WebSphere. software

IBM

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 2: Scenario

Learn to create business integration applications

Learn to create mediations

Learn end-to-end test techniques

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

Redbooks

ibm.com/redbooks

International Technical Support Organization

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 2: Scenario

July 2008

SG24-7642-00

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

First Edition (July 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:

IBM®	OS/2®	Redbooks (logo) ®
Lotus®	Rational®	Tivoli®
MVS™	Redbooks®	WebSphere®

The following terms are trademarks of other companies:

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

Expression, and the Windows logo are trademarks of Microsoft Corporation 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 illustrates the concepts and techniques that are associated with building business integration applications and mediations by example. It starts by designing a solution for an order management process. The solution includes a business process and several mediations. It then shows how each component of the solution is created and tested in a development environment. This book also illustrates the use of three adapters:

- The IBM WebSphere® Adapter for JDBC™
- The IBM WebSphere Adapter for Flat Files
- The IBM WebSphere Adapter for Email Version

This book is the second 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

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

Amanda Watkinson
IBM UK

Giancarlo Giannini
IBM Italy

Federico Senese
IBM Italy

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

Scenario high-level design

This chapter describes the logic and high-level design for the example scenario that we use throughout this book. *ITSOCompany* company uses two different ordering and fulfillment processes to process orders for regular and special (out of catalog) items.

In the first phase of development, the customer requirement is to automate the regular order management process. The new system (referred to as *Order Management System*) must be built in a way that allows future development for handling special orders without having a major impact on the system developed in the first phase.

The Order Management System must accept orders from sales representatives and must approve or deny the order based on the customer's financial status and the availability of the item in warehouses.

This chapter includes the following topics:

- Scenario description
- Scenario high-level design
- Scenario architecture
- Development process for the scenario
- Code repository
- Design limitation

1.1 Scenario description

> **Additional material:** The scenario that we describe in this book and all the files that are required to build it are included in the additional materials that are available for download. For information about downloading and using this material, see Appendix B, “Additional material” on page 461.

This section describes the scenario that we developed and documented in this book in terms of workflow and general activities. The starting point for our design is similar to what you might see if a business analyst had created a business analysis model using WebSphere Business Modeler. For more information about business analysis and modeling, see *Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products*, SG24-7148.

Figure 1-1 summarizes the overall process for the scenario. We develop this workflow step-by-step throughout the next chapters.

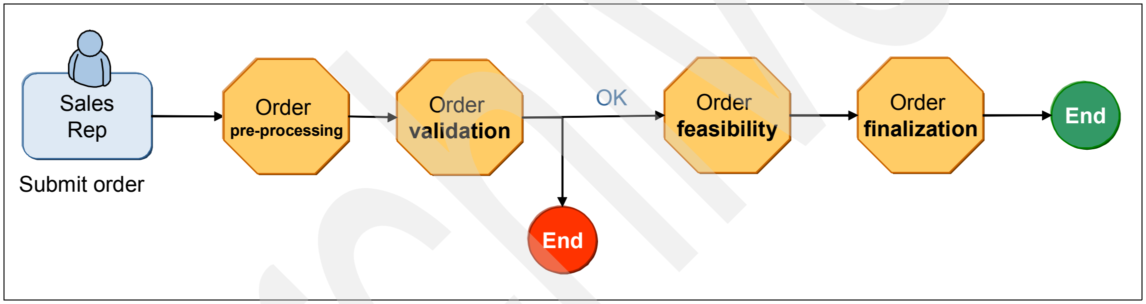


Figure 1-1 Overall order management flow

In this workflow, a sales representative submits an order. The order can pass through as many as four phases for processing:

- Order submission and pre-processing
- Order validation
- Order feasibility
- Order finalization

1.1.1 Order submission and pre-processing phase

Figure 1-2 shows logic for order pre-processing.

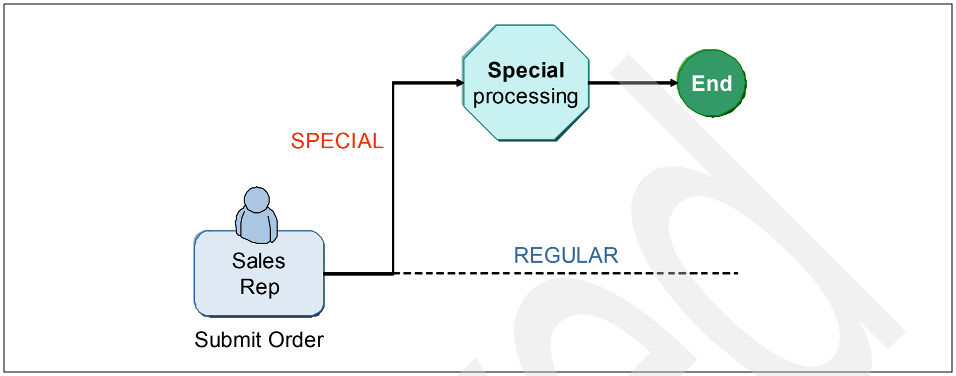


Figure 1-2 Order pre-processing phase logic

To submit an order, the sales representative must provide the following information:

- A unique customer identification (referred to as *customer ID* throughout this book)
- An item identification (referred to as *item ID* throughout this book) or a description of the item for special orders
- A quantity of items to be ordered
- An indicator if this is a special order

A *special order* is an order for an item that is not in the regular catalog. When a special order is placed, a request for special handling is submitted. Special order handling is out of the scope of the Order Management System for now, so the process terminates. The application can be extended later to handle the special processing.

Orders that are for items contained in the catalog are considered regular orders and the process continues to the order validation phase.

1.1.2 Order validation phase

Figure 1-3 shows logic for the order validation.

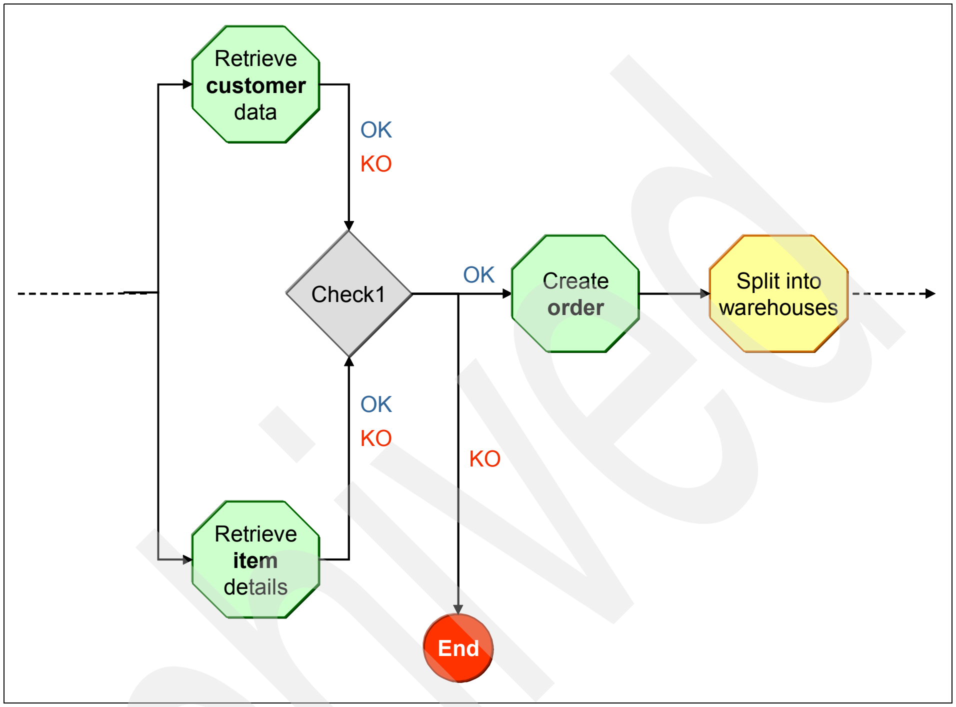


Figure 1-3 Order validation phase logic

In the order validation phase, the customer ID and item ID provided in the request go through a validation check, which are performed in parallel, as follows:

- The customer ID validation ensures that the customer is registered by verifying the existence of the ID in a customer data repository. If the ID exists, information about the customer is retrieved.
- The item ID validation ensures that an item by that number exists in the catalog repository.

If either the customer ID or item ID do not have corresponding entries in the repository, the process is terminated. If both IDs are found in the repository, details are retrieved and an order is created in the repository.

The process goes then through a warehouse split algorithm that determines which warehouses are involved filling the order. There can be up to three warehouses involved, *Warehouse A*, *Warehouse B*, and *Warehouse C*. The order validation phase is terminated and the order feasibility phase is entered.

1.1.3 Order feasibility phase

Figure 1-4 shows the logic for the order feasibility phase.

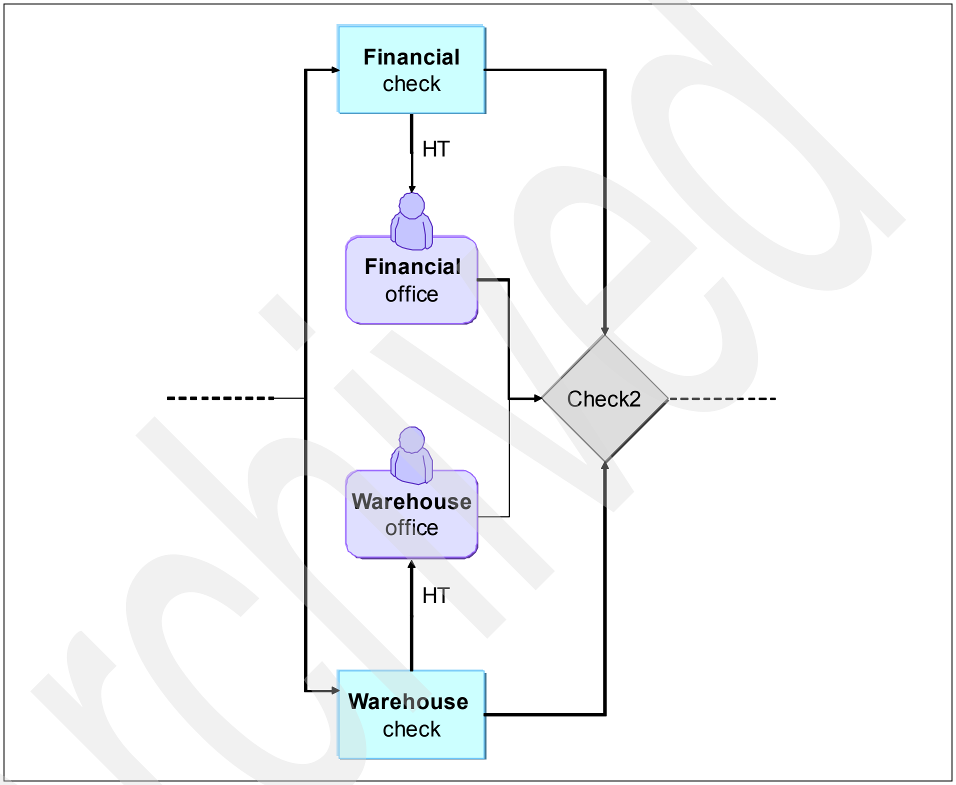


Figure 1-4 Order feasibility phase logic

Order feasibility is based on the following checks:

- A financial check of the customer
- A warehouse check for the availability of the item.

Financial check

The financial check is based on customer financial status in terms of exposure. Every customer has a yearly budget. Customer yearly expenses (from previous requests) are held in a sold-to-date field. Customer exposure is defined as follows:

```
exposure = order amount - (customer budget - customer sold-to-date)
```

The check uses the following logic:

- If customer is found to have no exposure or to have an exposure below a certain threshold (T1), the check is successful.
- If the customer exposure is greater than threshold (T2), the check will be unsuccessful.
- If the customer exposure is between the two thresholds, a financial officer decides whether to approve the order.

Table 1-1 summarizes the financial check logic.

Table 1-1 Financial check logic

Condition	Result
Exposure < T1	Accept
Exposure > T2	Refuse
T1 < Exposure < T2	Human decision

As an example, assume the default thresholds are 1000 for T1 and 10000 for T2. A customer has 15000 as budget and 0 sold-to-date at the beginning. The exposure is negative (-15000) which means no exposure is found. The following process occurs:

1. The customer submits an order where the amount is 7000, the exposure is negative (-8000) and lower than T1, so the order is approved. The sold-to-date value now becomes 7000.
2. Another order is submitted for the customer with 8500 as the amount. Now exposure is positive (500) but still under the T1 threshold and the order is accepted again. The value for sold-to-date becomes 15500.
3. Another order is submitted, with 1000 as the amount. The exposure is now 1500, in between T1 and T2 (which is 10000). So the order will go through a human decision process. It is approved and the sold-to-date value is 16500.
4. And finally, the customer submits another order with 9000 as the amount. Exposure now is 10500 and is higher than T2. The order gets automatically refused.

Figure 1-5 shows a graph for the exposure and sold-to-date trend of this example.

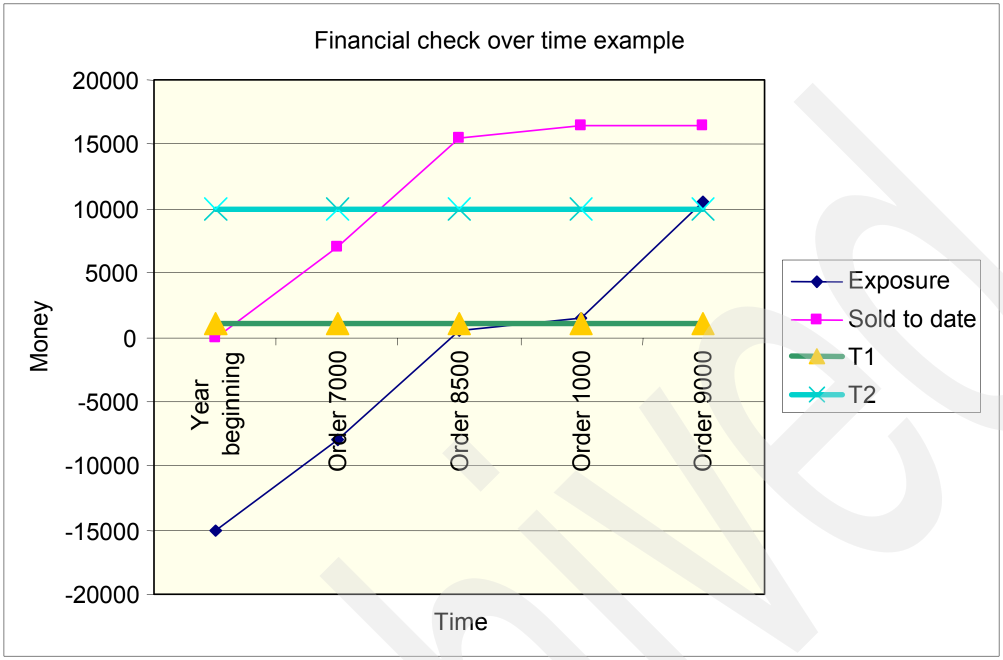


Figure 1-5 Financial check over time example

Warehouse check

The warehouse check is based on the item availability status in terms of deficiency. For an order, we define a warehouse deficiency as follows:

```
item deficiency = item availability - order warehouse requested quantity
```

The process for this check is as follows:

1. For any warehouse, if the deficiency is below a certain threshold T1, the check will be successful.
2. For any warehouse, if the deficiency is higher then a second threshold T2, the check will be unsuccessful.
3. If deficiency is within the two thresholds, a warehouse officer will be responsible for deciding whether or not to approve the order proposal for that warehouse.

Table 1-2 summarizes the warehouse check logic.

Table 1-2 Warehouse check logic

Condition	Result
Deficiency< T1	Accept
Deficiency> T2	Refuse
T1 < Deficiency < T2	Human decision

The entire warehouse check is successful if all of the warehouse deficiency values are under the threshold T1. The check is unsuccessful if just one warehouse deficiency value is over the T2 threshold. The process activates a human task if at least one warehouse deficiency is between the T1 and T2 thresholds, and no warehouse deficiency is over T2.

Finally, the feasibility phase reports a status of successful only if both the financial and warehouse checks are successful. Otherwise, it reports a status of unsuccessful to the last phase: the finalization phase.

1.1.4 Finalization phase

The finalization phase is responsible for completing the order or for rejecting the order as shown in Figure 1-6.

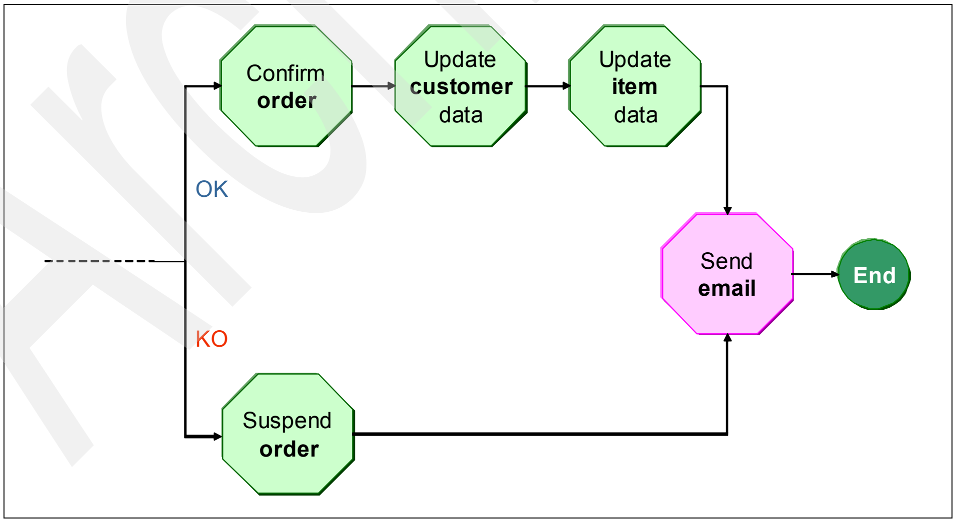


Figure 1-6 Order finalization phase logic

In case of success in the order feasibility phase, the activities are:

1. The order status is changed to “confirmed” in the order repository.
2. The customer data is updated with the new financial status in the customer repository.
3. The item data is updated with the new sold-to-date amount in the item repository.

In case of an unsuccessful feasibility phase, the order status is changed to *suspended* in the order repository.

In both cases an e-mail is sent to the order submitter that contains order status information.

This completes the process logic.

1.2 Scenario high-level design

This section describes the various high-level design choices related to the process described in 1.1, “Scenario description” on page 2. By *high-level design* we mean we do not go into details of single component design, such as mediation logic. Instead, we cover those topics in the specific chapters that provide the implementation details.

In 1.1, “Scenario description” on page 2, we describe the customer, item and order repositories. Potentially, these three repositories are separate from each other; however, as a design choice for the example, the same repository is used for the three object types. We refer to this repository, which is based on Apache Derby, simply as *the database* or *the repository* throughout this book.

1.2.1 Order submission and pre-processing

According to our design, a sales person starts the process by placing an order. This order might be done through a client Web application. For the purposes of this book, we use the Business Process Choreographer Explorer application to test the process.

The submitter is responsible for providing information that includes the customer ID, item ID, and quantity and indicates whether this is special handling request. If the sales person indicates the request requires special handling, the order is logged and is handled manually, external to the process.

From a design point of view, we account for the possible future substitution of this simple logging with a *special order handling* system. We use a Order Pre-Processing Mediation to determine whether the request requires special handling and to route the request appropriately. For special handling, the request is logged using the IBM WebSphere Adapter for Flat Files. Regular requests are routed to an Order Management process.

Figure 1-7 shows pre-processing through mediation.

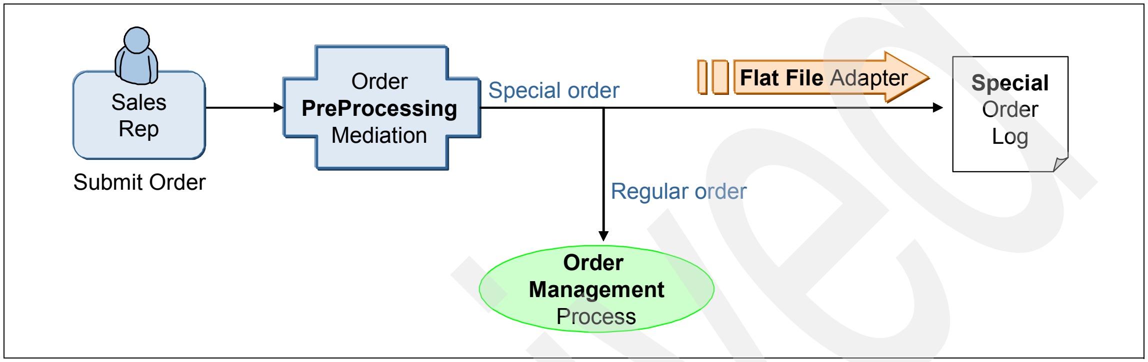


Figure 1-7 Order pre-processing through mediation

When a special order process is developed at a later date, a simple change to the routing in the mediation sends special orders to the new process.

We use mediations running in IBM WebSphere Enterprise Service Bus (WebSphere ESB) as a means of loose coupling among the general process and the services used. That is one of the most powerful patterns in Business Process Management design.

1.2.2 Order validation

Order validation checks for the existence of the customer ID and item ID against a database and retrieves details if these IDs exist in the database. Orders are also created in the database. For these database interactions, a DBMS Mediation is developed. This mediation uses the IBM WebSphere Adapter for JDBC to access the database.

Figure 1-8 on page 11 shows order validation activities.

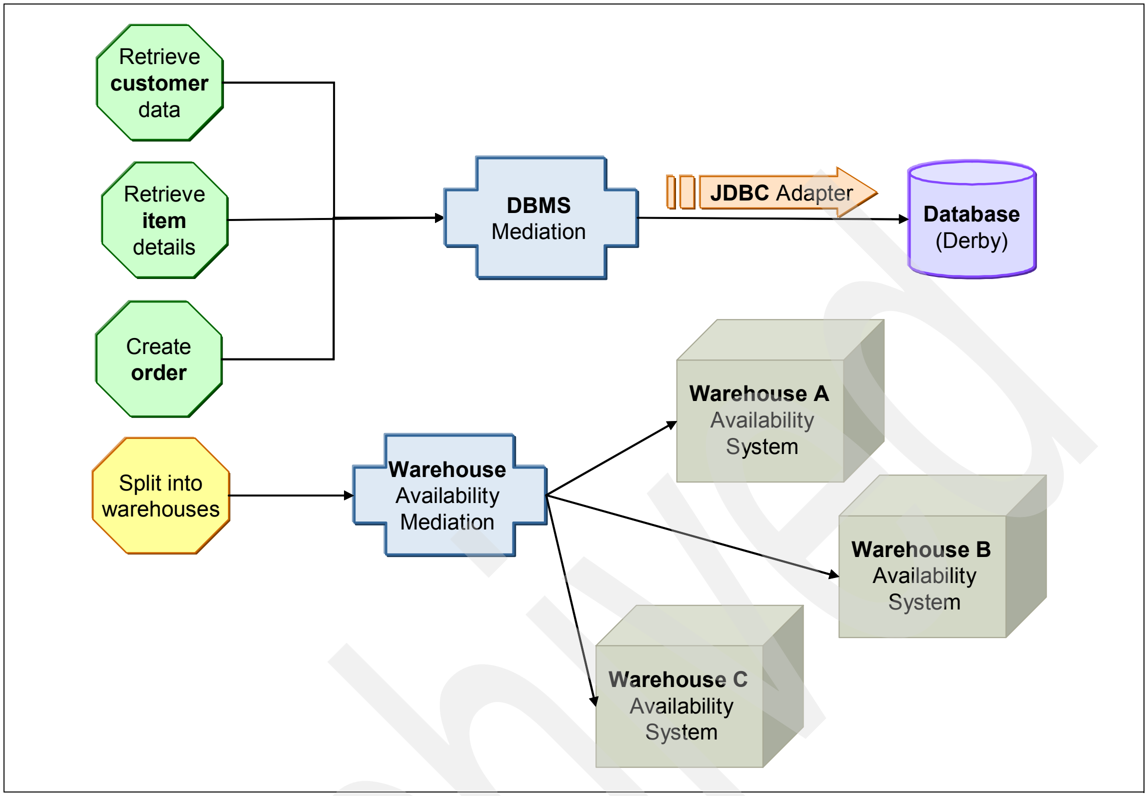


Figure 1-8 Order validation activities and related components

The Split Into Warehouses activity will be performed by dedicated Warehouse Availability Mediation. It first verifies how many of the three warehouses are generally available. The rationale that is used by the Warehouse Availability Systems is unknown; however, understanding the rationale is unnecessary for this discussion. We just know the warehouse provides a dedicated Web service that provides an *available* or *not available* response when invoked. The mediation must then calculate the quantity of the item that is ordered to be supplied by each available warehouse.

1.2.3 Order feasibility

The order feasibility phase consist of the financial and warehouse checks. The checks behave differently depending on certain exposure and availability thresholds, so they are implemented as business rules. This implementation gives business users the ability to update the thresholds over time (using the Business Rule Manager) without coming back to the development team.

Depending on the business rule and how the financial and warehouse data fall between the thresholds, the process can approve or reject the order, or it can ask a human to make a final decision. When a human is involved, we represent such a situation using *human tasks*.

Figure 1-9 shows order feasibility tasks.

