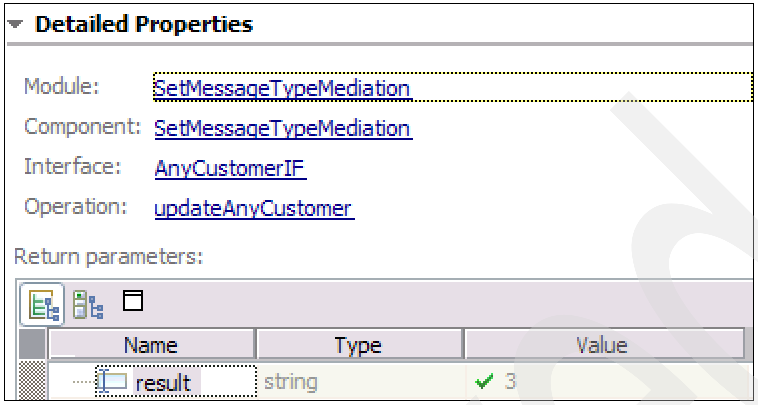
Name	Type	Value
custData	AnyCustomerBO	✓
customerData	Customer	✓
custID	string	✓ 10903
custName	string	✓ YXZ itso Corp.
address	string	✓ 4205 S. Miami Blvd
zipCode	string	✓ 27709
city	string	✓ RTP
state	string	✓ NC
budget	int	10000
soldToDate	int	

Press 'Alt+Enter' to add lines

Figure 6-79 New test input

6. Click **Continue** to run the test.

7. Verify the result, which should be 3 as expected from the CustomerServiceIF:updateCustomer operation, as shown in Figure 6-80.



Detailed Properties

Module: SetMessageTypeMediation

Component: SetMessageTypeMediation

Interface: AnyCustomerIF

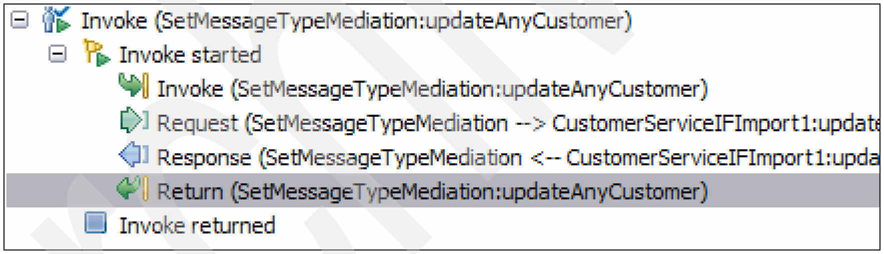
Operation: updateAnyCustomer

Return parameters:

Name	Type	Value
result	string	✓ 3

Figure 6-80 Test results

8. Verify the component invocation sequence. You should see that CustomerServiceIFImport1:updateCustomer was invoked appropriately (Figure 6-81).



```
⊟ Invoke (SetMessageTypeMediation:updateAnyCustomer)
    ⊟ Invoke started
        Invoke (SetMessageTypeMediation:updateAnyCustomer)
        Request (SetMessageTypeMediation --> CustomerServiceIFImport1:updat
        Response (SetMessageTypeMediation <-- CustomerServiceIFImport1:upda
        Return (SetMessageTypeMediation:updateAnyCustomer)
      Invoke returned
```

Figure 6-81 Test component sequence

9. Remove the projects from the server.

6.4 Message Filter example

Additional material: This example is included in the MessageFilterMediation project in the additional materials.

This sample demonstrates how to create a Message Filter primitive to route messages based on message content. It builds on the TestDBLookup mediation and adds additional processing for North Carolina customers using a message filter on the response flow.

6.4.1 Create the module

To create the module, you need to copy the TestDBLookup mediation and paste it in the workspace under a different name. The steps are as follows:

1. In the Business Integration view, right-click **TestDBLookup**, and select **Copy** from the pop-up menu.
2. Right-click anywhere in the white space of the Business Integration pane and select **Paste** from the pop-up menu.
3. Enter `MessageFilterMediation` as the new module name, and click **OK**.

These steps create a new MessageFilterMediation module with its own namespace that contains all the artifacts of the original TestDBLookup mediation module.

6.4.2 Build the response flow

To build the response flow, follow these steps:

1. Open the **MessageFilterMediation** assembly diagram.
2. Double-click TestDBLookup component to open it in the mediation flow editor.
3. Click the Response:retrieveCustomer tab (Figure 6-82).

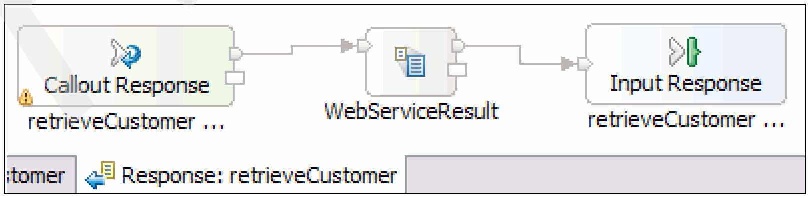


Figure 6-82 Response flow

The Callout Response node is wired to the Input Response node through the WebServiceResult XSL Transformation. The transformation simply sets “success from DBMS Web service” for the return code `RC`.

6.4.3 Message Filter primitive

Next, you add and configure a Message Filter primitive. Follow these steps:

1. In the Palette, expand the Routing folder. Click the Message Filter primitive, and then drag and drop it on the canvas (as shown in Figure 6-83).

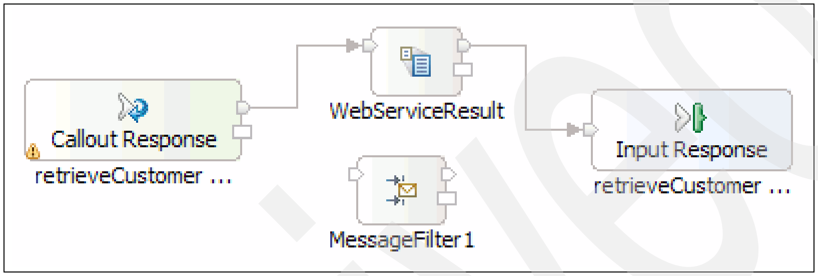
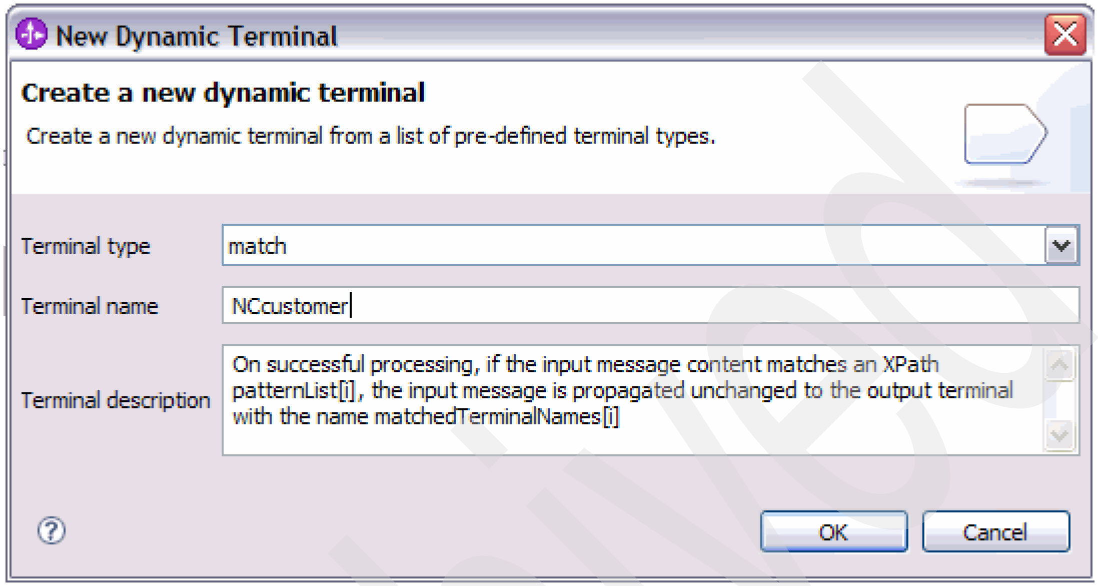


Figure 6-83 Add the Message Filter primitive

2. Right-click the Message Filter primitive and select **Add Output Terminal** from the pop-up menu. Name the terminal `NCcustomer` (as shown in Figure 6-84), and click **OK** when prompted. Then, click **OK** again.



New Dynamic Terminal

Create a new dynamic terminal

Create a new dynamic terminal from a list of pre-defined terminal types.

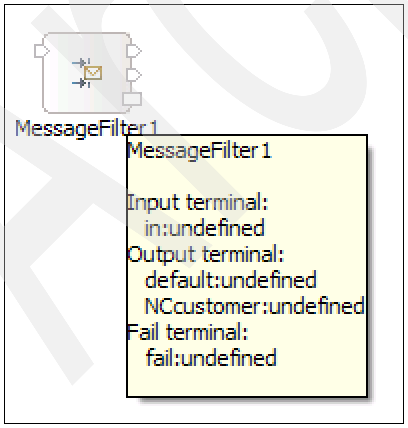
Terminal type: match

Terminal name: NCcustomer

Terminal description: On successful processing, if the input message content matches an XPath patternList[i], the input message is propagated unchanged to the output terminal with the name matchedTerminalNames[i]

Figure 6-84 New Message Filter terminal

3. The new terminal is added to the MessageFilter1 primitive, as shown in Figure 6-85. The primitive routes messages for North Carolina customers through this terminal. All other messages go through the default terminal.



MessageFilter1

Input terminal:
in:undefined
Output terminal:
default:undefined
NCcustomer:undefined
Fail terminal:
fail:undefined

Figure 6-85 Message filter terminals

4. Rename MessageFilter1 to `NorthCarolinaMessageFilter`.
5. Now, select and delete the wire that connects the Callout Response node with WebServiceResult (Figure 6-86).

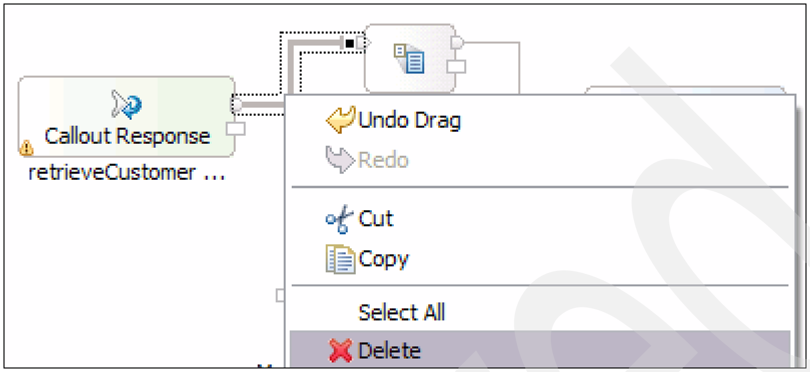


Figure 6-86 Delete the wire from the callout response

6. Then, connect the Callout Response node with NorthCarolinaMessageFilter instead (Figure 6-87).

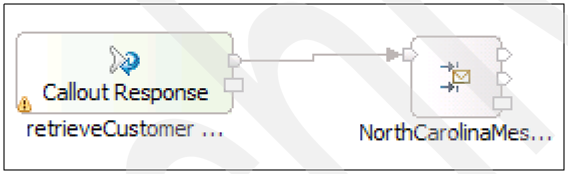
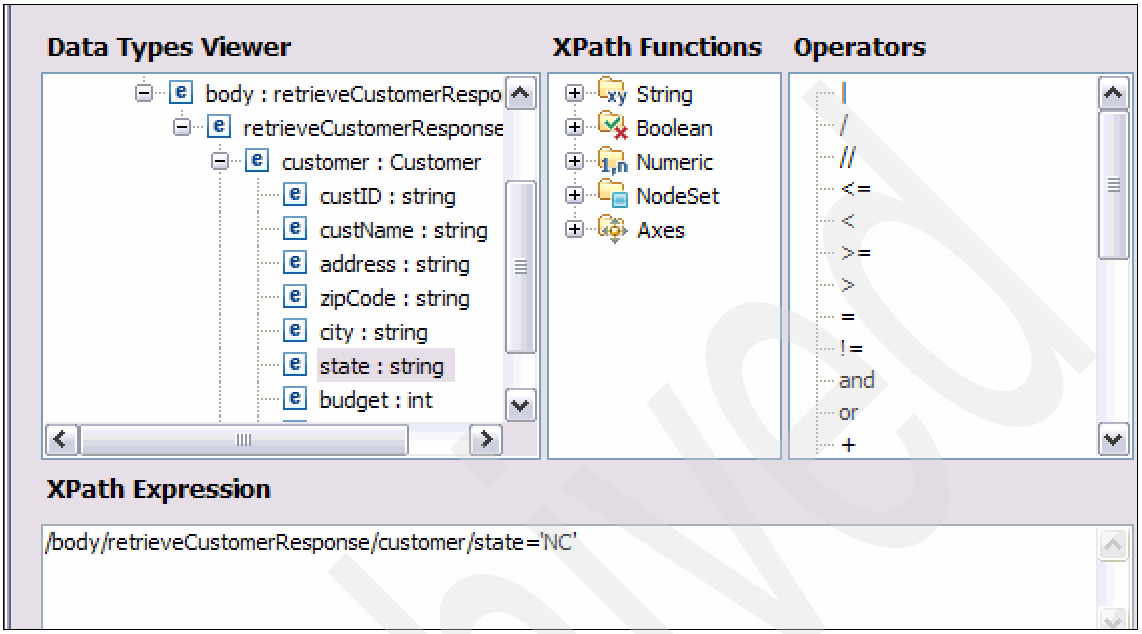


Figure 6-87 Rewire the callout response

7. Next, select **NorthCarolinaMessageFilter**, and click the Details tab in the Properties view to add a new filter.
8. Leave the distribution mode field set to `First`, which means that the message that satisfies the filter criterion is routed only to the first matching output terminal. If the distribution mode is set to `All`, the message goes to all matching terminals.The default terminal is used if the message meets none of the conditions.
9. Click **Add** to define a new filter. Then:
   a. On the Add/Edit properties window, select a matching terminal name, **NCcustomer** in our case, and click **Edit**. The XPath Expression Builder window opens (Figure 6-88 on page 423).
   b. In the Data Types Viewer expand the message elements until you find the **state** field and click it.
   c. Click the equal sign (=) in Operations pane.

d. Enter `NC` to the right of the equal sign (=) sign.

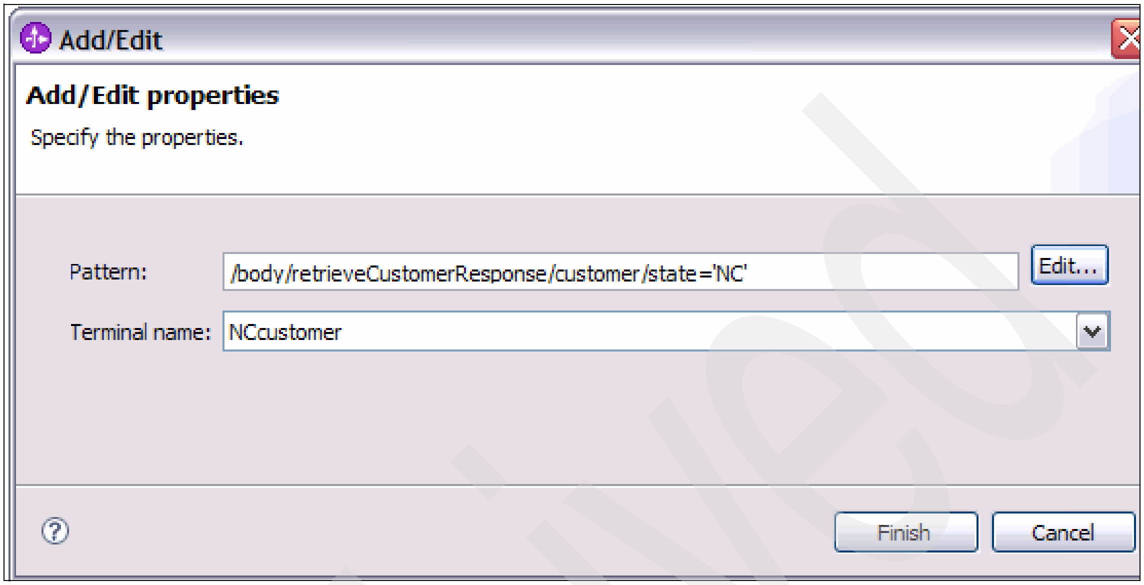
e. Click **Finish**.



/body/retrieveCustomerResponse/customer/state='NC'

Figure 6-88 Using the XPath Expression Builder to build the filter

10. The Pattern field has the new filter expression, as shown in Figure 6-89. Click **Finish**.


Add/Edit

Add/Edit properties

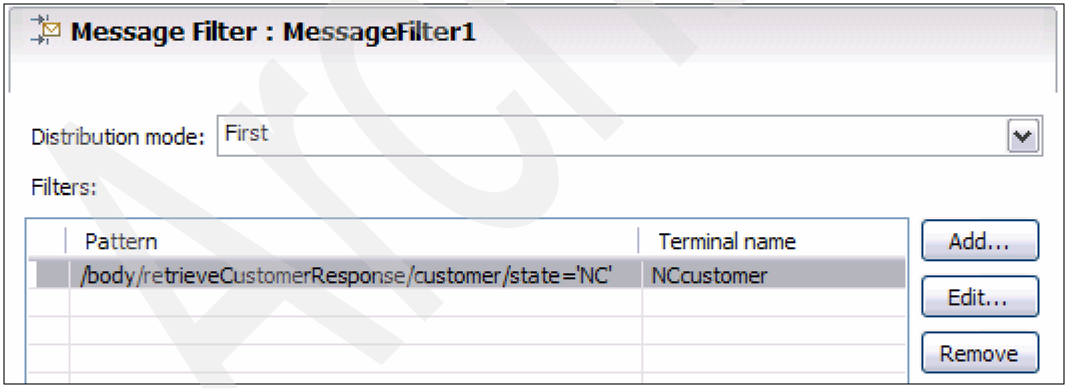
Specify the properties.

Pattern: /body/retrieveCustomerResponse/customer/state='NC'

Terminal name: NCcustomer

Figure 6-89 New filter pattern

The new filter is now in the list of the Properties view as shown in Figure 6-90.


Message Filter : MessageFilter1

Distribution mode: First

Filters:

Pattern	Terminal name
/body/retrieveCustomerResponse/customer/state='NC'	NCcustomer

Figure 6-90 New filter

11. Save the filter pressing Ctrl+S.

6.4.4 NorthCarolinaXSLT XSL Transformation

Now, you add and configure the XSL Transformation primitive that transforms the response message as follows:

1. Add a new XSL Transformation primitive to the canvas. Rename it to `NorthCarolinaXSLT`. You will need this primitive for special processing of the filtered message.
2. Connect the NCcustomer terminal of NorthCarolinaMessageFilter to NorthCarolinaXSLT and the default terminal with the in terminal of WebServiceResult. Connect the out terminal of NorthCarolinaXSLT to the in terminal of CustomerServiceIF_retrieveCustomer_InputResponse. See Figure 6-91.

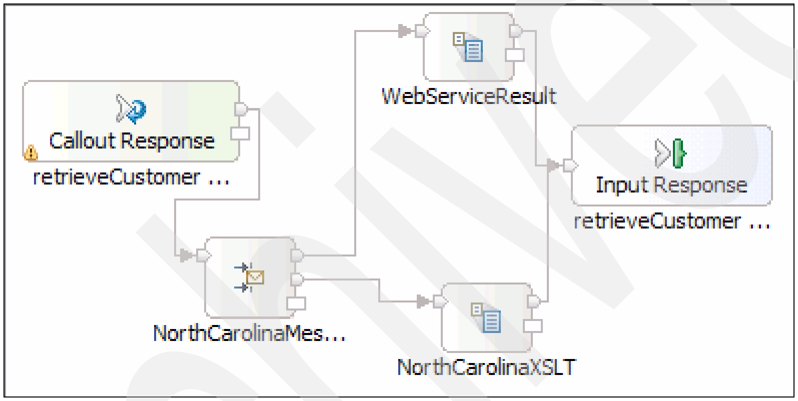
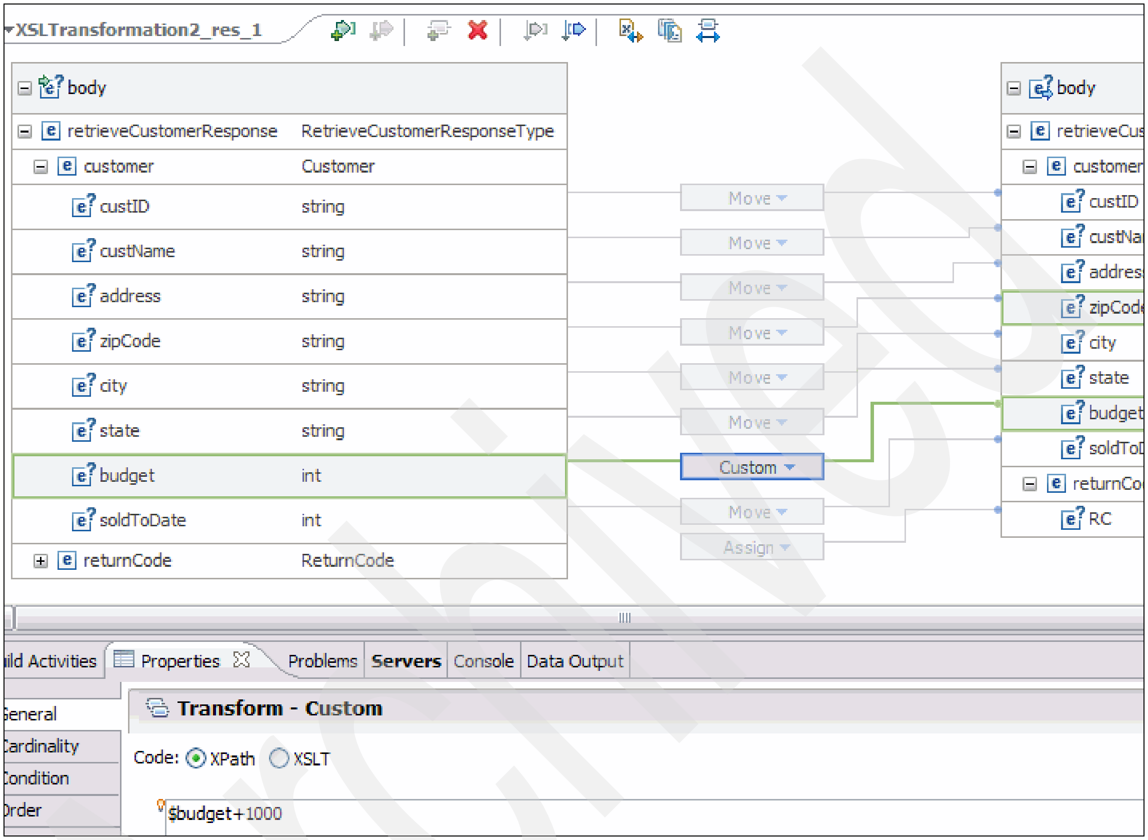


Figure 6-91 Add the NorthCarolinaXSLT primitive

3. Double-click **NorthCarolinaXSLT** and map the similar objects together with the exception for two fields: budget and RC.

4. Create a custom mapping for the budget fields and set it so that `1000` is added to the budget of a NC customer. (See the Properties view in Figure 6-92.)

5. Assign the string `successfully filtered NC customer` to the RC field.



XSLTransformation2_res_1

Source	Type
body	
retrieveCustomerResponse	RetrieveCustomerResponseType
customer	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int
returnCode	ReturnCode

Transform - Custom

Code: ● XPath ○ XSLT

$budget+1000

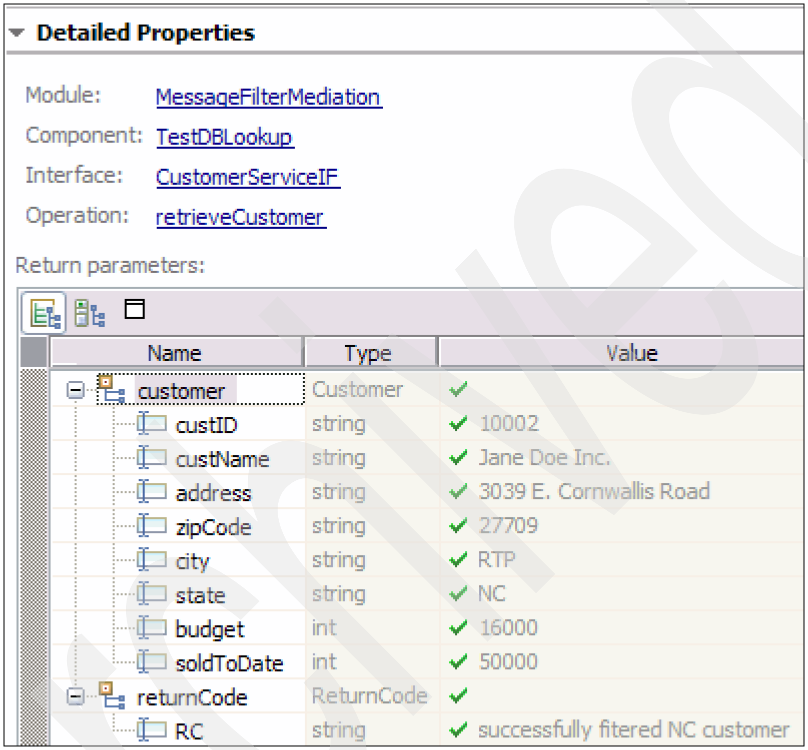
Figure 6-92 NorthCarolinaXSLT mapping

6. Save the assembly diagram and mediation flow.

6.4.5 Test the flow

Test both cases as follows:

1. Run a component test using customer ID 10002. The result shows an increased budget and a “successfully filtered NC customer” return code (as shown in Figure 6-93).



Detailed Properties

Module: MessageFilterMediation

Component: TestDBLookup

Interface: CustomerServiceIF

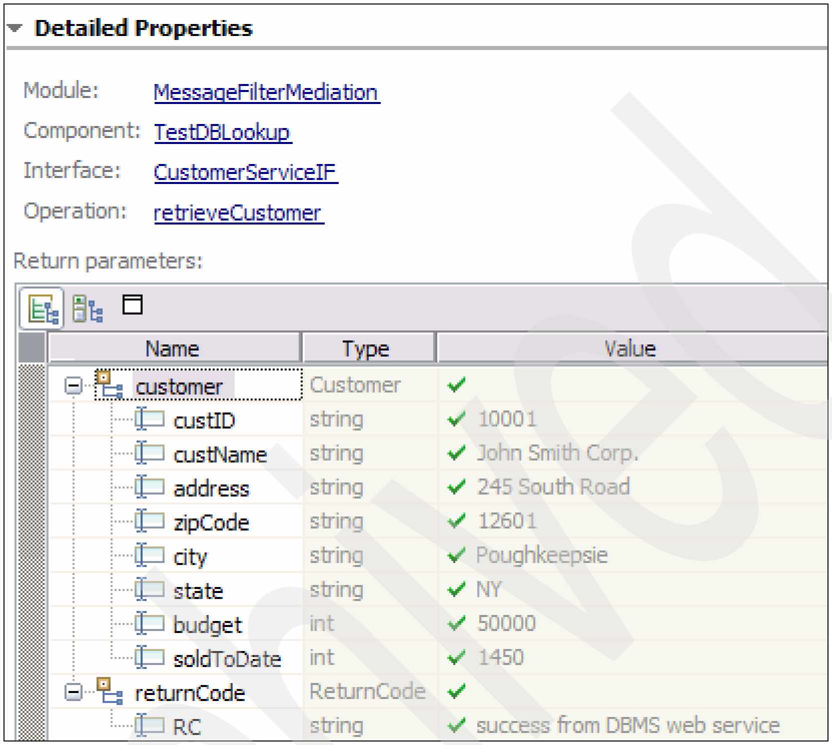
Operation: retrieveCustomer

Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10002
custName	string	✓ Jane Doe Inc.
address	string	✓ 3039 E. Cornwallis Road
zipCode	string	✓ 27709
city	string	✓ RTP
state	string	✓ NC
budget	int	✓ 16000
soldToDate	int	✓ 50000
returnCode	ReturnCode	✓
RC	string	✓ successfully fitered NC customer

Figure 6-93

2. Now test Customer ID 10001. The response message is not changed because that Customer ID is not a NC customer. See Figure 6-94.



Detailed Properties

Module: MessageFilterMediation

Component: TestDBLookup

Interface: CustomerServiceIF

Operation: retrieveCustomer

Return parameters:

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10001
custName	string	✓ John Smith Corp.
address	string	✓ 245 South Road
zipCode	string	✓ 12601
city	string	✓ Poughkeepsie
state	string	✓ NY
budget	int	✓ 50000
soldToDate	int	✓ 1450
returnCode	ReturnCode	✓
RC	string	✓ success from DBMS web service

Figure 6-94 Test non-North Carolina customer

It is interesting to observe that because the filter is set on the response flow and there is no response flow from the database lookup, the filter works only for the customer records that are retrieved through the Web service.

6.5 Endpoint Lookup example

Additional material: This example is included in the CustomerServiceEndpointMediation project in the additional materials.

This sample shows how to upload service definition to WebSphere Service Registry and Repository, create user properties, and invoke services based on selector criteria.

In this example, the customer information is stored in an EIS and can be accessed through an appropriate Web service. If the customer ID starts with *1*, the CustomerService Web service is called. If the customer ID starts with *5*, a ThirdPartyCustomerInfo service is to be called.

The CustomerService Web service is taken from the Order Management System scenario. The ThirdPartyCustomerInfo service is created here.

6.5.1 Building ThirdPartyCustomerInfo

Build the ThirdPartyCustomerInfo Web service as follows:

1. From the top menu bar select **File** → **New** → **Mediation Module**. Call the mediation module **ThirdPartyCustomerInfo**. Add **OrderManagementLib** and click **Finish**.
2. Right-click the component, and add the CustomerServiceIF interface from OrderManagementLib to the assembly diagram.
3. Select **ThirdPartyCustomerInfo**, and regenerate the implementation implementation. The Mediation Flow Editor opens.

   We do not do any actual mediating in this example. We need just set some default values for a ThirdPartyCustomer.
4. Add an XSL Transformation primitive to the mediation canvas, and rename it to `SetThirdPartyCustomerInfo`, as shown in Figure 6-95.