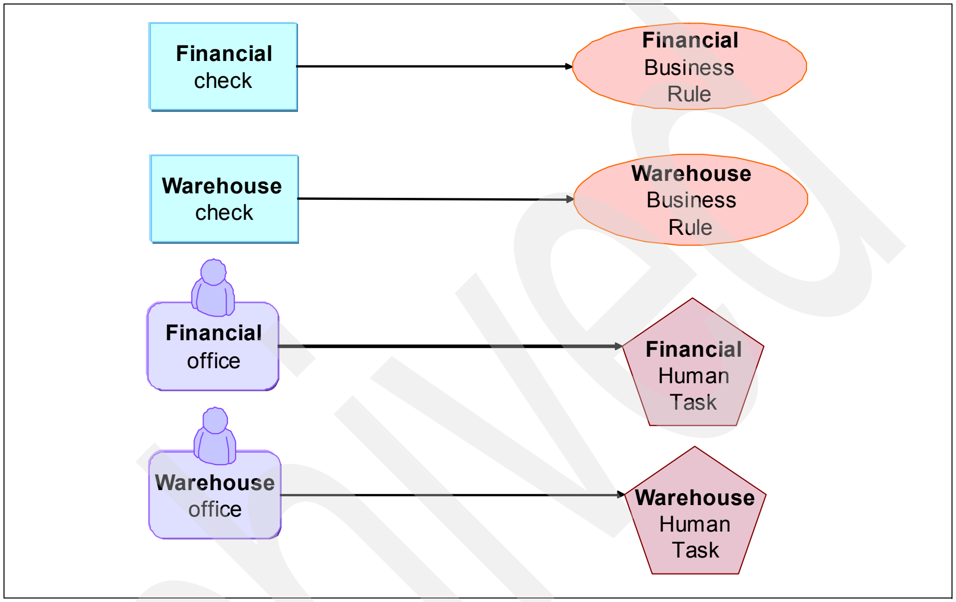


Figure 1-9 Order feasibility activities and related components

1.2.4 Order finalization

Order finalization activities differ depending on the result of the order feasibility phase.

Assuming the result of the order feasibility phase is approval of the order, data must be updated in the repository for customer, item and order information. These updates are performed by invoking the DBMS Mediation.

When an order is rejected by the order feasibility phase, only order data must be updated, again by invoking DBMS Mediation**.** An e-mail is also sent to the submitter. The e-mail address is retrieved in the *Customer data retrieve* activity in the order submission and validation phase. The e-mail is sent using the IBM WebSphere Adapter for Email, which connects to an e-mail system (SMTP server).

Figure 1-10 on page 13 shows the order finalization activities.

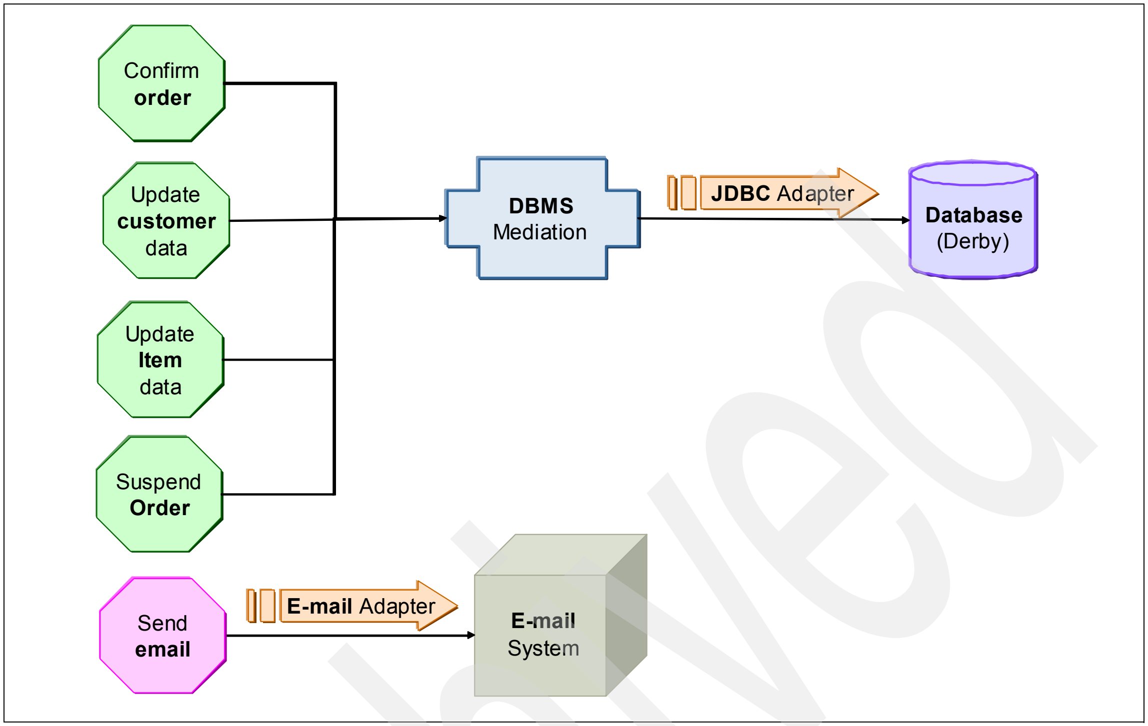


Figure 1-10 Order finalization activities and related components

1.3 Scenario architecture

From the high-level design, we determined which components are required to build our Order Management application and the nature of these components. In this section, we clarify how these components are deployed.

The middleware products that we use are:

- *IBM WebSphere Process Server V6.1* for running choreography in the form of a business process, using business rules, human tasks, and Java components in the appropriate containers.
- *IBM WebSphere Enterprise Service Bus V6.1* for running mediations.
- *IBM WebSphere Adapters* (JDBC, E-mail, and Flat File) for access to these technologies.

The design, development, and deployment of these application components across these three products is the focus of the entire book. The overall architecture also uses other systems such as:

- An e-mail system for sending mail.
- A User Registry system for authentication and authorization from WebSphere Process Server, including those functions that are required for handling human task assignment.
- A Database Management System (*Apache Derby*, which is provided with WebSphere Process Server) for storing application data.
- A Warehouse Availability general system that can be accessed through provided Web services.

Figure 1-11 shows the overall architecture of the solution.

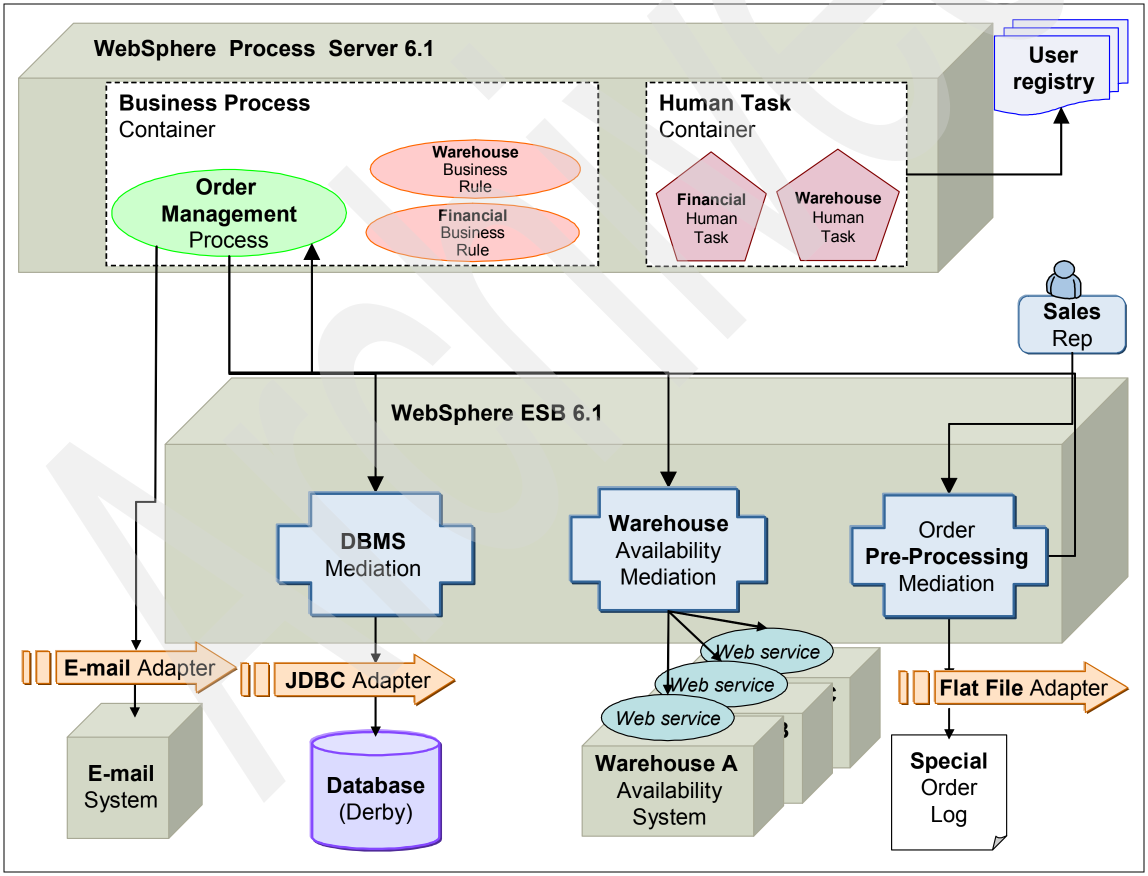


Figure 1-11 Overall architecture

The components and the connections shown in Figure 1-11 are as follows:

- The Sales Rep starts the process using a Web application (Business Process Choreographer Explorer in our case).
- The order request goes through the Order Pre-Processing Mediation running in WebSphere ESB.
- The Order Pre-Processing Mediation routes special orders to a file using the IBM WebSphere Adapter for Flat Files. It routes regular orders to the Order Management process running in WebSphere Process Server, specifically in the business process container.
- The Order Management process interacts with the following components in WebSphere Process Server and the mediations running in WebSphere ESB:
  - Warehouse and Financial business rules in the business process container.
  - Warehouse and Financial human tasks in the human task container.
  - DBMS mediation and Warehouse Availability mediation running in WebSphere ESB.
- The DBMS mediation provides access to the database repository through the IBM WebSphere Adapter for JDBC.
- The IBM WebSphere Adapter for EMail provides direct access to the e-mail system.
- The Warehouse Availability mediation running in WebSphere ESB provides access to the Warehouse Availability Systems using the provided Web services.

The architectural pieces of the solution can be deployed on one system or can run remotely from each other on separate systems.

1.4 Development process for the scenario

You might find it useful to know the basics of the organization and development process that we followed while developing the example. This example is a simple example and can serve as such for general use in Business Process Management design and development. For more in-depth information about service-oriented architecture development, see *Building SOA Solutions Using the Rational SDP*, SG24-7356.

In our sample solution, the architecture is divided clearly into two main blocks:

- Choreography and components to be developed for execution in WebSphere Process Server
- Mediations to be developed for execution in WebSphere ESB

In the project that produced this book, two development teams participated in the overall design, then worked separately to develop these two parts of the architecture as shown in Figure 1-12.

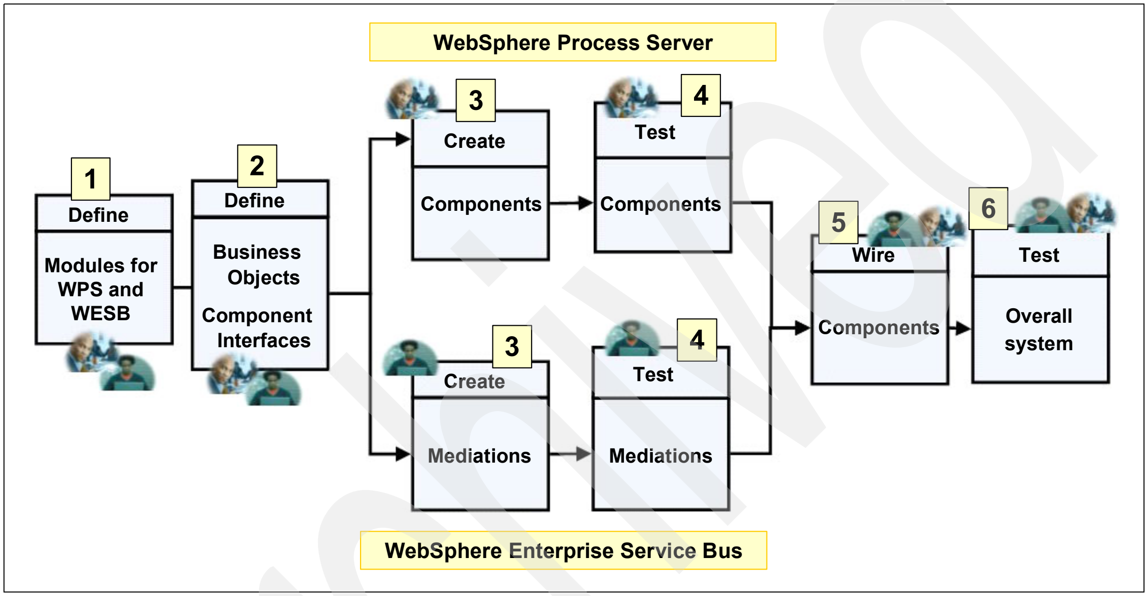


Figure 1-12 Overall development process for the scenario (simplified)

The modules that are defined for both WebSphere Process Server and WebSphere ESB are part of the high-level design.

The business objects and component interfaces are defined together, as a way to establish a technical *contract* between service requestor and service provider, allowing the developers of both sides to work independently of one another. In most cases, mediations play the role of service provider to WebSphere Process Server components; however, there are cases where mediations might see WebSphere Process Server components as a service provider. An example of this is the Order Pre-Processing Mediation that gets the order request from the submitter and then eventually passes it to the WebSphere Process Server Order Management process.

Business objects have been also defined together as a part of the “collaboration rules” between WebSphere Process Server components and WebSphere ESB mediations.

After designing the business objects and interfaces, the two teams proceeded in parallel to develop component and mediations top down from the interfaces.

Unit testing was performed on components and mediations along the way. Toward the end of the development cycle, the two teams joined again to wire the modules together into a single system similar to a production environment and to perform the overall testing.

Because the interactions between the modules is vital to the design, we describe the major components and interfaces in this section.

1.4.1 Module definitions

The first phase of the development process was the module definition.

A common library is used as a repository for business objects and component interfaces, because these components are accessed by every business integration and mediation module. This library is OrderManagementLib.

The following modules are defined:

- A single module for all of the WebSphere Process Server components named *OrderManagement*.
- Three mediation modules for hosting WebSphere ESB mediations:
  - DBMS Mediation
  - Warehouse Availability Mediation
  - Order Pre-Processing Mediation

1.4.2 Interfaces and business objects definition

The second phase of the development process defines the interfaces and business objects in terms of what is needed to satisfy the business process flow. The design of the business objects and interfaces goes hand-in-hand. However, we describe the business objects first because they are referred to by the interfaces.

Business objects

According to the design, the *business objects* that are required by the application include:

- An *OrderManagementInput* business object that contains the input properties for the OrderManagement module.
- A *Customer* business object that contains information about the customer. These properties are retrieved from the repository and updated in the OrderManagement module. The business object includes general properties describing the customer (name, address, and so forth) and financial information such as *budget* and *soldToDate*.
- An *Item* business object that contains information about the ordered item. This information is retrieved from the repository and updated in the OrderManagement module. It includes an array of warehouses that can potentially supply the item. This array is populated when the item information is retrieved from the database. Each Warehouse object in the array contains the current number of the ordered items in stock and the number that it will deliver.
- A *Warehouse* business object which represents an item's fulfillment by a warehouse. This holds information about item *stock* quantity, *in-delivery* quantity (the number being delivered to customers), and a *partial* quantity value that represents the quantity that is requested to be delivered from this warehouse as determined by the Split into warehouse activity.
- An *Order* business object that is created and updated with the order status as it passes through the process.
- A *ReturnCode* that contains a return code string. The interfaces will use this to provide information about the success or failure of an operation. Using a business object to contain this string provides the possibility of using business object maps for different return code types (that is integers) that might be returned from different services.

The details of these objects are worked out later in the design.

Interfaces

The business process and mediation developers worked together to define the following *interfaces* at the border of the two processing areas:

- The OrderManagement module interface (*OrderManagementIF*) defines the way the Order Pre-Processing Mediation accesses the OrderManagement module**.** The interface takes an object of type OrderManagementInput.
- The Customer Service interface (*CustomerServiceIF*) defines the way the OrderManagement module accesses the Customer data related activities, such as *Retrieve customer data* and *Update customer data*. As expected, this

interface provides retrieve and update services for the Customer business object and returns a return code in both cases.

- Order Service interface (*OrderServiceIF*) defines the way the OrderManagement module accesses the Order data related activities, such as *Create order*, *Confirm order*, and *Suspend*. For such activities the interface provides create and update services for an Order business object, getting Item and Customer details in both cases and returning a return code.
- Item Service interface (*ItemServiceIF*) defines the way the OrderManagement module accesses the Item data related activities, such as *Retrieve item details* and *Update item*. Retrieve and update services are provided, taking an Item ID as the parameter and returning a return code in both cases.
- Warehouse Item Split interface (*WarehouseItemSplitIF*) defines the way the OrderManagement module accesses the Warehouse Availability Mediation for the implementation of the *Split into warehouse* activity. As expected, just a split service is provided, with Item and Order as the request parameters and the warehouse split returned in an Item business object.

CustomerServiceIF, OrderServiceIF, and ItemServiceIF act as the boundary between the business process layer and the mediation layer for database access. They are implemented by DBMS Mediation.

WarehouseItemSplitIF acts as the boundary between the business process layer and the mediation layer for warehouse split algorithm. They are implemented by Warehouse Availability Mediation

1.4.3 Create and test the single components

We describe component development and testing of the single modules in the following chapters:

- Chapter 3, “Order Management Process business integration module” on page 67 describes the development of the OrderManagement module.
- Chapter 4, “OrderPreProcessingMediation module” on page 249, Chapter 5, “DBMSServiceMediation module” on page 297, Chapter 6, “WarehouseAvailabilityMediation module” on page 353 describe the development of the mediation modules.

1.4.4 Wiring the components

Modules are wired together using their import and export components. We describe this in detail in Chapter 7, “End-to-end test” on page 415.

1.4.5 Testing the overall system

The WebSphere Process Server and WebSphere ESB teams develop and test their components independently. At the end of the process, an overall test ensures that the Order Management System works from end-to-end. The entire application spans WebSphere Process Server and WebSphere ESB. These runtime processes can reside on one or multiple systems.

The primary test and deploy environment for this book is a single WebSphere Process Server instance.

> **Note:** Mediation modules and modules for business processes can both execute in a WebSphere Process Server application server.

In Part 3 of this series (*Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus:, Part 3: Run time,* SG24-7643), we deploy the same application to a distributed runtime environment. The mediations run in a cluster of WebSphere ESB capable application servers and the process runs in a cluster of WebSphere Process Server application servers.

1.5 Code repository

A Concurrent Version System (CVS) is used as the team artifact repository for sharing code.

The OrderManagementLib that contains the common interfaces and business objects is the first item checked into the CVS. The business process and mediation teams need this library to get started. They can access the CVS server and check out the library using WebSphere Integration Developer.

1.6 Design limitation

Finally, it is worth pointing out the obvious. The scenario that we use in this book is for illustration purposes only. It is not intended to be production ready. It is designed for simplicity and does not take into account many considerations that you have to consider in a production application (for example, concurrency of requests).

2

Preparing for development

This chapter recommends an approach for developing business integration applications. It discusses design decisions at a more detailed level than we discuss in Chapter 1, “Scenario high-level design” on page 1.

It also includes information about how to prepare the business objects and interfaces that provide the foundation for the development of each module and mediation in the Order Management System.

This chapter includes the following topics:

- Preparing to develop
- Map process steps to components and activities
- Database design and structure
- Business objects design
- Component design
- Interface design
- Division of work
- Gather endpoint information
- Obtain or agree upon the sample data
- Prepare the library
- Code repository check in

2.1 Preparing to develop

A recommended approach for developing business integration applications is to follow these steps:

1. Map your process steps to business integration and mediation components.
2. Decide on the business objects that best represent your data.
3. Map your component interactions.
4. Agree on the interfaces that best represent the interactions between your components.
5. Divide the work between developers.
6. Gather necessary information to integrate with an endpoint.
7. Obtain or agree on sample data that you can use to emulate end points.

In the remaining sections of this chapter, we go through these steps as we prepare to develop the scenario.

2.2 Map process steps to components and activities

To map the process steps to business integration and mediation components, it is best to understand the steps of the process. For our scenario, we identified the following steps:

1. A Regular Order is received.
2. The customer and item data are retrieved from a database. These activities are performed in parallel.
3. If both retrieves are successful (the item ID and customer ID exist), then the process continues as follows:
   a. Create the order in the database.
   b. Determine how to fulfill the order by splitting the items across the warehouses based on their availability.
   c. Determine the total stock available across the warehouses.

   If either retrieve fails, the process terminates.

4. A financial check and a fulfillment check (warehouse check) are performed in parallel as follows:
   a. The financial check is based on certain thresholds. The check can either APPROVE or REJECT a customer automatically. If a decision, cannot be made automatically, the customer's information is sent to a financial officer for a decision.
   b. The warehouse check can either APPROVE or REJECT an order automatically. If a decision, cannot be made automatically, the order is sent to a warehouse officer for a decision.

   If either the financial or fulfillment (warehouse) check is unsuccessful:

   – The order is SUSPENDED in the database.
   – An e-mail is sent to the submitter.
5. If both of the previous checks are successful, the order is CONFIRMED. The order, associated item, and customer's information are updated in the database.

2.2.1 Order Management System flow

Figure 2-1 depicts the Order Management System flow.

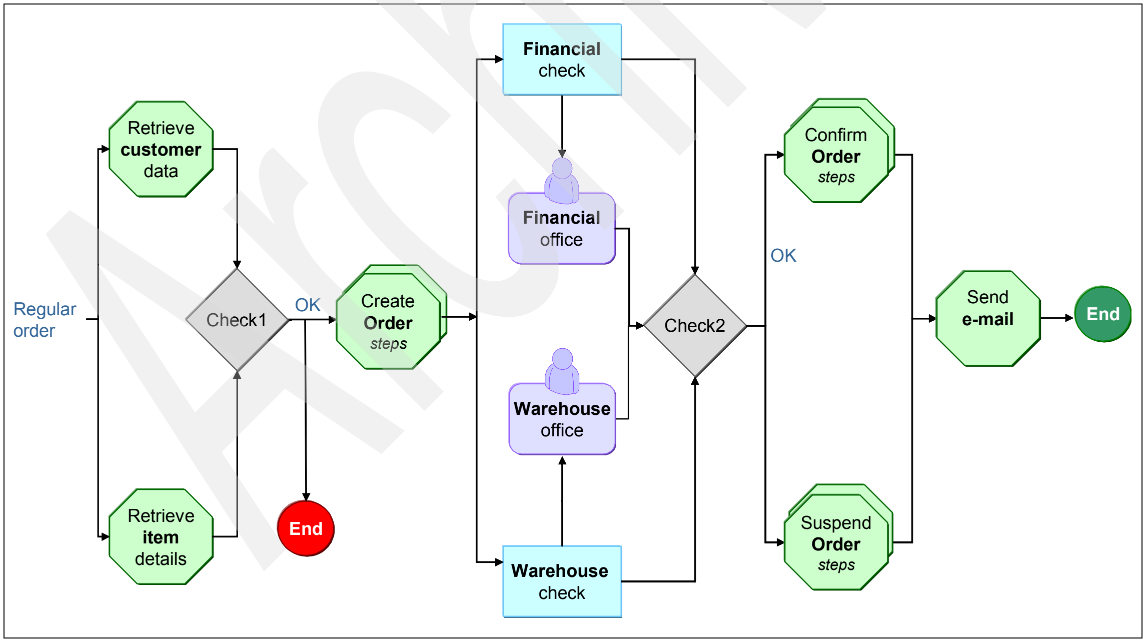


Figure 2-1 Order Management System flow

2.2.2 Component-to-activity map

In terms of components that we use in this scenario, Figure 2-2 identifies the business integration components that will be used.

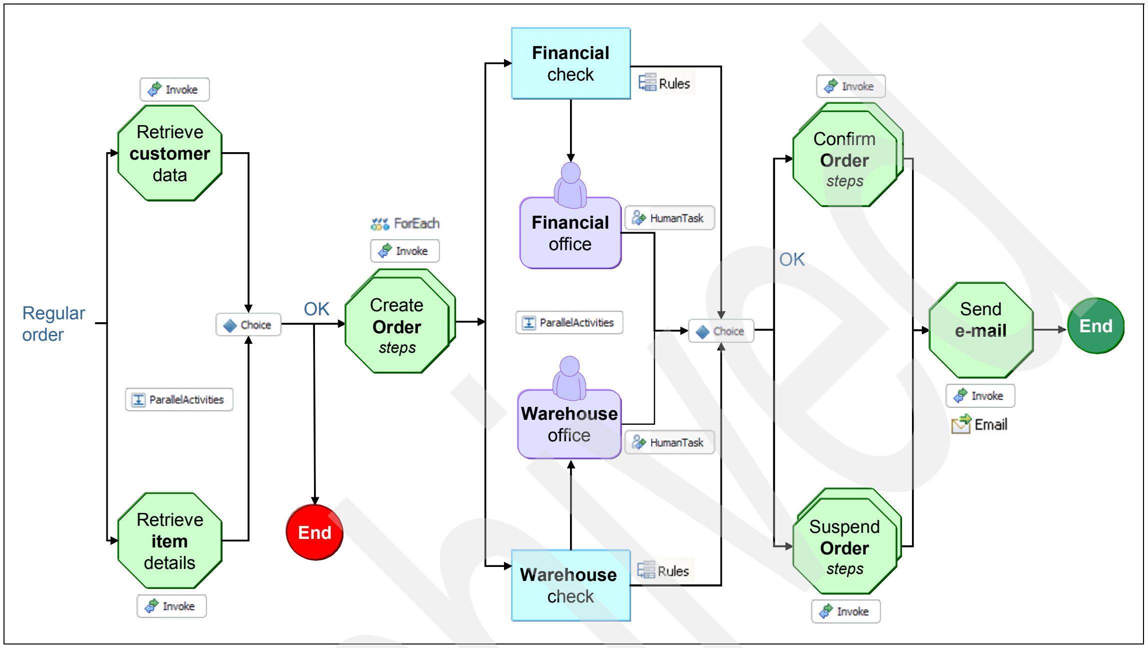


Figure 2-2 Order Management System flow mapped to WebSphere Process Server components

The components that we use include:

- Business integration module
- Library for common artifacts
- Business rules to determine the outcome of the financial and warehouse check
- Mediation modules to provide database access and the warehouse split information
- An e-mail adapter
- BPEL to implement the main process activities. Some specific BPEL activities that we use can be inferred from the diagram:
  - Receive activity
  - Invoke activity
  - Parallel Activities structured activity
  - Choice structured activity

- Human Task activity (inline)
- ForEach structured activity
- Assign activity
- Snippet activity
- Terminate activity

2.3 Database design and structure

Database design is outside the scope of this book, but the structure of the database plays a key role in defining business objects that have fields corresponding to data in database records.