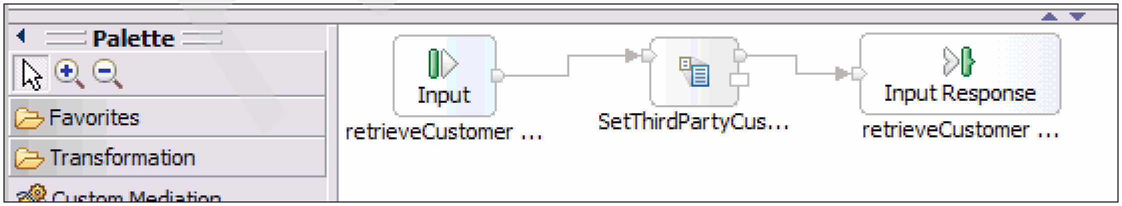
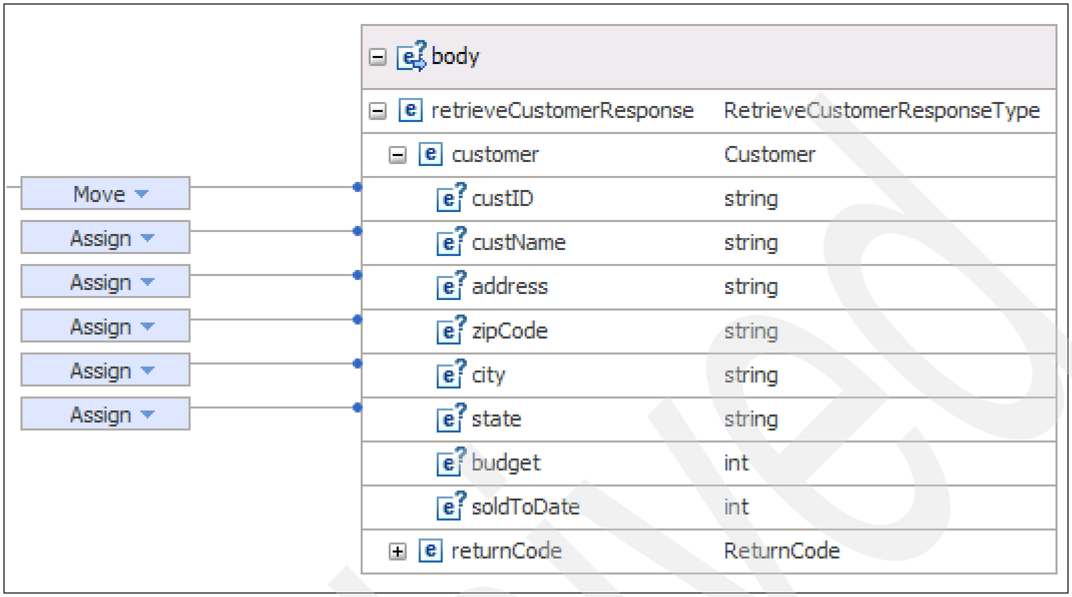


Figure 6-95 ThirdPartyCustomerInfo mediation flow

5. Double-click **SetThirdPartyCustomerInfo** to open the mapping and use the Assign transform to assign values (Figure 6-96).



body	
retrieveCustomerResponse	RetrieveCustomerResponseType
customer	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int
soldToDate	int
returnCode	ReturnCode

Figure 6-96 Assign values

The values to assign are:

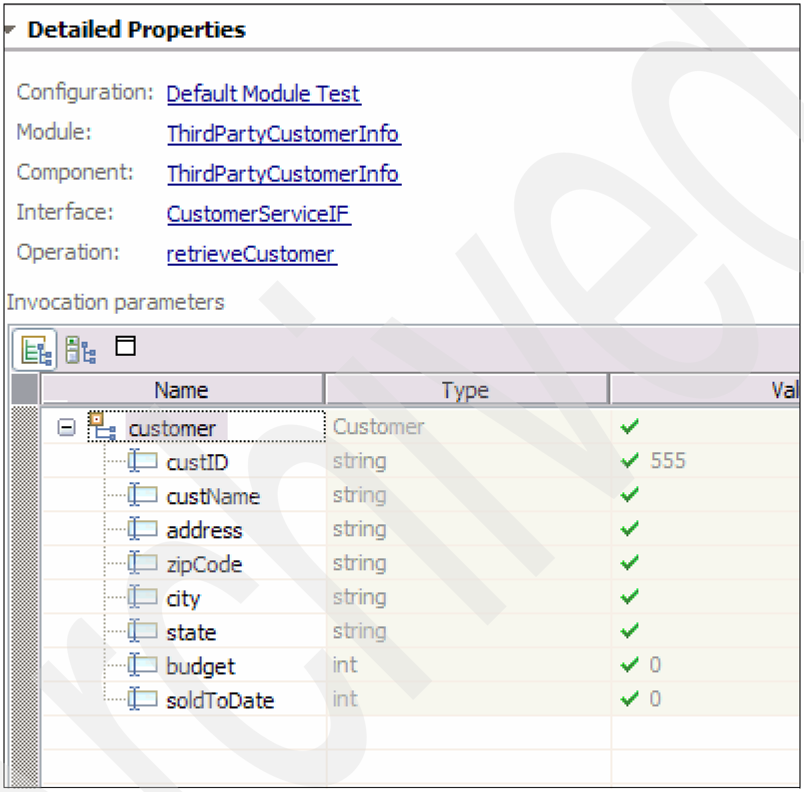
custName:	`Third Party Customer`
address:	`11111 Main Street`
zipCode:	`27514`
city:	`Chapel Hill`
state:	`NC`

6. Select **File** → **Save All**.

Test the service

Test the component to make sure the service returns these values. Follow these steps:

1. Select ThirdPartyCustomerInfo in the Business Integration view. Right-click, and select **Test** → **Test Module**.
2. Enter a customer ID, as shown in Figure 6-97, and click the Invoke icon.



Detailed Properties

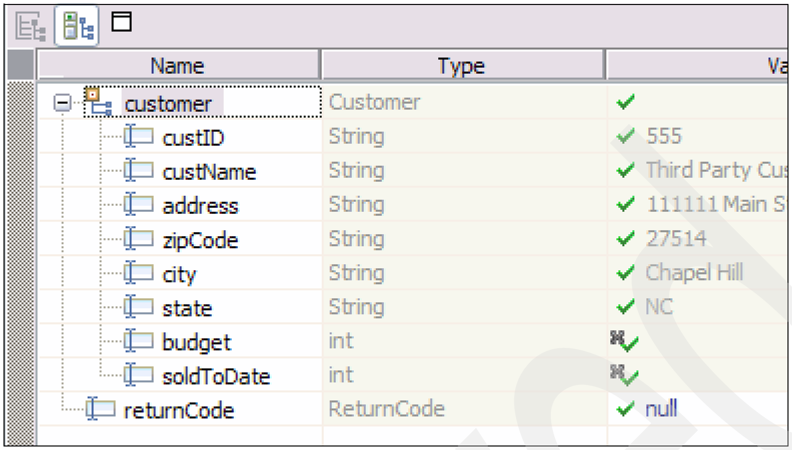
Configuration: Default Module Test
Module: ThirdPartyCustomerInfo
Component: ThirdPartyCustomerInfo
Interface: CustomerServiceIF
Operation: retrieveCustomer

Invocation parameters

Name	Type	Val
customer	Customer	✓
custID	string	✓ 555
custName	string	✓
address	string	✓
zipCode	string	✓
city	string	✓
state	string	✓
budget	int	✓ 0
soldToDate	int	✓ 0

Figure 6-97 Test values

The results should look similar to that shown in Figure 6-98.



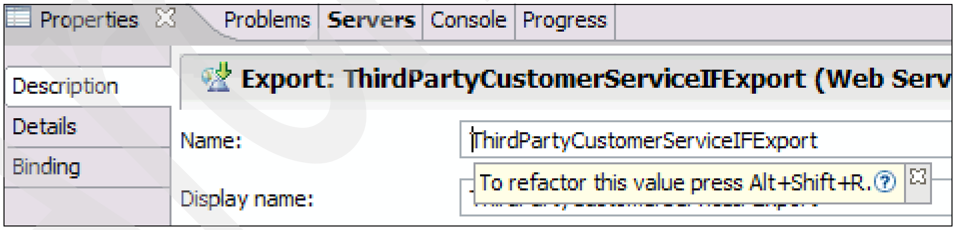
Name	Type	Va
customer	Customer	✓
custID	String	✓ 555
custName	String	✓ Third Party Cus
address	String	✓ 111111 Main S
zipCode	String	✓ 27514
city	String	✓ Chapel Hill
state	String	✓ NC
budget	int	✓
soldToDate	int	✓
returnCode	ReturnCode	✓ null

Figure 6-98 Test results

6.5.2 Export the service as a Web service

Export the ThirdPartyCustomerInfo mediation as a Web service as follows:

1. Right-click ThirdPartyCustomerInfo in the assembly diagram, and select **Generate Export** → **Web Service Binding** → **soap/http**.
2. Rename the new Export component to `ThirdPartyCustomerServiceIFExport`, and use Alt+Shift+R to refactor, as shown in Figure 6-99.



Export: ThirdPartyCustomerServiceIFExport (Web Serv

Name: ThirdPartyCustomerServiceIFExport

Display name:

To refactor this value press Alt+Shift+R.

Figure 6-99 Rename the export and refactor

The wiring looks similar to that shown in Figure 6-100

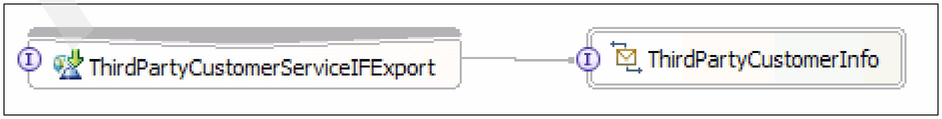
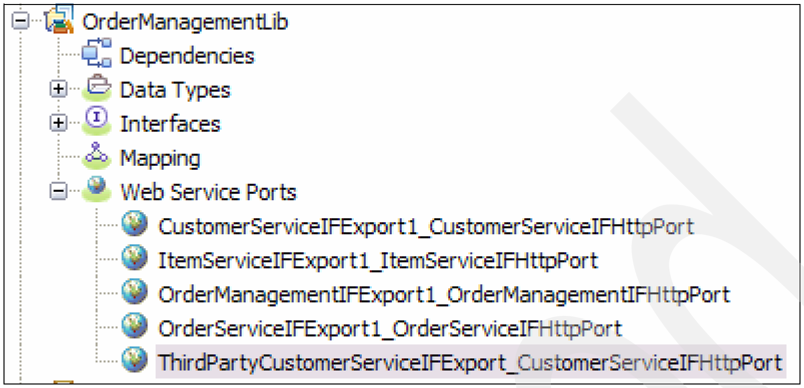


Figure 6-100 Assembly diagram

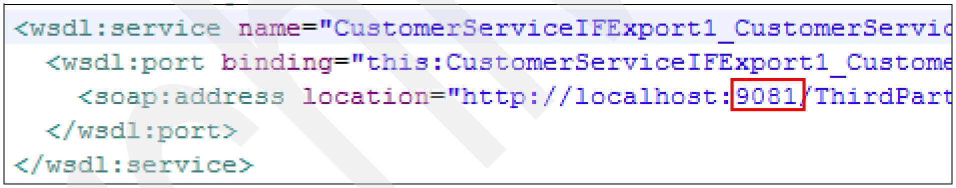
3. By default, the port that you create is added to the library. Expand **OrderManagementLib**, and then expand **Web Service Ports** (Figure 6-101).



```
OrderManagementLib
    Dependencies
    Data Types
    Interfaces
    Mapping
    Web Service Ports
        CustomerServiceIFExport1_CustomerServiceIFHttpPort
        ItemServiceIFExport1_ItemServiceIFHttpPort
        OrderManagementIFExport1_OrderManagementIFHttpPort
        OrderServiceIFExport1_OrderServiceIFHttpPort
        ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort
```

Figure 6-101 New Web service port

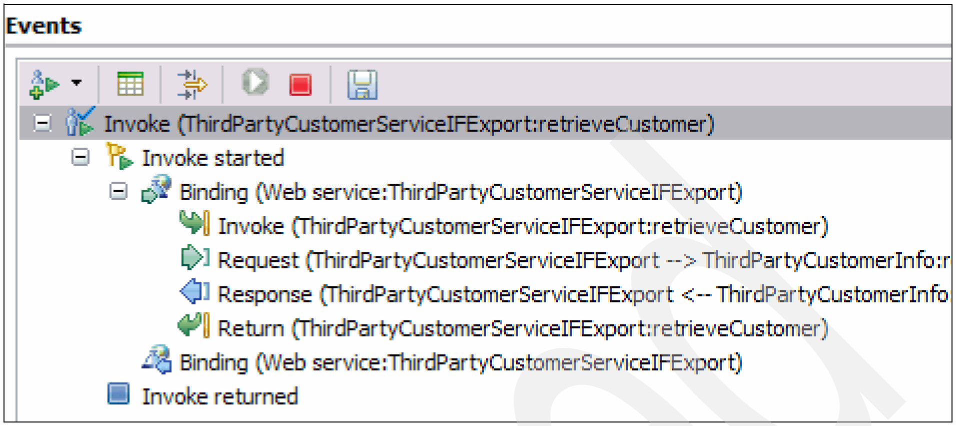
4. Double-click **ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort** to open a plain text file. Change the port number to match your integrated test environment server, as shown in Figure 6-102.



```
<wsdl:service name="CustomerServiceIFExport1_CustomerServic
  <wsdl:port binding="this:CustomerServiceIFExport1_Custome
    <soap:address location="http://localhost:9081/ThirdPart
  </wsdl:port>
</wsdl:service>
```

Figure 6-102 Update the Web service port URL

5. Test the Export component using a component test (Figure 6-103).

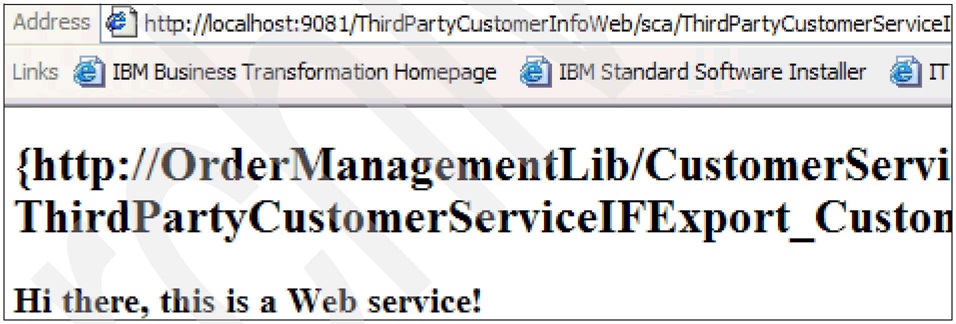


Events

```
Invoke (ThirdPartyCustomerServiceIFExport:retrieveCustomer)
  Invoke started
    Binding (Web service:ThirdPartyCustomerServiceIFExport)
      Invoke (ThirdPartyCustomerServiceIFExport:retrieveCustomer)
      Request (ThirdPartyCustomerServiceIFExport --> ThirdPartyCustomerInfo:r
      Response (ThirdPartyCustomerServiceIFExport <-- ThirdPartyCustomerInfo
      Return (ThirdPartyCustomerServiceIFExport:retrieveCustomer)
    Binding (Web service:ThirdPartyCustomerServiceIFExport)
  Invoke returned
```

Figure 6-103 Test export

6. Point your browser to the service URL and verify that the result is valid, as shown in Figure 6-104.



Address
Links
IBM Business Transformation Homepage
IBM Standard Software Installer
IT

http://localhost:9081/ThirdPartyCustomerInfoWeb/sca/ThirdPartyCustomerServiceI

{http://OrderManagementLib/CustomerServi
ThirdPartyCustomerServiceIFExport_Custon

Hi there, this is a Web service!

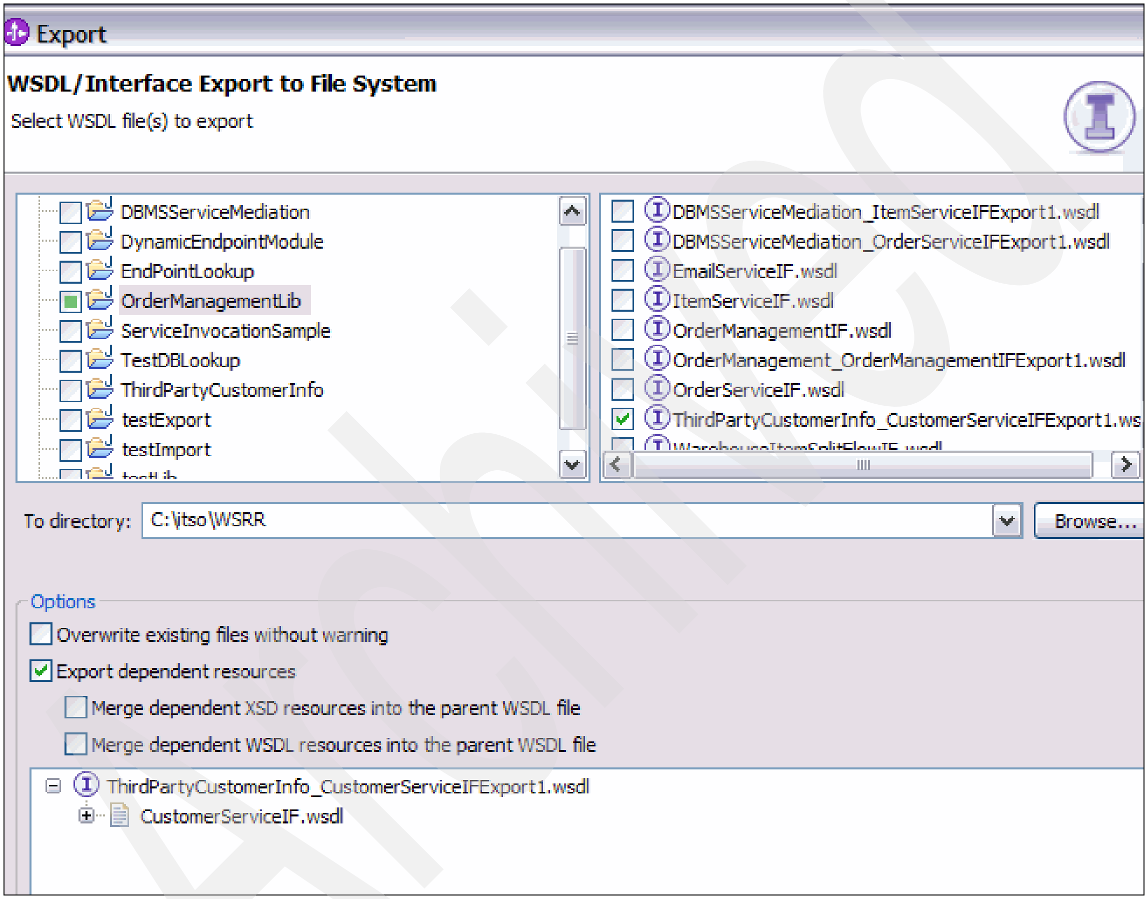
Figure 6-104 Test service presence with a browser

6.5.3 Export the service definitions

Now, there are two services that implement the same interface. Note that we have not implemented the updateCustomer operation, but this update is not important for the purpose of this sample scenario.

Export the service definitions by following these steps:

1. Right-click anywhere in Business Integration pane and choose **Export** from the pop-up menu. Select **Business Integration** → **WSDL/Interface**.
2. Select **ThirdPartyCustomerInfo_CustomerServiceIFExport1.wsdl.** Then, select **Export dependent resources**. Select a directory to export to (for example, c:\itso\WSRR) as shown in Figure 6-105.



Export

WSDL/Interface Export to File System

Select WSDL file(s) to export

- □ DBMSServiceMediation
- □ DynamicEndpointModule
- □ EndPointLookup
- ■ OrderManagementLib
- □ ServiceInvocationSample
- □ TestDBLookup
- □ ThirdPartyCustomerInfo
- □ testExport
- □ testImport
- □ testLib

- □ DBMSServiceMediation_ItemServiceIFExport1.wsdl
- □ DBMSServiceMediation_OrderServiceIFExport1.wsdl
- □ EmailServiceIF.wsdl
- □ ItemServiceIF.wsdl
- □ OrderManagementIF.wsdl
- □ OrderManagement_OrderManagementIFExport1.wsdl
- □ OrderServiceIF.wsdl
- ☑ ThirdPartyCustomerInfo_CustomerServiceIFExport1.ws
- □ WarehouseItemSplitFlowIF.wsdl

To directory: C:\itso\WSRR

Options

□ Overwrite existing files without warning

☑ Export dependent resources

□ Merge dependent XSD resources into the parent WSDL file

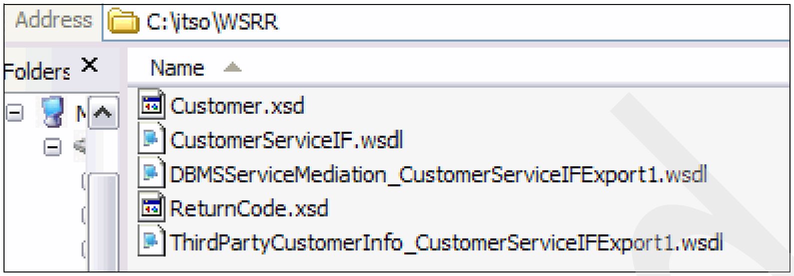
□ Merge dependent WSDL resources into the parent WSDL file

- ThirdPartyCustomerInfo_CustomerServiceIFExport1.wsdl
  - CustomerServiceIF.wsdl

Figure 6-105 Export the service interface

3. Click **Finish**.
4. Repeat this operation for CustomerServiceIFExport1_CustomerServiceIFHttpPort.

5. Verify that the definitions export to the directory correctly as shown in Figure 6-106.



Customer.xsd
CustomerServiceIF.wsdl
DBMSServiceMediation_CustomerServiceIFExport1.wsdl
ReturnCode.xsd
ThirdPartyCustomerInfo_CustomerServiceIFExport1.wsdl

Figure 6-106 Newly exported WSDL

6.5.4 Load the definitions to the registry

Now, you need to upload the two service definitions to the registry. Follow these steps:

1. Make sure that WebSphere Service Registry and Repository is up and running and access the registry. In our case the registry is at the following URL:

   ```
   http://192.168.157.132:9081/ServiceRegistry/
   ```

2. Click **Load Documents** under **Service Documents** (Figure 6-107).

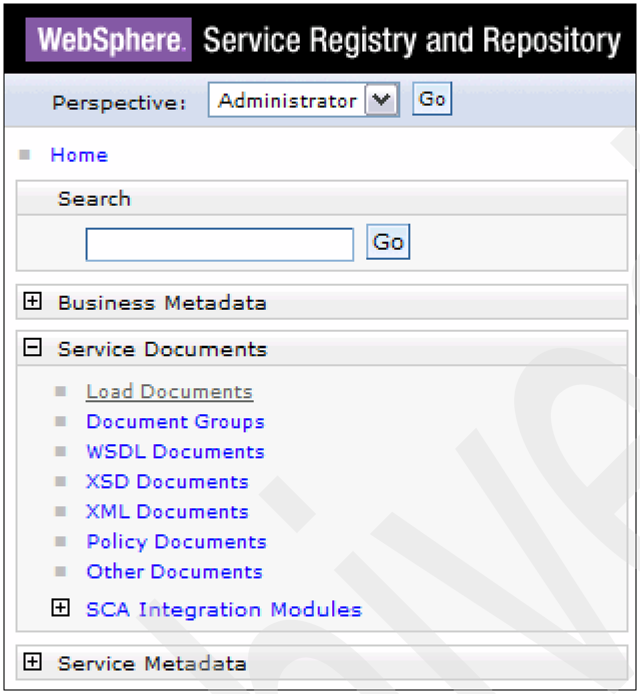


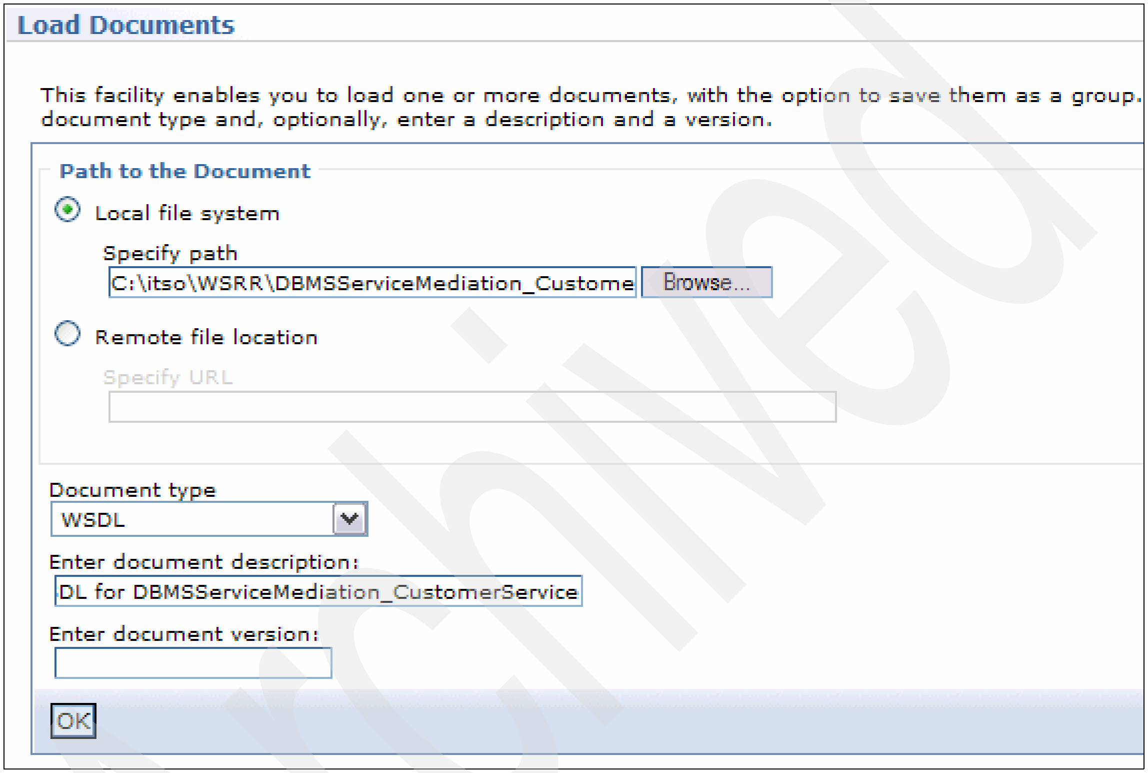
Figure 6-107 Load documents to WebSphere Service Registry and Repository

3. In the Load Documents dialog box (Figure 6-108):

   a. Click **Browse** and navigate to the c:\itso\WSRR directory.

   b. Select **DBMSServiceMediation_CustomerServiceFExport1.wsdl** and click **Open**.

   c. Add a document description. Then, click **OK**.



Load Documents

This facility enables you to load one or more documents, with the option to save them as a group. document type and, optionally, enter a description and a version.

Path to the Document

● Local file system

Specify path: C:\itso\WSRR\DBMSServiceMediation_Custome

○ Remote file location

Specify URL:

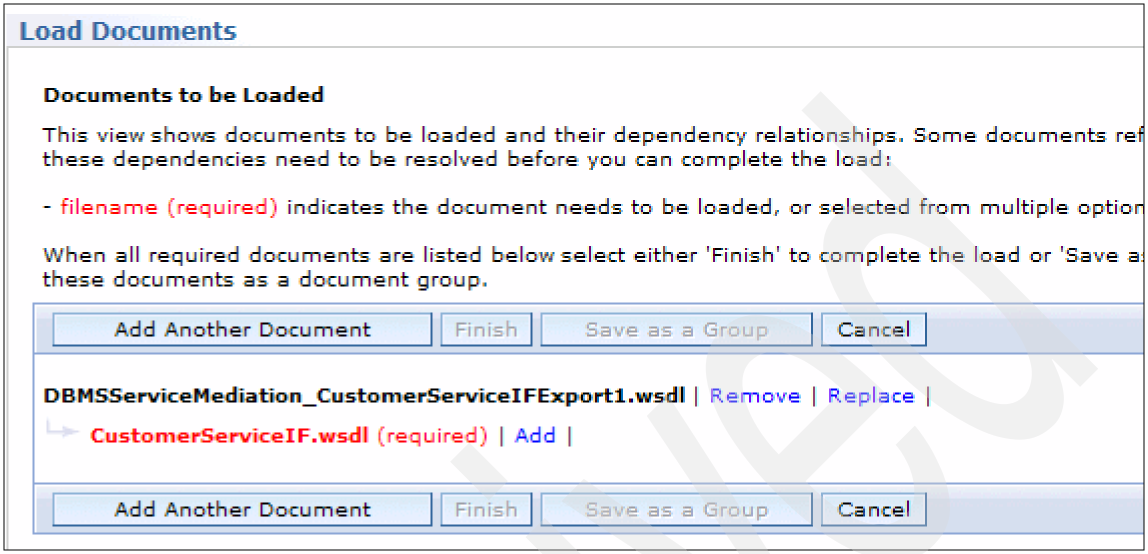
Document type: WSDL

Enter document description: DL for DBMSServiceMediation_CustomerService

Enter document version:

Figure 6-108 Identifying the WSDL to be loaded

The registry discovers that CustomerServiceIF.wsdl is required (Figure 6-109).



Load Documents

Documents to be Loaded

This view shows documents to be loaded and their dependency relationships. Some documents ref these dependencies need to be resolved before you can complete the load:

- filename (required) indicates the document needs to be loaded, or selected from multiple option

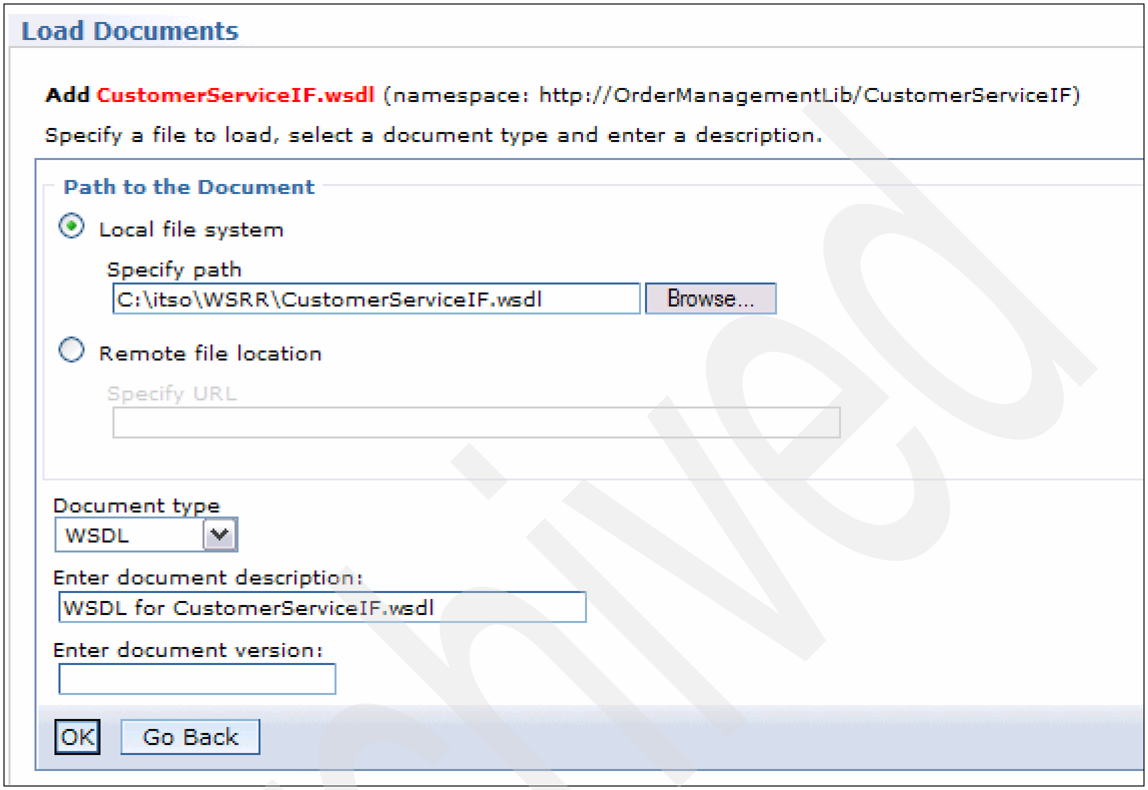
When all required documents are listed below select either 'Finish' to complete the load or 'Save a these documents as a document group.

DBMSServiceMediation_CustomerServiceIFExport1.wsdl | Remove | Replace |

CustomerServiceIF.wsdl (required) | Add |

Figure 6-109 Required WSDL is identified

d. Click **Add** to add the CustomerServiceIF.wsdl file to the list of files to be uploaded as shown in Figure 6-110.



Load Documents

Add CustomerServiceIF.wsdl (namespace: http://OrderManagementLib/CustomerServiceIF)

Specify a file to load, select a document type and enter a description.

Path to the Document

● Local file system

Specify path: C:\itso\WSRR\CustomerServiceIF.wsdl

○ Remote file location

Specify URL:

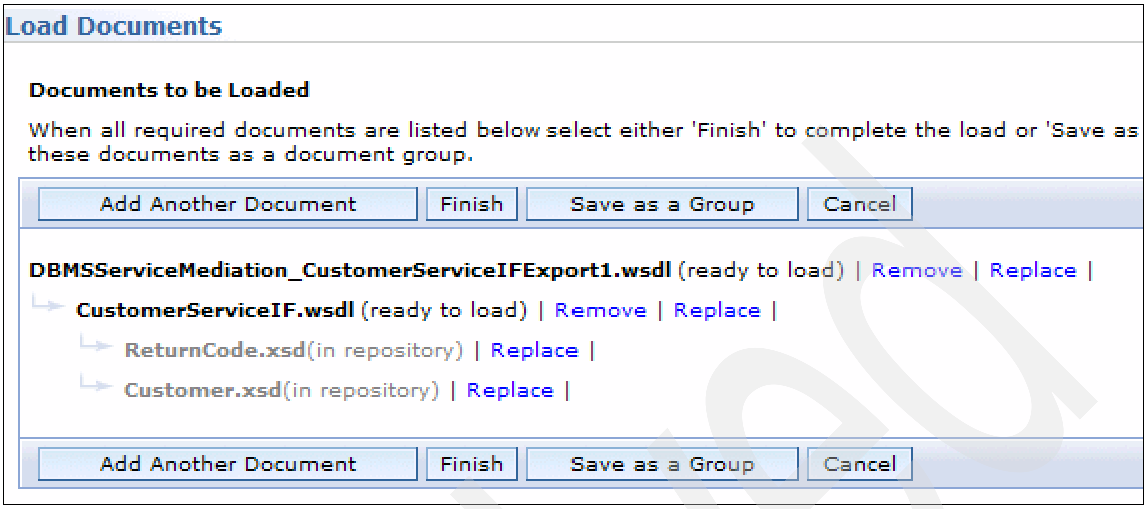
Document type: WSDL

Enter document description: WSDL for CustomerServiceIF.wsdl

Enter document version:

Figure 6-110 Load the required WSDL

e. Click **OK**. Figure 6-111 shows the list of files to be uploaded.



Load Documents

Documents to be Loaded

When all required documents are listed below select either 'Finish' to complete the load or 'Save as
these documents as a document group.

DBMSServiceMediation_CustomerServiceIFExport1.wsdl (ready to load) | Remove | Replace |

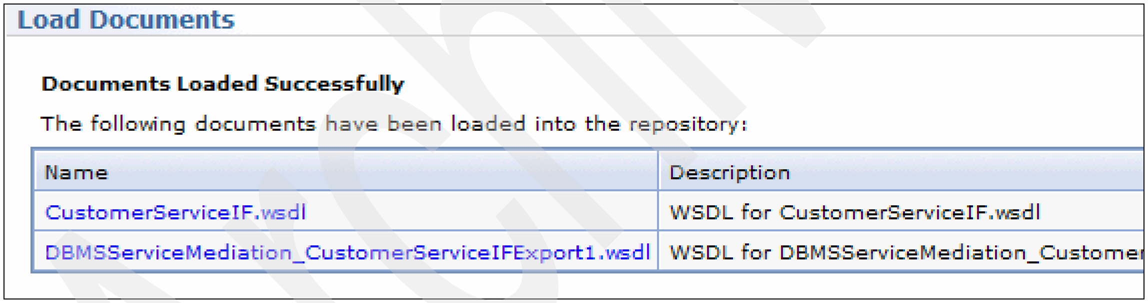
CustomerServiceIF.wsdl (ready to load) | Remove | Replace |

ReturnCode.xsd(in repository) | Replace |

Customer.xsd(in repository) | Replace |

Figure 6-111 List of files to be uploaded

f. Click **Finish** to upload the files.



Load Documents

Documents Loaded Successfully

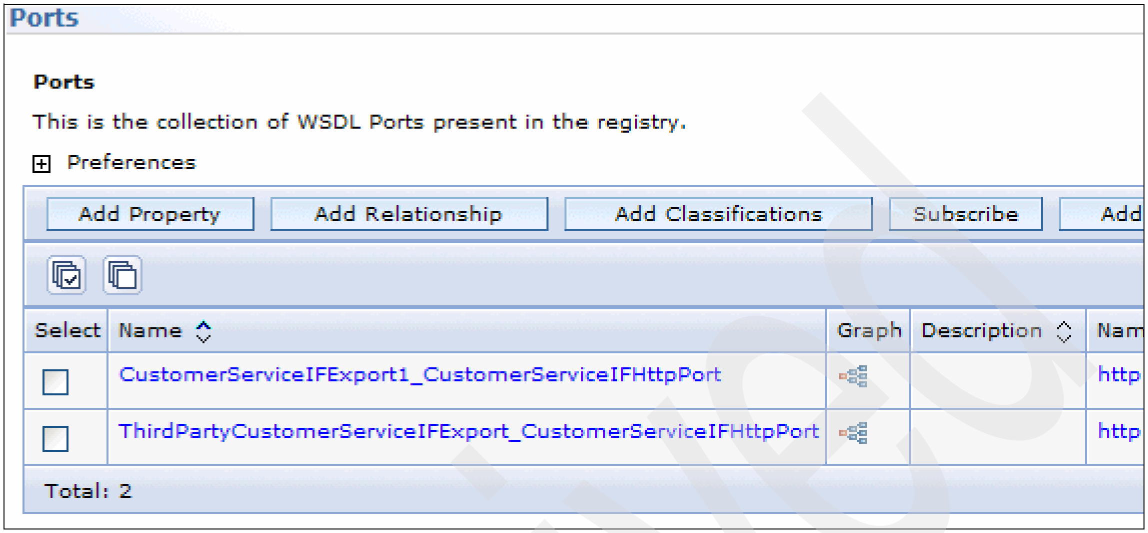
The following documents have been loaded into the repository:

Name	Description
CustomerServiceIF.wsdl	WSDL for CustomerServiceIF.wsdl
DBMSServiceMediation_CustomerServiceIFExport1.wsdl	WSDL for DBMSServiceMediation_Customer

Figure 6-112 Uploaded documents

4. Repeat step 2 to load the WSDL file for ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort.

5. When all the WSDL files are uploaded, select **Service Metadata** → **WSDL** → **Ports** in the navigation pane. The window shown in Figure 6-113 opens.



Ports

Ports

This is the collection of WSDL Ports present in the registry.

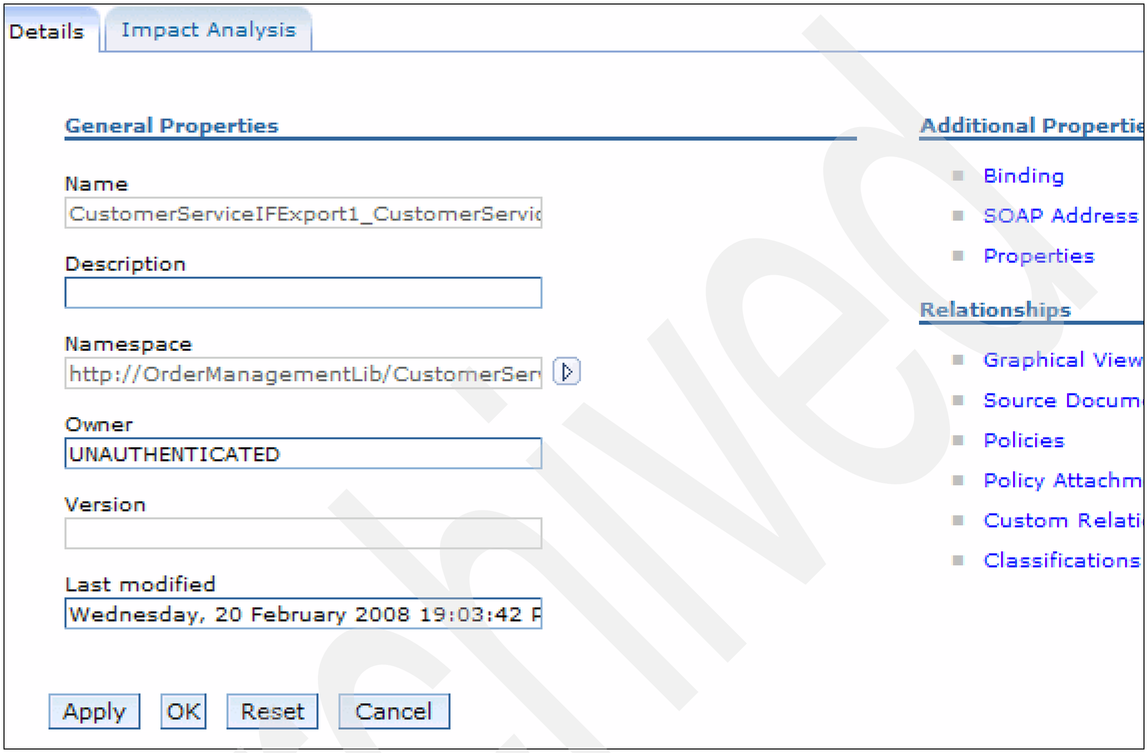
Select	Name	Graph	Description	Nam
□	CustomerServiceIFExport1_CustomerServiceIFHttpPort			http
□	ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort			http

Total: 2

Figure 6-113 Ports in the registry

6. Now, create a new property for each of the services. The registry uses the property value for selecting the right service. Follow these steps:

   a. Click **CustomerServiceIFExport1_CustomerServiceIFHttpPort** to open the Details page (Figure 6-114).



General Properties

Name: CustomerServiceIFExport1_CustomerServic

Description:

Namespace: http://OrderManagementLib/CustomerSer

Owner: UNAUTHENTICATED

Version:

Last modified: Wednesday, 20 February 2008 19:03:42 P

Additional Propertie

- Binding
- SOAP Address
- Properties

Relationships

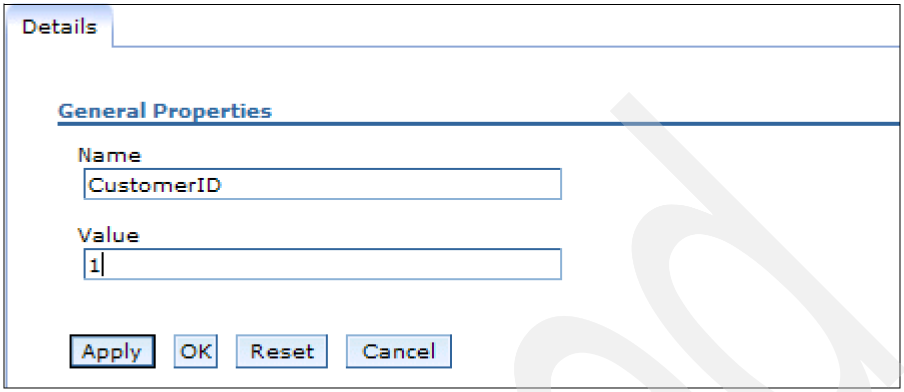
- Graphical View
- Source Docum
- Policies
- Policy Attachm
- Custom Relati
- Classifications

Figure 6-114 Detail properties for the port

   b. Click **Properties** under **Additional Properties**.

   c. Click **New**.

d. Create a property called `CustomerID`, and set its value to `1`, as shown in Figure 6-115.



General Properties

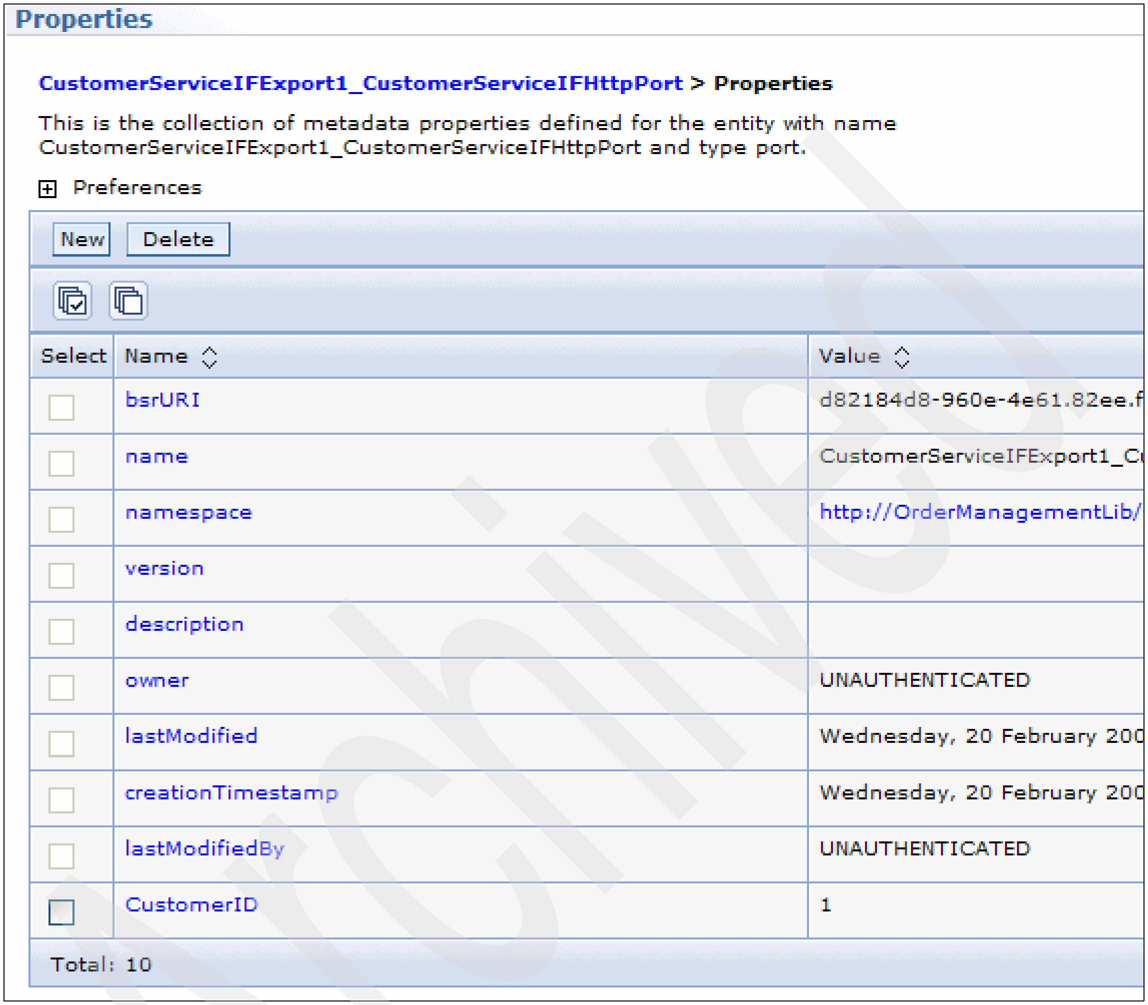
Name: CustomerID

Value: 1

Figure 6-115 CustomerID property

e. Click **OK**.

The new CustomerID displays as shown in Figure 6-116.



Properties

CustomerServiceIFExport1_CustomerServiceIFHttpPort > **Properties**

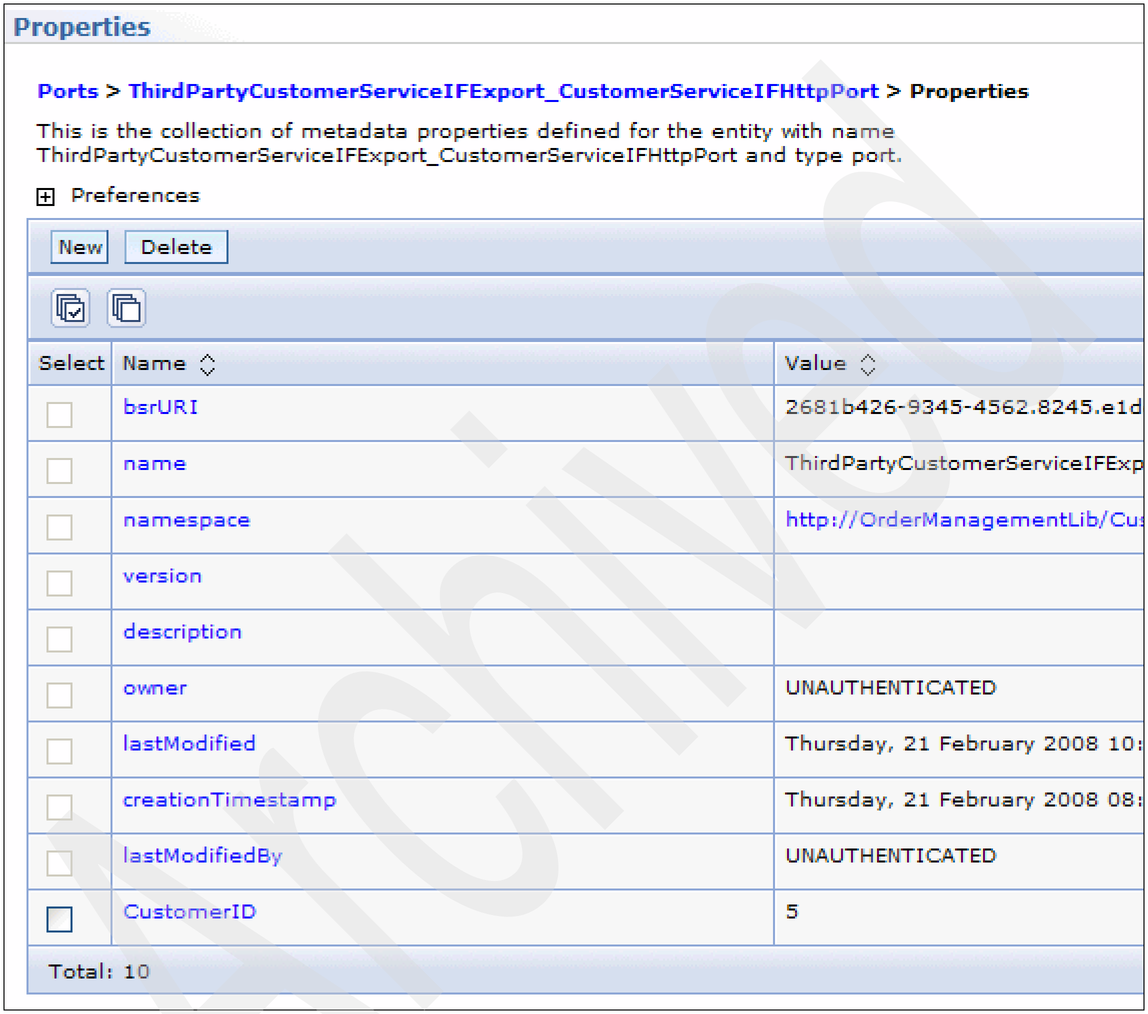
This is the collection of metadata properties defined for the entity with name CustomerServiceIFExport1_CustomerServiceIFHttpPort and type port.

Select	Name	Value
□	bsrURI	d82184d8-960e-4e61.82ee.f
□	name	CustomerServiceIFExport1_C
□	namespace	http://OrderManagementLib/
□	version	
□	description	
□	owner	UNAUTHENTICATED
□	lastModified	Wednesday, 20 February 200
□	creationTimestamp	Wednesday, 20 February 200
□	lastModifiedBy	UNAUTHENTICATED
□	CustomerID	1

Total: 10

Figure 6-116   The new CustomerID property in the port

7. Repeat the same operation for ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort. Assign 5 to its CustomerID property. See Figure 6-117.



Properties

Ports > ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort > **Properties**

This is the collection of metadata properties defined for the entity with name ThirdPartyCustomerServiceIFExport_CustomerServiceIFHttpPort and type port.

⊞ Preferences

Select	Name	Value
□	bsrURI	2681b426-9345-4562.8245.e1d
□	name	ThirdPartyCustomerServiceIFExp
□	namespace	http://OrderManagementLib/Cu:
□	version	
□	description	
□	owner	UNAUTHENTICATED
□	lastModified	Thursday, 21 February 2008 10:
□	creationTimestamp	Thursday, 21 February 2008 08:
□	lastModifiedBy	UNAUTHENTICATED
□	CustomerID	5

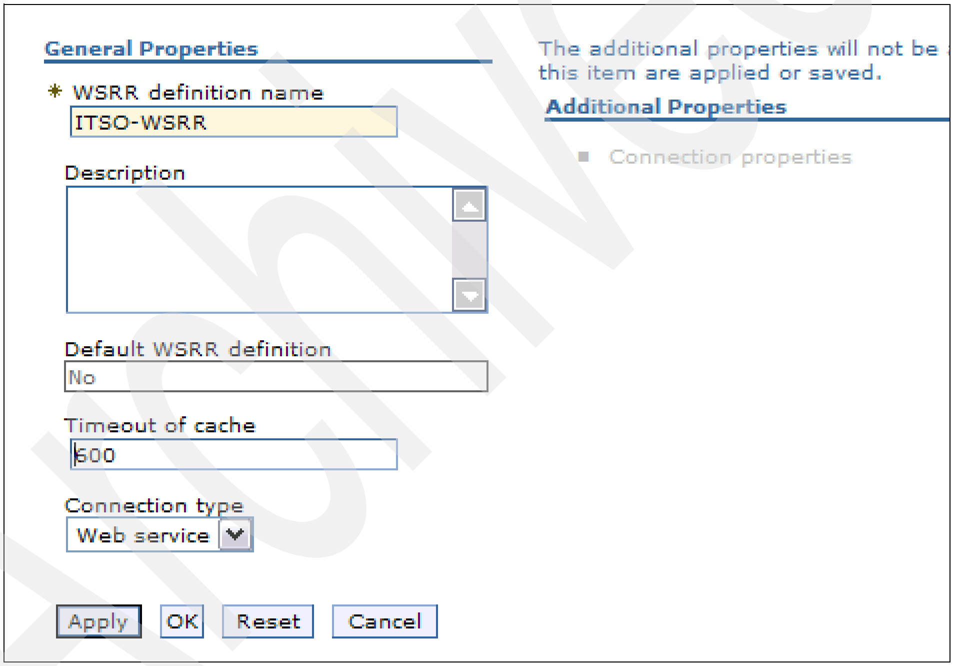
Total: 10

Figure 6-117 The new CustomerID property in the port

6.5.5 Configure the registry to WebSphere Enterprise Service Bus

To configure access to the registry from WebSphere Enterprise Service Bus, follow these steps:

1. Open the administrative console for the WebSphere Enterprise Service Bus server.
2. Expand **Service integration** and select **WSRR definitions**.
3. Click **New** to create a new definition.
4. Set the WebSphere Service Registry and Repository definition name to `ITSO-WSRR`, and increase the Timeout of cache to `600`, as shown in Figure 6-118.
5. Click **Apply**.



General Properties

* WSRR definition name: ITSO-WSRR

Description:

Default WSRR definition: No

Timeout of cache: 600

Connection type: Web service

The additional properties will not be this item are applied or saved.

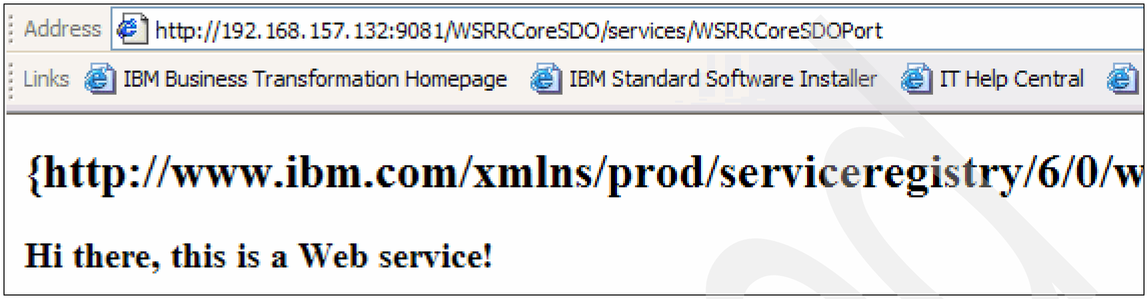
Additional Properties

- Connection properties

Figure 6-118 Define a new WSRR connection

6. **Click Connection properties** under **Additional Properties.**

It is a good practice to make sure that you know the correct URL to use as a connection to the registry by pointing to the URL with a browser as shown in Figure 6-119.



Address http://192.168.157.132:9081/WSRRCoreSDO/services/WSRRCoreSDOPort

Links
IBM Business Transformation Homepage
IBM Standard Software Installer
IT Help Central

{http://www.ibm.com/xmlns/prod/serviceregistry/6/0/w

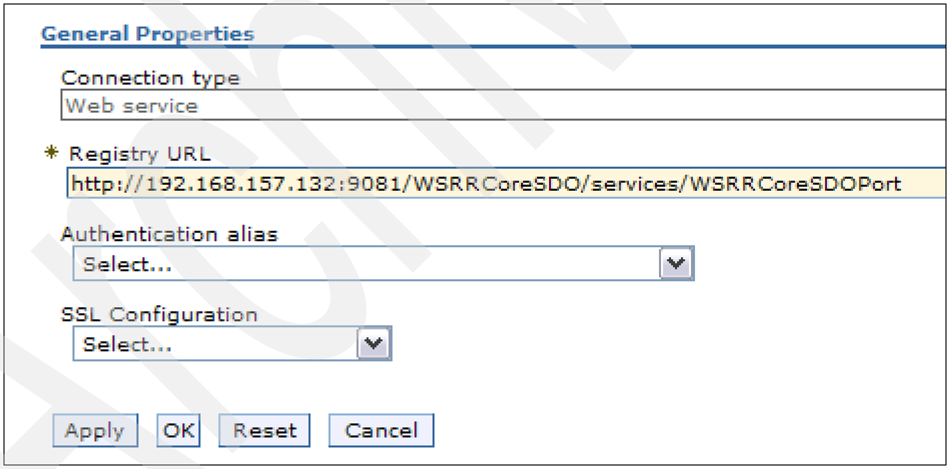
Hi there, this is a Web service!

Figure 6-119 Checking the registry port

Note that this example does not use a secured registry.

Set the Registry URL to the following address (as shown in Figure 6-120):

```
http://192.168.157.132:9081/WSRRCoreSDO/services/WSRRCoreSDOPort
```



General Properties

Connection type
Web service

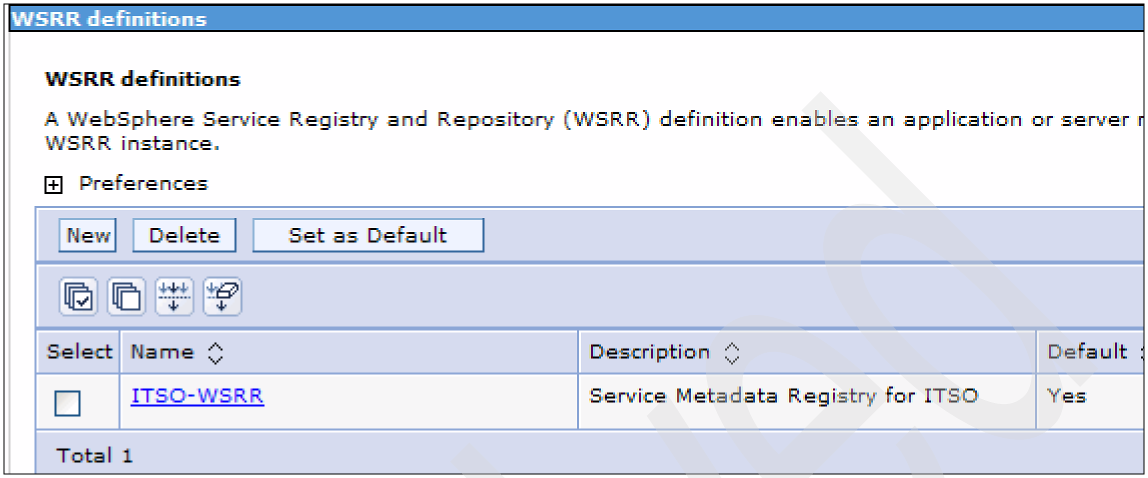
* Registry URL
http://192.168.157.132:9081/WSRRCoreSDO/services/WSRRCoreSDOPort

Authentication alias
Select...

SSL Configuration
Select...

Figure 6-120 Connection properties

7. Click **OK** to save all the changes. You now have an ITSO-WSRR service definition as shown in Figure 6-121.



WSRR definitions

A WebSphere Service Registry and Repository (WSRR) definition enables an application or server r WSRR instance.

Select	Name	Description	Default
☐	ITSO-WSRR	Service Metadata Registry for ITSO	Yes

Total 1

Figure 6-121 New WSRR definition

6.5.6 Create the mediation module

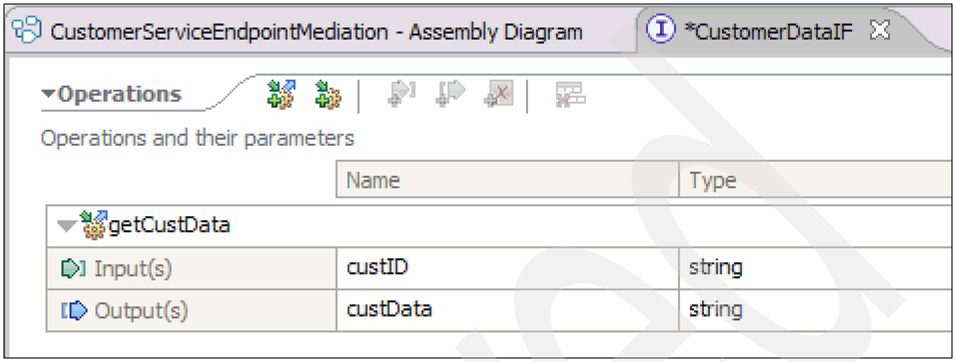
Now that the services are in place, you create a mediation module that can choose between two (or more) implementations of CustomerServiceIF interface.