The WebSphere Process Server and WebSphere ESB development teams have an agreement on interfaces and business objects. They also agree that the WebSphere Process Server team will not include any direct access to the database in their module. Requests for database access is passed to the DBMS Mediation mediation module.

For the database structure, we use a single database for the application, which contains the tables that we discuss in this section.

> **Tip:** Make sure that your table names are not the same as an SQL keyword. For example, we originally used ORDER as a table name but changed it to ORDERHEADER to avoid conflict.

2.3.1 Database structure overview

The entity relationship diagram (ERD) in Figure 2-3 provides a total view of the database.

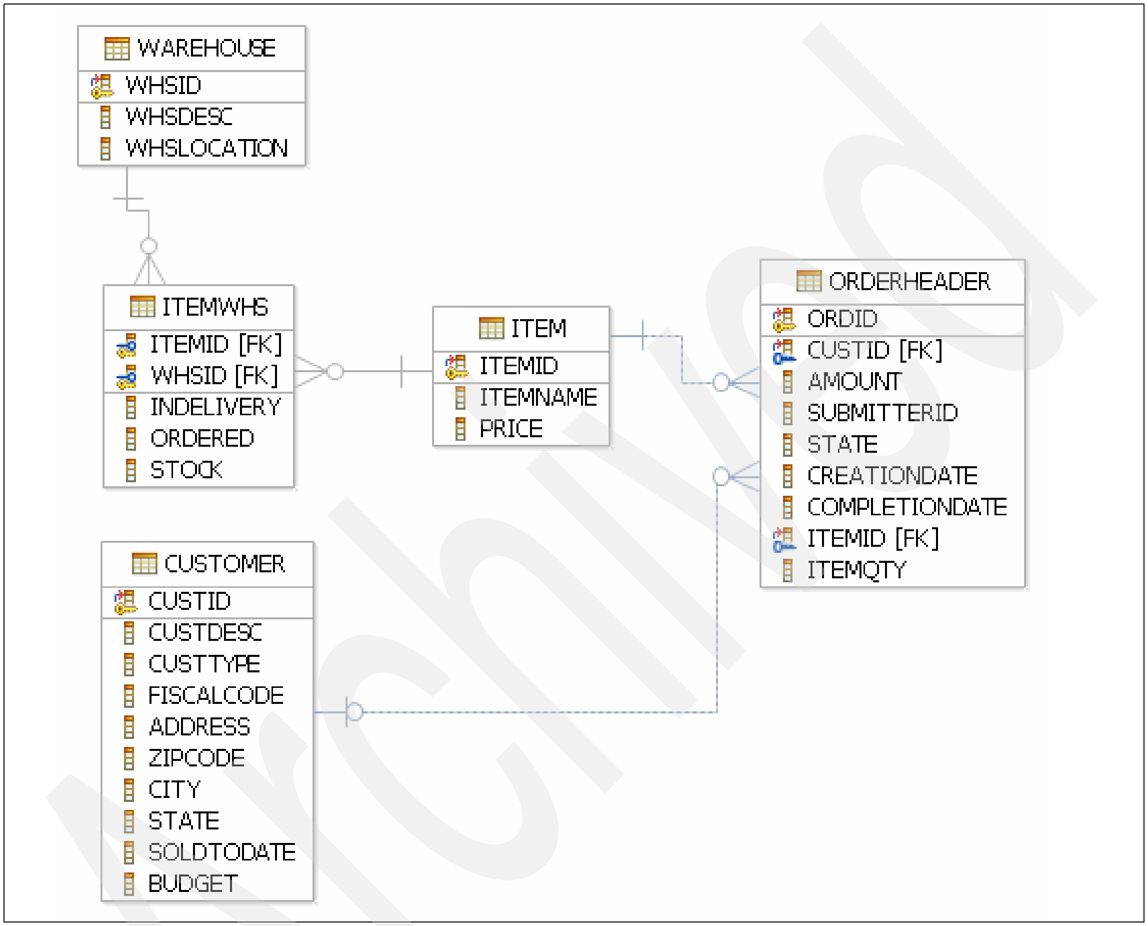


Figure 2-3 Database overall entry relationship diagram

2.3.2 CUSTOMER table

Customer data is stored in a CUSTOMER table with the structure shown in Figure 2-4. This table contains data about the customer including the *budget* and *sold-to-date* information that is used for the financial check.

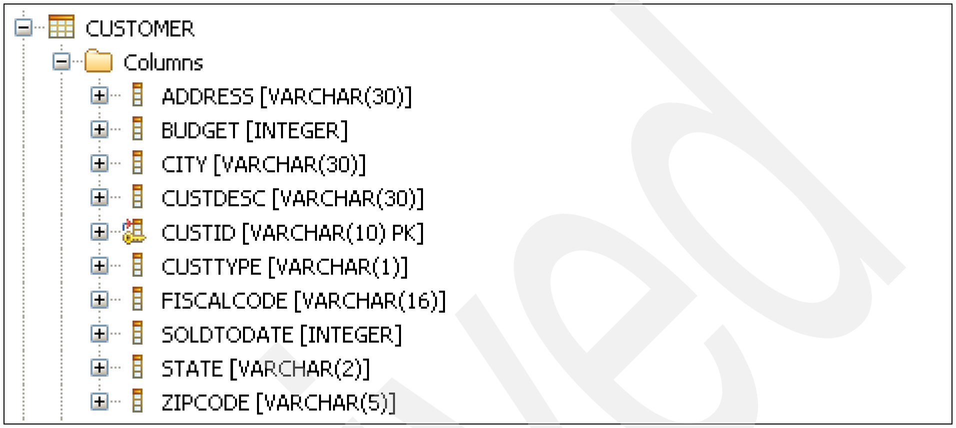


Figure 2-4 CUSTOMER table structure

2.3.3 ITEM table

An ITEM table contains the ID, name, and price of each item that can be ordered from the catalog. The structure is in Figure 2-5.

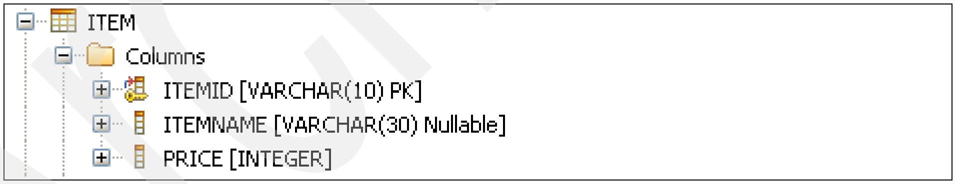


Figure 2-5 ITEM table structure

2.3.4 WAREHOUSE table

A WAREHOUSE table contains information about each warehouse, including the description, ID, and location. The structure is shown in Figure 2-6.

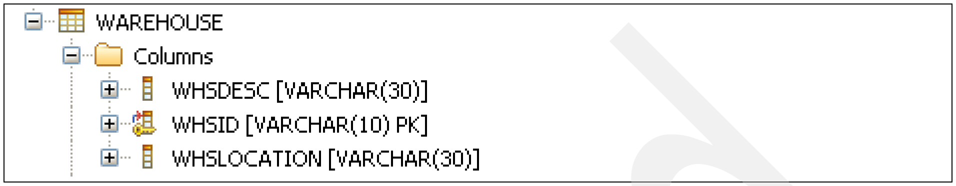


Figure 2-6   WAREHOUSE table structure

2.3.5 ITEMWHS table

Items and Warehouses have a many-to-many relationship because an item can be stored in multiple warehouses and, of course, warehouses store multiple items. A table called ITEMWHS (Figure 2-7) contains the information about the items held in a warehouse, including *Stock* (how many items of a kind are in that warehouse), *Ordered* (how many are about to arrive in the warehouse), and *InDelivery* (how many are about to leave the warehouse and be delivered to the customers).

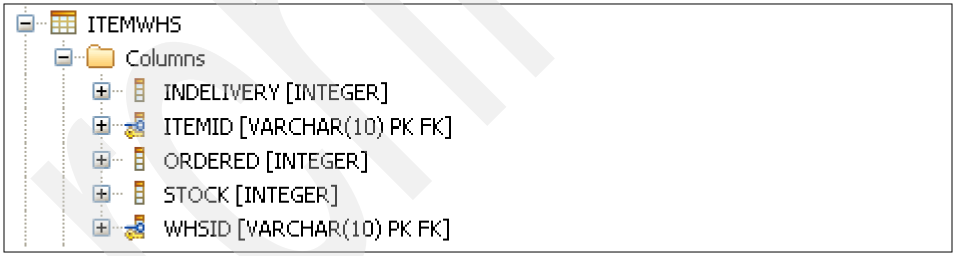


Figure 2-7   ITEMWHS table structure

2.3.6 ORDERHEADER table

An ORDERHEADER table holds data for Orders, including item ID and customer ID, item quantity ordered, and the state of the order, which changes all along the order life in the application. The structure is shown in Figure 2-8.

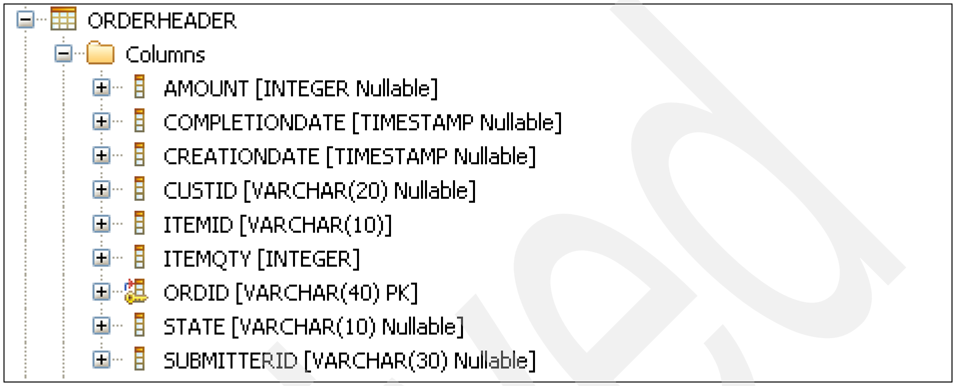


Figure 2-8 ORDERHEADER table structure

> **Note:** We use the name *ORDERHEADER* instead of ORDER because ORDER is a reserved word for some DBMS.

2.4 Business objects design

The data that moves across our application represents our business objects. If we look at our previous diagrams, we can identify the following data immediately:

- Customer
- Order
- Item
- Warehouse

If we look further, we can also see the following:

- The input to the process is a set of data (OrderManagementInput).
- The backend services return the result of lookups, data creation, or update (ReturnCode).
- We need to send an e-mail at the end of the process (EmailOrder).

All of these objects are shared in the common library.

Business objects for our scenario

In this simple example, our back-end database is *Derby*. We chose Derby because it comes as part of the installation.

For this scenario, we can look at the ERD diagram in Figure 2-3 on page 26. The ERD diagram represents the pertinent tables in Derby. We use it as a starting point for the Customer, Order, Item, and Warehouse objects.

> **Note:** In many cases the actual representation of the business object is not the same as the one in the database. The reason for this is that we want our business objects to represent process data not necessarily physical data in the database.

For the business objects based on Derby, the JDBC adapter has an Enterprise Service Discovery (ESD), which is a wizard that can help you create business objects from your database tables.

For the other business objects that we identified (OrderManagementInput, ReturnCode, and EmailOrder), we define their details manually.

2.4.1 Customer

The fields that we need for the process are slightly different than the database fields. For our scenario, we need only the subset of fields shown in Table 2-1.

Table 2-1 Customer business object

Name	Type	Default Value	Min Occurs	Max Occurs
custID	string		0	1
custName	string		0	1
address	string		0	1
zipCode	string		0	1
city	string		0	1
state	string		0	1
budget	int		0	1
soldToDate	int		0	1

2.4.2 Order

Again, the fields that we need for the process are slightly different than the database fields. For our scenario, we need only the subset of fields listed in Table 2-2. We send the customer and item informatoin along with the order, so that those fields are not needed within the process.

Table 2-2 Order business object

Name	Type	Default Value	Min Occurs	Max Occurs
ordID	string		0	1
amount	int		0	1
submitterID	string		0	1
state	string		0	1
creationDate	datetime		0	1
completionDate	datetime		0	1
itemQty	int		0	1

2.4.3 Warehouse

The warehouse fields represent the Item-to-Warehouse holdings. Again, the fields that we need for the process are slightly different than the database fields. For our scenario, we need only the subset of fields listed in Table 2-3.

Table 2-3 Warehouse business object

Name	Type	Default Value	Min Occurs	Max Occurs
whsID	string		0	1
stock	int		0	1
indelivery	int		0	1
itemQtyPartial	int		0	1

2.4.4 Item

The Item business object is different from the other business objects in that we build this as a hierarchical business object that has another business object, Warehouse, as a child. For our scenario, we need all the fields plus we need to include the associated warehouse (see Table 2-4).

Table 2-4 Item business object

Name	Type	Default Value	Min Occurs	Max Occurs
itemID	string		0	1
itemName	string		0	1
price	int		0	1
warehouses	Warehouse		0	unbounded

2.4.5 OrderManagementInput

The OrderManagementInput business object captures the Sales Representative's input into the OrderManagement application. Information entered includes the submitter's ID, their e-mail (for e-mail verification purposes), the customer's ID for whom they are placing the order, the item ID (what is the customer ordering), and the quantify ordered. For our scenario, we need only the subset of fields listed in Table 2-5.

Table 2-5 OrderManagementInput business object

Name	Type	Default Value	Min Occurs	Max Occurs
custID	string		0	1
itemID	string		0	1
itemQty	int		0	1
submitterID	string		0	1
submitterEmail	string		0	1

2.4.6 ReturnCode

The ReturnCode business object is used to capture responses from the database mediation and the rules. To simplify things, we agree upon a few simple codes that are of type string (see Table 2-6).

Note: In this scenario, the error code typically used is 99.

Table 2-6 ReturnCode business object

Name	Type	Default Value	Min Occurs	Max Occurs
RC	string		0	1

2.4.7 EmailOrder

The EmailOrder business object represents the data that you need to send an e-mail to the submitter to say that the order is confirmed or suspended. In this case, we need to know to whom we are sending the e-mail (to), to know who is sending the e-mail (from), to set a subject for the e-mail (subject), and to set the body of the e-mail (body), as shown in Table 2-7.

Table 2-7 EmailOrder business object

Name	Type	Default Value	Min Occurs	Max Occurs
to	string		0	1
from	string		0	1
subject	string		0	1
body	string		0	1

Tip: In real scenarios, the source of your business objects could come from many existing sources (WSDLs, XSDs, and so forth). The process is also iterative. You might first discover a few business objects from certain systems and, as you develop, you might need to refine one or more business objects because you did not know all the information up front. For our scenario, we had to modify our objects several times until we got the final representation, but for documentation purposes, we only provide the final versions.

2.5 Component design

In terms of the components of the application, you have to think about which components will interact with other components. This information is used to determine the assembly diagram of your modules.

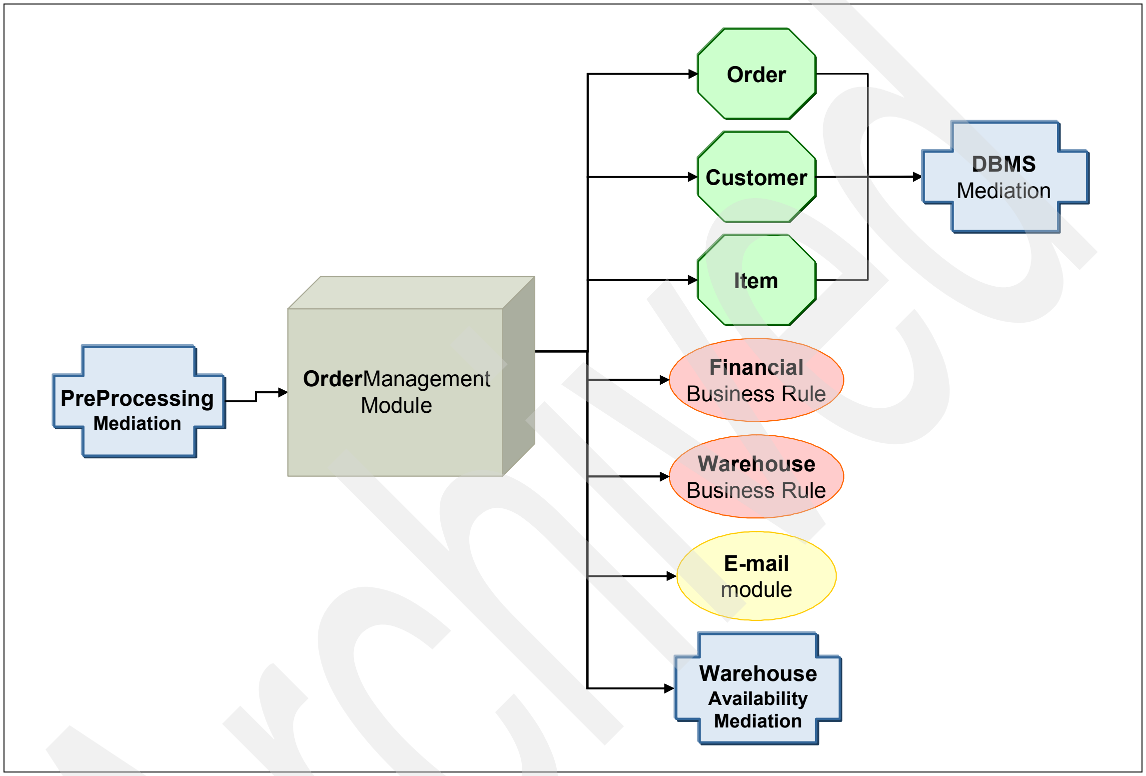


Figure 2-9 Component diagram of the OrderManagement system

Figure 2-9 depicts the interactions that we identified previously. The primary driver for orders (OrderManagement module) receives a regular order from the Order Pre-Processing mediation. It performs the orchestration across WebSphere Process Server and WebSphere ESB modules. This process, in turn, orchestrates the following services (*not necessarily in the following order*):

- It sends order, customer, and item data to be handled by the DBMS mediation.
- It sends order, customer, and item data to the rules for evaluation.
- If the order cannot be placed, an e-mail is sent to the submitter.
- It sends order and item data to a Warehouse Availability mediation to determine the warehouse split (fulfillment of order across warehouses).

2.6 Interface design

Now that we have identified the components and their interactions, the component design lets us know the interfaces that we need for our application. A decision is made as to whether the interfaces are placed in a library for shared use or are specific to a module or mediation and are placed there. Table 2-8 lists the component and interface mapping.

Table 2-8 Component and interface mapping

Component	Interface	Reference(s)
Order Pre-Processing Mediation	OrderPreProcessingIF	OrderManagementIF SpecialOrderIF
OrderManagement	OrderManagementIF	CustomerServiceIF ItemServiceIF OrderServiceIF FinancialOfficeBRIF WarehouseOfficeBRIF EmailServiceIF WarehouseItemSplitIF
DBMS Mediation	CustomerServiceIF OrderServiceIF ItemServiceIF	JDBCOutboundInterface (JDBC adapter)
FinancialOfficeRG	FinancialOfficeBRIF	N/A
WarehouseRG	WarehouseOfficeBRIF	N/A
EmailOutbound	EmailServiceIF	EmailImport
Warehouse Availability Mediation	WarehouseItemSplitIF	GetWarehouseA_Availability GetWarehouseB_Availability GetWarehouseC_Availability

In the next sections, we define the interfaces that each of the modules implements and, for the business integration components, define their references. We discuss the interfaces and references for the mediation modules in later chapters when the mediations are built.

2.6.1 Order Pre-Processing Mediation

This mediation has an interface called *OrderPreProcessingIF* that it exposes for other services to use. When a message arrives through this interface, the mediation determines the type of order based on a field in the input message.

If the order is a regular order, the mediation forwards the order to the OrderManagement module (the bulk of this example) for processing through a reference to the OrderManagement module interface, OrderManagementIF.

Table 2-9 shows the properties for the OrderPreProcessingIF interface. This interface is not shared with any other modules and is defined within the OrderPreProcessingMediation module.

Table 2-9 OrderPreProcessingIF definition

Operation	Input	Output
createOrder	name: orderManagementInputSb type: OrderManagementInputSb	None (one way operation)

OrderManagementInputSb is a superset of the OrderManagementInput business object. It contains one additional field that includes the special order indicator.

Table 2-10 shows the properties for the OrderManagementIF interface. This is a shared interface and is placed in a library.

Table 2-10 OrderManagementF definition

Operation	Input	Output
createOrder	name: orderManagementInput type: OrderManagementInput	None (one way operation)

2.6.2 OrderManagement module

The OrderManagement module implements the OrderManagementIF interface, listed in Table 2-10. It needs references to the appropriate interfaces to retrieve and update customer and items and to create and update orders. The following common interfaces are defined by our high-level design and are placed in a library:

- Three interfaces to interact with the DBMS mediation:
  - CustomerServiceIF
  - ItemServiceIF
  - OrderServiceIF
- For the warehouse availability mediation, the WarehouseItemSplitIF

Table 2-11 shows the CustomerServiceIF interface definition.

Table 2-11 CustomerServiceIF definition

Operation	Input	Output
retrieveCustomer	name: customer type: Customer	name: customer type: Customer name: returnCode type: ReturnCode
updateCustomer	name: customer type: Customer	name: returnCode type: ReturnCode

Table 2-12 shows the ItemServiceIF interface definition.

Table 2-12 ItemServiceIF definition

Operation	Input	Output
retrieveItem	name: item type: Item	name: item type: Item name: returnCode type: ReturnCode
updateItem	name: item type: Item	name: returnCode type: ReturnCode

Table 2-13 shows OrderServiceIF interface definition.

Table 2-13 OrderServiceIF definition

Operation	Input	Output
createOrder	name: order type: Order name: item type: Item name: customer type: Customer	name: order type: Order name: returnCode type: ReturnCode
updateOrder	name: order type: Order name: item type: Item name: customer type: Customer	name: returnCode type: ReturnCode

Table 2-14 shows the Interface WarehouseItemSplitIF definition.

Table 2-14 WarehouseItemSplitIF definition

Operation	Input	Output
split	name: item type: Item name: order type: Order	name: item type: Item

In addition, the following interfaces are specific to the OrderManagement module and are defined within that module:

- Two interfaces to evaluate the financial and warehouse rules, which are specific to the OrderManagement module:
  - FinancialOfficeBRIF
  - WarehouseOfficeBRIF
- If the order is suspended, an e-mail is sent using the interface EmailServiceIF, which is also specific to the OrderManagement module.

2.6.3 DBMS mediation

The DBMS mediation also implements the CustomerServiceIF (Table 2-11 on page 37), ItemServiceIF (Table 2-12 on page 37), and OrderServiceIF (Table 2-13 on page 38) interfaces.

2.6.4 Financial rules

Within the OrderManagement module, the FinancialOfficeRG implements the rule that is defined in the interface FinancialOfficeBRIF.

Table 2-15 shows the FinancialOfficeBRIF interface definition.

Table 2-15 FinancialOfficeBRIF definition

Operation	**Input**	**Output**
authorize	name: customer type: Customer name: order type: Order name: item type: Item	name: financialRuleRC type: ReturnCode

This interface is defined later when we create the OrderManagement module (see 3.2.3, “Creating the interfaces” on page 72).

2.6.5 Warehouse rules

The WarehouseRG implements the rule that is defined in the interface WarehouseOfficeBRIF.

Table 2-16 shows the WarehouseOfficeBRIF interface definition.

Table 2-16 WarehouseOfficeBRIF definition

Operation	**Input**	**Output**
authorize	name: order type: Order name: totalStock type: integer	name: warehouseRuleRC type: ReturnCode

This interface is defined later when we create the OrderManagement module (see 3.2.3, "Creating the interfaces" on page 72).

2.6.6 E-mail module

We implement the e-mail adapter within EmailOutbound and expose it through the EmailServiceIF. Because a new module is used for this function, the interface is placed in the library for sharing.

Table 2-17 shows the properties for the EmailServiceIF interface.

Table 2-17 EmailServiceIF definition

Operation	Input	Output
sendEmail	name: emailOrderSuspend type: EmailOrder	None (one way operation)

2.6.7 Warehouse Availability mediation

Finally, the Warehouse Availability mediation implements the WarehouseItemSplitIF, as described in Table 2-14 on page 38. This interface is defined in the library (see 3.2.3, "Creating the interfaces" on page 72).

2.7 Division of work

Now that we have identified the business objects, component interactions, and interfaces, we recommended creating a CVS repository that holds the "shell" of your development.

You can create skeleton modules, business objects, and interfaces in CVS. Development assignments are made, and each team member develops a module and then checks it in periodically.

Another recommended practice is to assign one team member to the overall end-to-end scenario. This person takes the various parts as they are developed and makes sure that the parts work end-to-end.

Tip: This end-to-end test process should start from day one. Do not wait until all the parts are developed. Otherwise, you run the risk that you will be at the end of development and these pieces cannot be run from end-to-end because someone might have overlooked something.

2.8 Gather endpoint information

Looking at this scenario from a WebSphere Process Server perspective, the only endpoint we are directly accessing is the e-mail server. Therefore, we need to determine up front what we need to know to be able to use the e-mail adapter. If you are working with another adapter, Web service, you want to go through a similar exercise.

Although, human tasks are not an endpoint, per se, we do need to know from where the users, group, and role information is retrieved.