To create the mediation module:

1. You create a new mediation module called `CustomerServiceEndpointMediation`.
2. Then, select WebSphere Enterprise Service Bus as the target run time.
3. Finally, choose OrderManagementLib as the required library.

6.5.7 Create the interface

Create a CustomerDataIF interface with one getCustData operation that takes a custID string for input and returns a string of custData, as shown in Figure 6-122.



CustomerServiceEndpointMediation - Assembly Diagram | *CustomerDataIF

Operations

Operations and their parameters

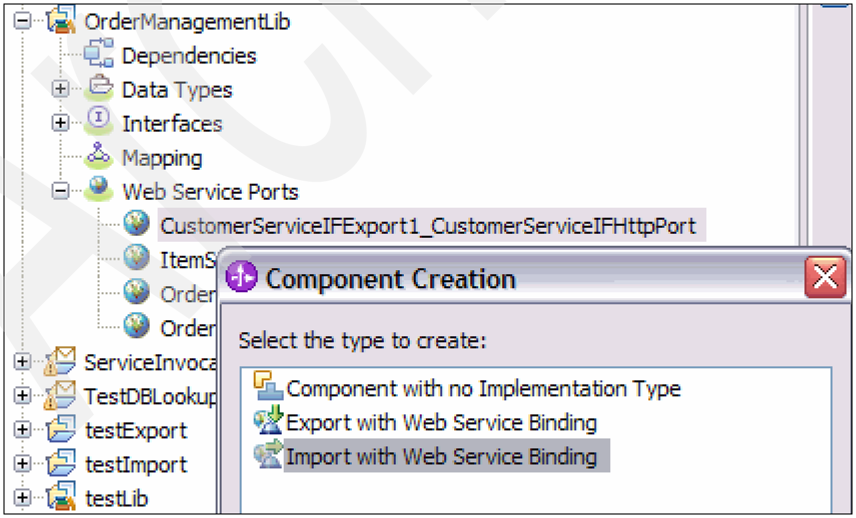
	Name	Type
getCustData		
Input(s)	custID	string
Output(s)	custData	string

Figure 6-122 CustomerDataIF interface

6.5.8 Build the assembly diagram

Next, populate the assembly diagram as follows:

1. Add CustomerDataIF to the CustomerServiceEndpointMediation component.
2. Import CustomerServiceIFExport1_CustomerServiceIFHttpPort with a Web service binding, as shown in Figure 6-123.



```
OrderManagementLib
    Dependencies
    Data Types
    Interfaces
    Mapping
    Web Service Ports
        CustomerServiceIFExport1_CustomerServiceIFHttpPort
        ItemS
        Order
        Order
ServiceInvoca
TestDBLookup
testExport
testImport
testLib
```

Component Creation

Select the type to create:

- Component with no Implementation Type
- Export with Web Service Binding
- Import with Web Service Binding

Figure 6-123 Create the Import component

3. Wire CustomerServiceEndpointMediation to CustomerServiceIFExport1_CustomerServiceIFHttpPort (Figure 6-124).

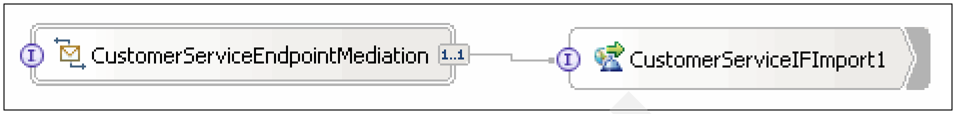


Figure 6-124 Assembly diagram

4. Regenerate the implementation.

6.5.9 Build the operation connections

In the Mediation Flow Editor, wire getCustData operation to retrieveCustomer operation, as shown in Figure 6-125.

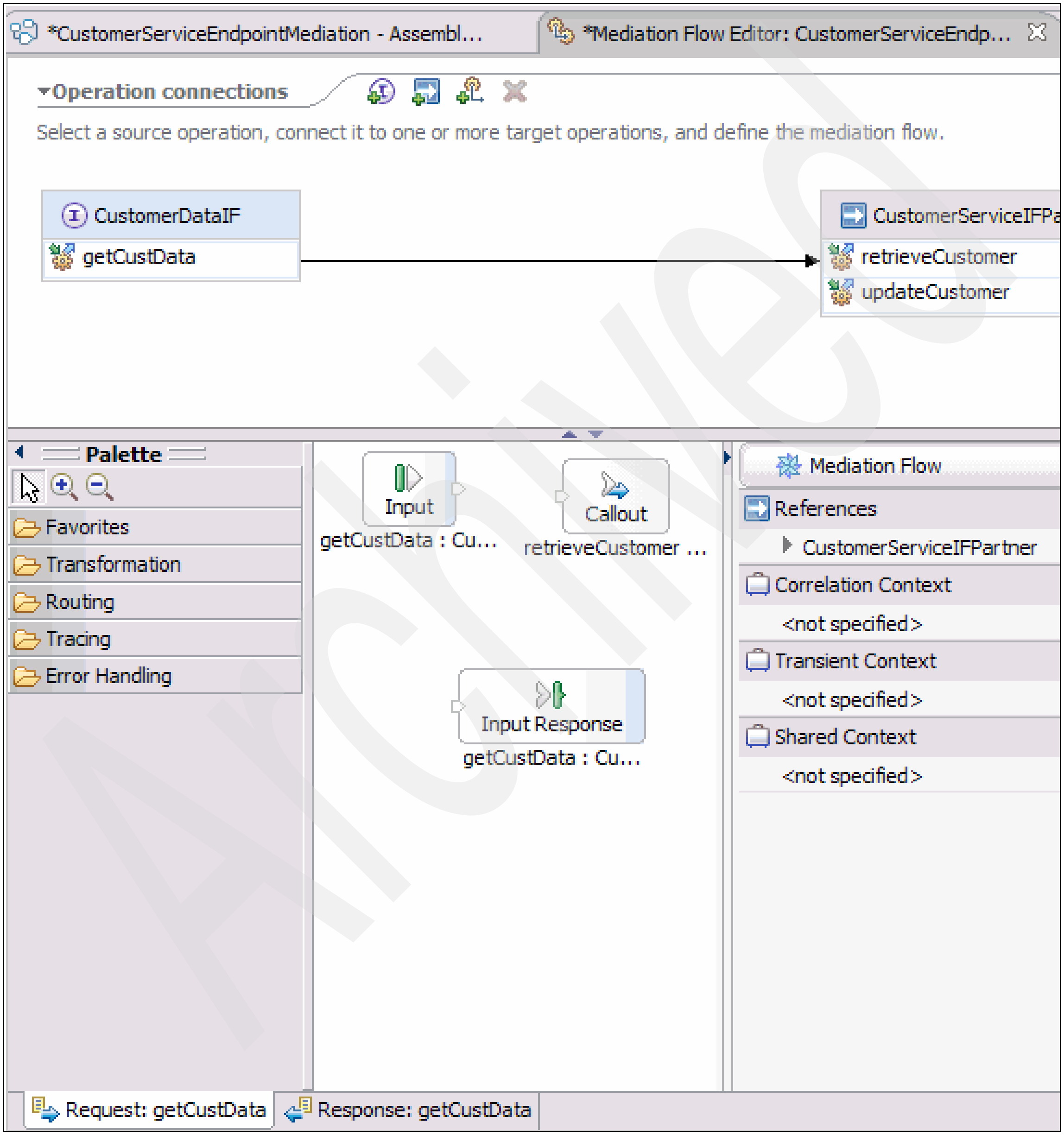


Figure 6-125 Connections and mediation flow

6.5.10 Build the request flow

The request flow looks similar to that shown in Figure 6-126.

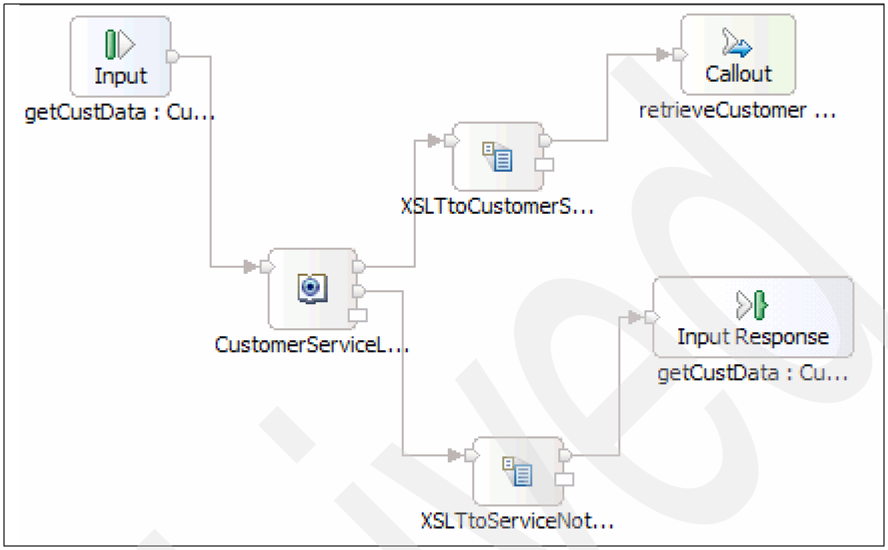
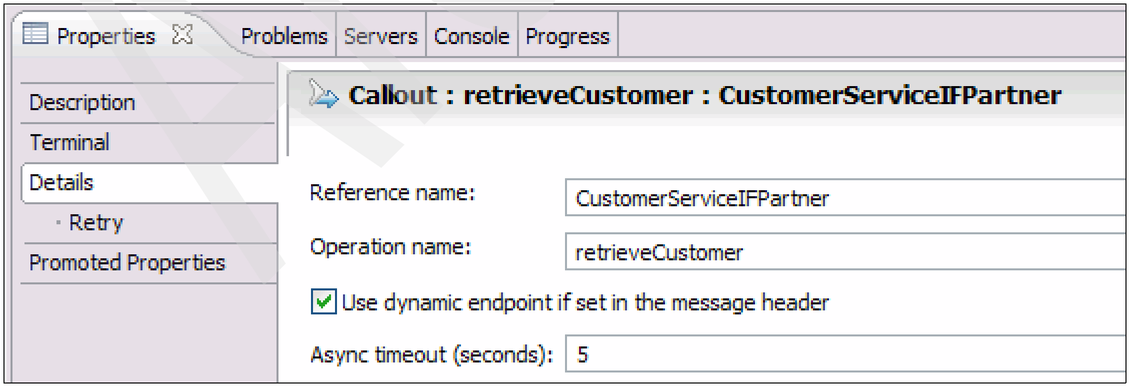


Figure 6-126 Request flow

Update the Callout node

Update the Callout node to use a dynamic endpoint for the service call. Follow these steps:

1. Click the **Callout** node to open its properties.
2. On the Details tab, make sure that the “Use dynamic endpoint if set in the message header” option is selected, as shown in Figure 6-127.



Callout : retrieveCustomer : CustomerServiceIFPartner

Reference name: CustomerServiceIFPartner

Operation name: retrieveCustomer

☑ Use dynamic endpoint if set in the message header

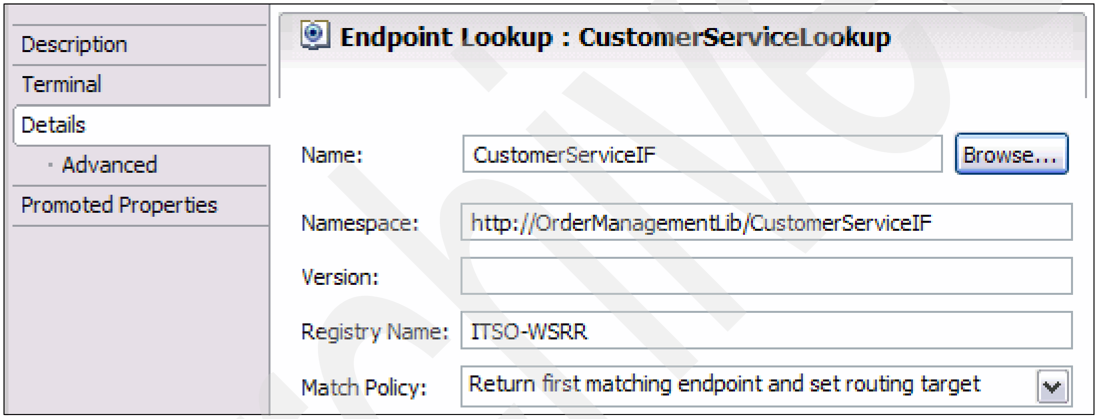
Async timeout (seconds): 5

Figure 6-127 Callout properties: Use dynamic endpoint

Endpoint Lookup primitive

Add the Endpoint Lookup primitive, as follows:

1. Find the Endpoint Lookup primitive under the Routing folder in the palette and drop it to the canvas. Name it `CustomerServiceLookup`.
2. Wire the Input node to CustomerServiceLookup
3. Go to the Details tab of the CustomerServiceLookup Properties (Figure 6-128).
   a. Click **Browse** to search for the service interface to use. Select CustomerServiceIF.
   b. Enter `ITSO-WSRR` as the Registry Name**.**
   c. Set the Match Policy to **Return first matching endpoint and set routing target**.



Endpoint Lookup : CustomerServiceLookup

Name: CustomerServiceIF

Namespace: http://OrderManagementLib/CustomerServiceIF

Version:

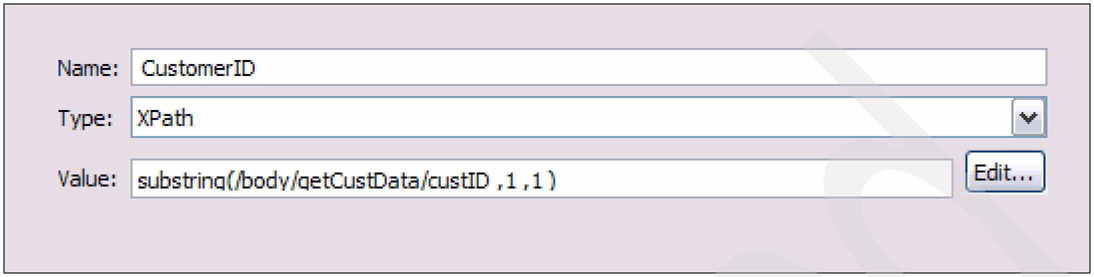
Registry Name: ITSO-WSRR

Match Policy: Return first matching endpoint and set routing target

Figure 6-128 Endpoint Lookup properties

4. Click **Advanced** under the Details tab.
5. Click **Add** under **User Properties**.

6. Create the CustomerID property and set a value to it. The property name has to match the property name that you defined previously on the WebSphere Service Registry and Repository server. For the value, take the first digit of the customer ID. See Figure 6-129.



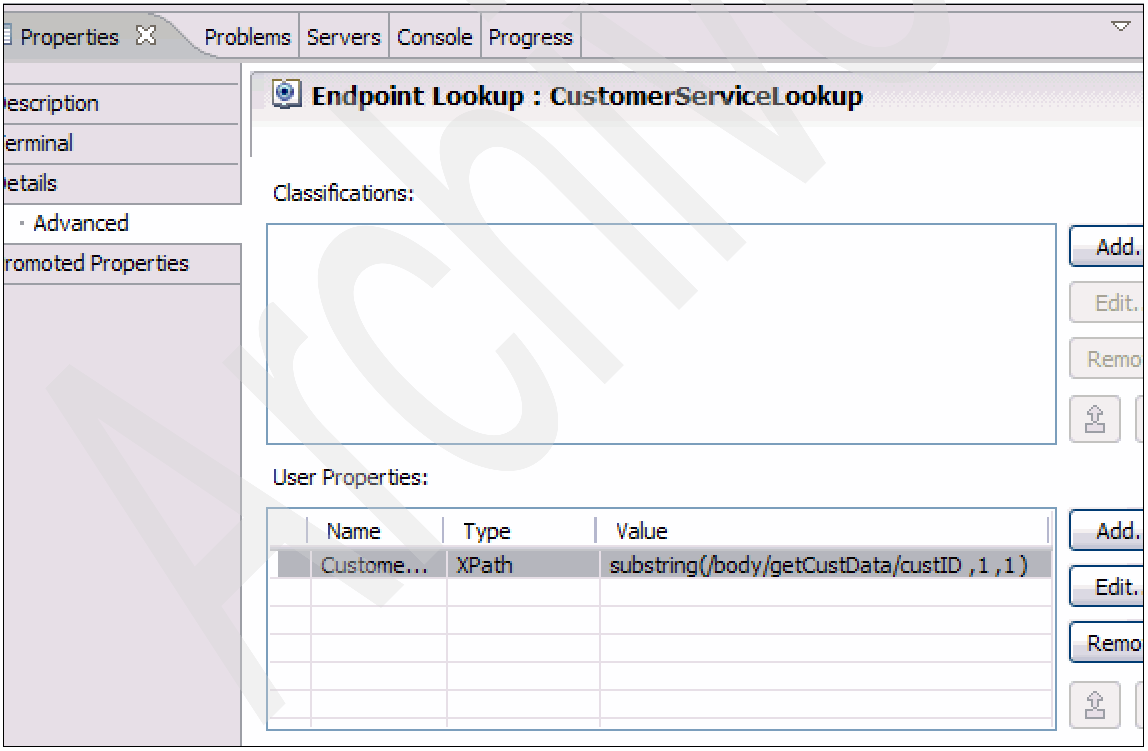
Name: CustomerID

Type: XPath

Value: substring(/body/getCustData/custID ,1 ,1)

Figure 6-129 Define a new property

7. Click **Finish**. The properties display as shown in Figure 6-130.



Endpoint Lookup : CustomerServiceLookup

Classifications:

User Properties:

Name	Type	Value
Custome...	XPath	substring(/body/getCustData/custID ,1 ,1)

Figure 6-130 Endpoint Lookup properties

XSL Transformation primitives

Add the two XSL Transformation primitives as follows:

1. Add an XSL Transformation primitive to the mediation flow, and call it `XSLTtoCustomerService`. Wire the out terminal of the CustomerServiceLookup node to the in terminal of XSLTtoCustomerService. Wire XSLTtoCustomerService to the Callout node.
2. Add another XSL Transformation primitive to the canvas, and call it `XSLTtoServiceNotFound`. Wire it between the noMatch terminal of the CustomerServiceLookup node and Input Response node.
3. Open the **XSLTtoServiceNotFound** map. Use a Custom mapping between custID and custData, as shown in Figure 6-131.

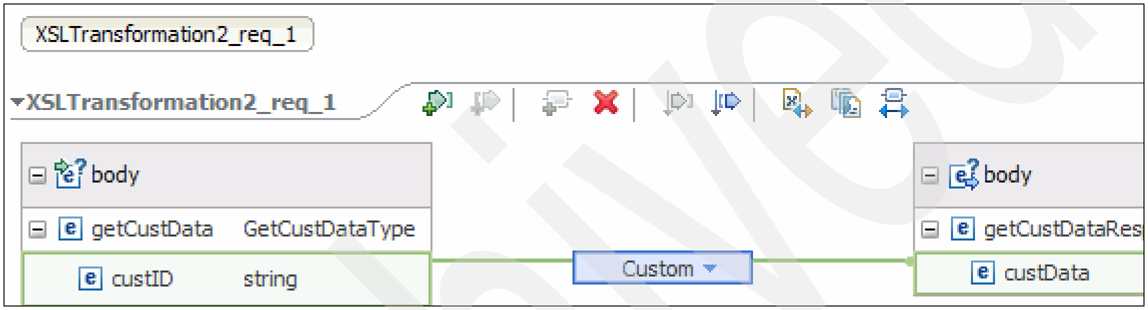


Figure 6-131 XSLTtoServiceNotFound mapping

Use the following XPath expression:

```
concat('Data for customerID=', /body/getCustData/custID, ' not
found')
```

4. Open the XSLTtoCustomerService Map. Use a Move mapping between custID in the source and custID field of customer on the target, as shown in Figure 6-132.
5. Save the changes.

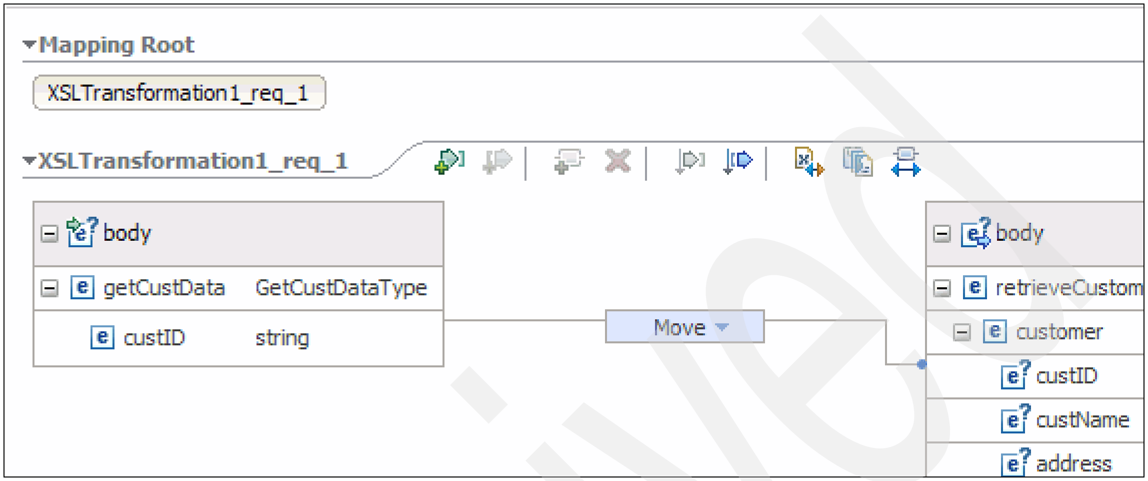
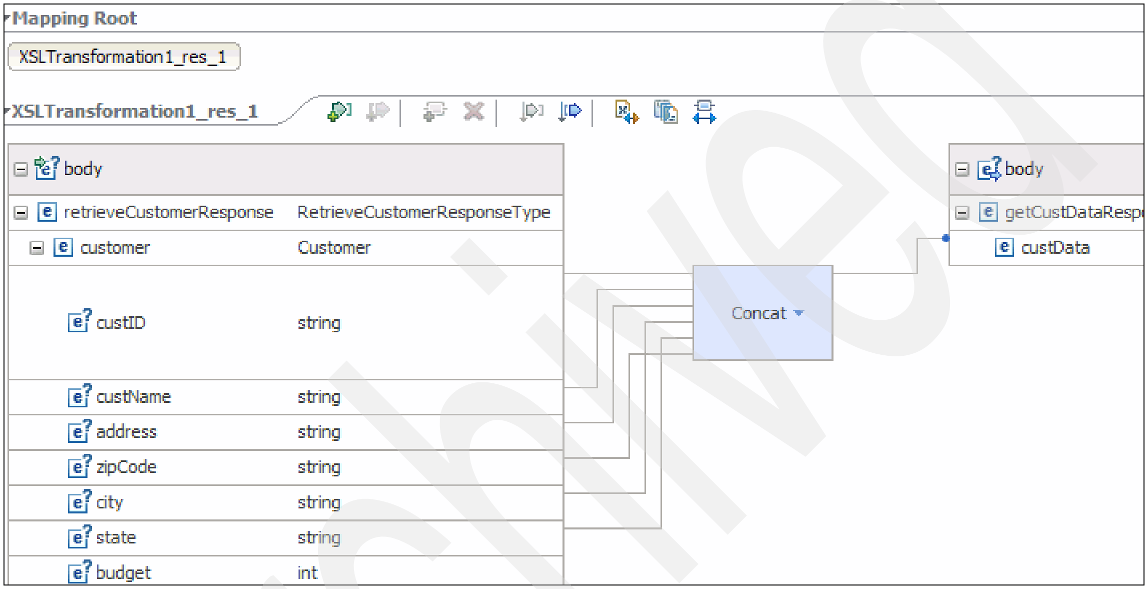


Figure 6-132   XSLTtoCustomerService mapping

6.5.11 Build the response flow

To build the response flow:

1. Go to the Response:getCustData tab.
2. Add an XSL Transformation primitive to the canvas.
3. Map all fields of the source to one field of the target using the Concat mapping action, as shown in Figure 6-133.



Mapping Root

XSLTransformation1_res_1

XSLTransformation1_res_1

body	
retrieveCustomerResponse	RetrieveCustomerResponseType
customer	Customer
custID	string
custName	string
address	string
zipCode	string
city	string
state	string
budget	int

Concat

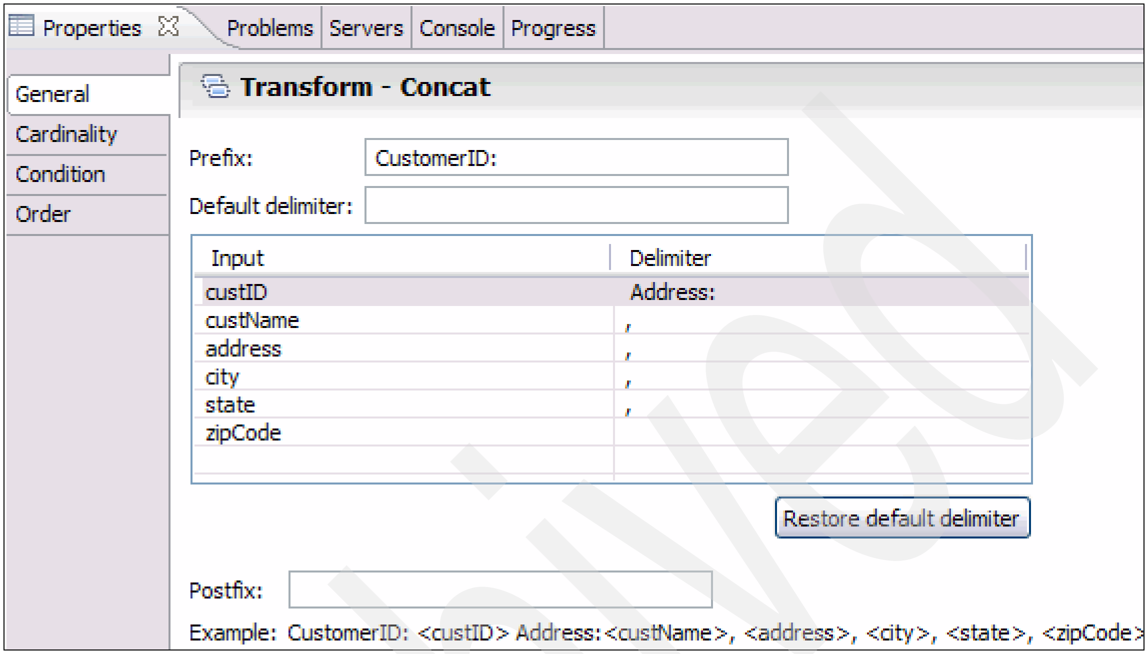
body

getCustDataResp

custData

Figure 6-133 XSL Transformation mapping

4. Click **Concat** and add delimiters and a prefix to better format the result (Figure 6-134).



Transform - Concat

Prefix: CustomerID:

Default delimiter:

Input	Delimiter
custID	Address:
custName	,
address	,
city	,
state	,
zipCode	

Postfix:

Example: CustomerID: <custID> Address:<custName>, <address>, <city>, <state>, <zipCode>

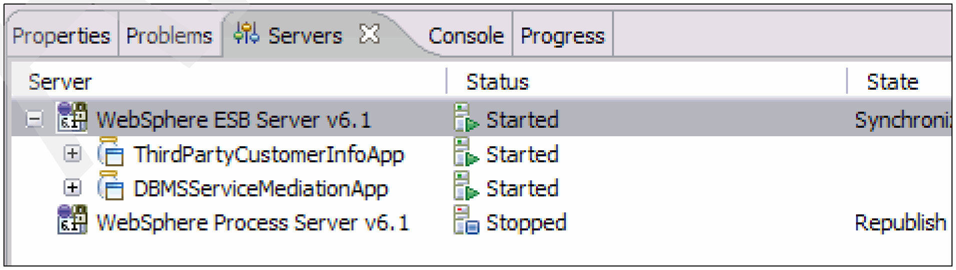
Figure 6-134 Add delimiters and a prefix to the Concat mapping action

5. Save all by selecting **File** → **Save All**.

6.5.12 Test the flow

To test the flow, follow these steps:

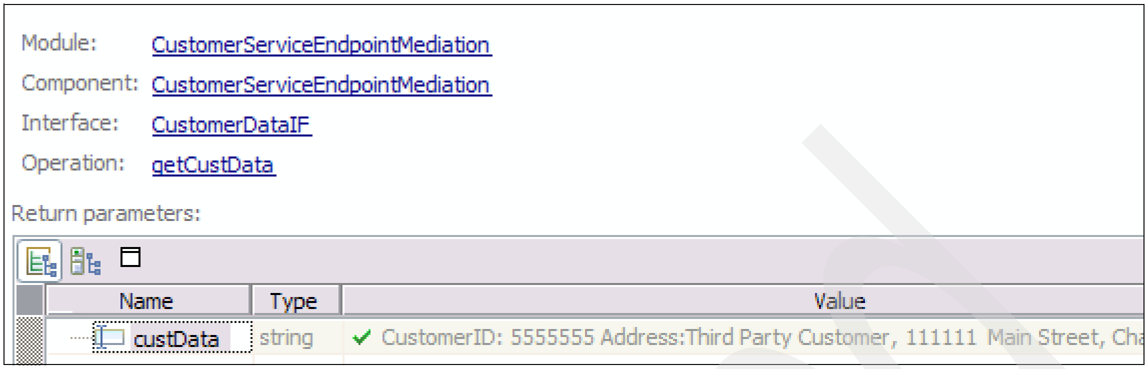
1. Make sure that the WebSphere Enterprise Service Bus server is up and running and both Web services are deployed and started. See Figure 6-135.



Server	Status	State
WebSphere ESB Server v6.1	Started	Synchroni
ThirdPartyCustomerInfoApp	Started	
DBMSServiceMediationApp	Started	
WebSphere Process Server v6.1	Stopped	Republish

Figure 6-135 Start the applications

2. Run the test using custID `5555555`. Figure 6-136 shows the results.



Module: CustomerServiceEndpointMediation

Component: CustomerServiceEndpointMediation

Interface: CustomerDataIF

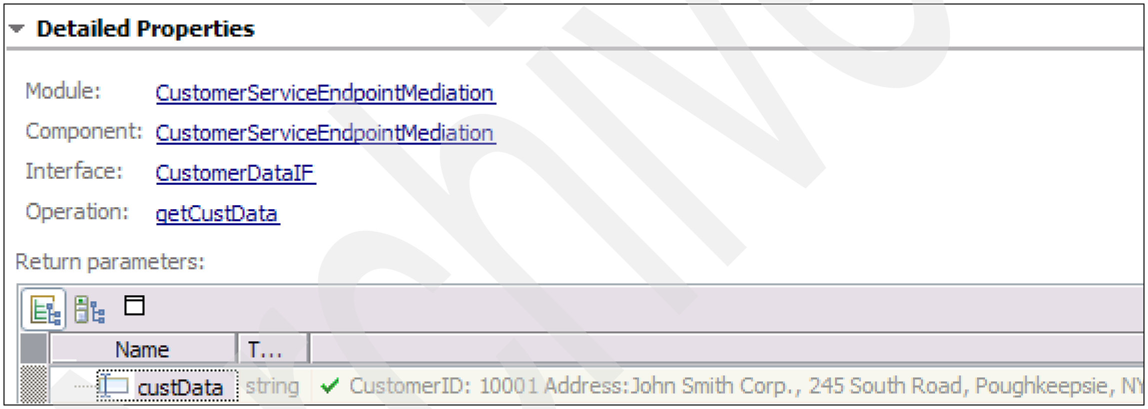
Operation: getCustData

Return parameters:

Name	Type	Value
custData	string	✓ CustomerID: 5555555 Address:Third Party Customer, 111111 Main Street, Cha

Figure 6-136 Test results

3. Run the test using custID `10001`. Figure 6-137 shows the results.



Detailed Properties

Module: CustomerServiceEndpointMediation

Component: CustomerServiceEndpointMediation

Interface: CustomerDataIF

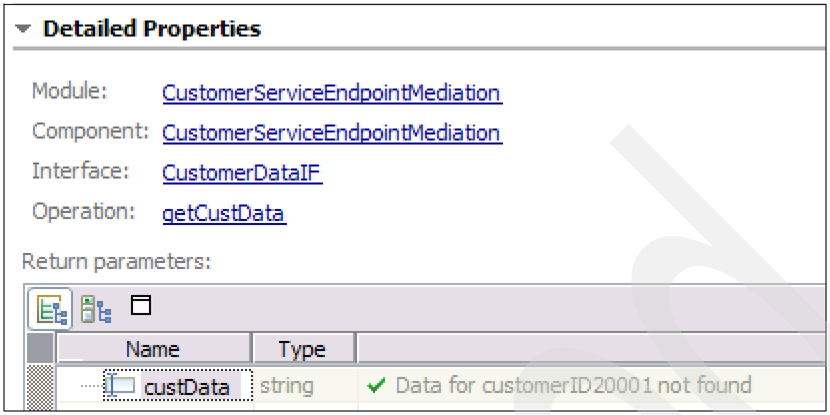
Operation: getCustData

Return parameters:

Name	T...	
custData	string	✓ CustomerID: 10001 Address:John Smith Corp., 245 South Road, Poughkeepsie, NY

Figure 6-137 Test results

4. Run the test using custID `20001`. Figure 6-138 shows the results.



Detailed Properties

Module: CustomerServiceEndpointMediation
Component: CustomerServiceEndpointMediation
Interface: CustomerDataIF
Operation: getCustData

Return parameters:

Name	Type	
custData	string	✓ Data for customerID20001 not found

Figure 6-138 Test results

6.6 The Event Emitter primitive

> **Additional material:** This example is included in the EventEmitterMediation project in the additional materials.

This sample illustrates the use of the Event Emitter primitive. In this sample, a mediation module called EventEmitterMediation uses a message filter to determine how to forward a message through the mediation flow.

If the customer ID in the message starts with a *1*, the mediation flow calls the retrieveCustomer operation of the DBMSServiceMediation. If the customer ID starts with any number other than *1*, the mediation flow emits an event and stops the mediation.

To build the mediation:

1. Create a mediation module called `EventEmitterMediation`.
2. Import the CustomerServiceIF Web service from OrderManagementLib by selecting **CustomerServiceIFExport1_CustomerServiceIFHttpPort** from the Web Service Ports folder in OrderManagementLib and dropping it to the assembly diagram.

   By default, it is assigned a name of **CustomerServiceIFImport1**.

3. Wire **EventEmitterMediation** to **CustomerServiceIFImport1** and add the **CustomerServiceIF** interface to EventEmitterMediation, as shown in Figure 6-139.

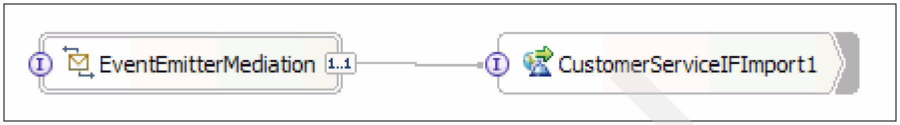


Figure 6-139 EventEmitterMediation assembly diagram

4. Right-click EventEmitterMediation and select **Regenerate Implementation** from the pop-up menu to open the Mediation Flow Editor with the new mediation flow.

6.6.1 Operation connections

Wire **retrieveCustomer** to **retrieveCustomer**, as shown in Figure 6-140.

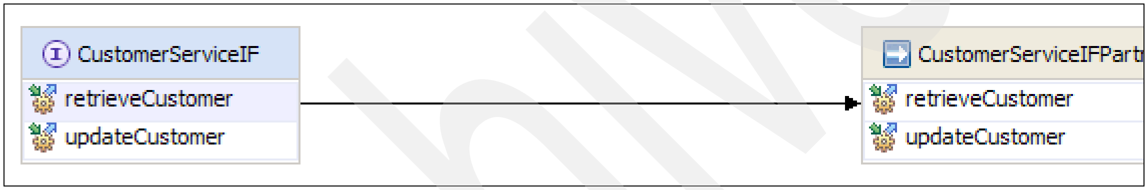


Figure 6-140 Wire the operations

6.6.2 Build the mediation flow

The mediation flow looks similar to that shown in Figure 6-141.

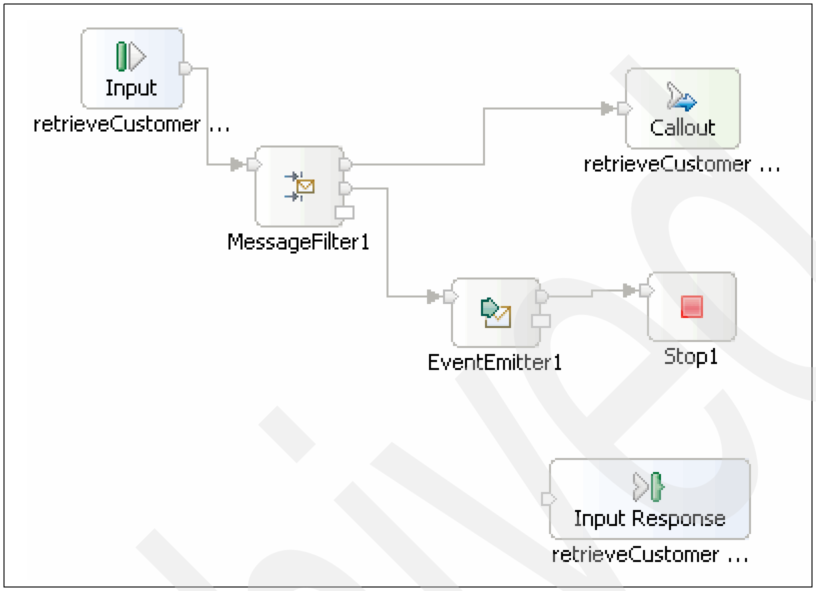


Figure 6-141 EventEmitterMediation mediation flow

Add the following primitives to the mediation flow canvas and take the default names:

- Message Filter
- Event Emitter
- Stop

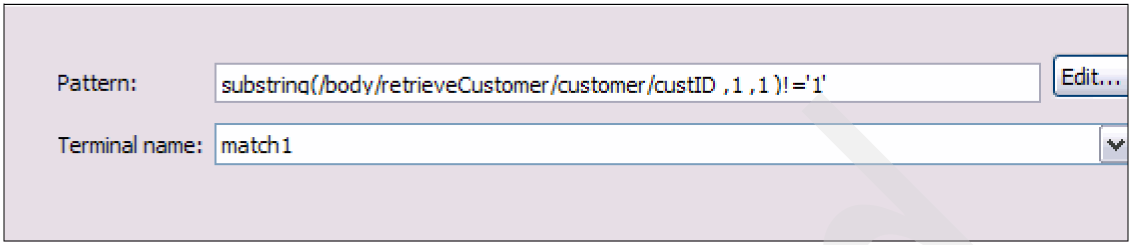
Message Filter primitive

The Message Filter primitive is used to validate the first character of the customerID. If the first character is not *1*, the flow emits an event and stops the mediation. Otherwise, it proceeds with a Web service call.

To build the mediation:

1. Wire the Input node to the input terminal of the MessageFilter1.
2. Right-click **MessageFilter1** and select **Add Output Terminal** from the pop-up menu. The new terminal is named `match1`.
3. Select the MessageFilter1 primitive, and go to the Details tab of the Properties view. Click **Add** to add a new pattern.

4. Set the pattern to compare the first character of custID to *1*. Set the terminal name to `match1`, and click **Finish**. See Figure 6-142.

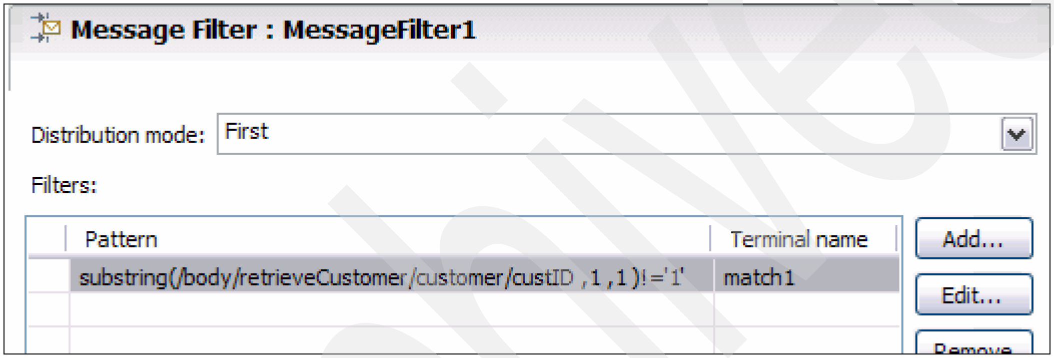


Pattern: substring(/body/retrieveCustomer/customer/custID ,1 ,1)!='1'

Terminal name: match1

Figure 6-142 Add a pattern to the Message Filter primitive

5. In the Details tab, set the distribution mode to **First**, as shown in Figure 6-143.



Message Filter : MessageFilter1

Distribution mode: First

Filters:

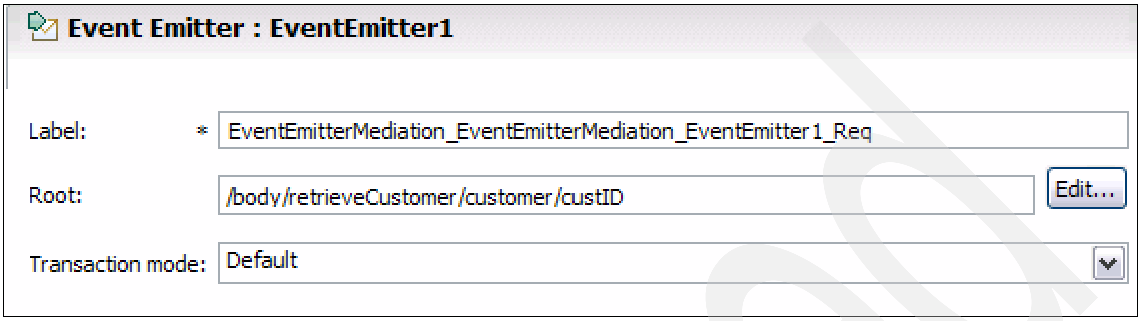
Pattern	Terminal name
substring(/body/retrieveCustomer/customer/custID ,1 ,1)!='1'	match1

Figure 6-143 New pattern and distribution mode

6. Complete the wiring in the flow as follows:
   - Wire the match1 terminal to EventEmitter1.
   - Wire the default terminal of MessageFilter1 to the Callout node.
   - Wire the output terminal of the EventEmitter1 to the Stop1 node.

Event Emitter primitive

In the Details tab of the Properties view for the EventEmitter1, set the Root value to /body/retrieveCustomer/customer/custID, as shown in Figure 6-144. Save the mediation and assembly diagram.



Event Emitter : EventEmitter1

Label: * EventEmitterMediation_EventEmitterMediation_EventEmitter1_Req

Root: /body/retrieveCustomer/customer/custID

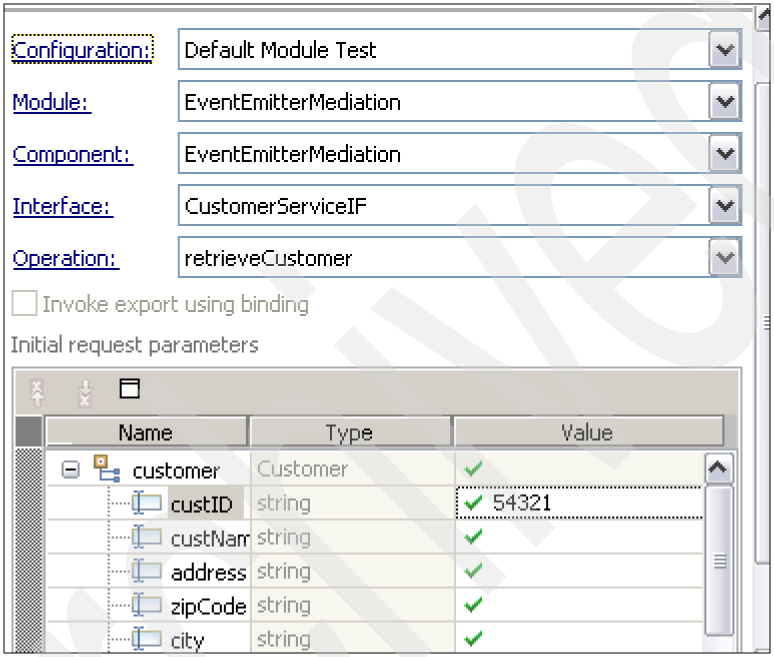
Transaction mode: Default

Figure 6-144 Event Emitter properties

6.6.3 Test the mediation

To test the mediation:

1. In the assembly diagram, right-click the EventEmitterMediation component and choose **Test Component** from the pop-up menu.
2. Set custID to `54321` (as shown in Figure 6-145), and click **Continue** to run the test.



Configuration: Default Module Test

Module: EventEmitterMediation

Component: EventEmitterMediation

Interface: CustomerServiceIF

Operation: retrieveCustomer

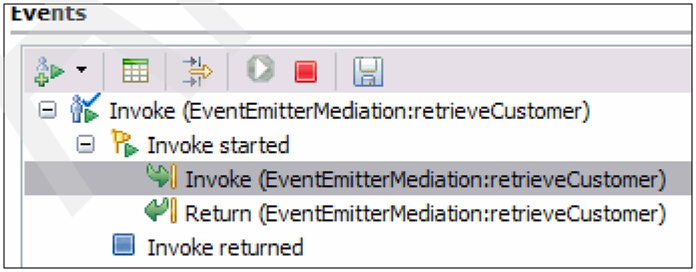
☐ Invoke export using binding

Initial request parameters

Name	Type	Value
customer	Customer	✓
custID	string	✓ 54321
custNam	string	✓
address	string	✓
zipCode	string	✓
city	string	✓

Figure 6-145 Test input

The test run should complete without errors, as shown in Figure 6-146.



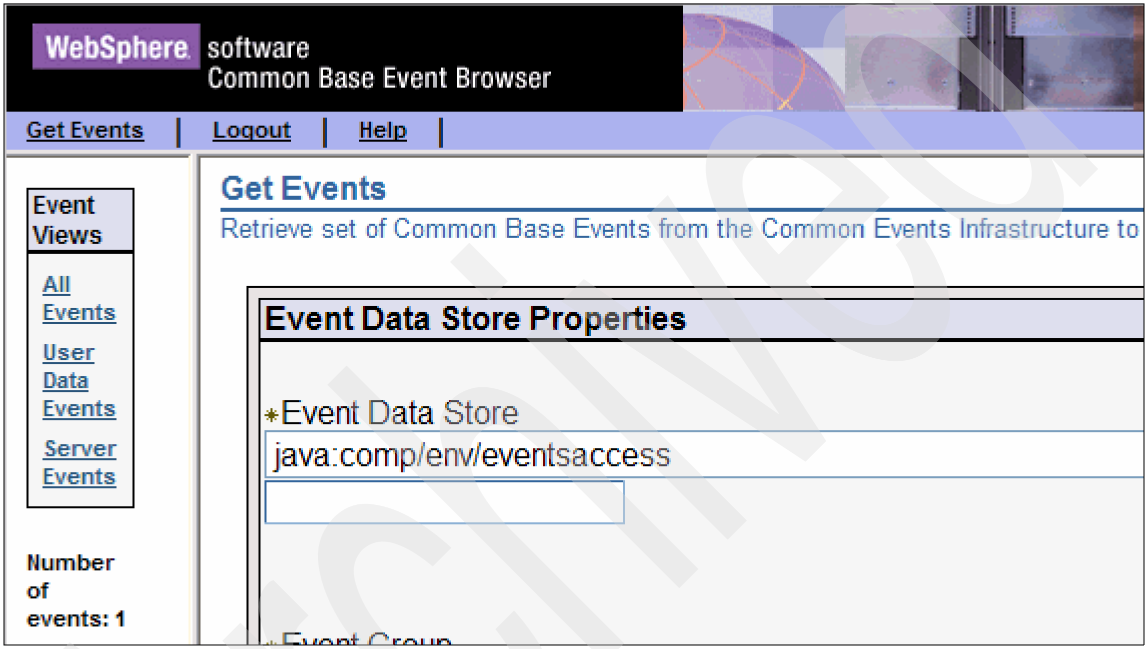
Events

- Invoke (EventEmitterMediation:retrieveCustomer)
  - Invoke started
    - Invoke (EventEmitterMediation:retrieveCustomer)
    - Return (EventEmitterMediation:retrieveCustomer)
  - Invoke returned

Figure 6-146 Test event sequence

To see the event, open the Common Base Event Browser:

1. Switch to the Servers tab.
2. Right-click the server and select **Launch** → **Common Base Event Browser** from the pop-up menu.
3. Enter the user ID and password (admin/admin by default). The Common Base Event Browser window opens (Figure 6-147). On the left, you see the number of events.



Number of events: 1

Get Events

Retrieve set of Common Base Events from the Common Events Infrastructure to

Event Data Store Properties

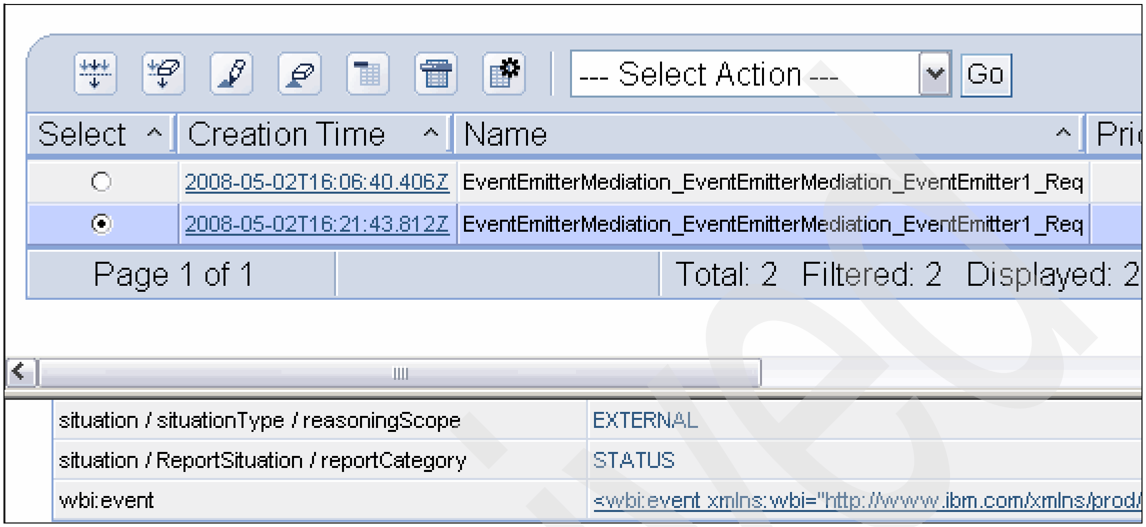
*Event Data Store

java:comp/env/eventsaccess

*Event Group

Figure 6-147 Common Base Event Browser

4. Click **All Events** under **Event Views** in the left navigation bar. Select the event that you want to view, and the data for the event displays in the bottom portion of the screen (as shown in Figure 6-148).



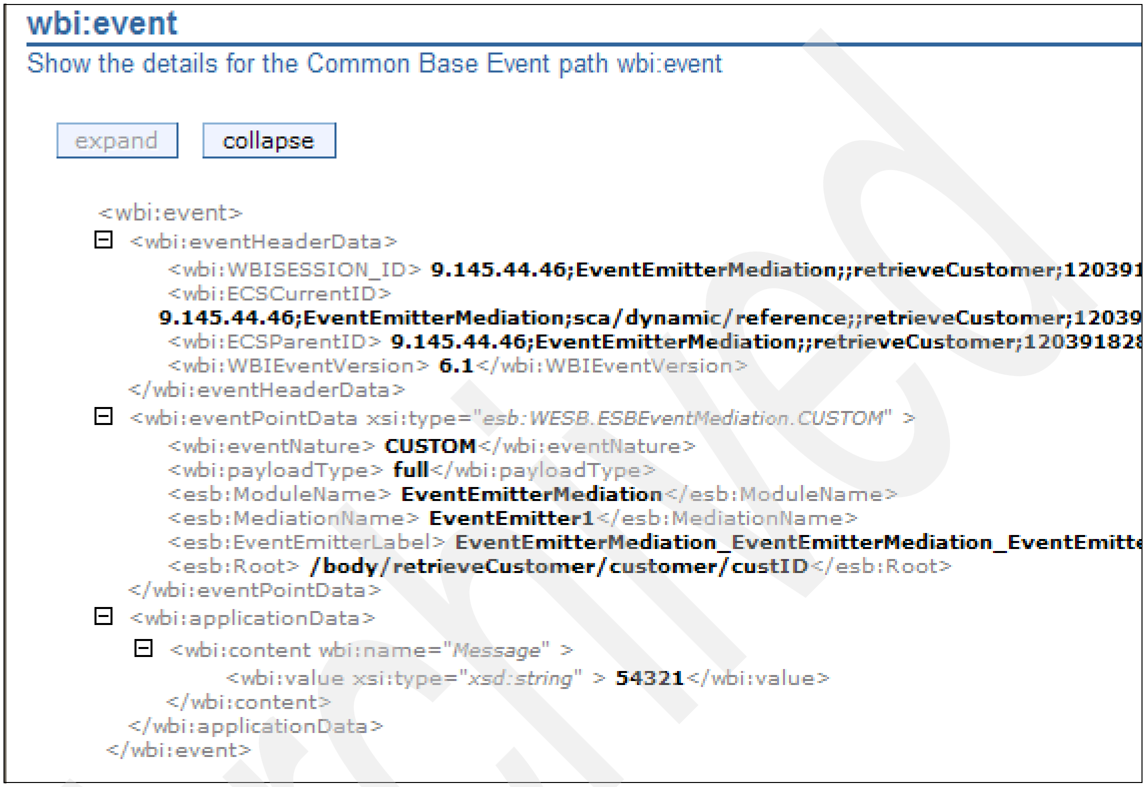
Select	Creation Time	Name	Pri
○	2008-05-02T16:06:40.406Z	EventEmitterMediation_EventEmitterMediation_EventEmitter1_Req	
●	2008-05-02T16:21:43.812Z	EventEmitterMediation_EventEmitterMediation_EventEmitter1_Req	

Page 1 of 1 Total: 2 Filtered: 2 Displayed: 2

situation / situationType / reasoningScope	EXTERNAL
situation / ReportSituation / reportCategory	STATUS
wbi:event	<wbi:event xmlns:wbi="http://www.ibm.com/xmlns/prod/

Figure 6-148 List of events

5. At the very bottom of the Event Data find the wbi:event field. Click the `<wbi:event xmlns:wbi=...>` value on the right to display the event data (as shown in Figure 6-149). The `<wbi:eventPointData>` and `<wbi:application Data>` tags are of interest.



wbi:event

Show the details for the Common Base Event path wbi:event

```
<wbi:event>
⊟ <wbi:eventHeaderData>
      <wbi:WBISESSION_ID> 9.145.44.46;EventEmitterMediation;;retrieveCustomer;120391
      <wbi:ECSCurrentID>
     9.145.44.46;EventEmitterMediation;sca/dynamic/reference;;retrieveCustomer;12039
      <wbi:ECSParentID> 9.145.44.46;EventEmitterMediation;;retrieveCustomer;12039182
      <wbi:WBIEventVersion> 6.1</wbi:WBIEventVersion>
   </wbi:eventHeaderData>
⊟ <wbi:eventPointData xsi:type="esb:WESB.ESBEventMediation.CUSTOM" >
      <wbi:eventNature> CUSTOM</wbi:eventNature>
      <wbi:payloadType> full</wbi:payloadType>
      <esb:ModuleName> EventEmitterMediation</esb:ModuleName>
      <esb:MediationName> EventEmitter1</esb:MediationName>
      <esb:EventEmitterLabel> EventEmitterMediation_EventEmitterMediation_EventEmitte
      <esb:Root> /body/retrieveCustomer/customer/custID</esb:Root>
   </wbi:eventPointData>
⊟ <wbi:applicationData>
   ⊟ <wbi:content wbi:name="Message" >
           <wbi:value xsi:type="xsd:string" > 54321</wbi:value>
      </wbi:content>
   </wbi:applicationData>
  </wbi:event>
```

Figure 6-149 Event data

6. Deploy DBMSServiceMediationApp and rerun the test with `custID=10001`, which should pass the MessageFilter1 and proceed with a successful Web service call. No event is emitted.

7

Using adapters

This chapter introduces the IBM WebSphere JCA Adapter support in IBM WebSphere Integration Developer and illustrates how to incorporate adapters into business integration modules and mediations.

This chapter contains the following topics:

- IBM WebSphere JCA Adapter architecture
- IBM WebSphere Adapter Toolkit
- Enterprise Metadata Discovery
- Tools for creating JCA adapters
- Example: Using the WebSphere Adapter for JDBC

7.1 IBM WebSphere JCA Adapter architecture

There are two main interfaces to a JCA adapter (Figure 7-1):

- The Service Provider Interface (SPI)
- The Common Client Interface (CCI)

The SPI is the application server's view of the adapter. It contains the contracts that are necessary to work with an application server, including connection creation and matching, security, and work management.

The CCI is designed to provide a common view of data and interaction with the adapter. The CCI defines the data model and provides a common mechanism to interact with the adapter.

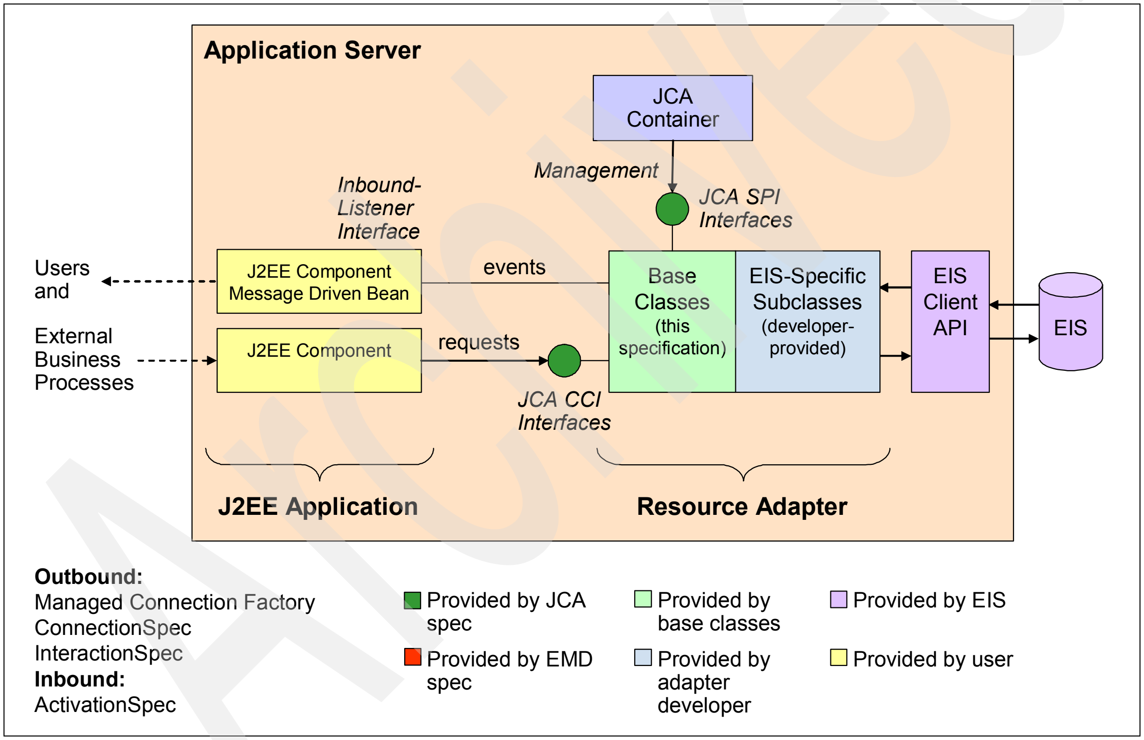


Figure 7-1 JCA adapter architecture

Import and export components with an EIS binding provide SCA components with the uniform view of the services external to the module. This view allows components to communicate with a variety of external EIS systems using consistent SCA programming model.

You can find all resource adapters that come with the product in the *WID_root*\Resource Adapters directory.

Exports (inbound)

An EIS service export is an export component that makes a service available to clients outside of the module. The EIS export interface defines the interface of the exported service. See Figure 7-2.

Within the EIS export, the EISExportBinding binds the exported service to the external EIS service to:

- Allow the service to subscribe to (listen for) EIS service requests.
- Specify the mapping between the definition of inbound events as it is understood by the Resource Adapter (using JCA interfaces) and the SCA operation invocation.

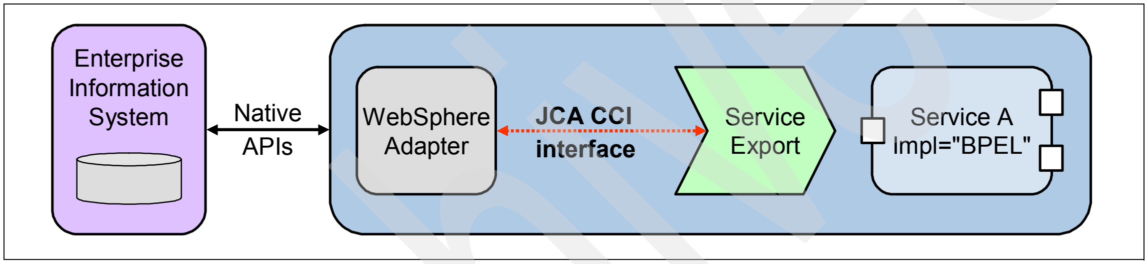


Figure 7-2 EIS Inbound to WebSphere Process Server

Imports (outbound)

The EIS service import is an import component that allows components in the module to use an EIS service that is defined outside of the module. The interfaces that are specified in the import, either WSDL or Java, represent the interface of the external service available through the import. Within the EIS import, the EISImportBinding binds external EIS services to the SCA module. See Figure 7-3.

The binding specifies the mapping between the definition of the outbound SCA operation invocation and the interaction information as understood by the Resource Adapter (using JCA interfaces).