2.8.1 Outbound e-mail

Our e-mail server is running locally. We use the e-mail outbound pattern, so we need the information listed in Table 2-18 for the e-mail adapter.

Table 2-18 E-mail server information

Property	Value
Host name	localhost
Port number	25
Protocol	smtp
User name	admin
Password	admin

2.8.2 People directory

For the users and groups, we use WebSphere Integration Developer's pre-configured people directory. We create our own user and group information to represent the Financial Office and Warehouse Office. See Figure 2-10.

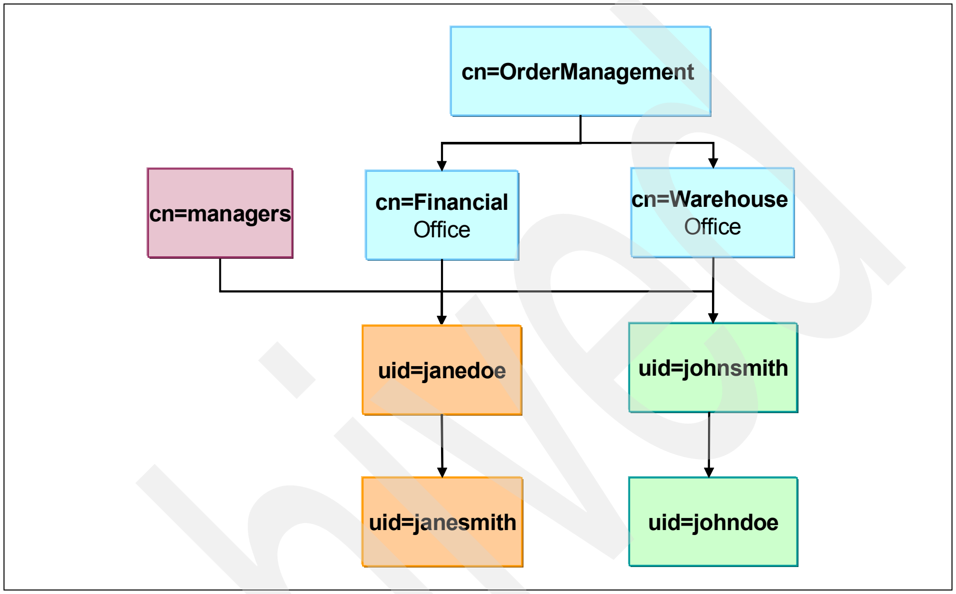


Figure 2-10 Users and groups for OrderManagement scenario

2.9 Obtain or agree upon the sample data

It is essential to obtain sample data so that you can use it to emulate end points while you develop. For existing systems or interfaces, obtain sample data. For new systems, agree upon sample data.

> **Additional material:** The test data shown in this section is available as XML files in the Sample Data directory of the additional materials. See Appendix B, "Additional material" on page 461.

2.9.1 orderManagementInput business object

While testing the Order Management System, we need to test the following scenarios:

- Approve
- Human Task
- Reject
- Warehouse split

The following sections show the values that start the scenario while testing only the WebSphere Process Server components. For the full end-to-end scenario, including WebSphere ESB, these values are pre-populated from the Order Pre-Processing Mediation.

Approve

Figure 2-11 shows the input data for the OrderManagementIF interface (orderManagementInput) to test the *approve* use case.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 15
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com

Figure 2-11 Sample input for the "approve" use case

Human

Figure 2-12 shows the input data for the OrderManagementIF interface (orderManagementInput) to test the *human task* use case.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com

Figure 2-12 Sample input for the "human task" use case

Reject

Figure 2-13 shows the input data for the OrderManagementIF interface (orderManagementInput) to test the *reject* use case.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10002
itemID	string	✔ IT_002
itemQty	int	✔ 10
submitterID	string	✔ user2
submitterEmail	string	✔ user2@kcg16hw.itso.ral.ibm.com

Figure 2-13 Sample input for the "reject" use case

Warehouse split

Figure 2-14 shows the input data for the OrderManagementIF interface (orderManagementInput) to test the *warehouse split* mediation.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 19
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com

Figure 2-14 Sample input to test the warehouse split

2.9.2 Customer retrieve samples

For the customer interactions, we show the customer retrieve sample input data followed by three customer retrieve responses for customers 10001, 10002, and 10003. The DBMS Mediation returns a value of `99` if there is an error.

Sample customer retrieve for custID 10001

Figure 2-15 shows the Customer Retrieve sample for custID 10001.

Name	Type	Value
customer	Customer	
custID	string	10001
custName	string	
address	string	
zipCode	string	
city	string	
state	string	
budget	int	
soldToDate	int	

Figure 2-15 Sample customer retrieve input data for custID 10001

Sample customer retrieve response for custID 10001

Figure 2-16 shows the Customer Retrieve response sample for custID 10001.

Name	Type	Value
customer	Customer	
custID	string	10001
custName	string	John Smith Corp.
address	string	2455 South Road
zipCode	string	12601
city	string	Poughkeepsie
state	string	NY
budget	int	50000
soldToDate	int	1000

Figure 2-16 Sample customer retrieve response for custID 10001

Sample customer retrieve response for custID 10002

Figure 2-17 shows the customer retrieve response sample for custID 10002.

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	✔ 15000
soldToDate	int	✔ 50000

Figure 2-17 Sample customer retrieve response for custID 10002

Sample customer retrieve response for custID 10003

Figure 2-18 shows the customer retrieve sample for custID 10003.

Name	Type	Value
customer	Customer	✔
custID	string	✔ 10003
custName	string	✔ YXZ itso Corp.
address	string	✔ 4205 S. Miami Blvd
zipCode	string	✔ 27709
city	string	✔ RTP
state	string	✔ NC
budget	int	✔ 5000
soldToDate	int	✔ 10000

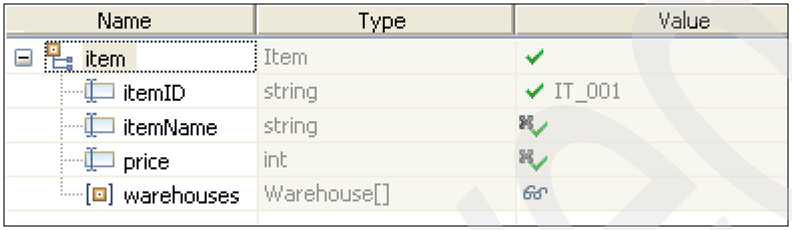
Figure 2-18 Sample customer retrieve response for custID 10003

2.9.3 Item interactions samples

For the item interactions, we show an item retrieve example followed by three item retrieve responses for items IT_001, IT_002, and IT_003. The DBMS Mediation returns a value of 99 if there is an error.

Sample item retrieve request for itemID IT_001

Figure 2-19 shows the item retrieve request for itemID IT_001.



Name	Type	Value
item	Item	
itemID	string	IT_001
itemName	string	
price	int	
warehouses	Warehouse[]	

Figure 2-19 Sample item retrieve request for itemID IT_001

Sample item retrieve response for itemID IT_001

Figure 2-20 shows the item retrieve response for itemID IT_001.

Name	Type	Value
item	Item	✔
itemID	string	✔ IT_001
itemName	string	✔ Item001
price	int	✔ 15
warehouses	Warehouse[]	6o
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_A
stock	int	✔ 10
indelivery	int	✔ 50
itemQtyPartial	int	✔ 100
warehouses[1]	Warehouse	✔
whsID	string	✔ WHS_B
stock	int	✔ 50
indelivery	int	✔ 150
itemQtyPartial	int	✔ 100
warehouses[2]	Warehouse	✔
whsID	string	✔ WHS_C
stock	int	✔ 30
indelivery	int	✔ 50
itemQtyPartial	int	✔ 100

Figure 2-20 Sample item retrieve response for itemID IT_001

Sample item retrieve response for itemID IT_002

Figure 2-21 shows the item retrieve response for itemID IT_002.

Name	Type	Value
item	Item	
itemID	string	IT_002
itemName	string	Item002
price	int	10
warehouses	Warehouse[]	
warehouses[0]	Warehouse	
whsID	string	WHS_A
stock	int	-100
indelivery	int	45
itemQtyPartial	int	4
warehouses[0]	Warehouse	
whsID	string	WHS_B
stock	int	-100
indelivery	int	50
itemQtyPartial	int	3
warehouses[0]	Warehouse	
whsID	string	WHS_C
stock	int	-100
indelivery	int	5
itemQtyPartial	int	3

Figure 2-21 Sample item retrieve response for itemID IT_002

Sample item retrieve response for itemID IT_003

Figure 2-22 shows the item retrieve response for itemID IT_003.

Name	Type	Value
item	Item	✔
itemID	string	✔ IT_003
itemName	string	✔ Item003
price	int	✔ 5
warehouses	Warehouse[]	✔
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_A
stock	int	✔ 0
indelivery	int	✔ 10
itemQtyPartial	int	✔ 8
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_B
stock	int	✔ 0
indelivery	int	✔ 45
itemQtyPartial	int	✔ 6
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_C
stock	int	✔ 0
indelivery	int	✔ 5
itemQtyPartial	int	✔ 6

Figure 2-22 Sample item retrieve response for itemID IT_003

2.9.4 Order test case samples

For the order test case, we do not need to define the test data that we get from the DBMS Mediation because we pre-populate all order values in the main process prior to sending it to the database. However, the DBMS Mediation needs to know what to expect from the OrderManagement system for creates and updates. The main information that we need to know is that the DBMS Mediation returns a value of 99 if there is an error.

Sample order create request

Figure 2-23 shows a sample order create request.

Name	Type	Value
order	Order	✔
ordID	string	✔ _PI:90030118.323c1706.4f55d5f6.8b44008d
amount	int	✔ 225
submitterID	string	✔ user1
state	string	✔ CREATED
creationDate	dateTime	✔ 2008-02-19T10:07:37.890 -0500
completionDate	dateTime	✔ 2008-02-19T10:07:37.890 -0500
itemQty	int	✔ 15

Figure 2-23 Sample order create request

Sample order update request

Figure 2-24 shows a sample order update request.

Name	Type	Value
order	Order	✔
ordID	string	✔ _PI:90030118.323d7e85.4f55d5f6.8b4400d3
amount	int	✔ 225
submitterID	string	✔ user1
state	string	✔ CONFIRMED
creationDate	dateTime	✔ 2008-02-19T15:08:57.546 -0500
completionDate	dateTime	✔ 2008-02-19T10:08:59.031 -0500
itemQty	int	✔ 15

Figure 2-24 Sample order update request

2.9.5 orderManagementInputSb

The next set of sample data is needed for end-to-end testing when we bring together the WebSphere Process Server components and WebSphere ESB components. The testing starts with the interface to the Order Pre-Processing Mediation. The interface, OrderPreProcessingIF, requires input of data type OrderManagementInputSb.

In addition to the four regular order scenarios (Approve, Reject, Human, and Item split), we need a case for Special Orders.

Regular Order: Approve

Figure 2-25 shows the input data for the OrderPreProcessingIF interface to test the *approve* use case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 15
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 2-25 Sample input for the "Regular Order: Approve" use case

Regular Order: Warehouse split

Figure 2-26 shows the input data for the OrderPreProcessingIF interface to test the *warehouse split* use case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 19
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 2-26 Sample input for the "Regular Order: Warehouse split"

Regular Order: Reject

Figure 2-27 shows the input data for the OrderPreProcessingIF interface to test the *reject* use case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10002
itemID	string	✔ IT_002
itemQty	int	✔ 10
submitterID	string	✔ user2
submitterEmail	string	✔ user2@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 2-27 Sample input for the "Regular Order: Reject" use case

Regular Order: Human task

Figure 2-28 shows the input data for the OrderPreProcessingIF interface to test the *human task* use case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 2-28 Sample input for the "Regular Order: Human task" use case

Special Order

Figure 2-29 shows the input data for the OrderPreProcessingIF interface to test the *Special Order* use case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ true

Figure 2-29 Sample input for the "Special Order" use case

Other modules

We do not need to generate sample data for the rules or for e-mail because these modules are developed within the business integration example. Thus, there is no need to emulate those interactions.

2.10 Prepare the library

Now that we have the basic design in place, we can build the objects and interfaces that are used to develop the modules and mediations.

2.10.1 Build OrderManagementLib

In the Order Management System scenario, a library is used as a common repository for the business objects and interfaces.

To begin the development process we created this library using the following steps:

1. Open a WebSphere Integration Developer with a new workspace and switched to the Business Integration perspective.
2. Create a new library:
   a. Right-click the Business Integration view and select **New** → **Library** from the context menu.
   b. Enter `OrderManagementLib` as the name for the library and ensure that **Use default location** is selected. Click **Finish**.

   The new library is created, and you can view the structure of the library in the Business Integration view.

The business objects and interfaces that are stored in this library are listed in 1.4.2, “Interfaces and business objects definition” on page 17. We create these interfaces and business objects next. We start with the business objects because they are used to create the interfaces.

2.10.2 Create the common business objects

The common business objects created in the library include:

- OrderManagementInput
- Customer
- Warehouse
- Order
- ReturnCode
- Item
- EmailOrder

Additional business objects that are required only by a single module are created later during the module development.

Create the business object using the following steps:

1. In the OrderManagementLib, right-click **Data Types** in the Business Integration view and select **New** → **Business Object**.
2. Name the object and click **Finish**. The business object editor opens.
3. Add fields to the object and assign a type to each field.
4. Save and close the object.

OrderManagementInput

OrderManagementInput has the properties listed in Table 2-5 on page 32. The resulting object looks similar to Figure 2-30.

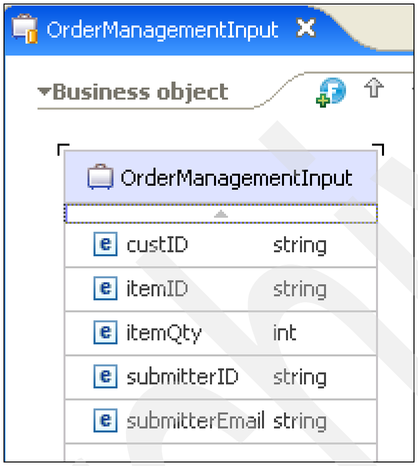


Figure 2-30 OrderManagementInput business object

Customer

The Customer business object has the properties listed in Table 2-1 on page 30. The resulting object looks similar to Figure 2-31.

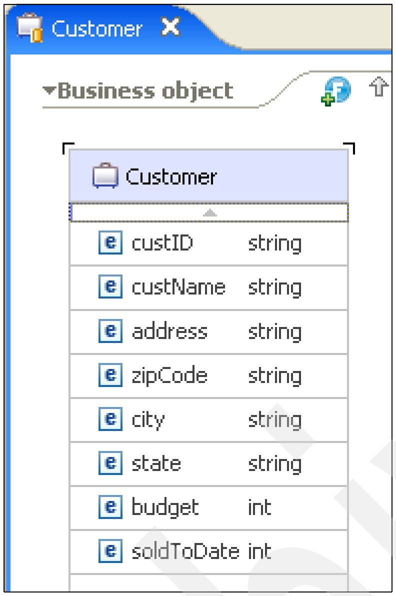


Figure 2-31 Customer business object

Warehouse

The Warehouse business object has the properties listed in Table 2-3 on page 31. The resulting object looks similar to Figure 2-32.

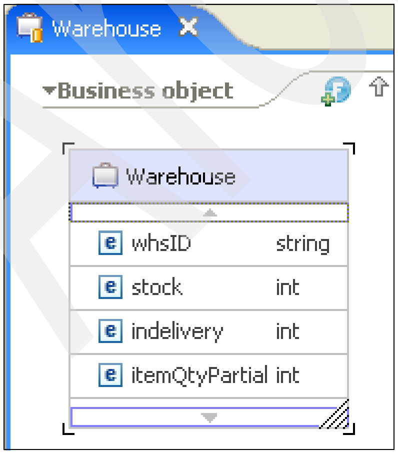


Figure 2-32 Warehouse business object

Order

The Order business object has the properties listed in Table 2-2 on page 31. The resulting object looks similar to Figure 2-33.

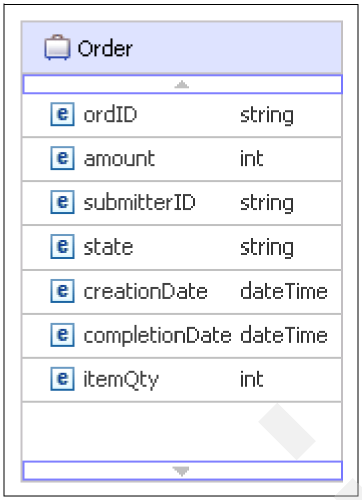


Figure 2-33 Order business object

ReturnCode

The ReturnCode business object has the properties listed in Table 2-6 on page 33. The resulting object looks similar to Figure 2-34.

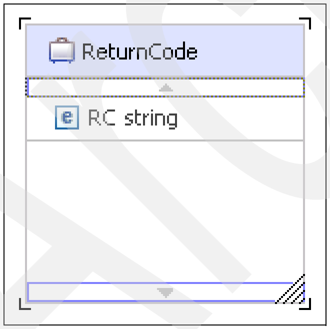


Figure 2-34 ReturnCode business object just hosts a string

Item

The Item object contains the properties listed in Table 2-4 on page 32. It is similar to the other business objects, with one exception. The Item object contains an array. To put an array in your Item object, follow these steps:

1. First, create the Item business object and add the attributes listed in Table 2-19.

Table 2-19 Item business object structure

Attribute name	Attribute type
itemID	string
itemName	string
price	int

2. Now add the **warehouses** attribute.
3. Change the attribute type to the Warehouse object.
4. Select the warehouses attribute and go to the Description tab of the Properties view. Select **Array** as shown in Figure 2-35.

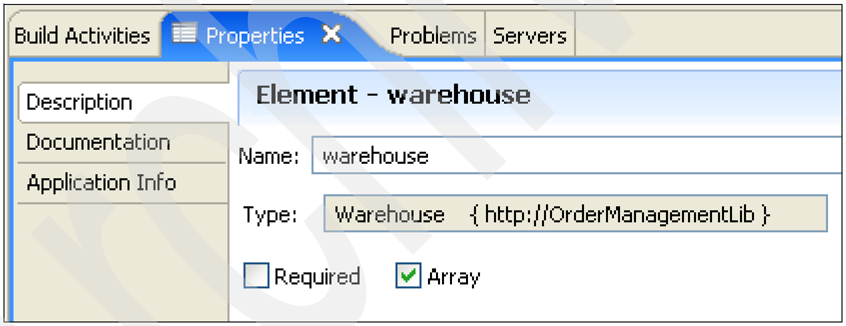


Figure 2-35 How to put an array data type in your Item business object

Figure 2-36 shows the results.

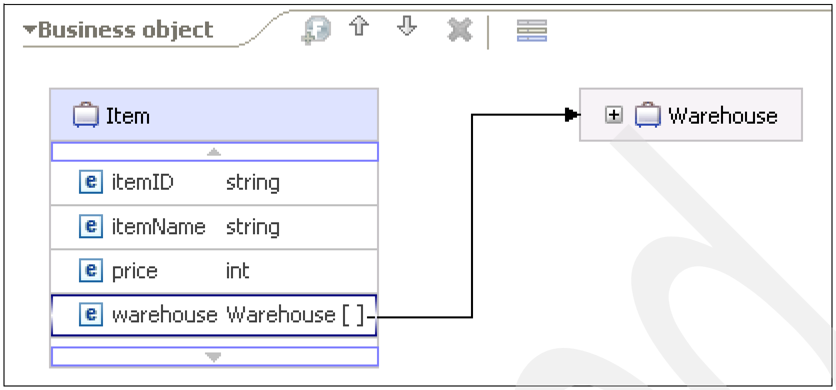


Figure 2-36 Item business object holding a warehouse array

EmailOrder

The EmailOrder business object has the properties listed in Table 2-7 on page 33. The resulting object looks similar to Figure 2-37.

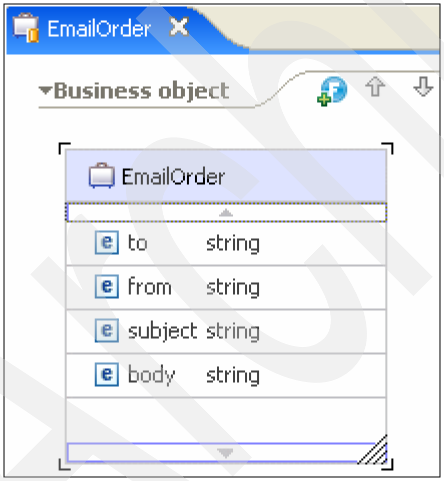


Figure 2-37 EmailOrder business object

2.10.3 Create interfaces for the library

The interfaces that we need to create in the library include:

- OrderManagementIF
- CustomerServiceIF
- OrderServiceIF
- ItemServiceIF
- FinancialOfficeBRIF
- WarehouseOfficeBRIF
- EmailServiceIF

If any additional interfaces that are required only by a single module are created as the module is developed.

Create the interface using these steps:

1. In the OrderManagementLib, right-click **Interfaces** in the Business Integration view and select **New** → **Interface**.
2. Name the interface and click **Finish**. The interface editor opens.
3. Add the operation to the interface using the icons.
   - Add a one-way operation:
   - Add a request-response operation:
4. Rename the operation.
5. Name the input and output fields and assign types to them. Types can be assigned by clicking the default type and selecting from a drop-down list.

OrderManagementIF

The OrderManagementIF interface has the properties shown in Table 2-10 on page 36. It has a one-way operation called createOrder as shown in Figure 2-38.

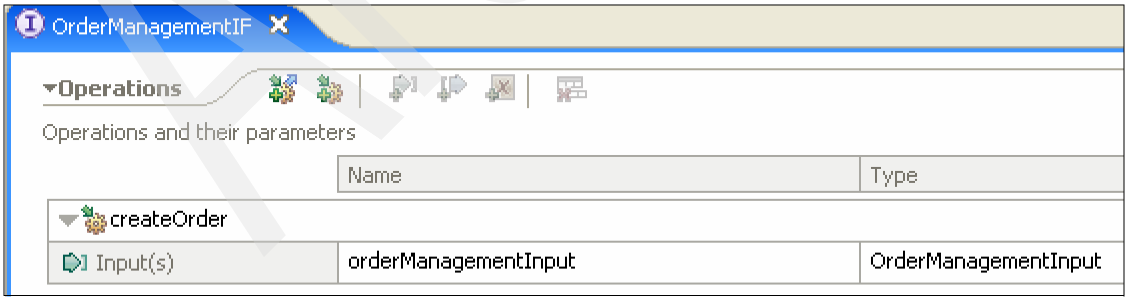


Figure 2-38 OrderManagementIF interface

CustomerServiceIF

The CustomerServiceIF interface has the properties shown in Table 2-11 on page 37. It has two operations: retrieveCustomer and updateCustomer. Both are request-response operations as shown in Figure 2-39.

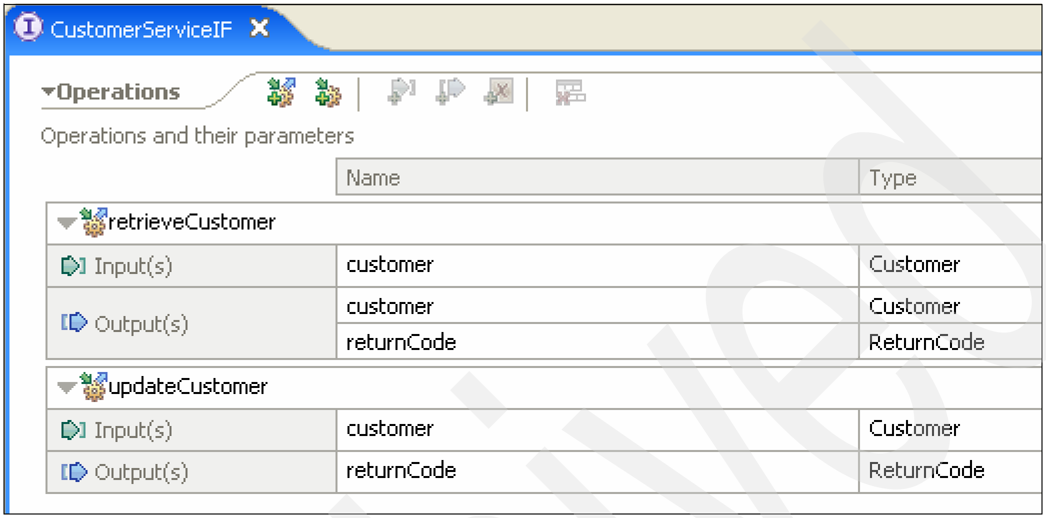


Figure 2-39 Customer Service interface

ItemServiceIF

The ItemServiceIF interface has the properties shown in Table 2-12 on page 37. It has two operations: retrieveItem and updateItem. Both are request-response operations as shown in Figure 2-40.

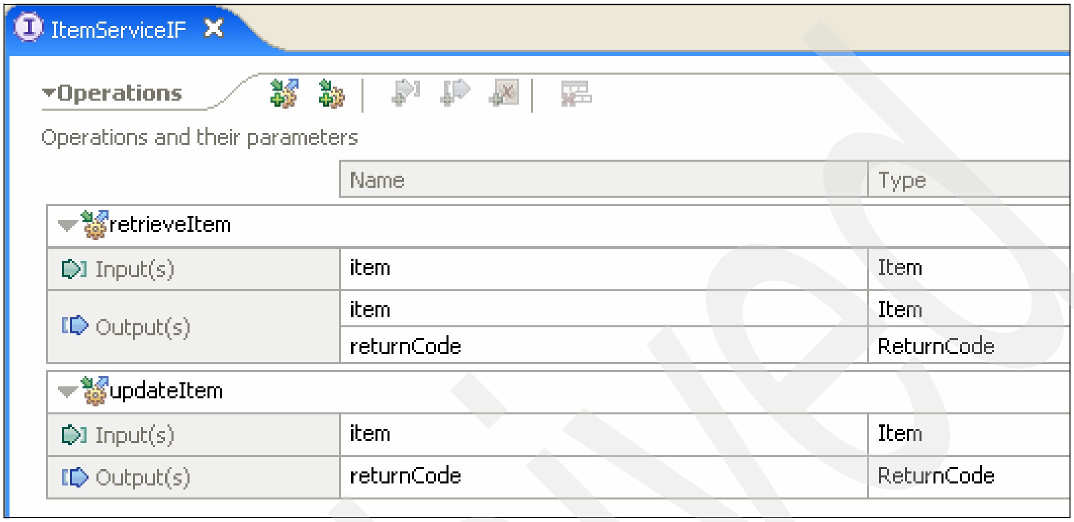


Figure 2-40 Item Service interface

OrderServiceIF

The OrderServiceIF interface has the properties shown in Table 2-13 on page 38. It has two operations: createOrder and updateOrder. Both are request-response operations as shown in Figure 2-41.