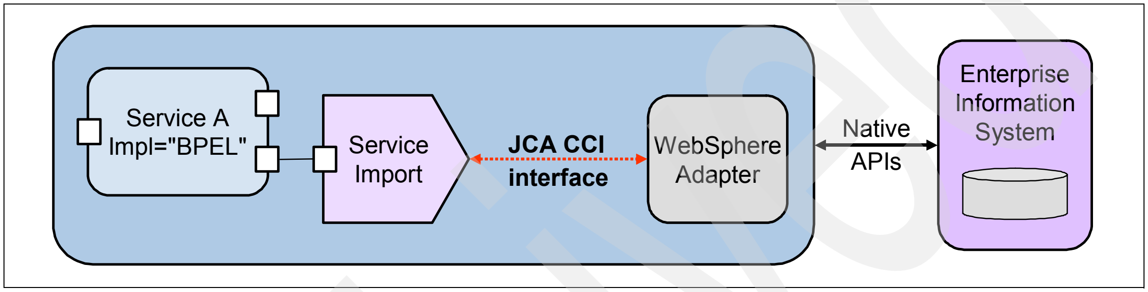


Figure 7-3 WebSphere Process Server to EIS outbound

Consult the documentation that comes with these resource adapters for more information, such as details about their availability on platforms such as Linux.

7.1.1 Different types of WebSphere JCA Adapters

WebSphere JCA Adapters implement the Enterprise MetaData Discovery specification to provide a simple integration experience with graphical discovery tools without resorting to writing code.

There is a distinction between two categories of WebSphere JCA adapter:

- Application adapter
- Technology adapters

Application adapters

WebSphere Adapters V6.1 connect enterprise business application suites to IBM Business Process Management, enterprise service bus, and application server solutions in a service-oriented architecture (SOA).

Application adapters include:

- JD Edwards EnterpriseOne

  Provides bidirectional, real-time integration between JD Edwards Enterprise One and OneWorld Xe.

- Oracle E-Business Suite

  Supports real-time, bidirectional integration to all Oracle applications modules and other systems.

- PeopleSoft Enterprise

  Provides bidirectional, real-time integration between PeopleSoft and other applications.

- SAP Exchange Infrastructure

  Allows business applications to send and receive business data and events as XML messages asynchronously.

- SAP Software

  Provides bidirectional, multi-threaded, real-time integration between SAP and other applications, using SAP's all interfaces capabilities.

- Siebel Business Applications

  Provides comprehensive bidirectional, real-time integration between Siebel and order management, ERP, e-business, and existing systems.

Technology adapters

WebSphere Adapters V6.1 technology adapters deliver file and database connectivity solutions for IBM Business Process Management and enterprise service bus solutions in an SOA.

The technology adapters include:

- Email

  Enables the exchange of business objects with a variety of applications.

- Flat Files

  Enables the communication with an application through the exchange of text files.

- File Transfer Protocol (FTP)

  Extend ESB with business document exchange via FTP server.

- JDBC

  Allows the exchange of objects with applications built on any database supported by JDBC Driver.

7.2 IBM WebSphere Adapter Toolkit

IBM WebSphere Adapter Toolkit enables customers and business partners to develop custom JCA adapters to meet unique business requirements. The toolkit helps to create either a basic JCA 1.5 adapter or an adapter that takes advantage of the additional capabilities of the Adapter Foundation Classes utilized by WebSphere Adapters.

The Eclipse-based toolkit includes:

- A wizard to create an adapter project, including the Java code stubs for the appropriate adapter classes.
- The Adapter Foundation Classes that provide a consistent implementation for WebSphere Adapters based on JCA 1.5.
- A graphical Resource Adapter Deployment Descriptor Editor to ease creation and modification of the XML deployment descriptor file.
- A sample adapter and its associated enterprise application with source. The sample adapter takes advantage of the Adapter Foundation Classes and implements the Enterprise Metadata Discovery (EMD) specification for wizard-driven configuration and the Service Data Objects (SDO) specification for exchanging data.

IBM WebSphere Adapter Toolkit is provided as a no-fee download from IBM developerWorks® to customers and business partners who secure licenses to WebSphere Integration Developer and Rational Application Developer. To access WebSphere Adapter Toolkit, visit IBM developerWorks.

`http://www.ibm.com/developerworks/websphere/downloads/wat/`

7.3 Enterprise Metadata Discovery

To realize the benefits of an SOA, it is critical to have easy interoperability with EIS systems.

The Enterprise Metadata Discovery specification introduces a new metadata discovery and import model for resource adapters and the enterprise application integration (EAI) tools framework. The model allows resource adapters to easily plug into an integration framework and improves the usability of adapters within the framework.Tools that support Enterprise Metadata Discovery specification have become the standard way to implement JCA-based applications.

The *external service wizard* in WebSphere Integration Developer supports the specification. The wizard browses the metadata information of an EIS system in a process called *discovery*. Then the artifacts that are of interest are imported into a *service descriptio*n. The service description contains all the information that is required to generate an implementation of the service that can then be deployed to an application server.

In addition to generating a service, you might be able to edit the configuration of the resource adapter or service after it has been created and configured. The tools in WebSphere Integration Developer that perform this editing function, the assembly editor and the business object editor, are demonstrated in the task help.

7.4 Tools for creating JCA adapters

WebSphere Integration Developer provides tools that support using JCA adapters in modules and mediation modules.

7.4.1 Using the external service wizard

WebSphere Integration Developer provides the *external service wizard* to implement resource adapters in projects.

To start the wizard:

1. Create the module that will use the adapter, and open the assembly diagram.
2. Select **New** → **External Service** from the context menu of the module in the Business Integration view.

   Alternatively, you can select an adapter from the Inbound or Outbound adapter list in the palette and drop it into the assembly diagram.

The wizard provides following capabilities:

- Allows you to select the resource adapter and to include it in the workspace, either bundled with the module or as a separate RAR to be deployed.
- Builds the connector project to hold the files that are associated with the module.
- Finds the application or data on the EIS system and builds the import component (for outbound services) or export component (for inbound services) required for the service.

  An import component lets your application invoke a service on the EIS system.

An export component lets an application on an EIS system invoke a service in WebSphere Process Server or WebSphere Enterprise Service Bus. Exports can only be created for EIS systems that support initiating external function calls.

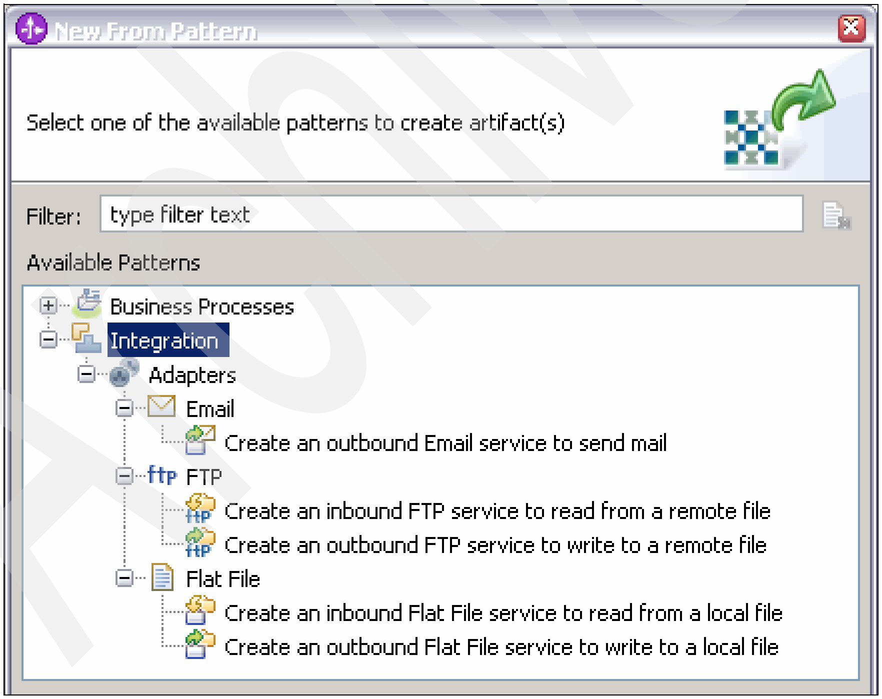
- Generates business objects representing data structures on the EIS systems.

7.4.2 Using the adapter pattern wizard

Patterns enable the creation of integrated solutions based on predefined pieces. WebSphere Integration Developer provides a number of patterns for integrating adapters into your solution. These patterns can be used by invoking the *adapter pattern wizard*.

You first create the module that will use the adapter. Then, invoke the wizard to populate the module with the necessary artifacts by selecting **New** → **From Patterns** from the context menu of the module in the Business Integration view.

Figure 7-4 shows the patterns that are available.



Select one of the available patterns to create artifact(s)

Filter: type filter text

Available Patterns

- Business Processes
- Integration
  - Adapters
    - Email
      - Create an outbound Email service to send mail
    - FTP
      - Create an inbound FTP service to read from a remote file
      - Create an outbound FTP service to write to a remote file
    - Flat File
      - Create an inbound Flat File service to read from a local file
      - Create an outbound Flat File service to write to a local file

Figure 7-4 Adapter patterns

7.4.3 Adapter deployment options

When planning for implementation, you must to decide whether to bundle the adapter within the module that uses it or to deploy it separately. You typically bundle the adapter within the module when that module is the only application that will use the adapter or if that module uses a unique version of the adapter. When the Adapter is bundled with the module, it is packaged in the same EAR file as the module, as shown in Figure 7-5.

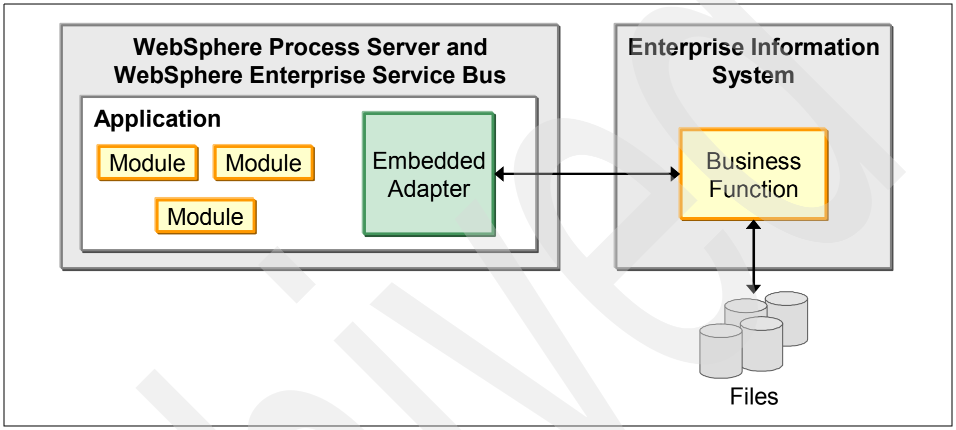

WebSphere Process Server and
WebSphere Enterprise Service Bus
Application
Module
Module
Module
Embedded Adapter
Enterprise Information System
Business Function
Files

Figure 7-5 Embedding the adapter in your module

You can also choose to deploy the adapter separately. In this case, a Resource Adapter Archive (RAR) is deployed and is accessible to any application in the run time.

7.5 Example: Using the WebSphere Adapter for JDBC

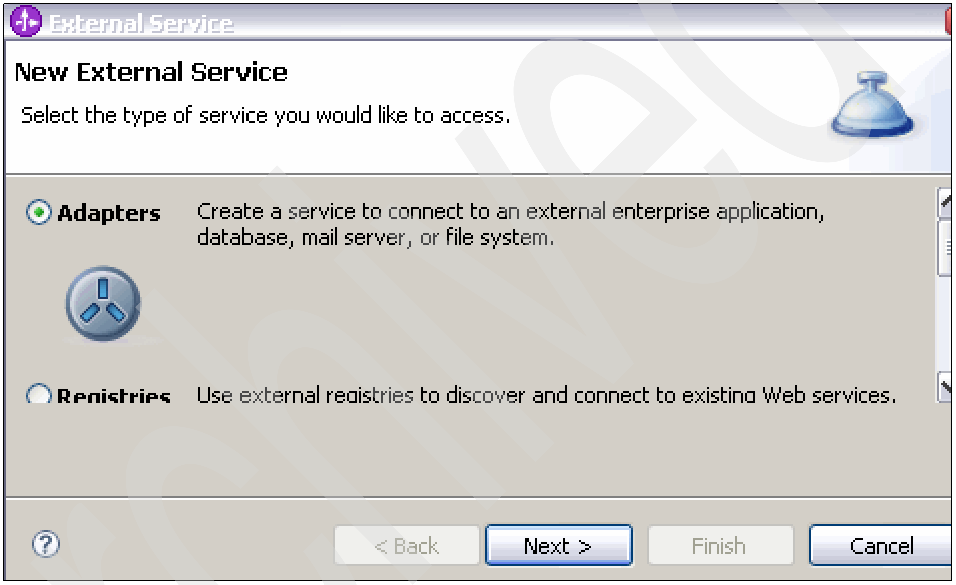
The JDBC resource adapter enables bidirectional connectivity for integration to any database application. The exchange of data for such applications happens at the database level. The JDBC resource adapter can integrate with any database, as long as there is a JDBC driver that supports the JDBC 2.0 (or higher) Specification, available for the database. Examples of such databases include DB2, Oracle, Microsoft SQLServer, Sybase, and Informix.

For complete information about this adapter, see the WebSphere Adapter for JDBC documentation, which is available at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.wsadapters.610.jca_jdbc.doc/doc/stbp_jdb_welcome.html

This example demonstrates how to build a business object using the JDBC Resource Adapter, as follows:

1. Create the database, and ensure that the system that hosts WebSphere Integration Developer has access to this database.
2. In the WebSphere Integration Developer, switch to the Business Integration perspective, and create a new module called **ITSOModule.**
3. Right-click **ITSOModule** in the Business Integration view, and select **New** → **External Service**.
4. Select **Adapters**, and click **Next** (Figure 7-6).


External Service

New External Service

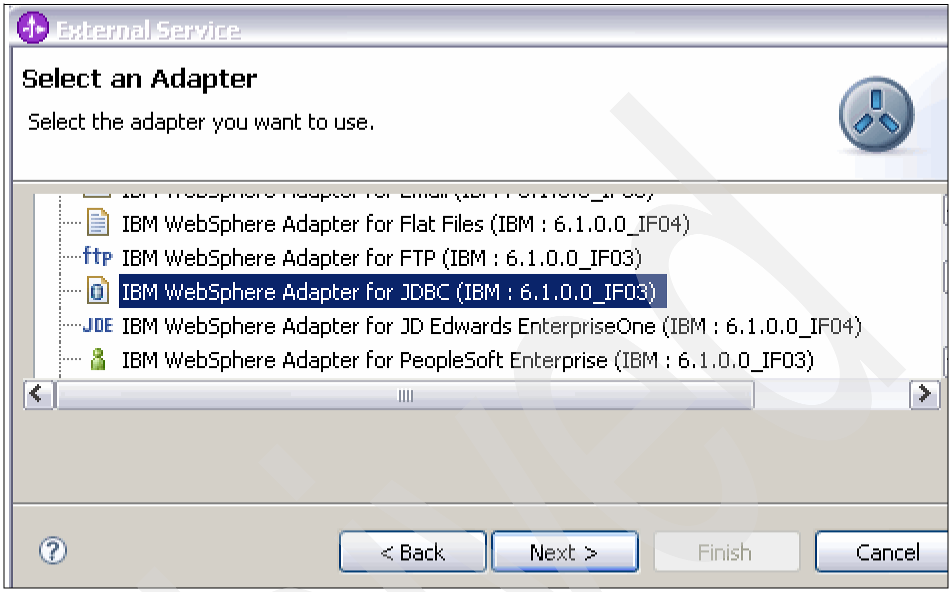
Select the type of service you would like to access.

● Adapters — Create a service to connect to an external enterprise application, database, mail server, or file system.

○ Registries — Use external registries to discover and connect to existing Web services.

Figure 7-6 Select Adapters as the external service type

5. Select IBM WebSphere Adapter for JDBC (IBM: 6.1), as shown in Figure 7-7, and click **Next**.



External Service

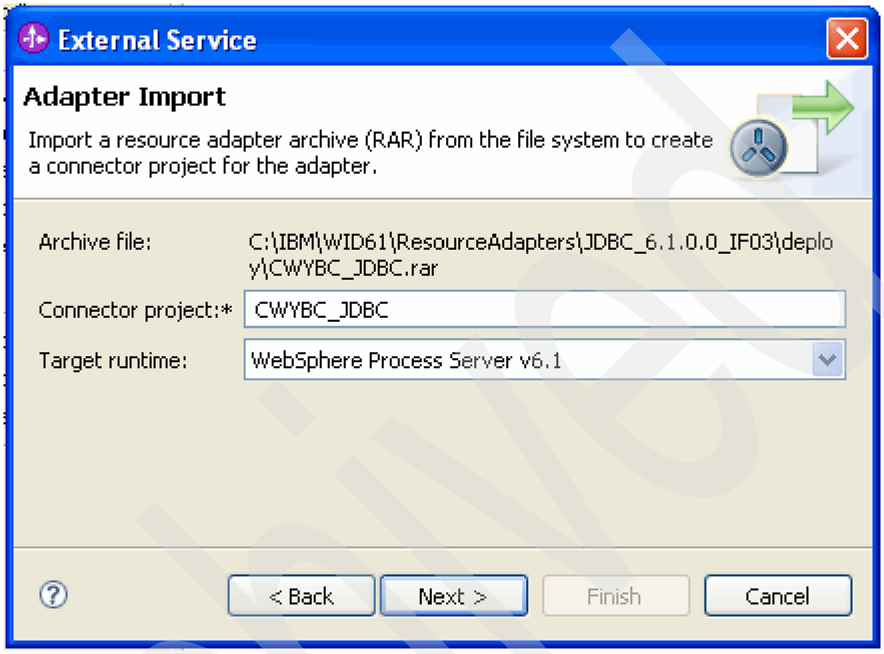
Select an Adapter

Select the adapter you want to use.

- IBM WebSphere Adapter for Email (IBM : 6.1.0.0_IF03)
- IBM WebSphere Adapter for Flat Files (IBM : 6.1.0.0_IF04)
- IBM WebSphere Adapter for FTP (IBM : 6.1.0.0_IF03)
- IBM WebSphere Adapter for JDBC (IBM : 6.1.0.0_IF03)
- IBM WebSphere Adapter for JD Edwards EnterpriseOne (IBM : 6.1.0.0_IF04)
- IBM WebSphere Adapter for PeopleSoft Enterprise (IBM : 6.1.0.0_IF03)

Figure 7-7 External Service Adapter for JDBC

6. The archive file to be imported and the connector project name that is used displays (Figure 7-8). You can specify a new connector project or use an existing one. Also, select a target run time. Then, click **Next**. The connector project is created, and the archive is imported to it.



External Service

Adapter Import

Import a resource adapter archive (RAR) from the file system to create a connector project for the adapter.

Archive file: C:\IBM\WID61\ResourceAdapters\JDBC_6.1.0.0_IF03\deploy\CWYBC_JDBC.rar

Connector project:* CWYBC_JDBC

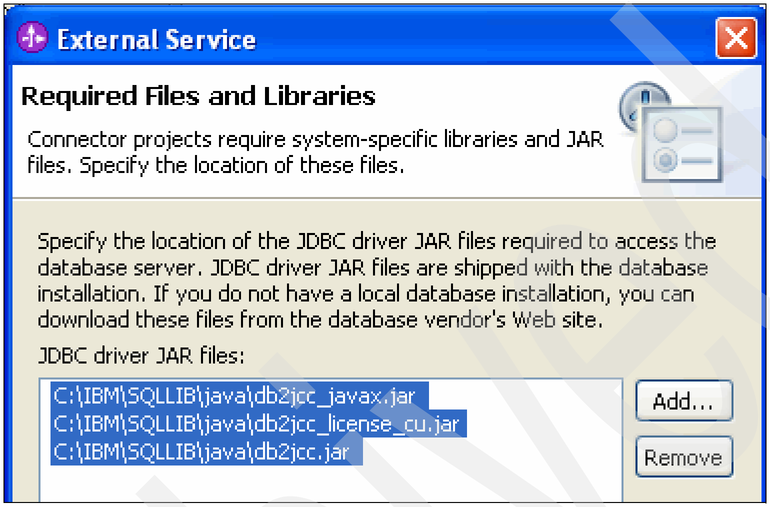
Target runtime: WebSphere Process Server v6.1

Figure 7-8 External Service Adapter Import

7. Next, you select the JDBC driver JAR files to be used to access the database.

   For each driver, click **Add**, browse to the appropriate JAR files, and click **Open**. Figure 7-9 shows the JAR files that we selected for IBM DB2 V9.1.

   Click **Next**.



Required Files and Libraries

Connector projects require system-specific libraries and JAR files. Specify the location of these files.

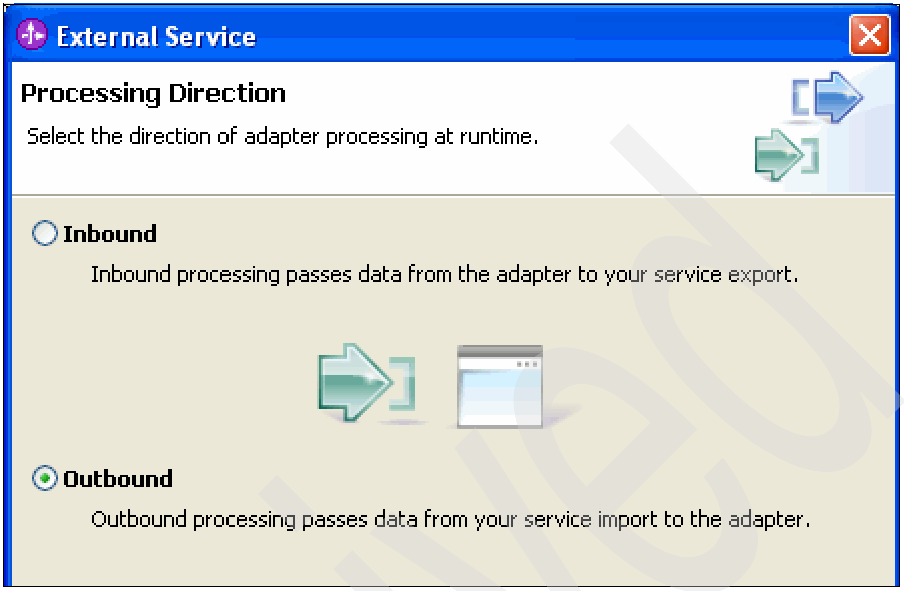
Specify the location of the JDBC driver JAR files required to access the database server. JDBC driver JAR files are shipped with the database installation. If you do not have a local database installation, you can download these files from the database vendor's Web site.

JDBC driver JAR files:

C:\IBM\SQLLIB\java\db2jcc_javax.jar
C:\IBM\SQLLIB\java\db2jcc_license_cu.jar
C:\IBM\SQLLIB\java\db2jcc.jar

Figure 7-9 Required JDBC driver files for the Adapter

8. Click **Outbound**, and click **Next** (Figure 7-10).



External Service

Processing Direction

Select the direction of adapter processing at runtime.

○ Inbound

Inbound processing passes data from the adapter to your service export.

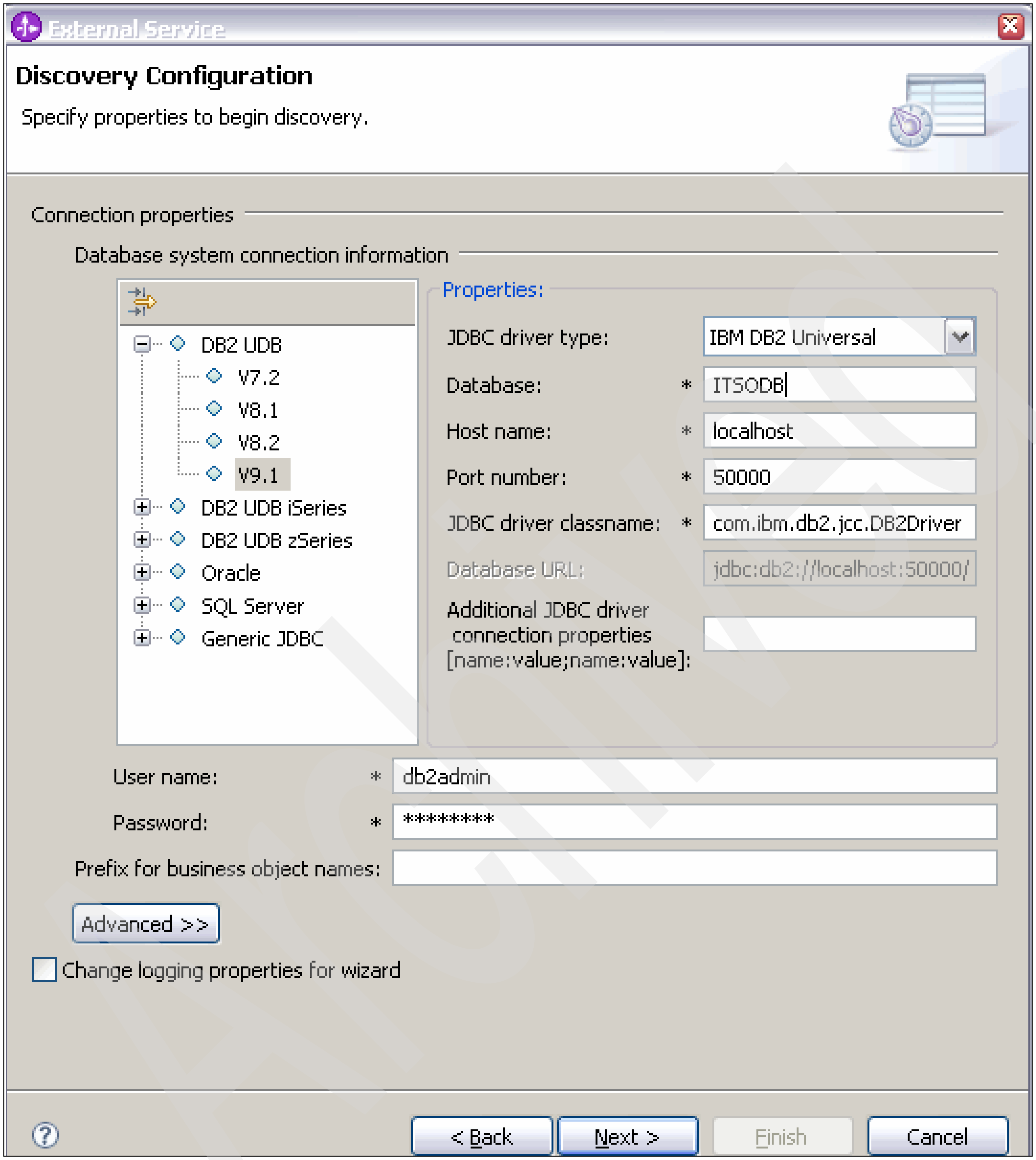
● Outbound

Outbound processing passes data from your service import to the adapter.

Figure 7-10 The Adapter Style options

9. The next panel specifies the connection properties for the database server (Figure 7-11 on page 485).
   a. Use the left panel to select the database vendor, driver, and version to which you want to connect.
   b. In the right panel, enter the information that is required to connect to the database (ITSO).
   c. Then, click **Next**. A connection to the database opens.

   The user ID and password are temporary settings that are used to access the database during the wizard.



JDBC driver type: IBM DB2 Universal

Database: * ITSODB

Host name: * localhost

Port number: * 50000

JDBC driver classname: * com.ibm.db2.jcc.DB2Driver

Database URL: jdbc:db2://localhost:50000/

Additional JDBC driver connection properties [name:value;name:value]:

User name: * db2admin

Password: * *********

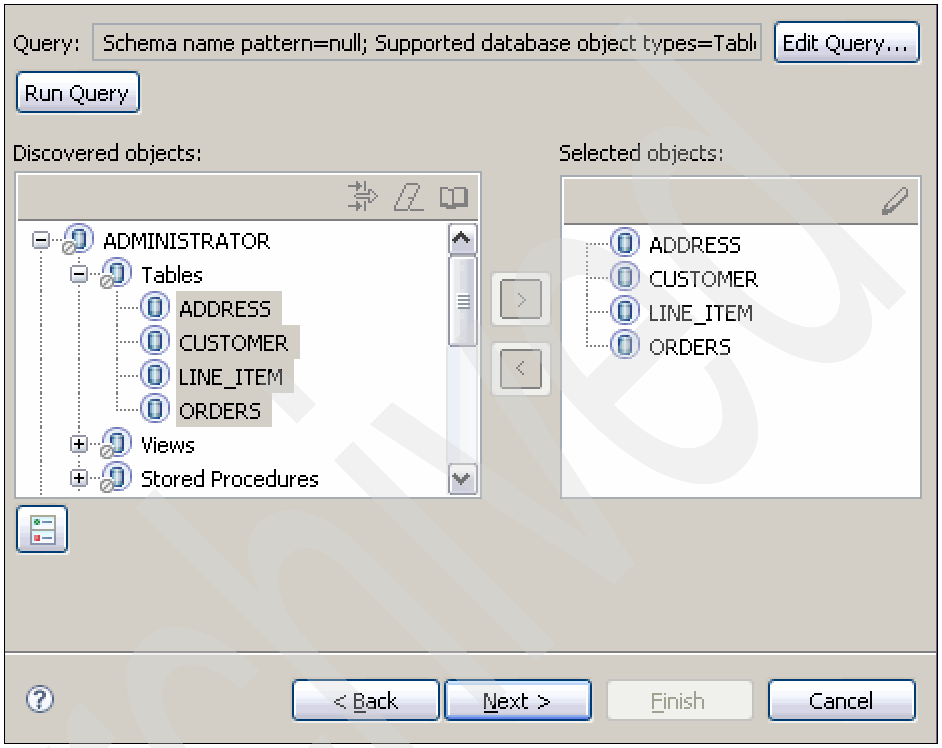
Prefix for business object names:

□ Change logging properties for wizard

Figure 7-11 Connection properties for database system and connection information

10. Click **Run Query** to list the tables, stored procedures, views, and synonyms for each schema in the database.

11. Select the tables for which you want the wizard to create operations and click the **>** (Add) button (as shown in Figure 7-12). Click **Next**.

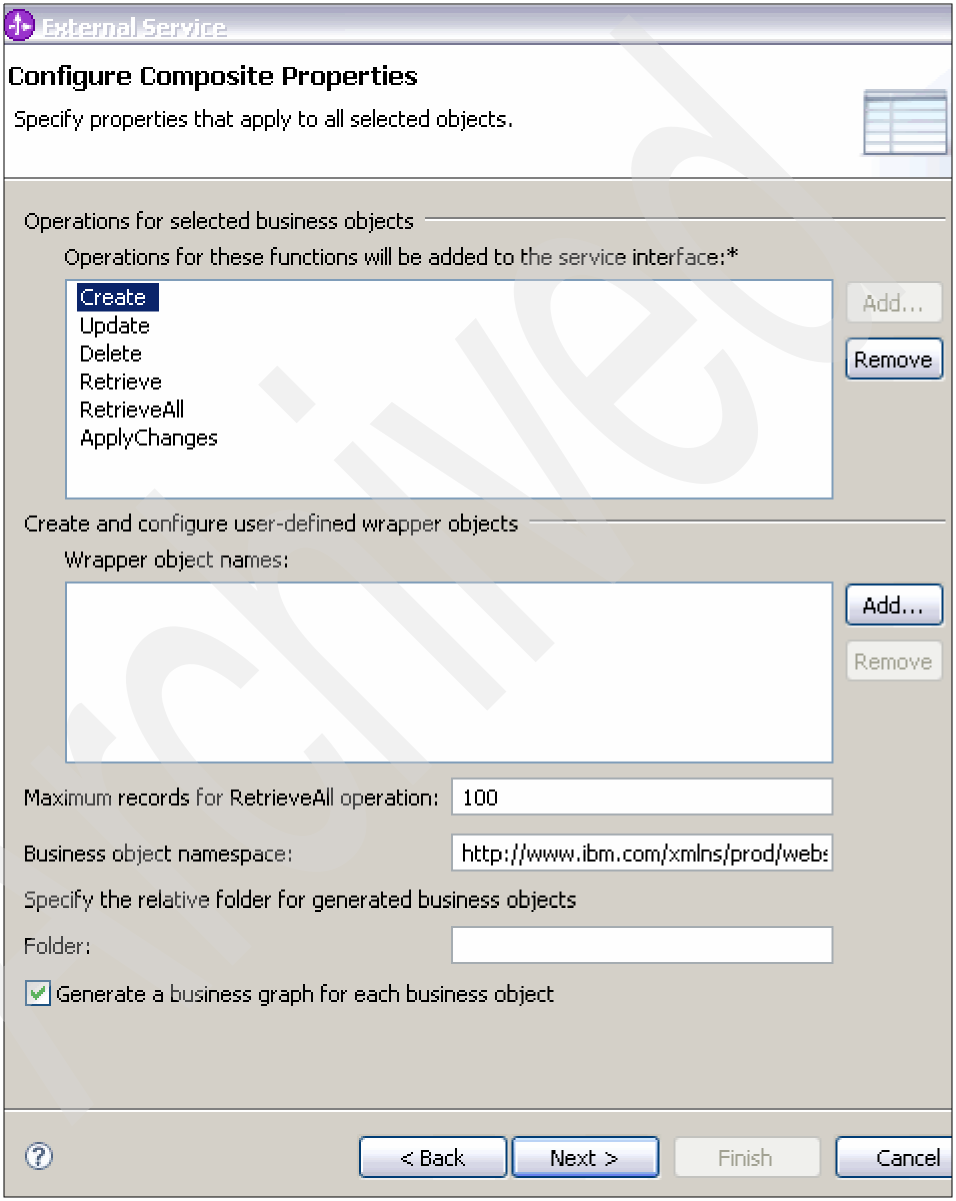


Query: Schema name pattern=null; Supported database object types=Tabl

Figure 7-12 Object discovery and selection

12. Select the operations that will apply to the business objects and business graphs that are generated.

    In the Configure Composite Properties screen (Figure 7-13), leave all the default settings and click **Next.**



External Service

Configure Composite Properties

Specify properties that apply to all selected objects.

Operations for selected business objects

Operations for these functions will be added to the service interface:*

- Create
- Update
- Delete
- Retrieve
- RetrieveAll
- ApplyChanges

Create and configure user-defined wrapper objects

Wrapper object names:

Maximum records for RetrieveAll operation: 100

Business object namespace: http://www.ibm.com/xmlns/prod/webs

Specify the relative folder for generated business objects

Folder:

☑ Generate a business graph for each business object

Figure 7-13 Configure Composite Properties

13. Next, enter the database security credentials, as well as deployment options and the JNDI name to use to access the database. Select one of the following deployment options:

    - With module for use by single application (our choice in this example)
    - On server for use by multiple applications

14. Specify how you want the adapter to get the database user name and password at run time (see Figure 7-14 on page 489).

    There are three options:

    - Use an authentication alias.

      Select **Specify a Java Authentication and Authorization Services (JAAS) alias security credential** and enter the name of the alias in the J2C Authentication Data Entry field. The alias can be an existing alias, or it can be created later.

    - To use an existing data source on the server:

      i. Clear the “Specify a Java Authentication and Authorization Services (JAAS) alias security credential” option.

      ii. Click **Advanced**.

      iii. Expand **Alternate ways** to specify connection information and enter either a data source JNDI name or an XA data source name and XA database name.

    - To specify the database user name and password to be saved in the adapter properties:

      i. Clear the “Specify a Java Authentication and Authorization Services (JAAS) alias security credential” option.

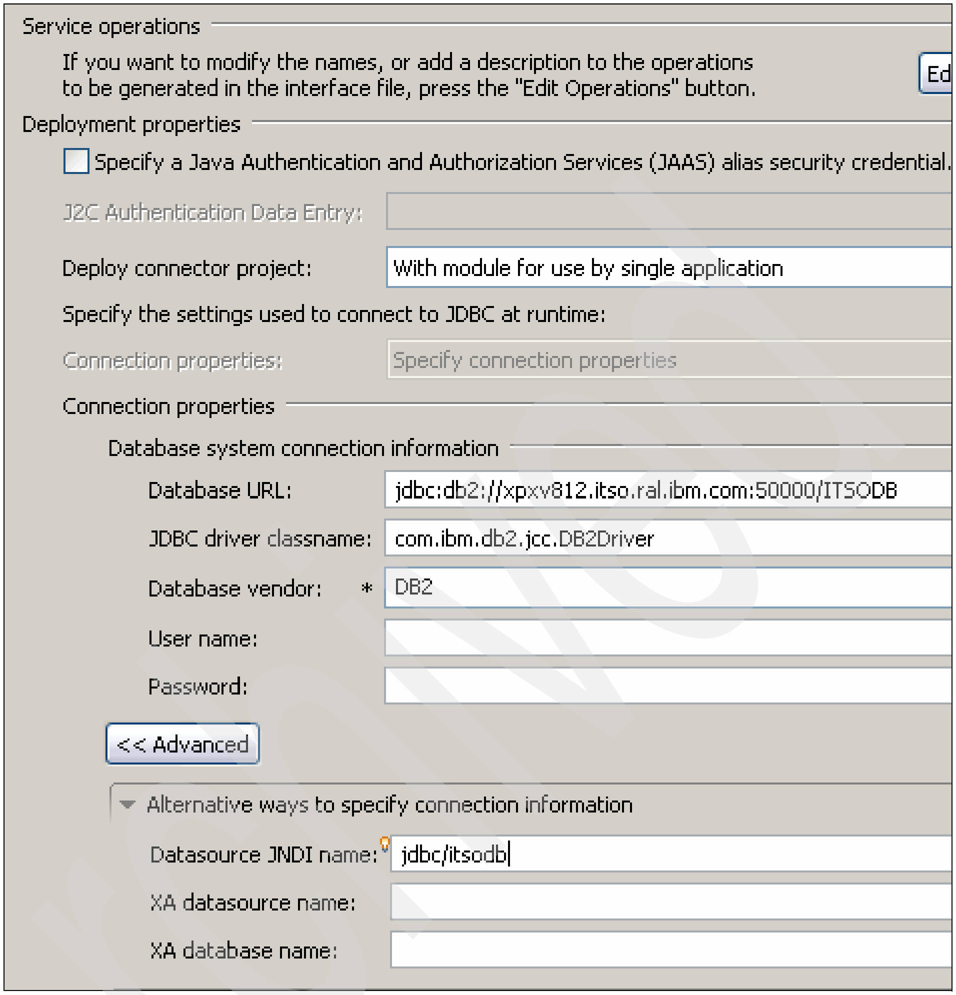
      ii. Click **Advanced**.

      iii. Under Database system connection properties, enter the User name and Password.

      > **Note:** When you specify the password here, it is saved as clear text in an adapter property, which unauthorized users might be able to see.

    In this example, we use a data source entry. So, we enter a JNDI name (jdbc/itsodb) for the data source. You must create this data source manually. The wizard does not create it.

    Click **Next**.



Service operations

If you want to modify the names, or add a description to the operations
to be generated in the interface file, press the "Edit Operations" button.

Deployment properties

□ Specify a Java Authentication and Authorization Services (JAAS) alias security credential.

J2C Authentication Data Entry:

Deploy connector project: With module for use by single application

Specify the settings used to connect to JDBC at runtime:

Connection properties: Specify connection properties

Connection properties

Database system connection information

Database URL: jdbc:db2://xpxv812.itso.ral.ibm.com:50000/ITSODB

JDBC driver classname: com.ibm.db2.jcc.DB2Driver

Database vendor: * DB2

User name:

Password:

Alternative ways to specify connection information

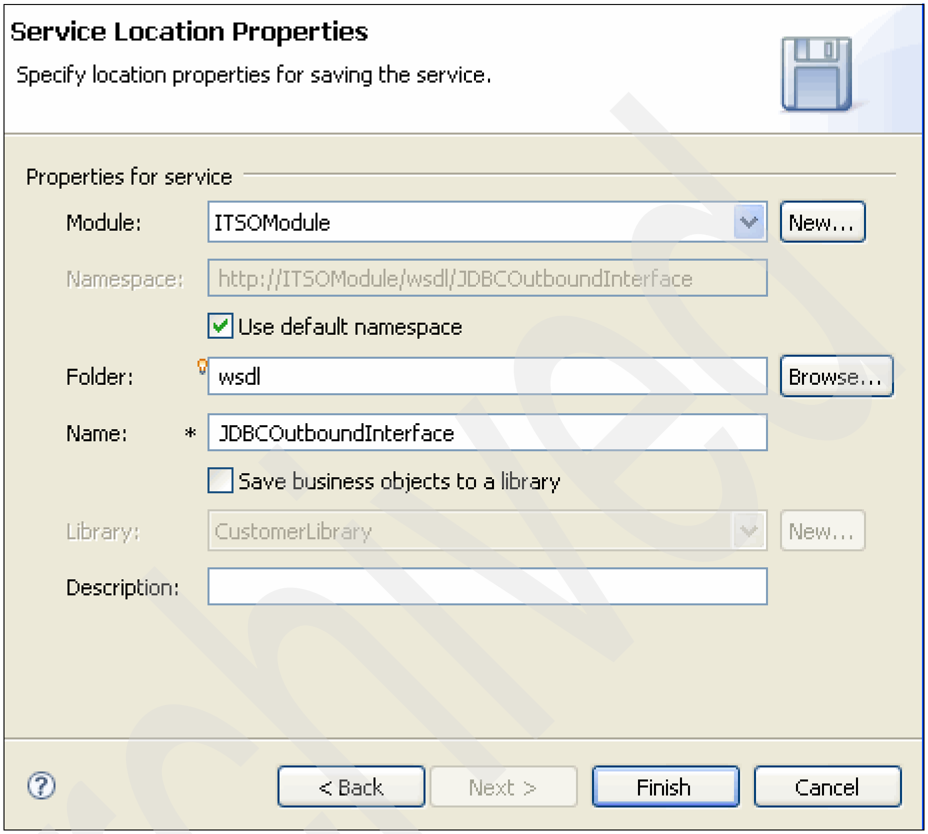
Datasource JNDI name: jdbc/itsodb

XA datasource name:

XA database name:

Figure 7-14 Deployment properties

15. In the Service Location Properties panel (Figure 7-15), select the new module and click **Finish**.



Service Location Properties

Specify location properties for saving the service.

Properties for service

Module: ITSOModule [New...]

Namespace: http://ITSOModule/wsdl/JDBCOutboundInterface

☑ Use default namespace

Folder: wsdl [Browse...]

Name: * JDBCOutboundInterface

□ Save business objects to a library

Library: CustomerLibrary [New...]

Description:

Figure 7-15 Service Location Properties

When the wizard completes, a file listing of the newly created module and an assembly diagram of its interface displays as shown in Figure 7-16.

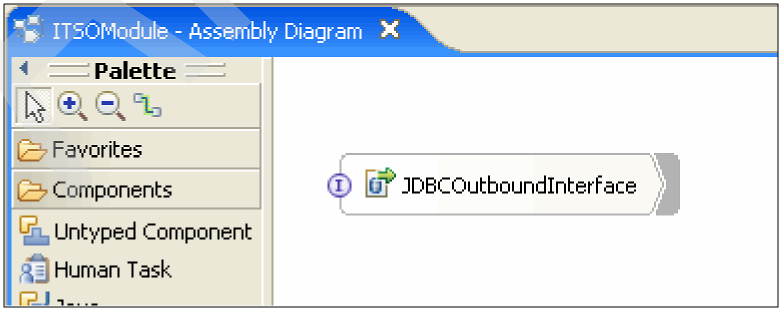


Figure 7-16 Generated Artifacts in the assembly diagram

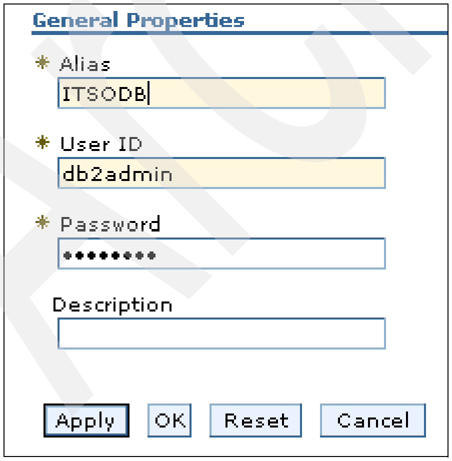
7.5.1 Creating an authentication alias for the database

Next, you need to create an authentication alias that includes the credentials required to access the database.

To set the authentication alias, you first need to start the server. You can start the server from the Servers view by right-clicking on the server and selecting **Start**.

After the server is started, follow these steps:

1. When the server status is Started, right-click the server, and then select **Run administrative console**. Log in to the administrative console.
2. In the console, select **Security** → **Secure Administration, applications and infrastructure**.
3. On the right, click **J2C Authentication Data** under Java Authentication and Authorization Service.
4. Create an authentication alias (Figure 7-17 on page 491):
   a. In the list of J2C authentication aliases that displays, click **New**.
   b. In the Configuration tab, type a name for the authentication alias in the Alias field.
   c. Enter the user ID and password that are required to establish a connection to the database.
   d. Optionally, type a description of the alias.
   e. Click **OK**.



General Properties

* Alias
ITSODB

* User ID
db2admin

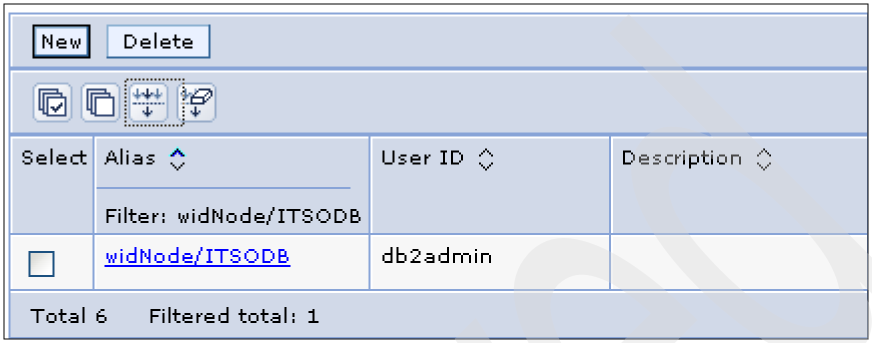
* Password
••••••••

Description

Figure 7-17 Configuration General Properties console

The newly created alias displays (Figure 7-18). Note the full name of the alias, which includes the node name. This full name is the one that you use in subsequent configuration windows.

f. Save the configuration.



Select	Alias Filter: widNode/ITSODB	User ID	Description
□	widNode/ITSODB	db2admin	

Total 6 Filtered total: 1

Figure 7-18 JAAS: JCA authentication data console

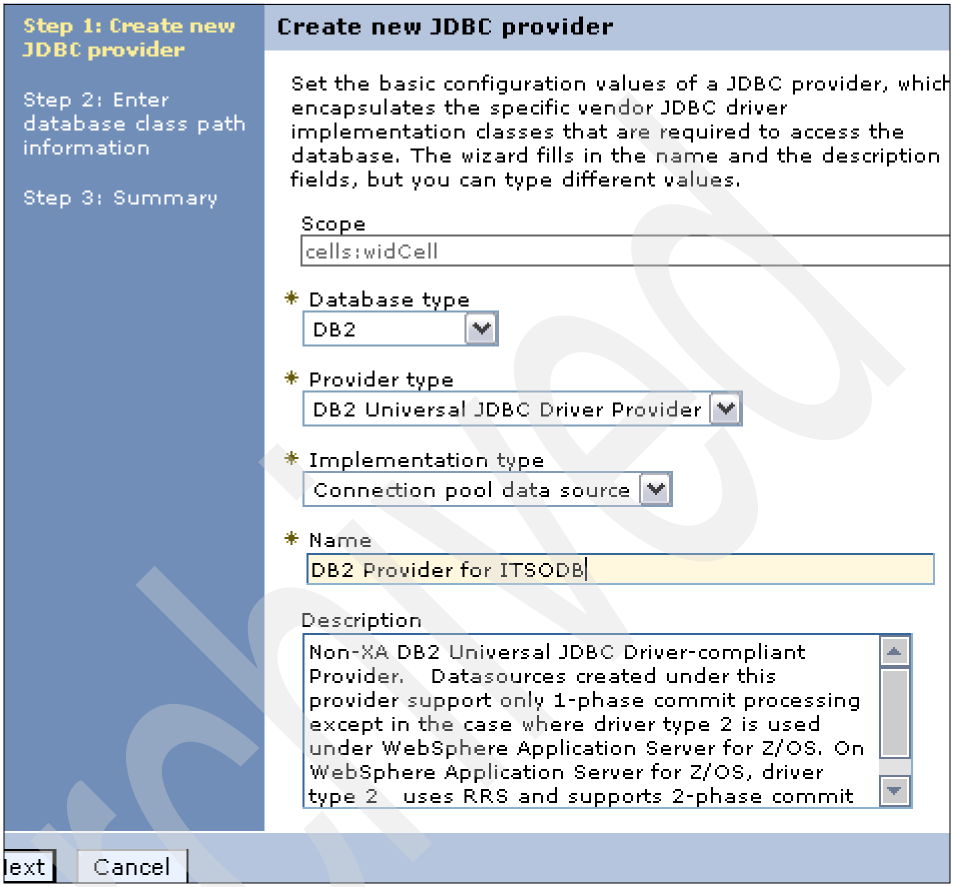
7.5.2 Creating the data source for the database

Using a data source lets applications share connections to a database. For example, if multiple applications access the same database with the same user name and password, the applications can be deployed using the same data source. Using a J2C authentication alias is a robust, secure way to deploy applications. An administrator creates the authentication alias that is used by one or more applications that need to access a system. The user name and password are known only to that administrator.

This process assumes that you have created the authentication alias with the credentials required to access the database. Follow these steps:

1. In the administrative console, select **Resources → JDBC → JDBC Providers**.
2. Select the scope at which you want to create the JDBC provider. Because JDBC providers point to files on a a server, the node scope is often the most appropriate.
3. Click **New** at the top of the list of JDBC providers.

4. Name the new JDBC provider and select the database and provider type as shown in Figure 7-19. Click **Next**.



Step 1: Create new JDBC provider
Step 2: Enter database class path information
Step 3: Summary
Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:widCell

* Database type: DB2

* Provider type: DB2 Universal JDBC Driver Provider

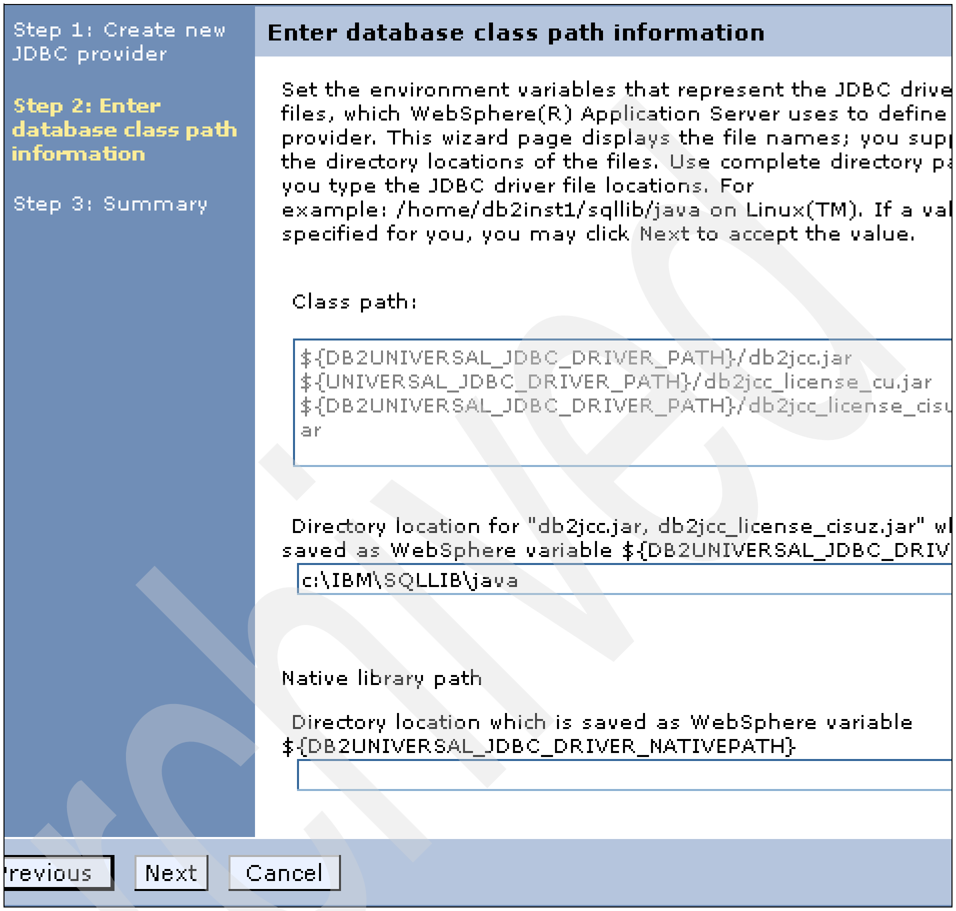
* Implementation type: Connection pool data source

* Name: DB2 Provider for ITSODB

Description: Non-XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support only 1-phase commit processing except in the case where driver type 2 is used under WebSphere Application Server for Z/OS. On WebSphere Application Server for Z/OS, driver type 2 uses RRS and supports 2-phase commit

Figure 7-19 Create new JDBC provider

5. Specify the directory location that contains the provider JAR files (Figure 7-20). Click **Next**.


Step 1: Create new JDBC provider
Step 2: Enter database class path information
Step 3: Summary
Previous
Next
Cancel

Enter database class path information

Set the environment variables that represent the JDBC drive
files, which WebSphere(R) Application Server uses to define
provider. This wizard page displays the file names; you sup
the directory locations of the files. Use complete directory p
you type the JDBC driver file locations. For
example: /home/db2inst1/sqllib/java on Linux(TM). If a val
specified for you, you may click Next to accept the value.