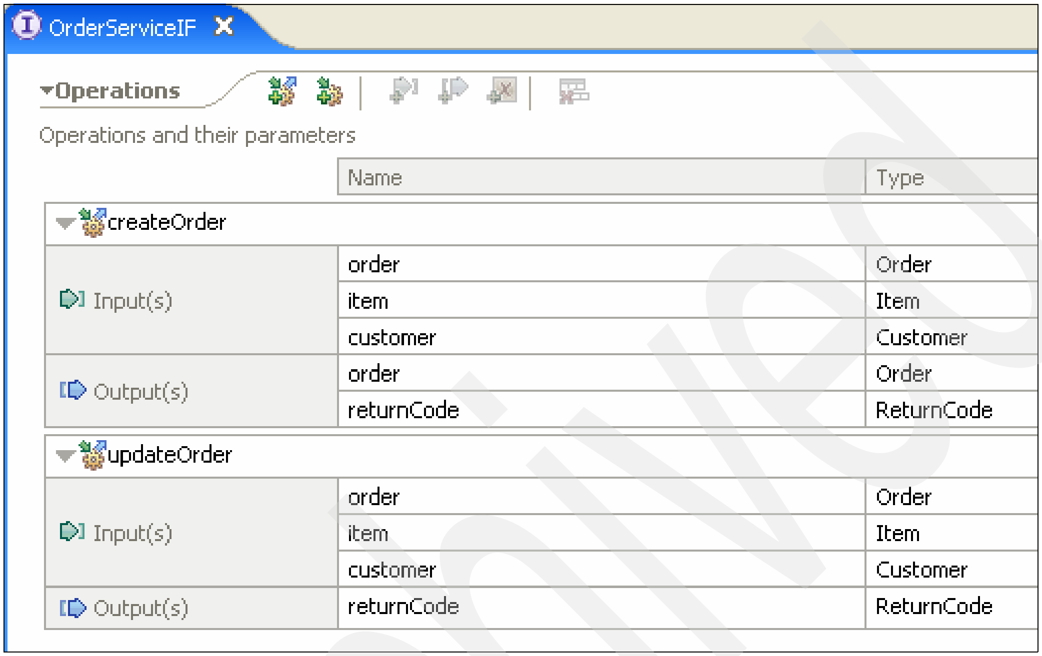


Figure 2-41 Order Service interface

WarehouseItemSplitIF

The WarehouseItemSplitIF interface has the properties shown in Table 2-14 on page 38. It has one request-response operation called split as shown in Figure 2-42.

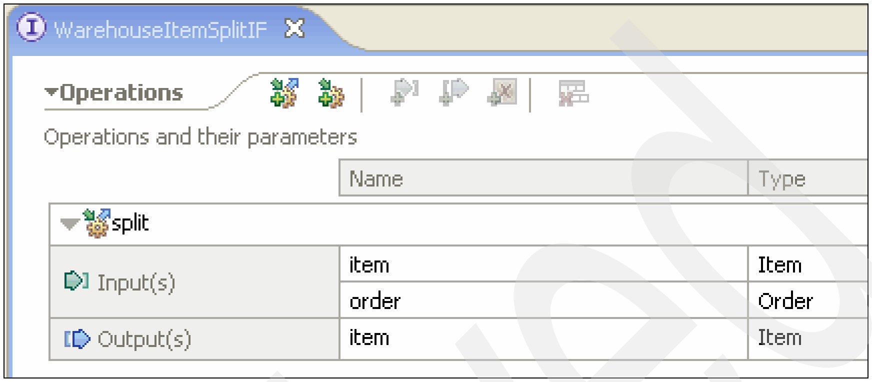


Figure 2-42 Warehouse split interface

EmailServiceIF

The EmailServiceIF interface has the properties shown in Table 2-17 on page 40 It has one request-response operation called split as shown in Figure 2-43.

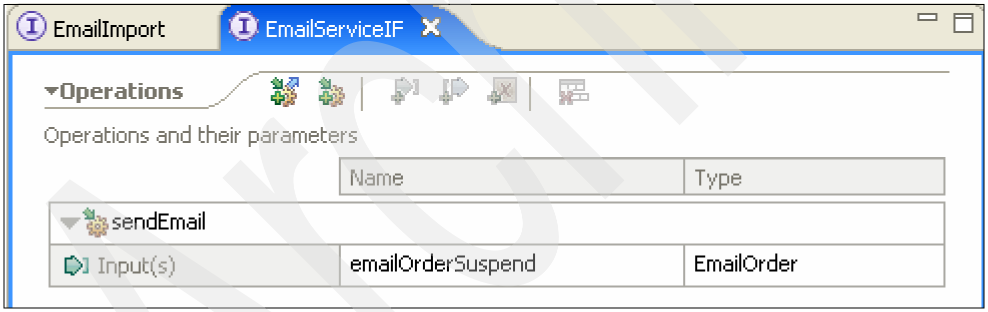


Figure 2-43 EmailService interface

2.11 Code repository check in

We use CVS as our team collaboration repository. Because the development of this scenario is spread among various development teams, the library is committed to the repository at this point.

Each developer can check in the common library from the repository. Figure 2-44 shows the workspace after library check in. This library is the common starting point for the two teams.

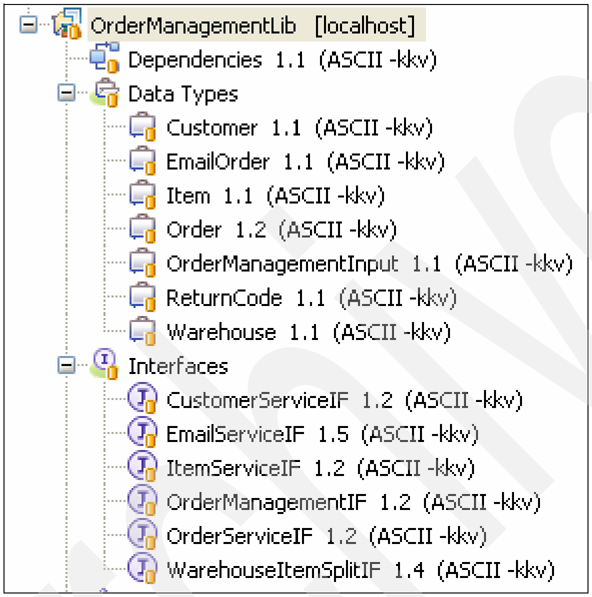


Figure 2-44 CVS Repository view after OrderManagementLib check in

3

Order Management Process business integration module

In this chapter, we implement the OrderManagement module incrementally. First, we construct a BPEL process that initially stubs out certain activities. Then, in subsequent steps, we implement and test each remaining activity.

This chapter contains the following topics:

- Business process development steps
- Preparing to develop
- Developing and testing the basic business process
- Adding business rules to the process
- Implementing and testing human tasks
- Implementing and testing the final verification steps
- Adding the ForEach activity and test
- Implementing and testing the e-mail
- Testing the module components end-to-end

3.1 Business process development steps

If we revisit the mapping of activities to the Order Management System flow (shown in Figure 3-1), we see the steps that are required to develop the business process.

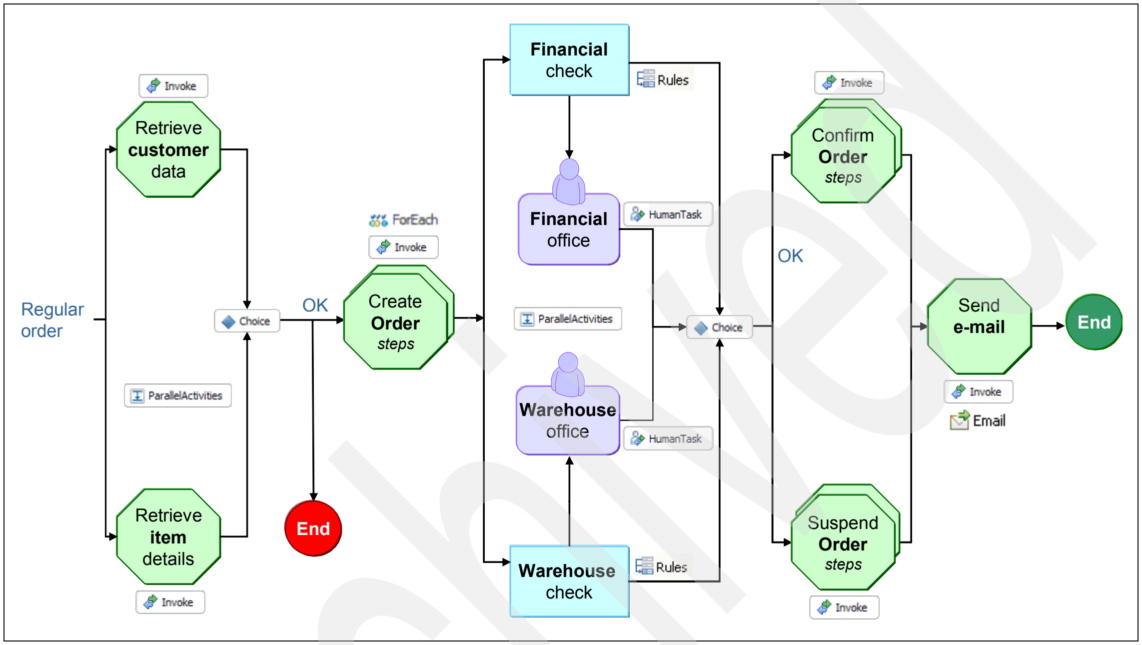


Figure 3-1 Mapping of activities to Order Management System flow

To develop the business process:

1. Prepare to develop by verifying the common library artifacts and then by creating any additional artifacts that apply only to the business integration module. See 3.2, "Preparing to develop" on page 69.
2. Create a basic business process where the human tasks, business rules, and final verification are stubbed out. See 3.3, "Developing and testing the basic business process" on page 74. We demonstrate how to unit test this process in the test environment in 3.3.2, "Testing the basic business process" on page 102.
3. Implement and test the business rules of the process as described in 3.4, "Adding business rules to the process" on page 112.

4. Implement the human tasks of the process using the pre-configured people directory in the integrated test environment as described in 3.5, “Implementing and testing human tasks” on page 156. In our example, we also create our own users and groups and test the rules with these.
5. Implement and test the final verification steps as described in 3.6, “Implementing and testing the final verification steps” on page 189.
6. Develop and test the ForEach construct after the Warehouse Item Split activity. See 3.7, “Adding the ForEach activity and test” on page 200.
7. Add the e-mail adapter to the process and test it as described in 3.8, “Implementing and testing the e-mail” on page 222.
8. Test the process end-to-end from a business integration point of view. See 3.9, “Testing the module components end-to-end” on page 246.

> **Tip:** First develop a basic flow that implements the logic of the process and then stub out external components. As external components are developed, add them to the overall end-to-end testing. Using this process, you can build an application iteratively that runs end-to-end beginning from day one.

3.2 Preparing to develop

In this section, we begin in a workspace that includes the shared library, OrderManagementLib. We describe how to create a new OrderManagement module and add a dependency to the shared library. Then, we explain how to create additional interfaces that are not in the shared library and add these interfaces to CVS.

3.2.1 Verifying the shared library

In the Chapter 1, “Scenario high-level design” on page 1, we built the common library and checked in the library to CVS. If this library is not in the workspace, you need to check out the library from the CVS repository.

Verify common artifacts in the shared library as shown in Figure 3-2.

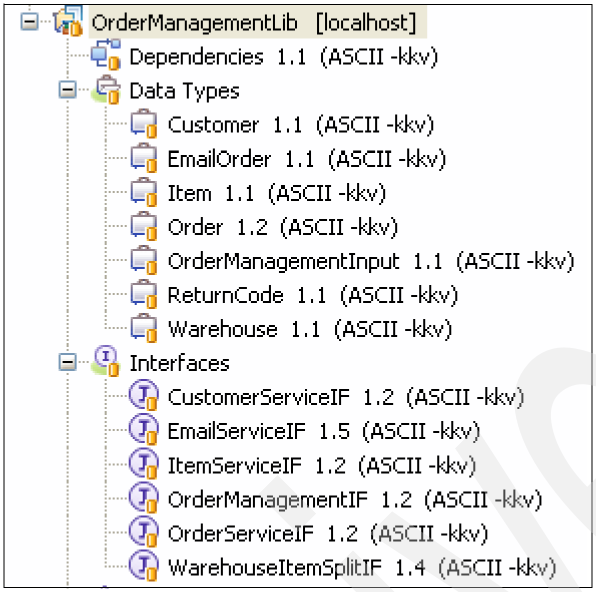


Figure 3-2 OrderManagementLib contents

All the business objects that we need for this scenario (Customer, EmailOrder, Item, Order, OrderManagementInput, ReturnCode, and Warehouse), were developed in the shared library.

For our example, we use all interfaces in the shared library and, in addition, we build a few more interfaces that are used only within the business integration example.

3.2.2 Creating the OrderManagement module and adding library dependency

The FinancialOfficeBRIF and WarehouseOfficeBRIF are used only within the OrderManagement module, so they are stored in the module versus the library. However, these two interfaces depend on business objects that are stored in the library. Therefore, you need to add a dependency for the library to the OrderManagement module. Follow these steps:

1. Select **File** → **New** → **Module** to create a new business integration module. Name the module `OrderManagement` and click **Next**. See Figure 3-3.

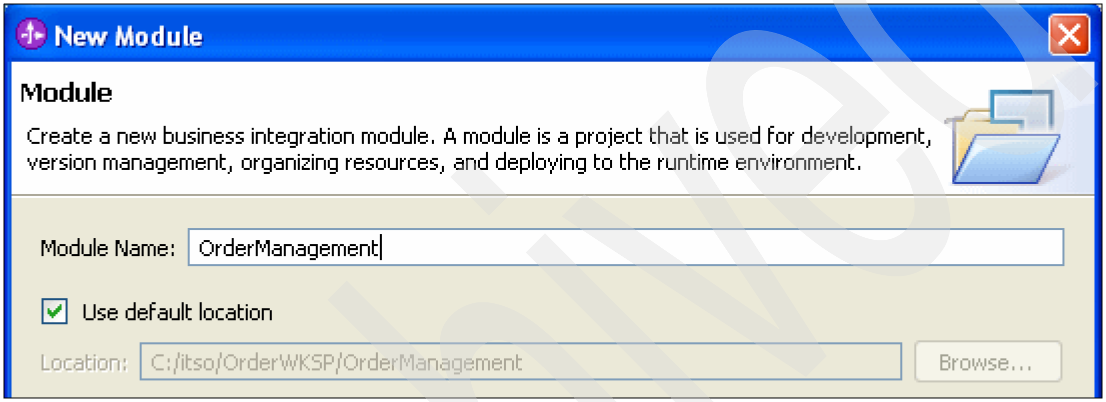


Figure 3-3 Creating OrderManagement module

2. Select **OrderManagementLib** as a required library and then click **Finish**.

3.2.3 Creating the interfaces

Next, you create the four interfaces that are stored in the OrderManagement module:

- Two interfaces for the business rules
- Two interfaces for the human tasks

Follow these steps:

1. Create the *FinancialOfficeBRIF* interface with the operations and parameters shown in Figure 3-4.

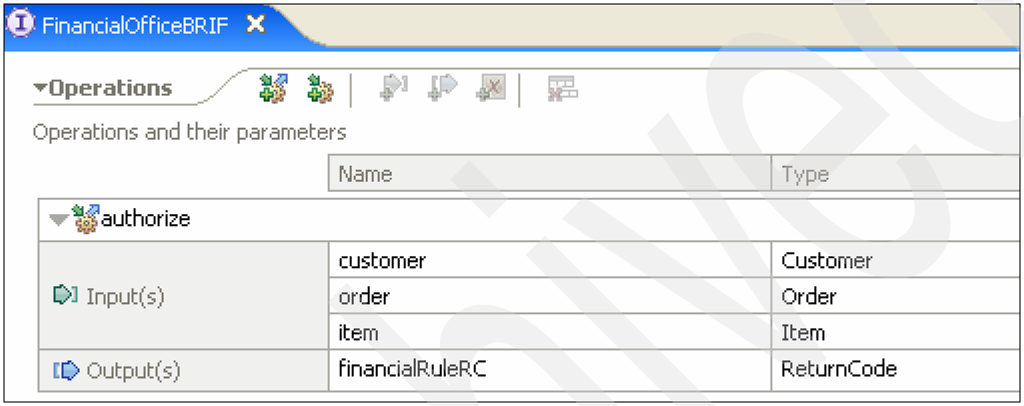


Figure 3-4 FinancialOfficeBRIF details

2. Create the *WarehouseOfficeBRIF* interface with the operations and parameters shown in Figure 3-5.

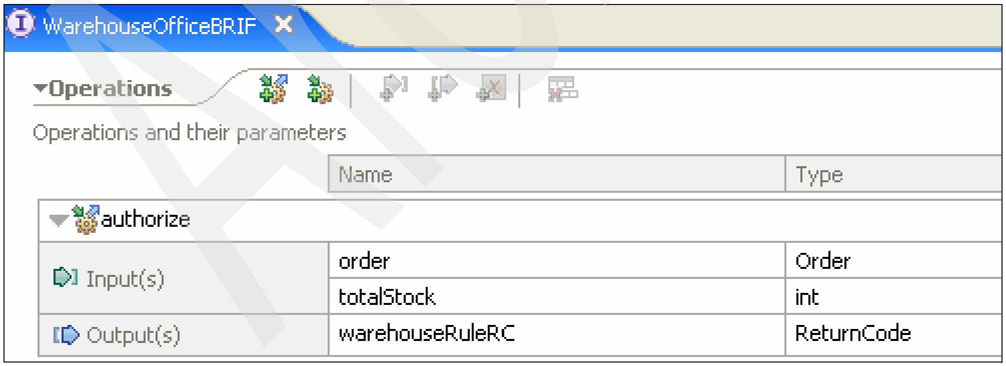


Figure 3-5 WarehouseOfficeBRIF details

3. Create the *FinancialAuthorizationIF* interface with the operations and parameters shown in Figure 3-6.

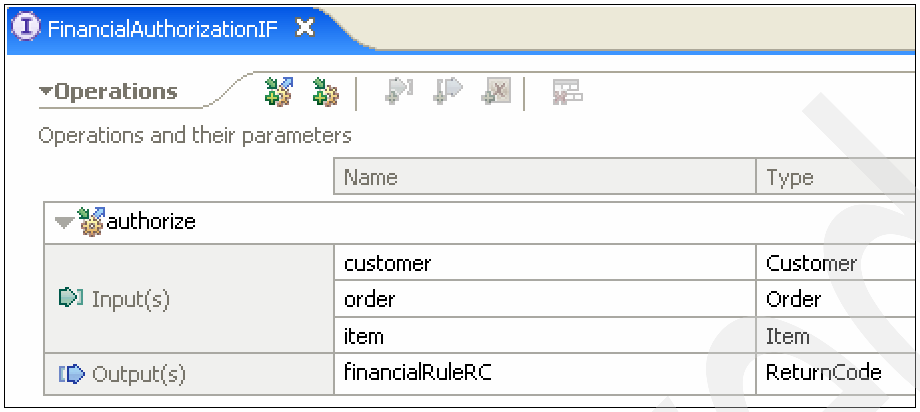


Figure 3-6 FinancialAuthorizationIF details

4. Create the *WarehouseAuthorizationIF* interface with the operations and parameters shown in Figure 3-7.

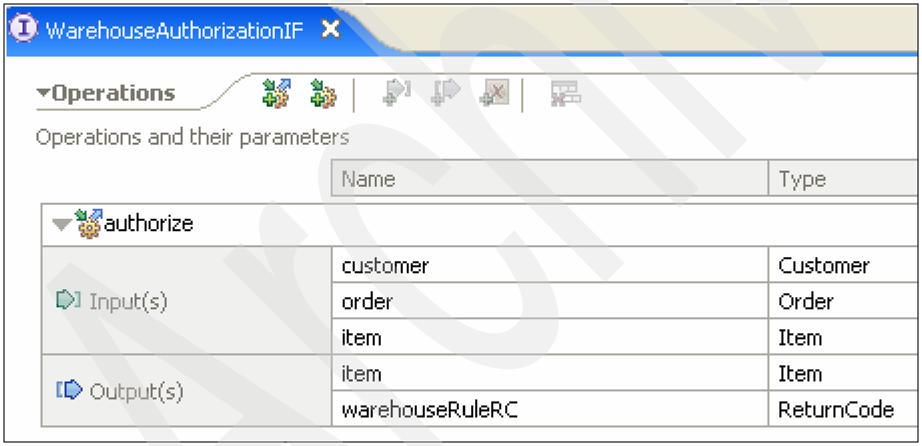


Figure 3-7 WarehouseAuthorizationIF details

3.2.4 Adding the module to CVS

If you have a CVS server, it is recommended that you store the module into CVS.

3.3 Developing and testing the basic business process

Next, you create the basic business process, *OrderManagementBP*. In this section, we describe how to stub out the human tasks, business rules, and final verification. Then, we demonstrate how to unit test this process in the test environment.

3.3.1 Developing the basic business process

To create the business process:

1. Expand the **OrderManagement** module and select **Business Logic** as shown in Figure 3-8.

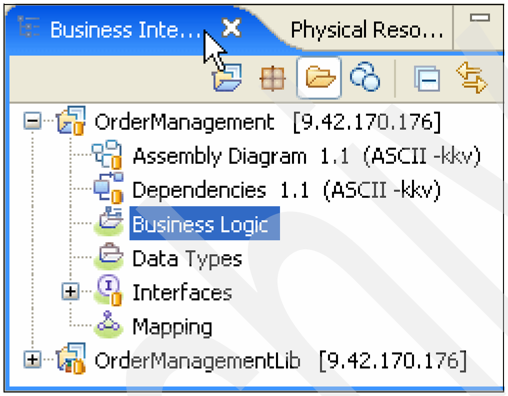


Figure 3-8 OrderManagement module in the Business Integration View

2. Right-click **Business Logic** → **New** → **Business Process**. In the New Business Process dialog box:
   a. Select **New default Business Process**.
   b. Select the **OrderManagement** module.
   c. Specify the name as `OrderManagementBP`.
   d. Click **Next**.

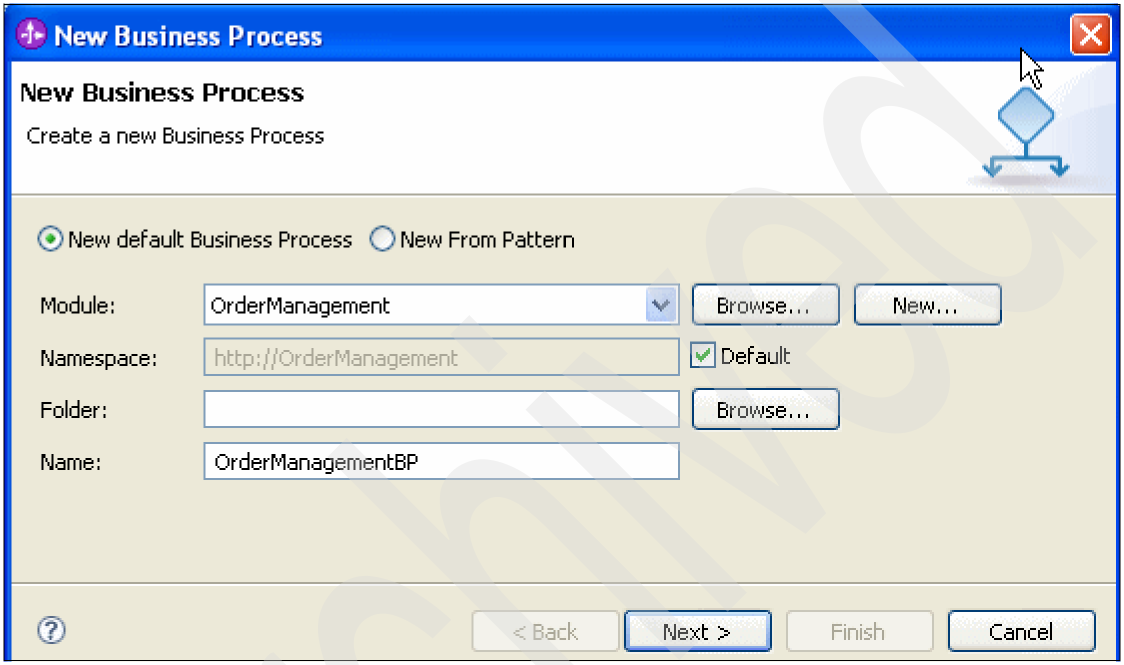


Figure 3-9 New Business Process wizard

3. Select **Long-running process** for the business process type (as shown in Figure 3-10, and then click **Next**.

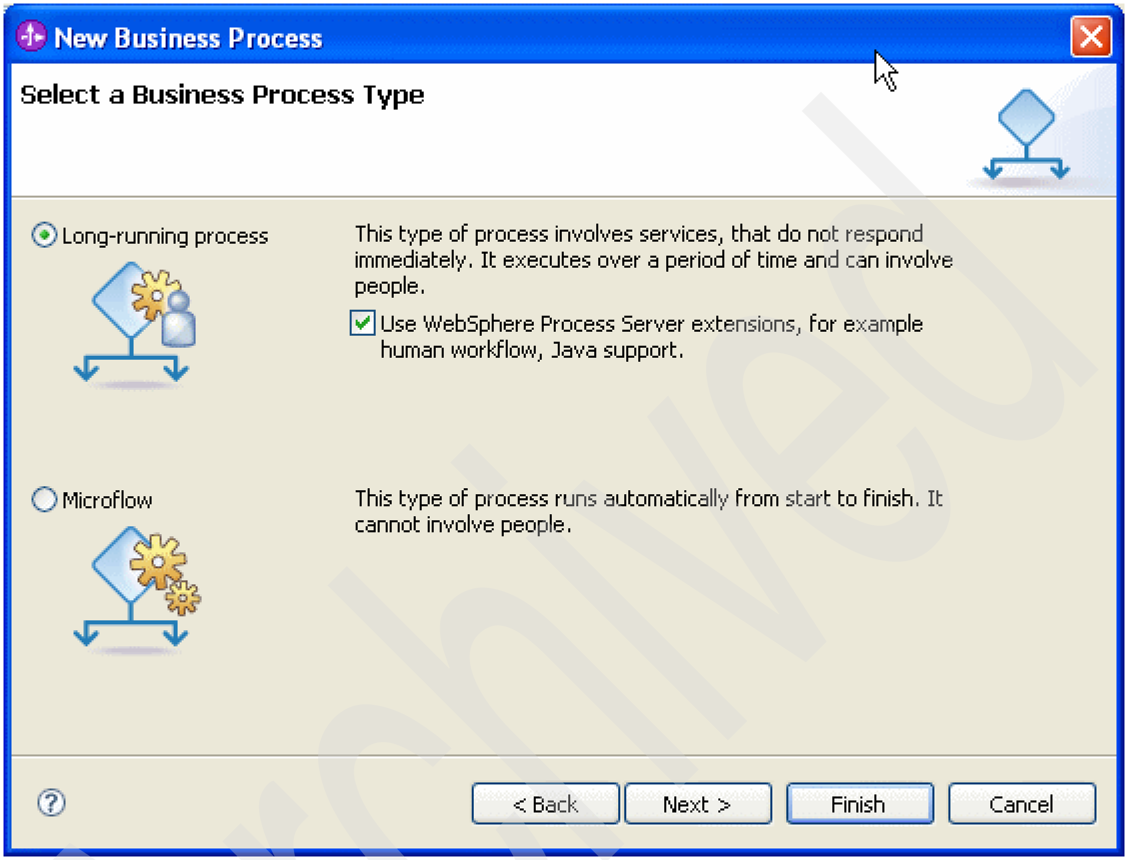


Figure 3-10 Selecting the business process type in the New Business Process wizard

4. In the next panel (shown in Figure 3-11):
   a. Choose **Select an existing interface**.
   b. Use the Browse button to select **OrderManagementIF** and then click **OK**.
   c. Select the **createOrder** operation.
   d. Click **Finish**.

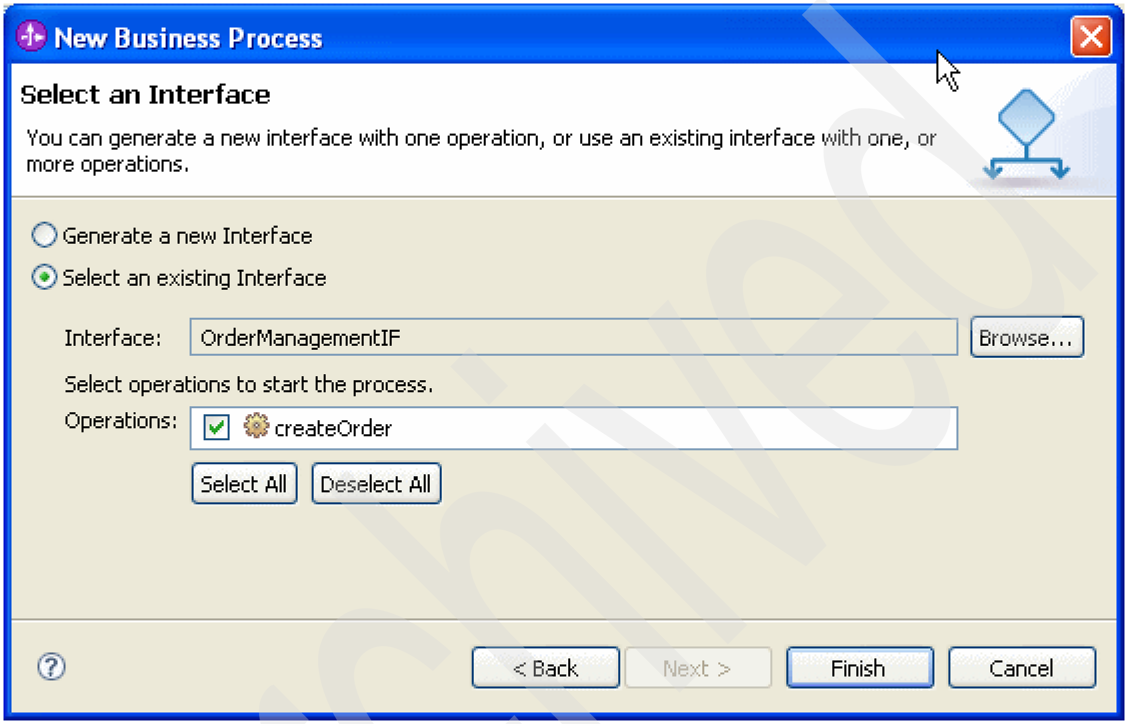


Figure 3-11 Selecting the interface operation the business process implements

The OrderManagementBP business process opens in the editor, as shown in Figure 3-12.

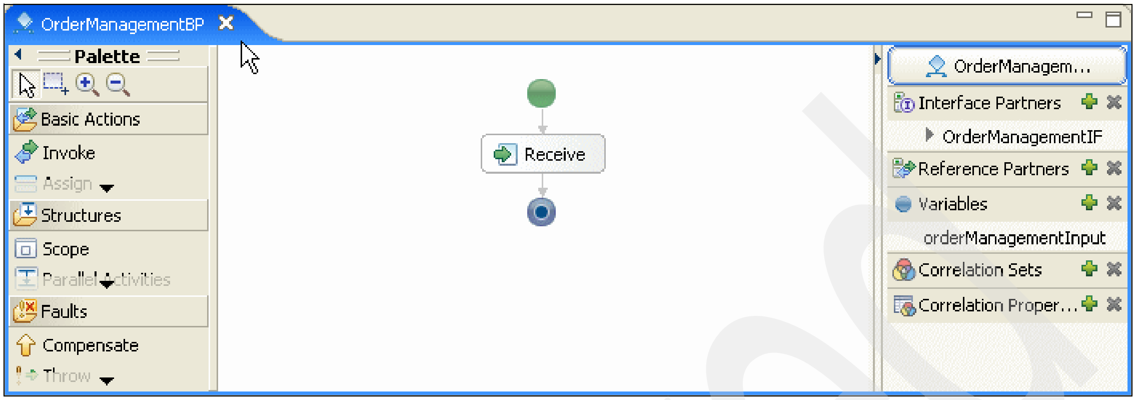


Figure 3-12   Default OrderManagementBP business process

Adding the partner interfaces

Next, you add the interfaces to the reference partners:

1. In the Business Integration view and expand the interfaces in OrderManagementLib. Click **CustomerServiceIF** and drag it over the OrderManagementBP's **Reference Partners** list.
2. Repeat this process to add the **ItemServiceIF** and **OrderServiceIF** to the OrderManagementBP. See Figure 3-13.

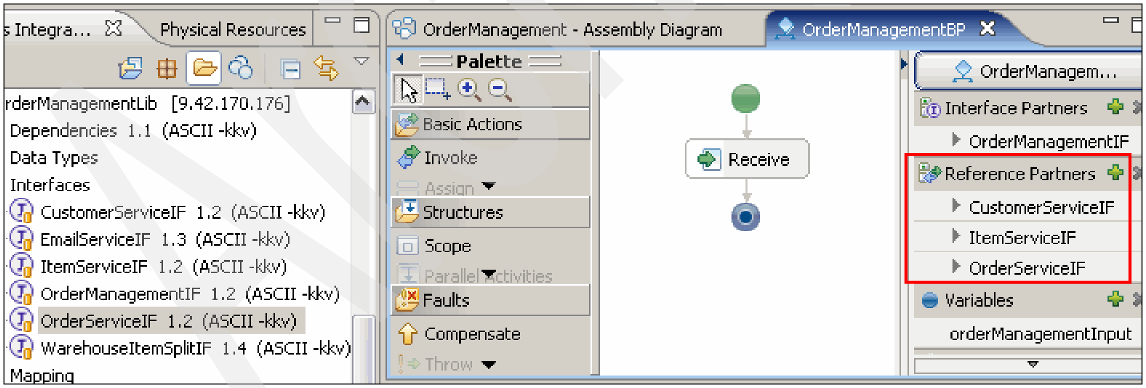


Figure 3-13   OrderManagementBP Reference Partners after adding CustomerServiceIF

3. Save the process using Ctrl+S.
4. Click the **Problems** view and verify that there are no errors, warnings or informational messages as shown in Figure 3-14.

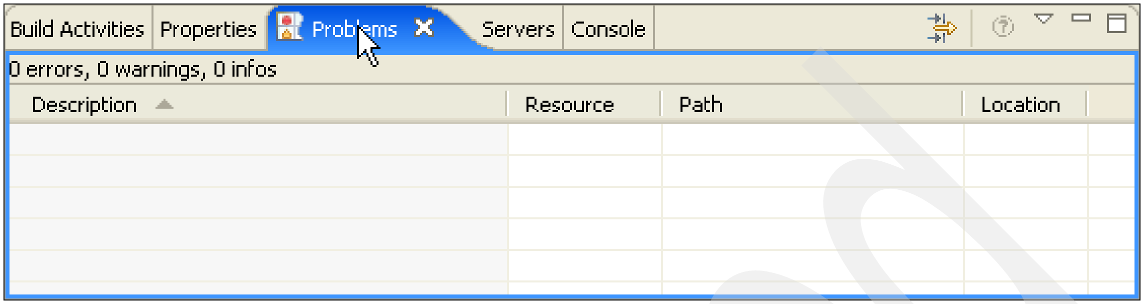


Figure 3-14 Problems view

> **Tip:** We added reference partners with one or more operations that are not implemented yet in the BPEL. However, we do not have any errors—a clear distinction between a BPEL's interface and its reference partner's interface.
>
> The interface of a BPEL is the contract that you agree to when developing this process. You *must* implement all operations of an interface of your BPEL. However, you can pick and choose the operations that you use of your reference partners. You can use all or none of their operations. If the interface of your BPEL has more than one operation and you do not implement all operations, then you see errors in the BPEL and in the Problems View. Similarly, the implementation of your reference partners must implement all operations of the interfaces that they are exposing (their contract).

Adding activities for logging

Now, add the activities for the process using the following steps:

1. Drag a Snippet activity from the Palette on to the business process and then rename it `Log - Begin`. Repeat this step to add a snippet named `Log - End`. See Figure 3-15.

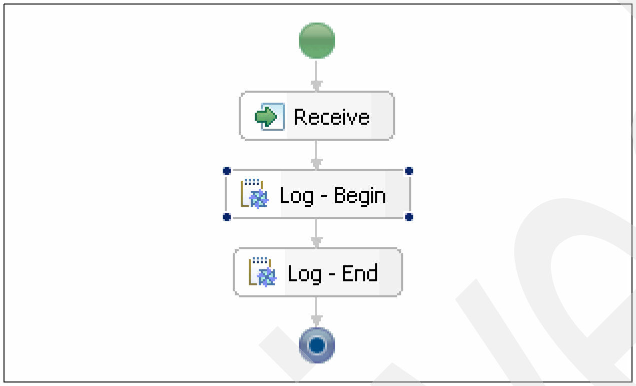


Figure 3-15 OrderManagementBP after adding Snippet Log - End

2. Click **Log - Begin** and then go to the Properties view (Figure 3-16).

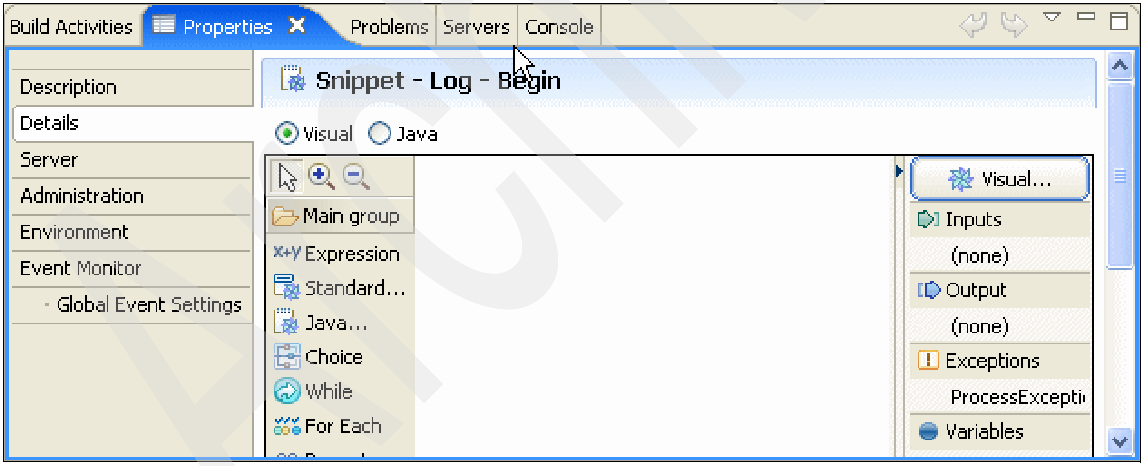


Figure 3-16 Properties view of Log - Begin

3. In the palette, click **Standard**. Expand **utility** → **print to log** → click **OK**. Then click the canvas to add the print to log utility to the snippet.
4. In the palette, click **Expression®** and then drag it on to the canvas. Enter `OrderManagement process - Begin` in the expression and then wire the two snippets as shown in Figure 3-17.

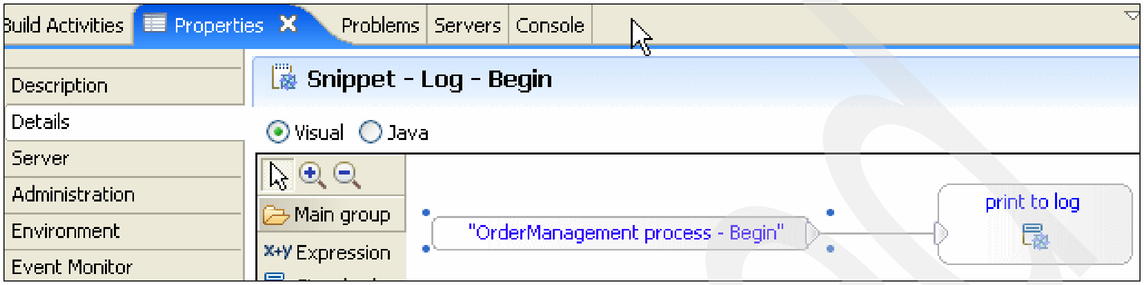


Figure 3-17 Log - Begin snippet

5. Repeat steps 2 through 4 to set the Log - End snippet as shown in Figure 3-18.

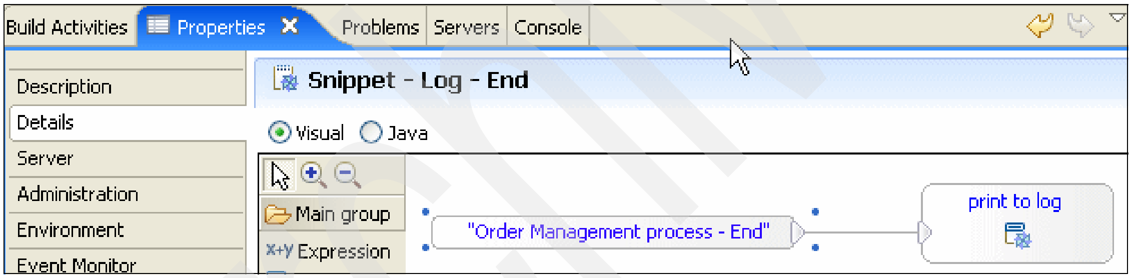


Figure 3-18 Log - End snippet

Creating the variables

Create the variables listed in Table 3-1 in the OrderManagementBP.

Table 3-1 OrderManagementBP variables

Variable name	Data type variable
customer	Customer
customerRC	ReturnCode
item	Item
itemRC	ReturnCode

Variable name	Data type variable
order	Order
orderRC	ReturnCode
totalStockAtWarehouses	int
processID	string

Save the process and verify that there are no errors, warnings, or informational messages in the Problems view.

Adding the Init Assign activity

Next, add an Assign activity to assign initial values to the variables. Follow these steps:

1. Drop an Assign activity between the Log - Begin and Log - End Snippet activities.
2. Name the Assign activity `Init`.
3. In the Properties view, set the Assign From and Assign To fields as depicted in Figure 3-19.

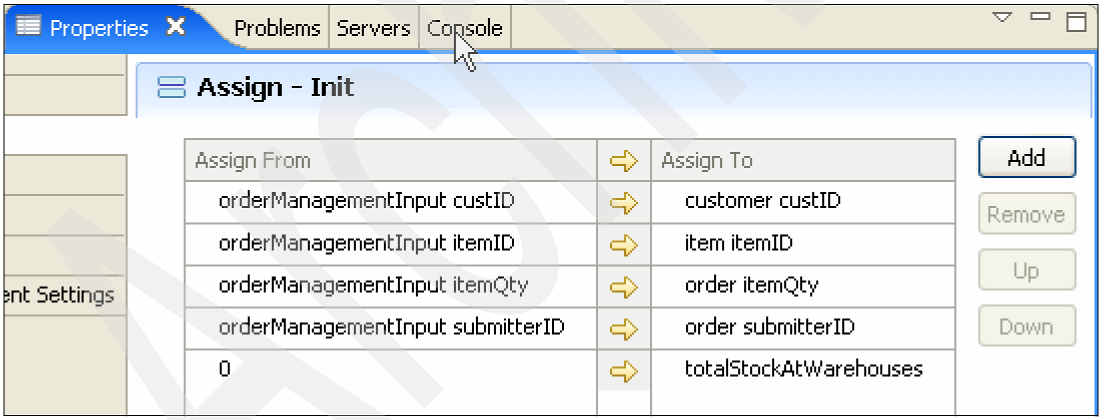


Figure 3-19 Init's properties view

Adding the OrderValidation Parallel Activities structure

Now, you add a Parallel Activity structure. This structure contains Invoke activities to retrieve the customer and item data from DBMS Mediation using the CustomerServiceIF and ItemServiceIF interfaces. Follow these steps:

1. Add a Parallel Activities below the Init.
2. Go to **Properties** → **Description**, and then set:
   - Name: `OrderValidation`
   - Display Name: `Order Validation`

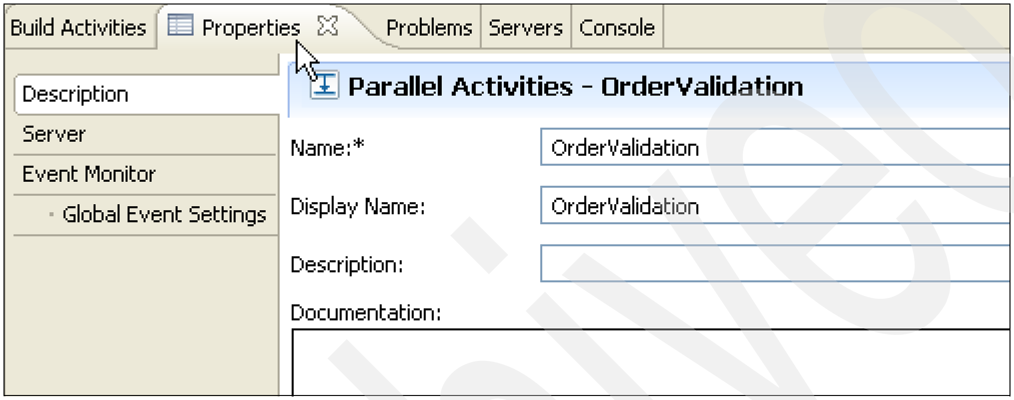


Figure 3-20 OrderValidation's Properties view

3. Add an Invoke activity into the Order Validation activity. Change the display name to `Retrieve Customer Data`, and set its properties as shown in Figure 3-21.

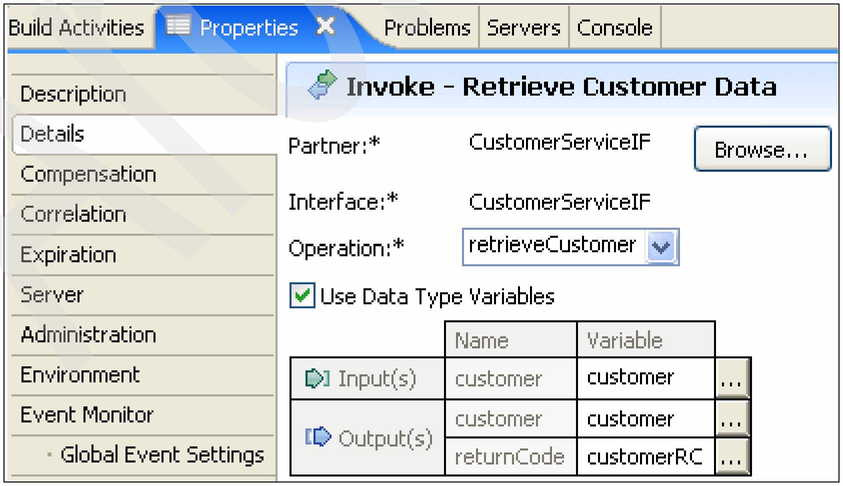


Figure 3-21 Retrieve Customer Data's Properties view

4. Add another Invoke activity into the OrderValidation activity. Change the display name to `RetrieveItemDetails`, and set its properties as shown in Figure 3-22.

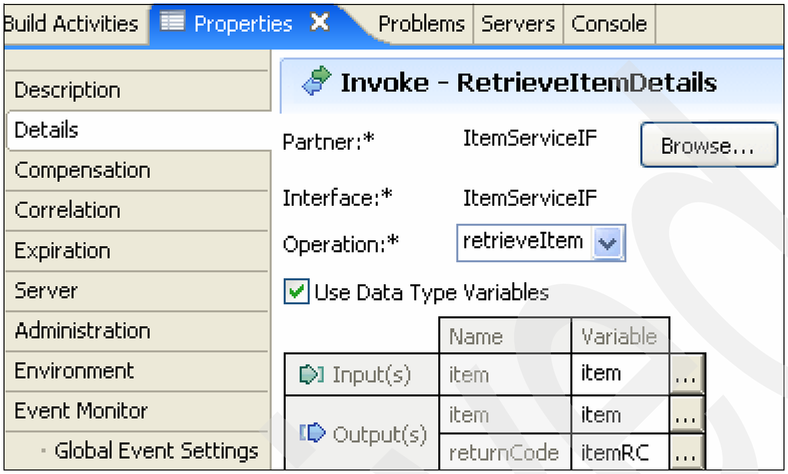


Figure 3-22 RetrieveItemDetails's Properties view

Your business process most likely shows the two Invoke activities partially hidden or not aligned in parallel, as shown in Figure 3-23.

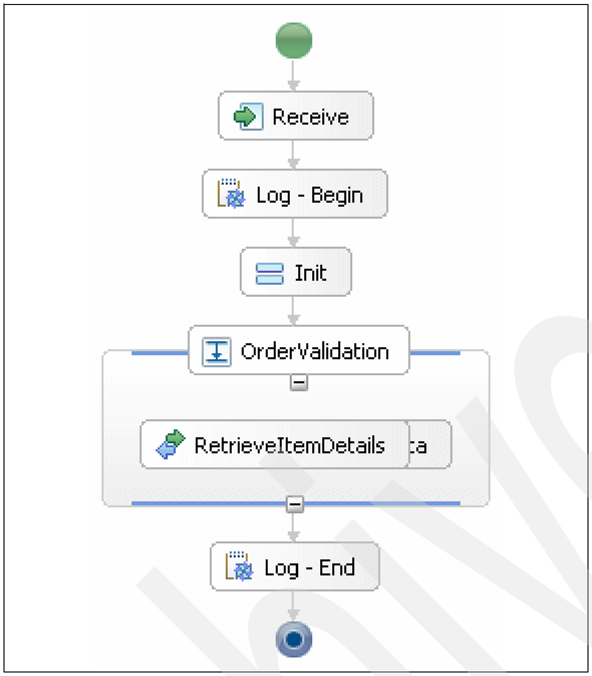


Figure 3-23 Parallel activities alignment

5. Right-click **OrderValidation**, and then select **Align Parallel Activities Contents Automatically**.

   The business process now shows the Invoke activities aligned in parallel, as shown in Figure 3-24.