Class path:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisu
ar
```

Directory location for "db2jcc.jar, db2jcc_license_cisuz.jar" w
saved as WebSphere variable ${DB2UNIVERSAL_JDBC_DRIV

c:\IBM\SQLLIB\java

Native library path

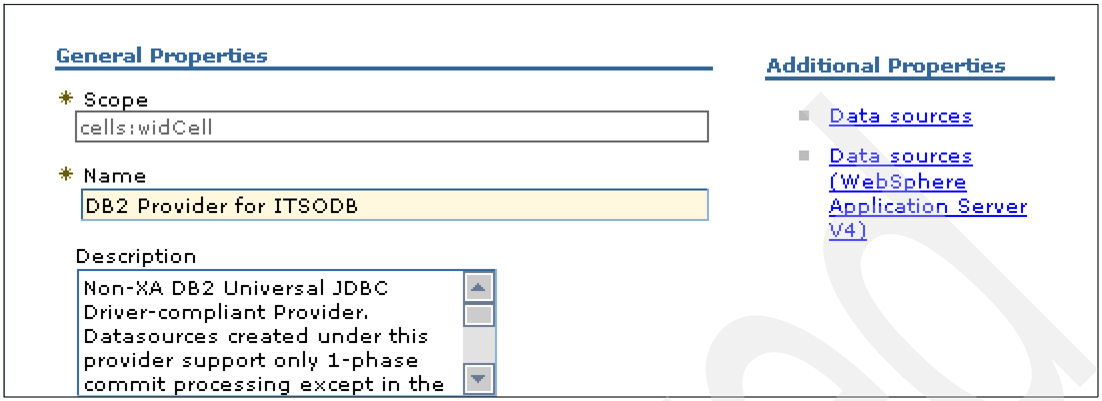
Directory location which is saved as WebSphere variable
${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}

Figure 7-20 Enter database class path information console

6. The last panel shows a summary of your selections. Click **Finish**.

   The driver is created and a list of providers displays, including the new driver.

7. Click the driver name to open the configuration, and then click **Data sources** as shown in Figure 7-21.



General Properties

* Scope

cells:widCell

* Name

DB2 Provider for ITSODB

Description

Non-XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support only 1-phase commit processing except in the

Additional Properties

- Data sources
- Data sources (WebSphere Application Server V4)

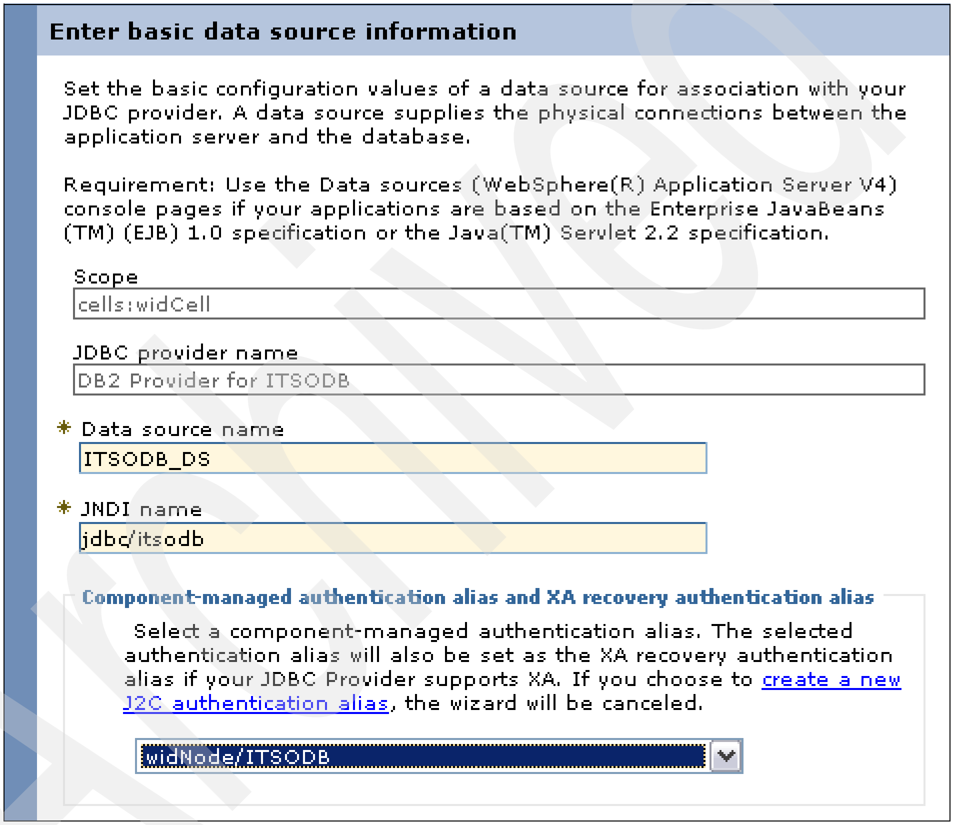
Figure 7-21 DB2 Universal JDBC Driver Provider

8. Click **New** to create a new data source. In the panel that displays (Figure 7-22):

   a. Enter a new name for the data source, the JNDI name for the data source. This name must match the name that is specified in the external service wizard (see Figure 7-14 on page 489).

   b. Select the authentication alias that you created earlier in 7.5.1, “Creating an authentication alias for the database” on page 491.

   c. Click **Next**.



Enter basic data source information

Set the basic configuration values of a data source for association with your JDBC provider. A data source supplies the physical connections between the application server and the database.

Requirement: Use the Data sources (WebSphere(R) Application Server V4) console pages if your applications are based on the Enterprise JavaBeans (TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope
cells:widCell

JDBC provider name
DB2 Provider for ITSODB

* Data source name
ITSODB_DS

* JNDI name
jdbc/itsodb

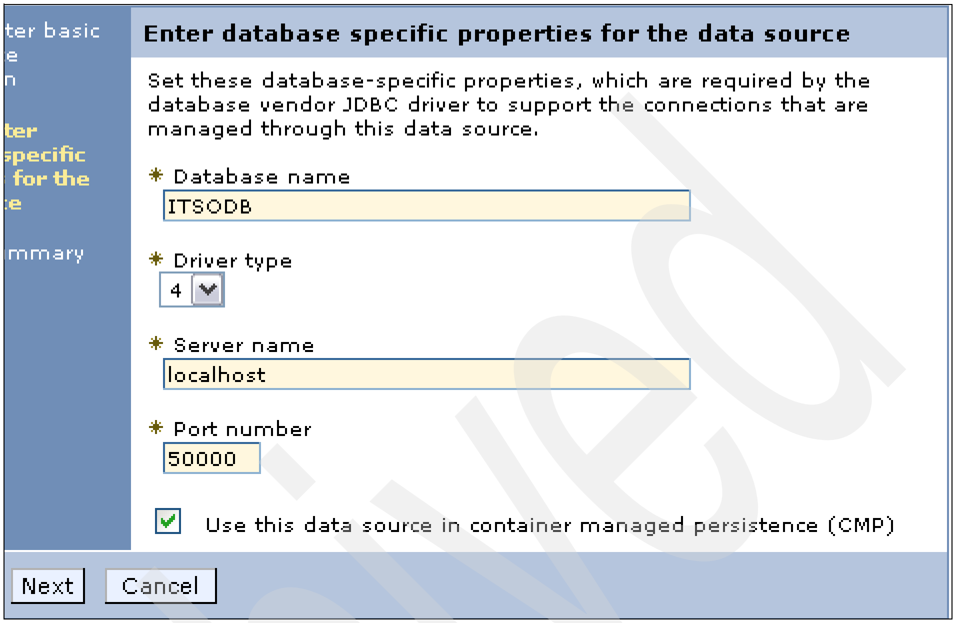
Component-managed authentication alias and XA recovery authentication alias

Select a component-managed authentication alias. The selected authentication alias will also be set as the XA recovery authentication alias if your JDBC Provider supports XA. If you choose to create a new J2C authentication alias, the wizard will be canceled.

widNode/ITSODB

Figure 7-22 Enter basic data source information

9. Enter the information that is required to connect to the database as shown in Figure 7-23. Click **Next**.



Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through this data source.

* Database name: ITSODB

* Driver type: 4

* Server name: localhost

* Port number: 50000

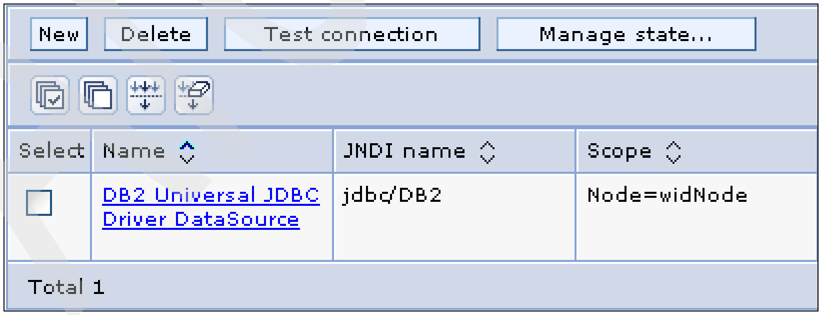
☑ Use this data source in container managed persistence (CMP)

Figure 7-23 Enter database specific properties for the data source

10. Review the summary, and click **Finish** to create the data source.

11. Save the changes.

12. Click Test connection to make sure the data source is defined correctly (Figure 7-24).



Select	Name	JNDI name	Scope
□	DB2 Universal JDBC Driver DataSource	jdbc/DB2	Node=widNode

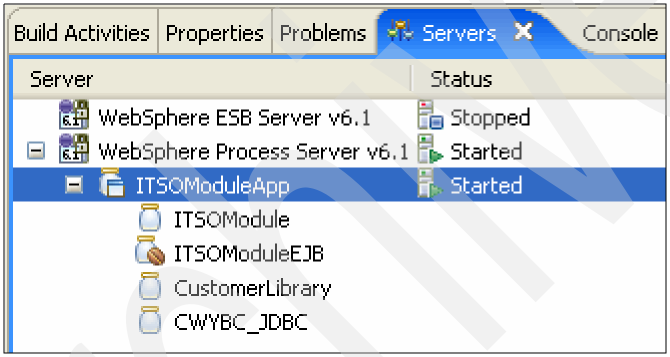
Total 1

Figure 7-24 Setting data sources

7.5.3 Deploying the module to the test environment

The result of running the external service wizard is a module that contains an EIS import or export. Install this SCA module in WebSphere Integration Developer integration test client. Follow these steps:

1. In the Servers view in the lower-right pane, right-click the server, and then select **Start**.
2. When the Server status changes to Started, add the module to the server.
   a. Right-click the server, and then select **Add and Remove Projects**.
   b. Select **ITSOModuleApp** and click the **>** (Add) button.
   c. Click **Finish**.
3. Verify that the server and ITSOModuleApp are in the `Started` state, as shown in Figure 7-25.



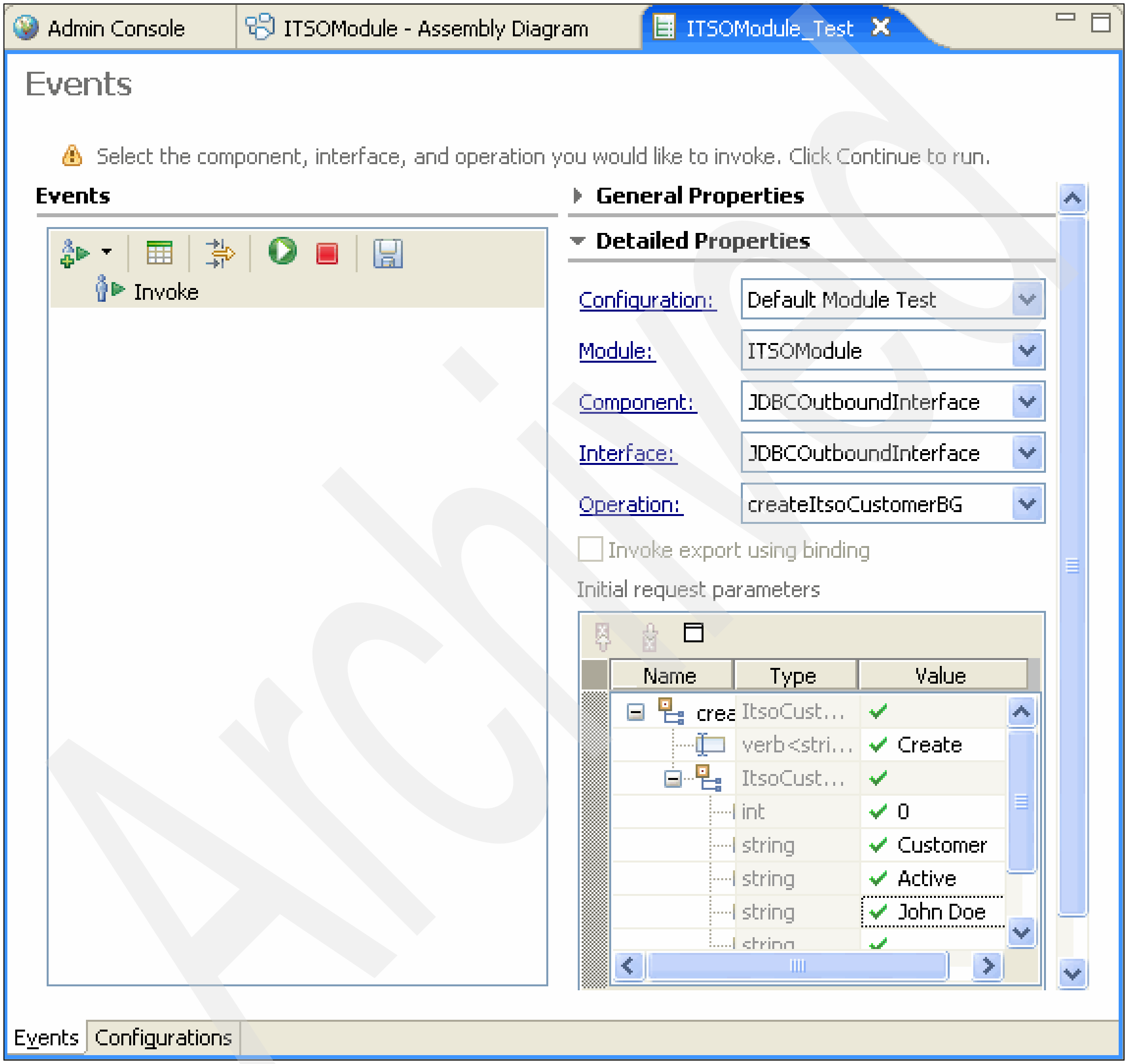
Server	Status
WebSphere ESB Server v6.1	Stopped
WebSphere Process Server v6.1	Started
ITSOModuleApp	Started
ITSOModule	
ITSOModuleEJB	
CustomerLibrary	
CWYBC_JDBC	

Figure 7-25 ITSOModuleApp started and ready for testing

Now, test the JDBCOutboundInterface component by creating and an outbound request:

1. Open the assembly diagram for ITSOModule. Right-click the **JDBCOutboundInterface** import and select **Test Component** from the context menu.

2. In the test client, set the properties as shown in Figure 7-26 on page 499:
   a. Make sure the Operation field is set to `createItsoCustomerBG`.
   b. Enter test data in the Initial request parameters table as shown in Figure 7-26.



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test

Module: ITSOModule

Component: JDBCOutboundInterface

Interface: JDBCOutboundInterface

Operation: createItsoCustomerBG

□ Invoke export using binding

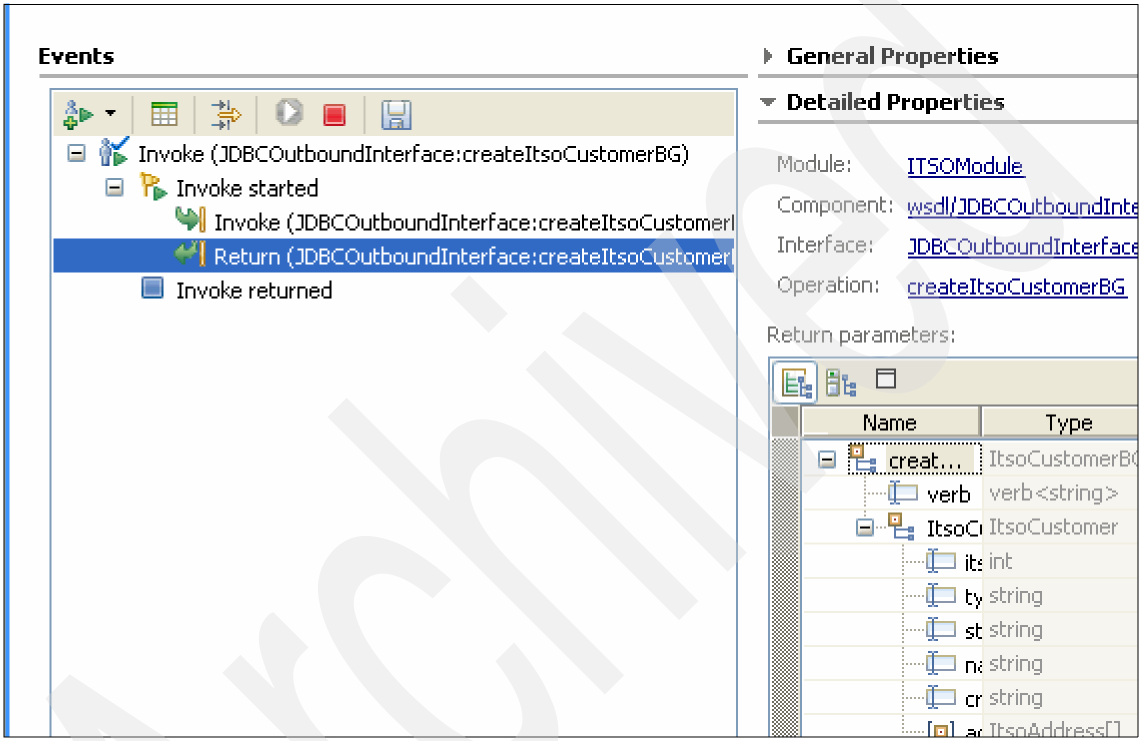
Initial request parameters

Name	Type	Value
crea	ItsoCust...	✓
	verb<stri...	✓ Create
	ItsoCust...	✓
	int	✓ 0
	string	✓ Customer
	string	✓ Active
	string	✓ John Doe
	string	✓

Figure 7-26 ITSOModule_Test mod

3. Click the Continue icon ▶.
4. Select **WebSphere Process Server v6.1** as the deployment location and click **Finish**.
5. If administrative security is enabled on server, log in using `admin` for the User ID and Password.

   The test runs. Figure 7-27 shows the results.



Events

- Invoke (JDBCOutboundInterface:createItsoCustomerBG)
  - Invoke started
    - Invoke (JDBCOutboundInterface:createItsoCustomerI
    - Return (JDBCOutboundInterface:createItsoCustomerI
  - Invoke returned

General Properties

Detailed Properties

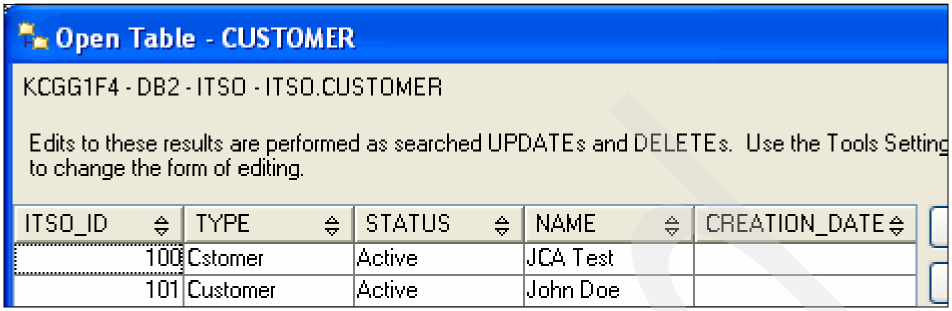
Module:	ITSOModule
Component:	wsdl/JDBCOutboundInte
Interface:	JDBCOutboundInterface
Operation:	createItsoCustomerBG

Return parameters:

Name	Type
creat...	ItsoCustomerBC
verb	verb<string>
ItsoCı	ItsoCustomer
its	int
ty	string
st	string
na	string
cr	string
ac	ItsoAddress[]

Figure 7-27 WebSphere Administration Console for testing

6. Using the DB2 Control Center, verify that the ITSO customer record was created to the CUSTOMER database table, as shown in Figure 7-28.



Open Table - CUSTOMER

KCGG1F4 - DB2 - ITSO - ITSO.CUSTOMER

Edits to these results are performed as searched UPDATEs and DELETEs. Use the Tools Setting to change the form of editing.

ITSO_ID	TYPE	STATUS	NAME	CREATION_DATE
100	Cstomer	Active	JCA Test	
101	Customer	Active	John Doe	

Figure 7-28 Creation of customer record to the ITSO database

A

WebSphere Integration Developer installation

This appendix includes an example of installing WebSphere Integration Developer on a Windows platform.

Installation of WebSphere Integration Developer

There are a number of scenarios that you can follow when installing WebSphere Integration Developer. This scenario illustrates just one of the methods.

Hardware and software requirements

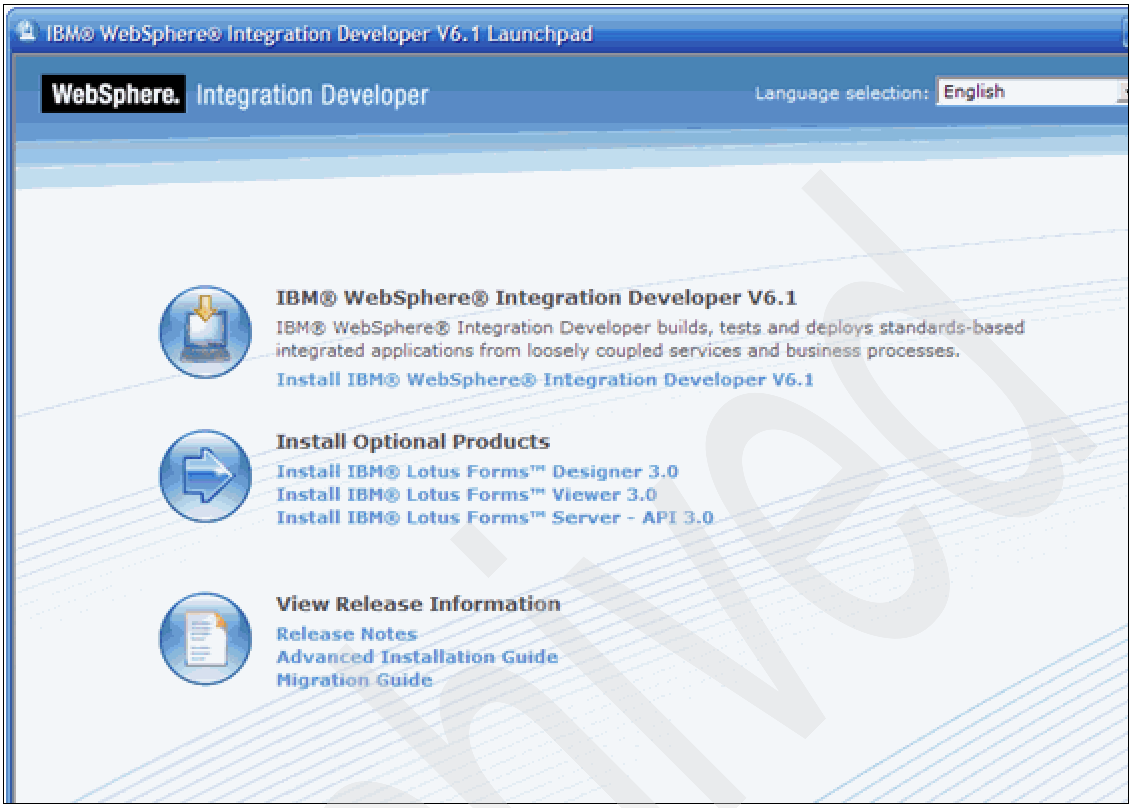
WebSphere Integration Developer can run on a Windows or Linux operating system. You can find specific hardware and software requirements at:

`http://www.ibm.com/software/integration/wid/sysreqs/`

Getting started with the installation

The following steps are an example of installing WebSphere Integration Developer on a Windows operating system:

1. Double-click **launchpad.exe** in the root directory of the installation media.
2. On the Launchpad (Figure A-1 on page 505), select **Install IBM WebSphere Integration Developer V6.1**.



IBM® WebSphere® Integration Developer V6.1 Launchpad

WebSphere. Integration Developer

Language selection: English

IBM® WebSphere® Integration Developer V6.1

IBM® WebSphere® Integration Developer builds, tests and deploys standards-based integrated applications from loosely coupled services and business processes.

Install IBM® WebSphere® Integration Developer V6.1

Install Optional Products

Install IBM® Lotus Forms™ Designer 3.0
Install IBM® Lotus Forms™ Viewer 3.0
Install IBM® Lotus Forms™ Server - API 3.0

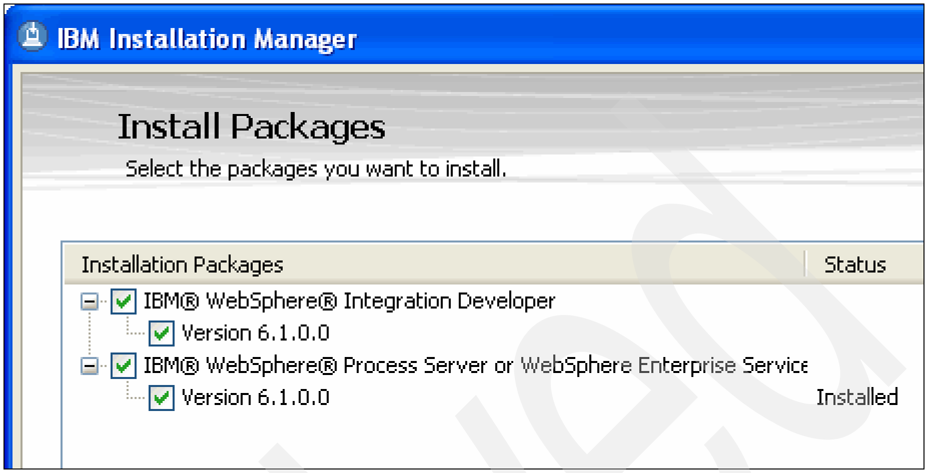
View Release Information

Release Notes
Advanced Installation Guide
Migration Guide

Figure A-1 IBM WebSphere Integration Developer V6.1 launchpad

3. If IBM Installation Manager is not detected on your workstation, then it is installed at the same time as the WebSphere Integration Developer package.

4. From the IBM Installation Manager Install Packages window, select all the options, as shown in Figure A-2.



IBM Installation Manager

Install Packages

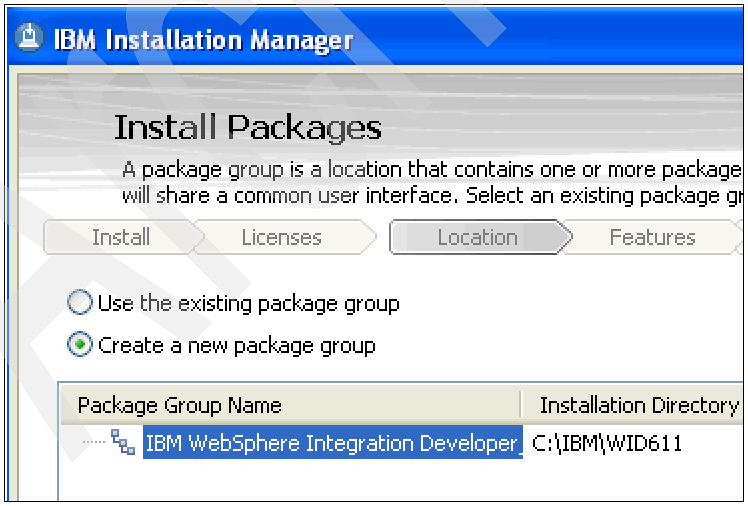
Select the packages you want to install.

Installation Packages	Status
⊟ ☑ IBM® WebSphere® Integration Developer	
☑ Version 6.1.0.0	
⊟ ☑ IBM® WebSphere® Process Server or WebSphere Enterprise Service	
☑ Version 6.1.0.0	Installed

Figure A-2 Select the packages to install

5. When you have read the terms in the license agreements, accept the license by selecting **I accept the terms in the license agreement**.

6. By default, the package group installation directory is C:\Program Files\IBM\WID61 for Windows XP and Windows Server®. In this scenario, we shorten this to C:\IBM\WID61.



IBM Installation Manager

Install Packages

A package group is a location that contains one or more package
will share a common user interface. Select an existing package gr

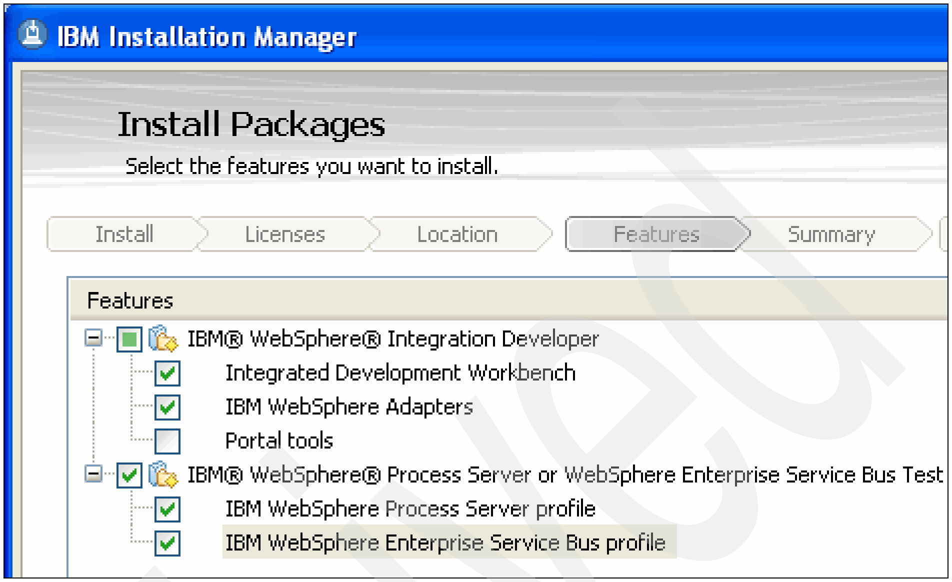
○ Use the existing package group

● Create a new package group

Package Group Name	Installation Directory
IBM WebSphere Integration Developer	C:\IBM\WID611

Figure A-3 IBM Installation Manager Installation Directory

7. Select the features appropriate for your installation. At minimum, we recommend those shown in Figure A-4.


IBM Installation Manager

Install Packages

Select the features you want to install.

Features

- ■ IBM® WebSphere® Integration Developer
  - ☑ Integrated Development Workbench
  - ☑ IBM WebSphere Adapters
  - □ Portal tools
- ☑ IBM® WebSphere® Process Server or WebSphere Enterprise Service Bus Test
  - ☑ IBM WebSphere Process Server profile
  - ☑ IBM WebSphere Enterprise Service Bus profile

Figure A-4 IBM Installation Manager Installation Features

The minimum features shown in Figure A-4 are:

– Integrated Development Workbench

  Provides tools in a comprehensive development environment to build integrated solutions.

– IBM WebSphere Adapters

  Adapters access programs and data on Enterprise Information Systems (EISs). Installing the following adapters and their documentation, which are included with WebSphere Integration Developer Version 6.1, is optional:

  - IBM WebSphere Adapter for Email
  - IBM WebSphere Adapter for FTP
  - IBM WebSphere Adapter for Flat Files
  - IBM WebSphere Adapter for JDBC
  - IBM WebSphere Adapter for JD Edwards EnterpriseOne
  - IBM WebSphere Adapter for Oracle E-Business Suite
  - IBM WebSphere Adapter for PeopleSoft Enterprise
  - IBM WebSphere Adapter for SAP Software
  - IBM WebSphere Adapter for Siebel Business Applications

- IBM WebSphere Process Server profile

  A profile is used to define a separate runtime environment, with separate command files, configuration files, and log files. The WebSphere Process Server profile enables you to run SCA applications, BPEL business processes, human tasks, transition tables, business rules, selectors, and other resources. You can also run mediation flows that are contained in mediation modules. If you intend to eventually deploy an integration module with one or more of these resources to a WebSphere Process Server production server, then you need to install the WebSphere Process Server profile.

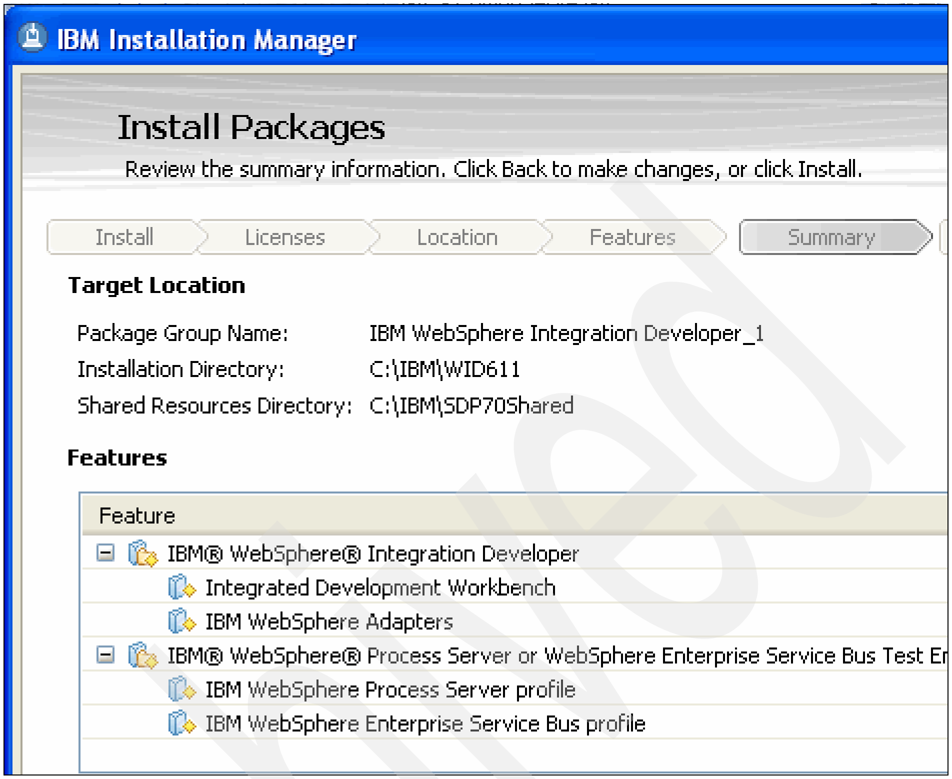
- IBM WebSphere Enterprise Service Bus profile

  A profile is used to define a separate runtime environment, with separate command files, configuration files, and log files. The WebSphere Enterprise Service Bus profile enables you to run mediation flows contained in mediation modules. However, you cannot run BPEL business processes, human tasks, business rules, selectors, and other resources. If you intend to eventually deploy your mediation module to a WebSphere Enterprise Service Bus production server, then you need to install the WebSphere Enterprise Service Bus profile.

- Portal tools

  Provides tools to create, customize, test, debug, and deploy portal applications. The Portal development tools support IBM WebSphere Portal versions 5.1 and 6.0.

8. Click **Finish** to start the installation.



Install Packages

Review the summary information. Click Back to make changes, or click Install.

Target Location

Package Group Name:	IBM WebSphere Integration Developer_1
Installation Directory:	C:\IBM\WID611
Shared Resources Directory:	C:\IBM\SDP70Shared

Features

Feature
⊟ IBM® WebSphere® Integration Developer
Integrated Development Workbench
IBM WebSphere Adapters
⊟ IBM® WebSphere® Process Server or WebSphere Enterprise Service Bus Test Er
IBM WebSphere Process Server profile
IBM WebSphere Enterprise Service Bus profile

Figure A-5 IBM Installation Manager Installation Summary

9. When installation is completed, you start WebSphere Integration Developer by clicking **Start** → **Programs** → **IBM WebSphere Integration Developer** → **IBM WebSphere Integration Developer 6.1** → **WebSphere Integration Developer 6.1**.

B

Additional material

This book refers to additional material that you can download from the Internet as described in this appendix.

Locating the Web material

The Web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247608`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247608.

Using the Web material

The Web material contains a zipped file with project interchange files that are intended for use with IBM WebSphere Integration Developer V6.1. They are provided on an “as-is” basis.

Create a subdirectory (folder) on your workstation, and decompress the contents of the Web material zipped file into this folder.

To import these files to WebSphere Integration Developer:

1. Select **File** → **Import** → **Other** → **Project Interchange**. Click **Next**.
2. Navigate to the *download_location*\MediationSamples\MediationSamples.zip file. Click **Open**.
3. Select all the files and click **Finish**.
   - OrderManagementLib
   - CWYBC_JDBC
   - DBMSServiceMediation

Several of the example mediations use a Derby database. This database is also included with this material. To use the database:

1. Create the C:\itso\sampleDB directory.
2. Copy the *download_location*\MediationSamples\OLDDB database to the new directory.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks” on page 514. Note that some of the documents referenced here might be available in softcopy only.

- *Getting Started with WebSphere Process Server and WebSphere Enterprise Service Bus Part 2: Scenario*, SG24-7642
- *Getting Started with WebSphere Process Server and WebSphere Enterprise Service Bus Part 3: Runtime*, SG24-7643
- *Human-Centric Business Process Management with WebSphere Process Server V6*, SG24-7477
- *Building SOA Solutions Using the Rational SDP*, SG24-7356
- *Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products*, SG24-7148

Online resources

The following Web sites are also relevant as further information sources:

- WebSphere Process Server V6.1 information center.

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wps.610.doc/welcome_top_wps.htm`
- WebSphere Enterprise Service Bus V6.1 information center

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.wesb.61x.root.doc/info/welcome.html`
- WebSphere Integration Developer V6.1 information center

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.wbit.610.help.nav.doc/topics/welcome.html`

- WebSphere Integration Developer V6.1 information center: Configuring and Using Adapters

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.610.help.adapter.emd.ui.doc/topics/tcreatecmps.html

- New platform support, including support for i5/OS and 64-bit Windows and UNIX platforms. A full list of supported platforms can be found at:

  http://www-1.ibm.com/support/docview.wss?rs=2307&context=SSQH9M&uid=swg27009829

- What's new in WebSphere Process Server V6.1

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_fasbinder_wps/0712_fasbinder.html

- IBM WebSphere Developer Technical Journal article, *A guided tour of WebSphere Integration Developer - Part 4 Unleashing visual snippets*

  http://www.ibm.com/developerworks/websphere/techjournal/0606_gregory/0606_gregory.html#VSE

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Getting Started with IBM WebSphere Process Server and IBM WebSphere ESB Part 1: Development

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus
Part 1: Development

Build business integration applications

Build mediations

Use adapters

This IBM Redbooks publication provides developers with information about building and testing applications for IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus. It helps developers with the tasks of creating business integration applications and mediations. It also includes information about the use of adapters.

This is the first book of a three-part series:

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus:

- *Part 1: Development*, SG24-7608
- *Part 2: Scenario*, SG24-7642
- *Part 3: Run time*, SG24-7643

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7608-00 ISBN 0738430013