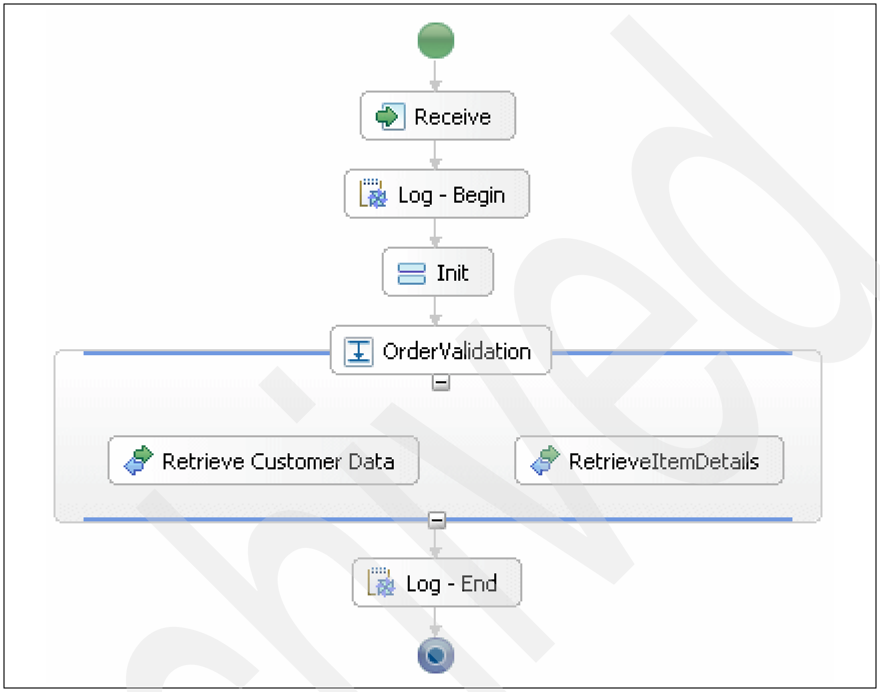


Figure 3-24 Parallel Activities with activities aligned in parallel

6. Save the process and verify that there are no errors, warnings, or informational messages in the Problems view.

Adding a Choice activity to verify the customer and item exist

The Choice activity is a structured activity that contains the logic that determines whether the customer and item data retrieval are successful. If either retrieval fails, then the process is terminated. If both retrievals are successful, the order is created in a database, the split of the items that is taken across the available warehouses is determined, and the total available stock is calculated.

Follow these steps to add a Choice activity:

1. Drop a Choice activity between OrderValidation and Log - End. Name the activity `Check if both customer and item exist`, as shown in Figure 3-25.

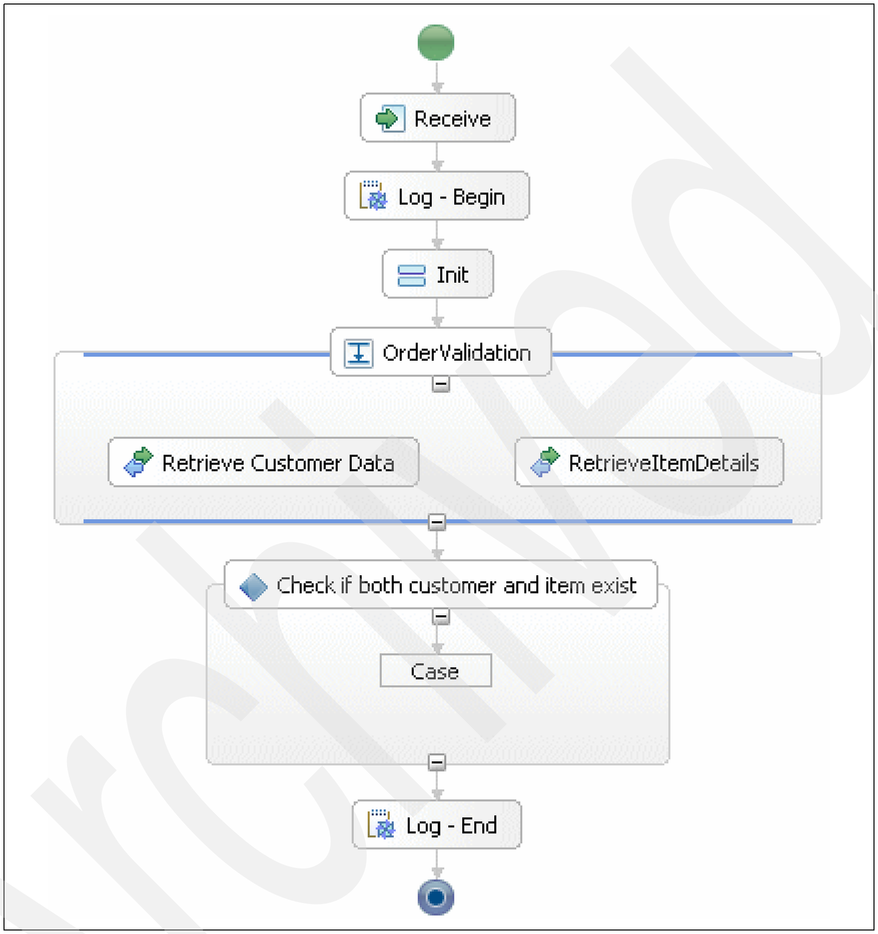


Figure 3-25 OrderManagementBP with Choice activity

2. In the Properties view of the Case element, set the expression language to `Same as Process (Java)`. Set the Expression type to `Visual`, which allows you to use the visual snippet editor to build the expression logic. See Figure 3-26.

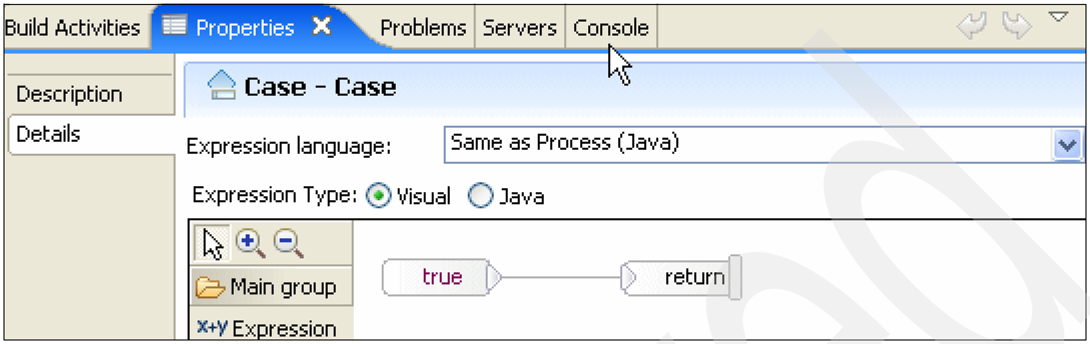


Figure 3-26 Case element's expression language

3. Switch to the Description tab and set the Display Name to `Customer or Item does not exist`, as shown in Figure 3-27.

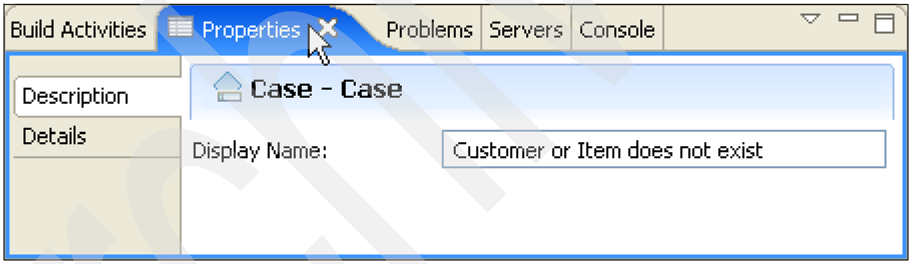


Figure 3-27 Setting the Display Name of a Case element

4. Go back to the Details tab and use the visual snippet editor to build the logic for the Case element as follows:
   a. Add a Standard visual snippet onto the canvas by selecting **text** → **text equal to**. Click **OK** then click the canvas to add the snippet.
   b. Add a second snippet to the canvas. Again, select **text** → **text equal to**.
   c. Add a third snippet onto the canvas. Select **logic** → **or**.
   d. Wire the **or** activity to the **return**, which leaves the true activity not wired to anything else. See Figure 3-28.

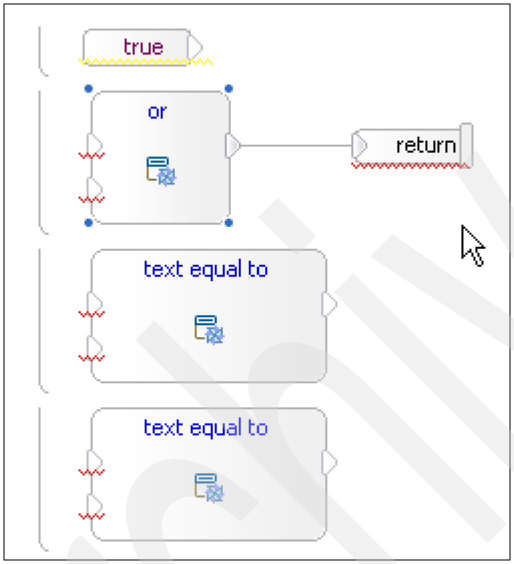


Figure 3-28 Case element design

5. Right-click the true activity and select **Delete**.
6. Wire the two **text equal to** activities to the **or** activity.
7. Drag **customerRC** and **itemRC** onto the canvas from the list of variables at the far right of the canvas. See Figure 3-29.

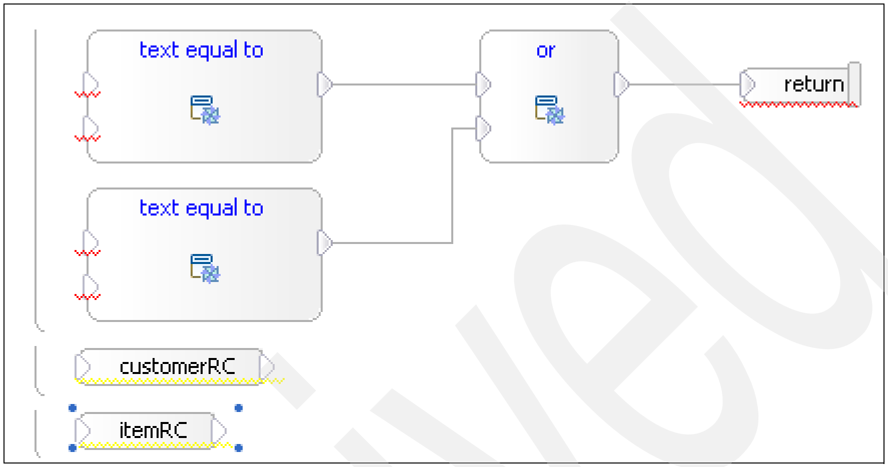


Figure 3-29 Case element design

8. Select the attribute of the **customerRC** and **itemRC**:
   a. Click **customerRC** and select **CustomerRC** → **RC**.
   b. Click **itemRC** and select **ItemRC** → **RC**.
9. Wire customerRC to the input of the *top* **text equal to** snippet.
10. Wire itemRC to the input of the *bottom* **text equal to** snippet.
11. Drag an Expression onto the canvas, then set it to the string “99”. Then, do this step a second time.

12. Wire each of these expressions to the second input of the **text equal to** snippets to complete the Case element design, as shown in Figure 3-30.

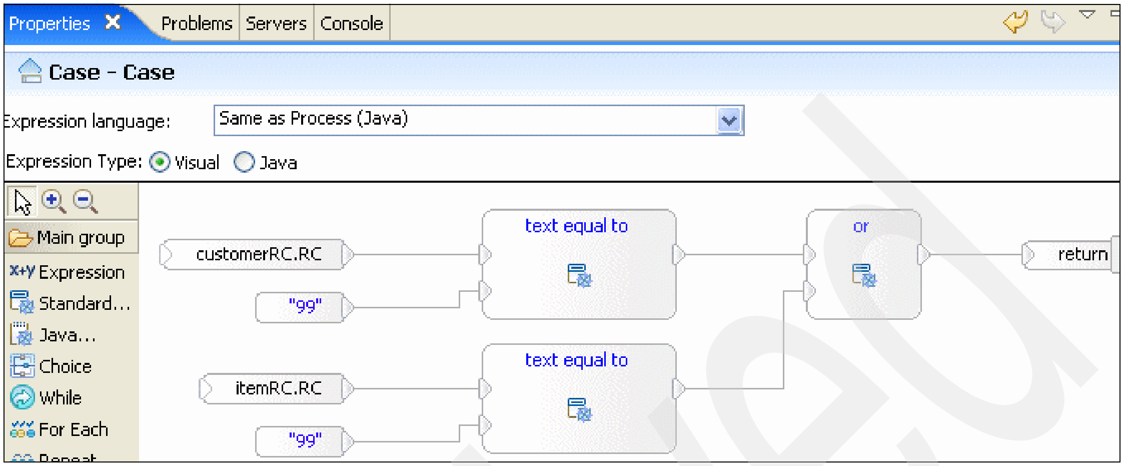


Figure 3-30 Case element design

13. Next, add an Otherwise element to the Choice activity (Figure 3-31):

a. Right-click the Choice activity and select **Add Otherwise**.
b. Verify that an Otherwise element was added to the Choice activity.

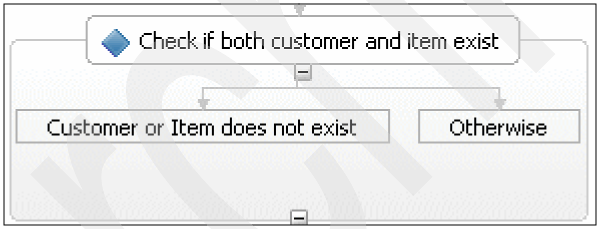


Figure 3-31 Choice activity with Otherwise

14. Complete the path **Customer of Item does not exist** as follows:

   a. Drag an Empty Action activity below **Customer or Item does not exist**. Name the activity `Error`.

   b. Drag a Terminate activity below Error, as shown in Figure 3-32.

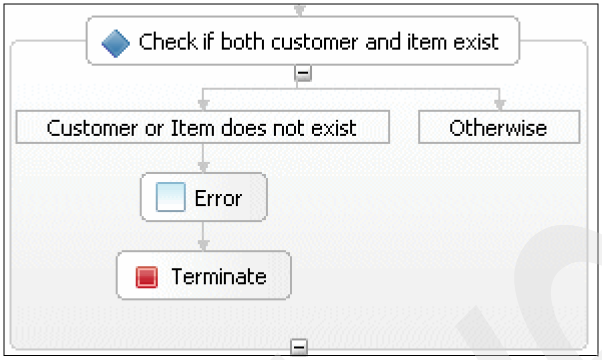


Figure 3-32 Customer of Item does not exist logic of Choice activity

15. Now, complete the Otherwise path:

   a. Drag a Snippet activity below the Otherwise element. Name the activity `Set Order creation parameters`.

   b. In the Properties details, switch from the Visual design view to the Java design view and complete the snippet as shown in Figure 3-33.

> **Additional material:** The Java code for this snippet is contained in the additional materials in the following file:
>
> BusExampleSnippets\SetOrderCreationParameters.txt
>
> See Appendix B, “Additional material” on page 461.

Snippet - Set Order creation parameters

Visual Java

```
order.setString("state", "CREATED");
order.setInt("amount",
    ( new java.lang.Integer(item.getInt("price") * order.getInt("itemQty")) ).intValue() );
java.util.Date dCurrent = new java.util.Date();
order.setDate("creationDate", dCurrent);
order.setDate("completionDate", dCurrent);
processID = processInstance().getID().toString();
order.setString("ordID", processID);
```

Figure 3-33 Set Order creation parameters

16. Verify that your BPEL looks similar to Figure 3-34.

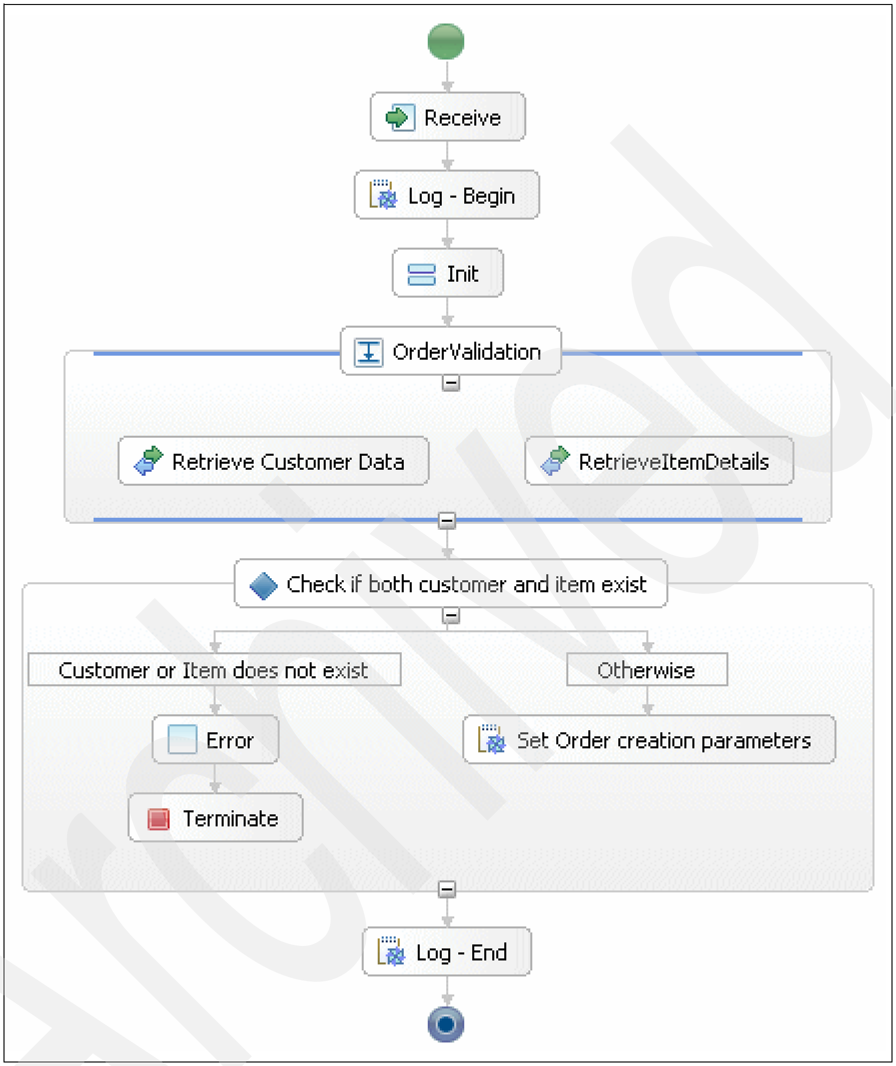


Figure 3-34 Basic OrderManagement BPEL

17. Add an Invoke activity below the **Set Order creation parameters** activity. Name the activity `Create Order` and set the properties as shown in Figure 3-35. This activity invokes the DBMSServiceMedation using the OrderServiceIF.

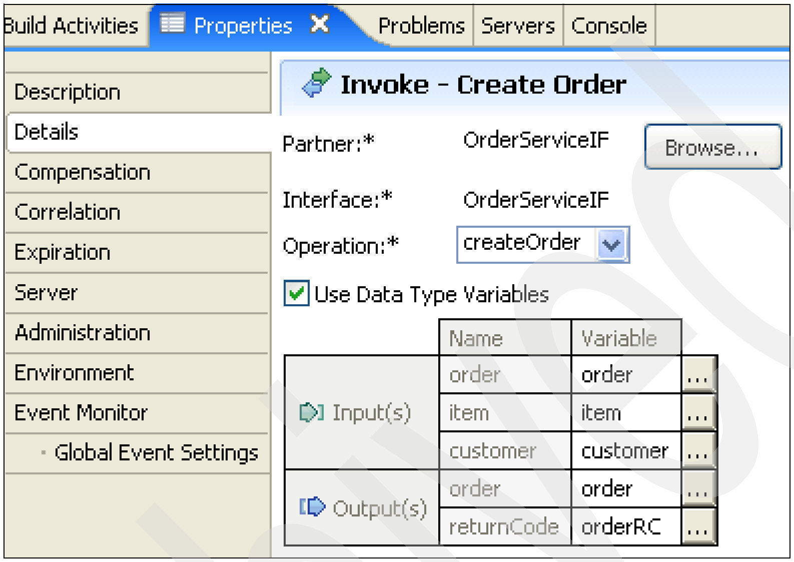


Figure 3-35 Create Order properties view

18. Add a Snippet activity to calculate the total cost of the order as follows:

   a. Drop a Snippet activity below the Create Order activity. Name the snippet: `Calculate Total Stock`.

   b. Add an expression for item.warehouses.

   c. Add a For Each item to the canvas.

d. Right-click the For Each item, then select **Rename**. Enter the new name as w, as shown in Figure 3-36, and then click **OK**. The name *w* is the name of the iterator variable.

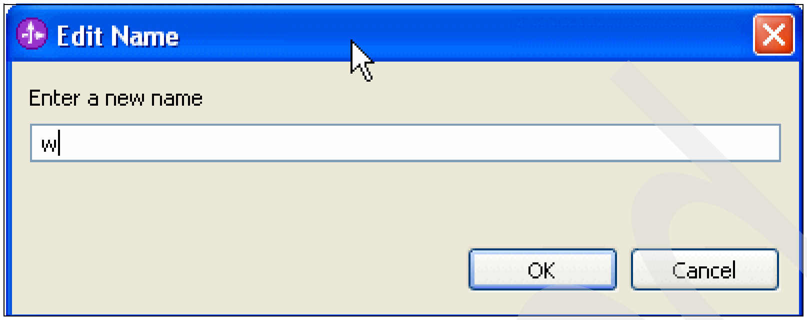


Figure 3-36 Enter For Each iterator variable name

e. Add two totalStockAtWarehouses expressions in the For Each block.

f. Add a standard snippet, **math** → **add** to the For Each block. Then, wire the add to totalStockAtWarehouses.

g. Right-click the **For Each** and select **Add Existing** → **w** to add a new element in the For Each block called *w*.

h. Set the type of the iterator variable by right-clicking this new element and selecting **Set Type** → **Business Object**. Select **Warehouse** as the **Data Type Selection**, then click **OK**.

i. Now that you have set the data type of the iterator, you can drill-down and set the field in which you are interested. Click the **w** in the new element and select **w : warehouse** → **stock**. Then, verify that the canvas looks similar to that shown in Figure 3-37.

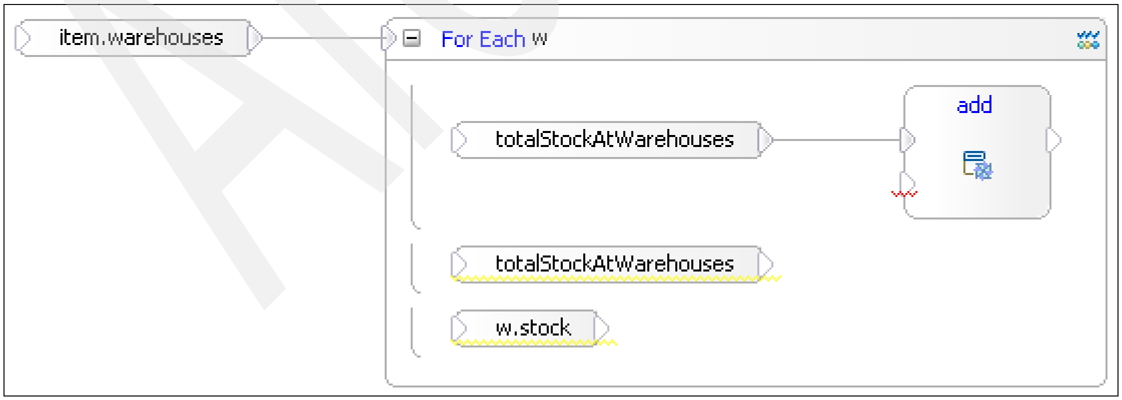


Figure 3-37 For Each intermediate design

j. Complete the For Each item as shown in Figure 3-38.

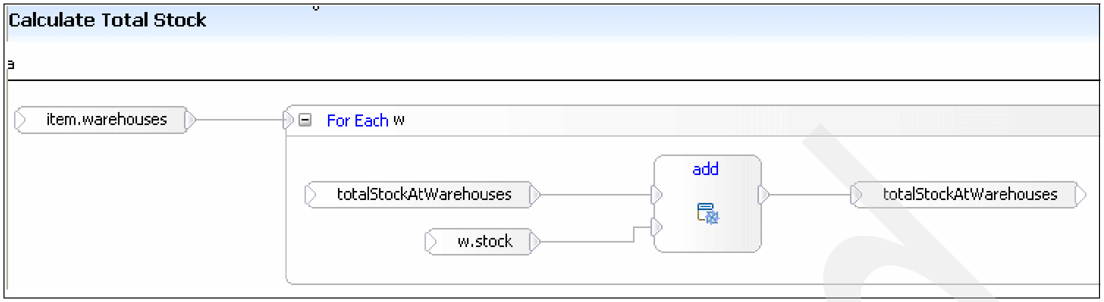


Figure 3-38 For Each design completed

> **Note:** In 3.7, “Adding the ForEach activity and test” on page 200 we convert this Java Snippet to a ForEach activity.

19. Verify that your BPEL looks as shown in Figure 3-39.

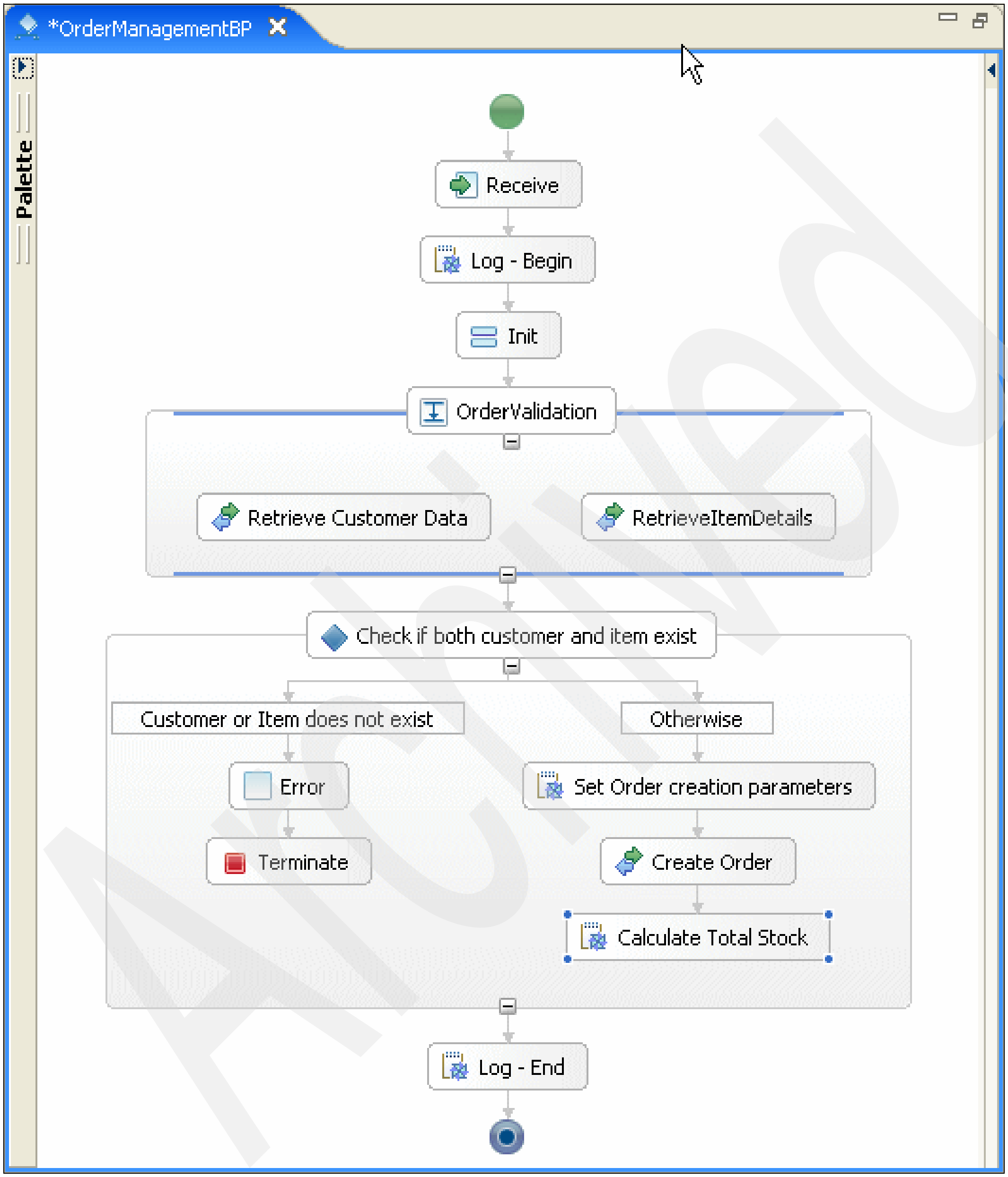


Figure 3-39 OrderManagementBP design

20. Save the process using Ctrl+S.

21. Click the Problems view and verify that there are no errors, warnings, or informational messages.

Adding the OrderFeasibility Parallel Activities structure

In the previous step, you created an order if both the customer and the item existed. The next step is to see whether the order is feasible by looking at the customer's financial information and the warehouse information to see if the order can be fulfilled. You can do both of these checks in parallel in a parallel activity named *OrderFeasibility*.

The following steps show only how to develop a placeholder for the actual steps that are implemented later in 3.4, "Adding business rules to the process" on page 112 and 3.5, "Implementing and testing human tasks" on page 156.

To add the OrderFeasibility activity:

1. Below the "Check if both customer and item exist" Choice activity, add a Parallel Activities structure, and name it `OrderFeasibility`.
2. Add a snippet to the Parallel Activities structure and name it `Log - Approve`. Use the same techniques described in "Adding activities for logging" on page 80 to build the snippet as shown in Figure 3-40.

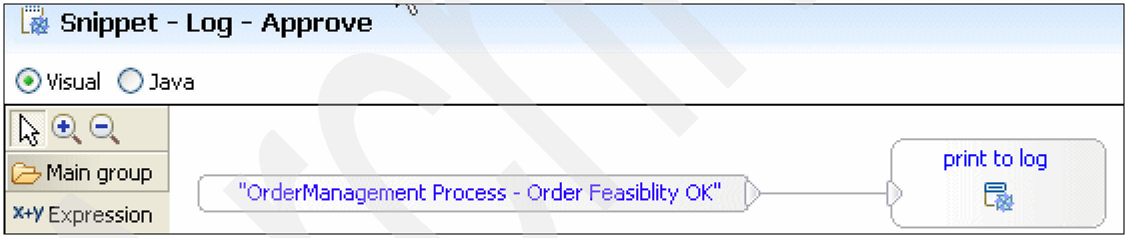


Figure 3-40 Log - Approve details

Adding the Final Verification Choice activity

The OrderFeasibility activity determines if the financial check and the warehouse checks are approved or rejected. Now, you update the order, item, and customer information if both the financial and warehouse check are approved. Otherwise, you update only the order information.

The following steps show only how to develop a placeholder for the actual steps that we implement later in 3.6, "Implementing and testing the final verification steps" on page 189.

To update the information:

1. Add a Choice activity below the OrderFeasibility structure, and then name it `Final Verification`.
2. Go to the Properties view of the Case element that is contained in the Choice activity, and then set the expression language to `Same as Process (java)` as shown in Figure 3-41.

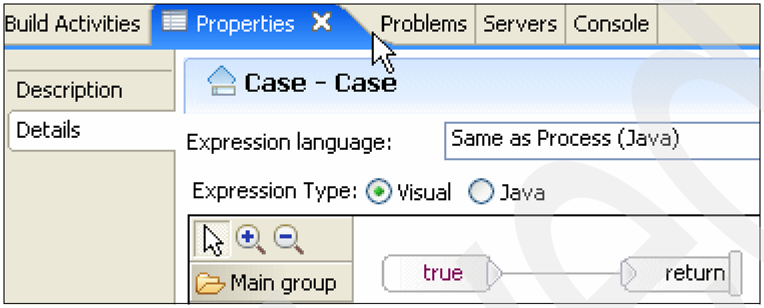


Figure 3-41 Final Verification Case element's Expression Language

3. Add an Empty Action activity below the Case activity.
4. Then, verify your BPEL one last time, as shown in Figure 3-42 on page 100.

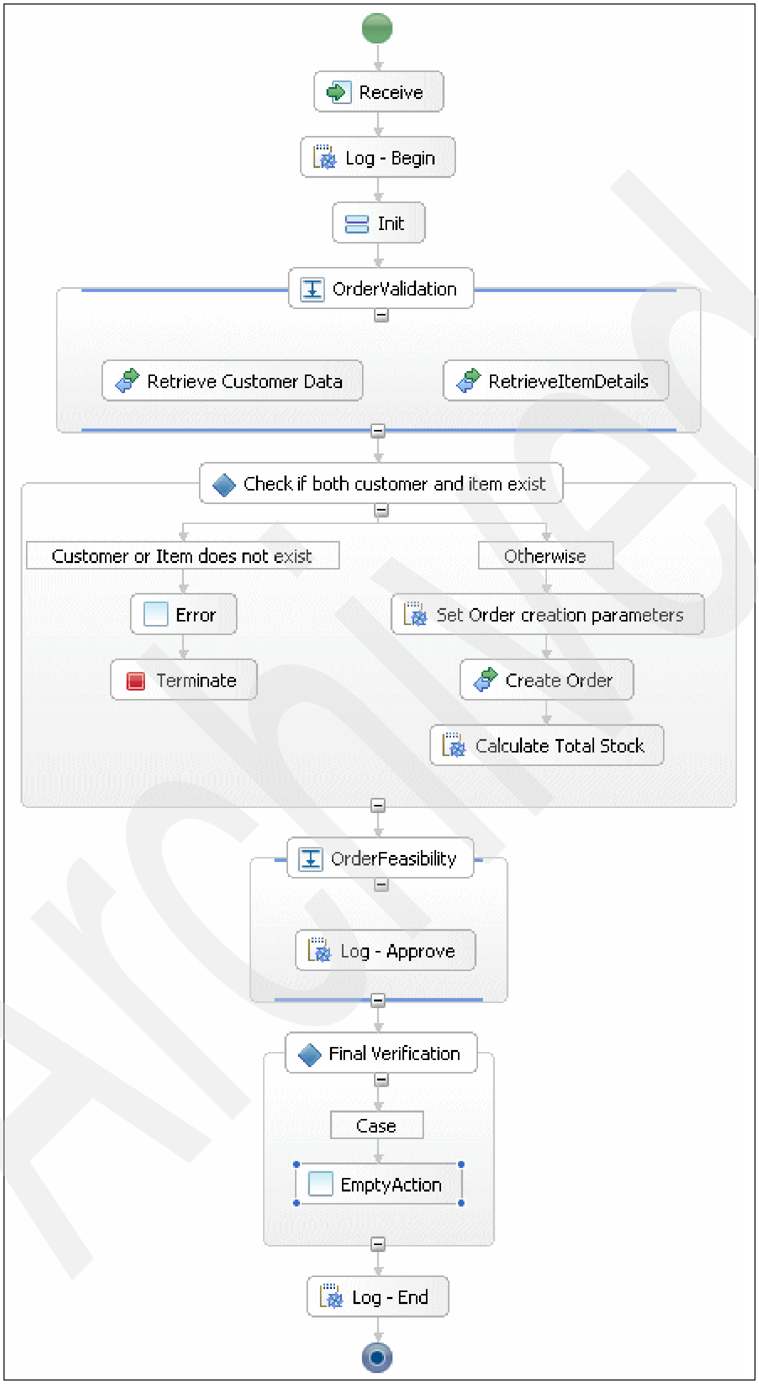


Figure 3-42 OrderManagementBP design

5. Save the process.
6. Click the Problems view, and verify that there are no errors, warnings, or informational messages.

Building the assembly diagram

Now, add the business process to the assembly diagram as follows:

1. Open the OrderManagement assembly diagram.
2. Drag the business process, OrderManagementBP, from the business integration view onto the canvas as shown in Figure 3-43.

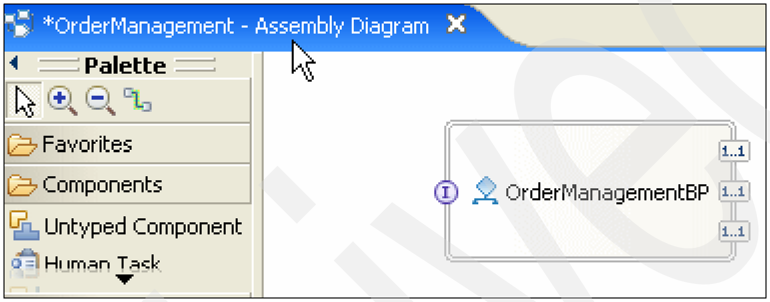


Figure 3-43 OrderManagement's assembly diagram

3. Save the assembly diagram.

> **Note:** Warning symbols display on the three references as shown in Figure 3-44.

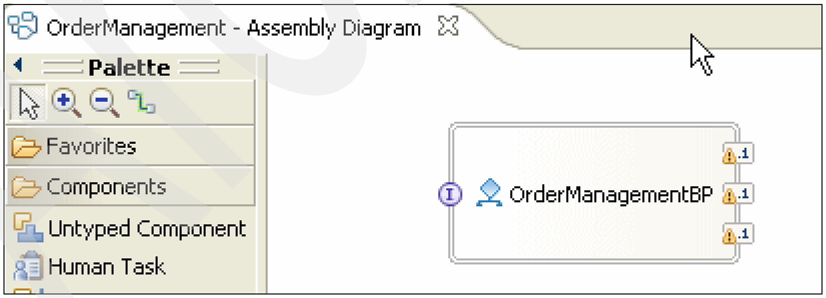


Figure 3-44 OrderManagement's assembly diagram

4. Switch to the Problems view, and notice the warnings, as shown in Figure 3-45.

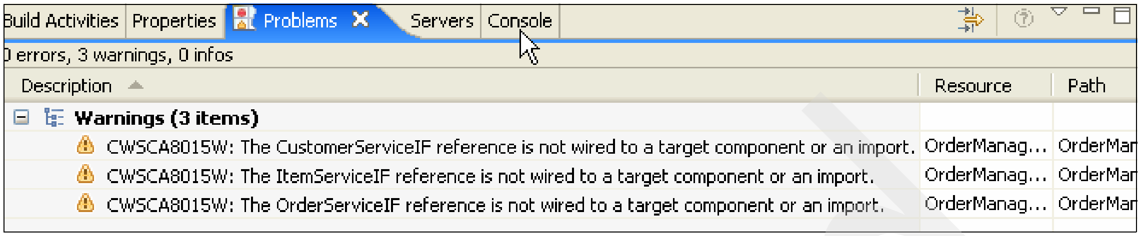


Figure 3-45 Warnings in the Problems view

> **Tip:** Even though these end points are not currently developed, WebSphere Integration Developer provides us with a feature of the integration test client that allows us to emulate modules that might or might not be developed. We demonstrate this emulate functionality in the next section.

3.3.2 Testing the basic business process

In this section, we explain how to ensure that you have obtained the sample data, shown in Figure 3-46 on page 103, and then describe how to use the sample data to test the basic business process.

> **Additional material:** For this scenario, we provide sample data for the input values (orderManagementInput*testcase*.xml) that starts the process for each of the scenarios. We also provide customer and item data (customer*testcase*.xml and item*testcase*.xml) because these values should already exist in the database. This customer and item data is used to enrich the data that is provided in orderManagementInput*testcase*.xml.
>
> This test data is available in the SampleData folder of the additional materials, which you can download as described in Appendix B, “Additional material” on page 461. If you are building this example as you read, make sure that you have downloaded and extracted the sample data.

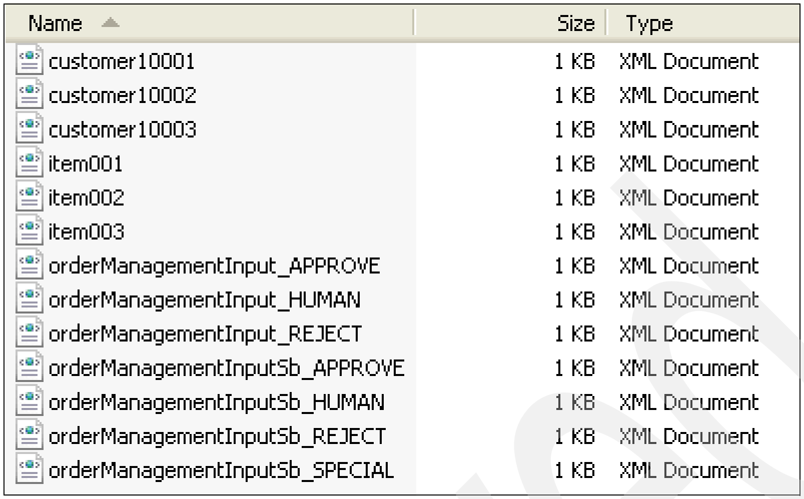


Figure 3-46 Sample Data for testing OrderManagement with other components emulated

Testing the basic process

To test the basic process:

1. Start the WebSphere Process Server test server. Then, wait until you see the `Server server1 open for e-business` message in the Console view.
2. Open the OrderManagement assembly diagram. Right-click **OrderManagementBP** → **Test Component**.

   The integration test client opens.

3. Switch to the Configurations tab to see the emulators that are defined (Figure 3-47). By default, if you do not have a reference wired to a component, the integration test client sets those endpoints automatically as emulated.

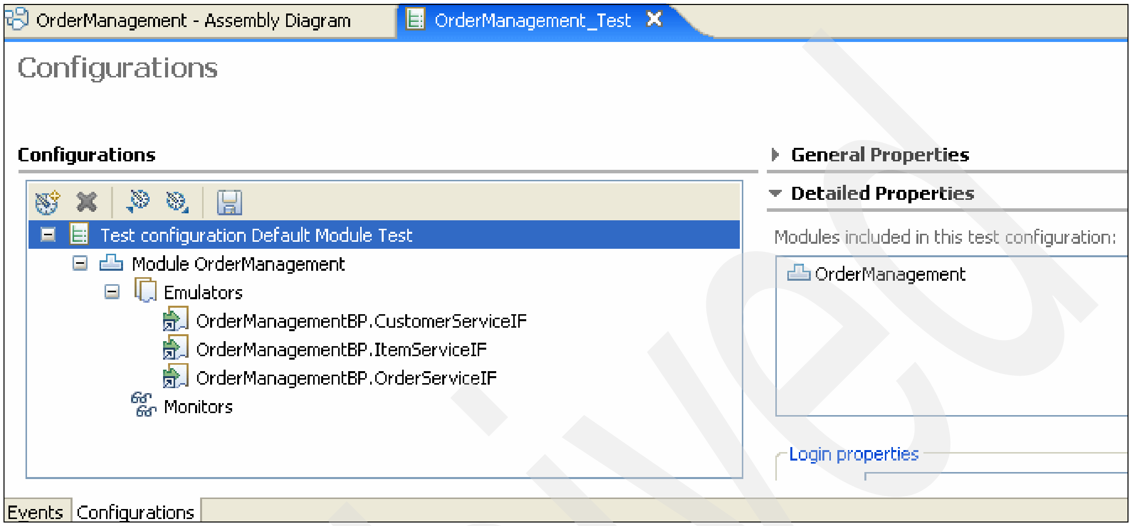


Figure 3-47 Integration test client's Configuration

4. Now, go back to the Events tab. In the Detailed Properties section, select:
   - Configuration: Default Module Test
   - Module: OrderManagement
   - Component: OrderManagementBP
   - Interface: OrderManagementIF
   - Operation: createOrder

5. Then, set the input values for the initial request parameters using the orderManagementInput_APPROVE.xml file as input. In the Initial request parameters, perform the following steps to set the request values:

   a. Right-click **orderManagementInput** and select **Import from XML file**.

   b. Browse to the directory and select the orderManagementInput_APPROVE.xml file. Click **Open**. The values for the initial request parameters are filled in as shown in Figure 3-48.

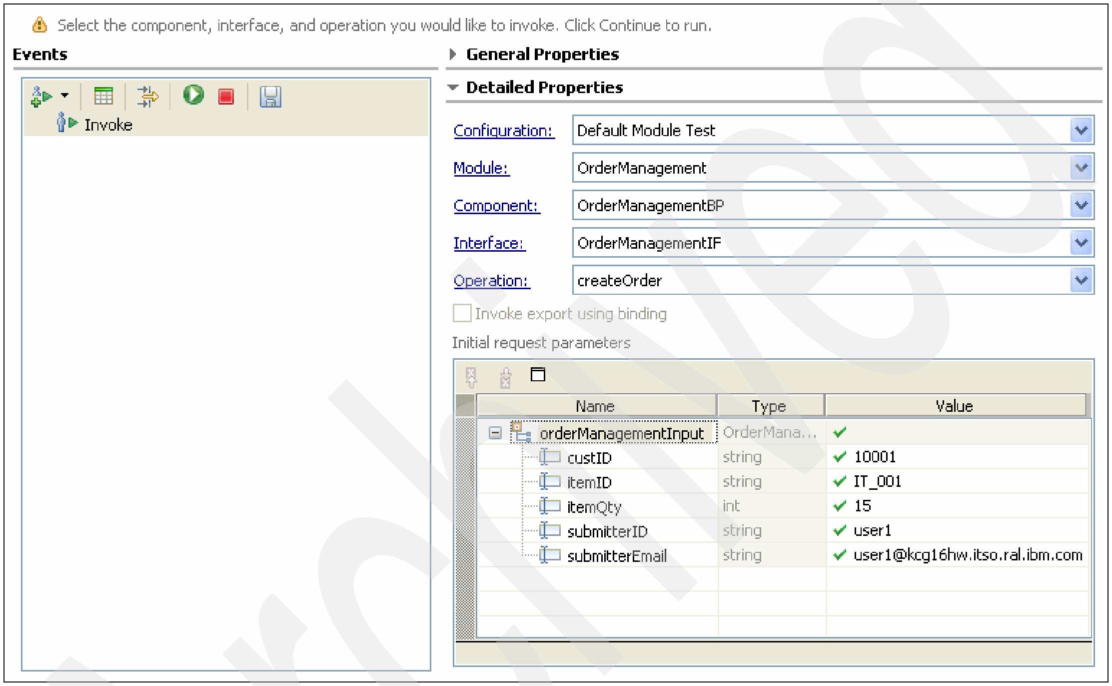


Figure 3-48 Initial request parameters of OrderManagement component test

6. Deploy the module to the server.

> **Tip:** Although starting a component test using the Test Component option deploys the module to the server for you, we sometimes prefer using the **Add and Remove Projects** on the server to deploy the module before starting the test. This method allows us to control what is published to the server as follows:
>
> - If you are testing multiple components using the Test Component option and if you had not deployed previously other components that required for the test, these components are published automatically for you. Instead you receive an error because the other components are not deployed.
> - An alternative is to use the Publish option for the server, but if you are working with multiple components, this option publishes all components. You do not have an option of selecting which modules to publish.

   Follow these steps:

   a. Go to the Servers view. Then, right-click the WebSphere Process Server test server and select **Add and Remove Projects**.

   b. Use the Add button to move OrderManagementApp from the list of Available projects to the list of Configured projects. Then, click **Finish**.

   c. Select **WebSphere Process Server v6.1** as the deployment location, and then click **Finish.**

   d. If prompted for a User ID and password, accept the defaults, then click **OK**. The application is deployed.

7. When the deploy is complete, click the **Continue** icon ( ) to begin testing the business application.

8. Switch to the Console view to look for a message that indicates that `OrderManagement process started`, which is the log from the first activity of the business process.

The integration test client shows the panel shown in Figure 3-49.

Note: The selection shows that for each of those references that were emulated, the integration test client executed all steps that it could automatically and then stopped at the emulation points. In this case, the test stopped at two emulation points because these two activities are executed in parallel.

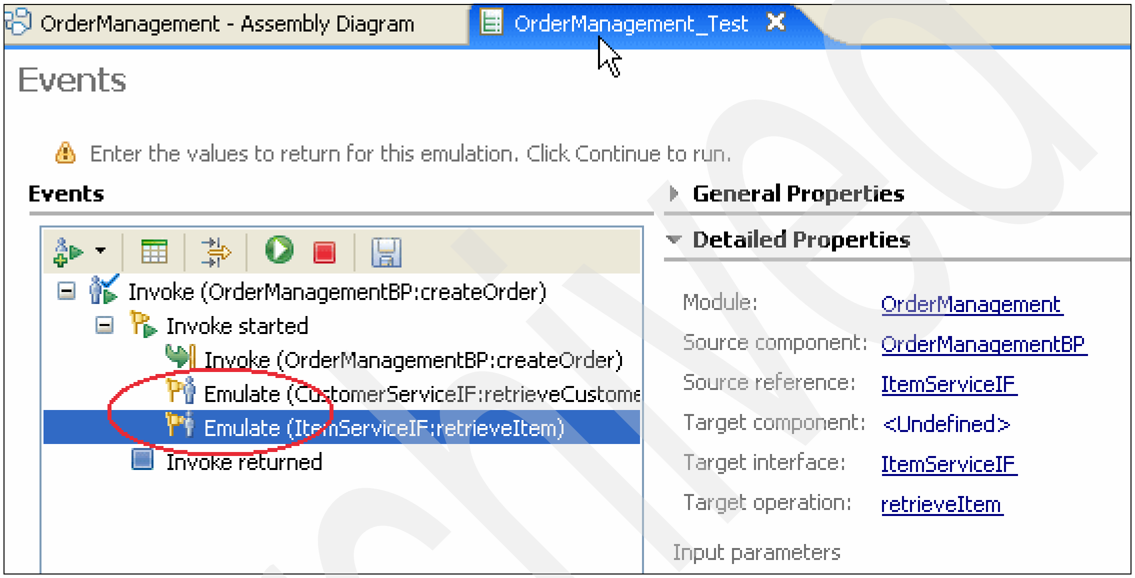


Figure 3-49 Integration test client stopping at emulation points

9. Select the first emulation point (CustomerServiceIF:retrieveCustomer). Complete the customer values as output parameters for both the customer and returnCode objects (Figure 3-50):
   - Use `Customer10001.xml` to set the customer output value.
   - Enter `0` for the returnCode.RC value.

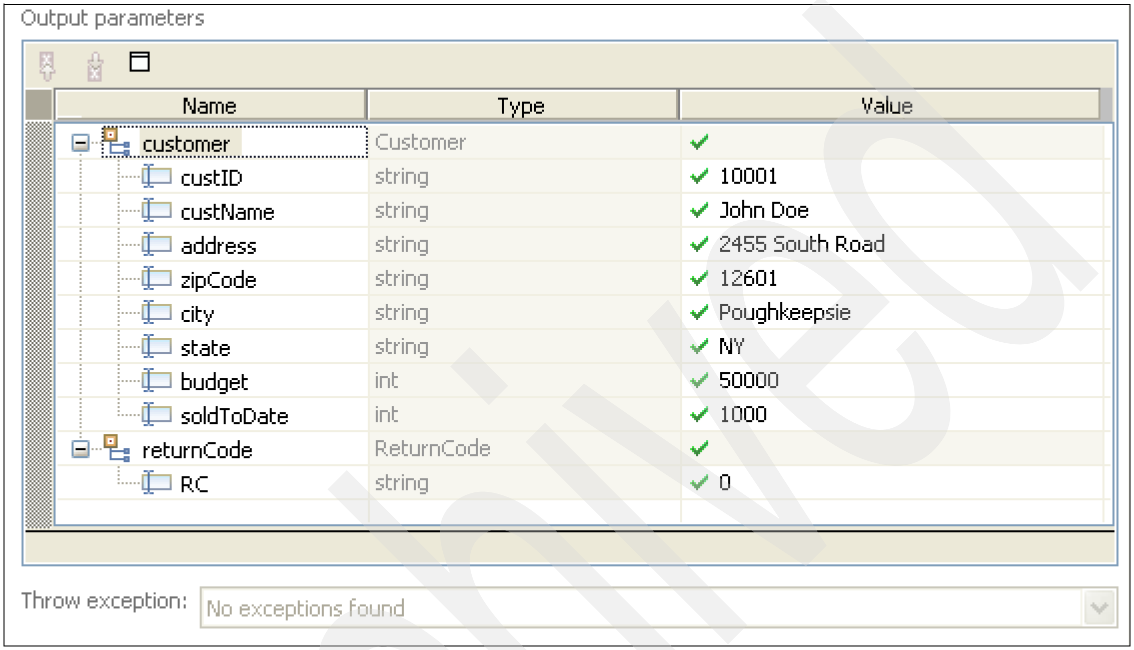


Figure 3-50 Customer information output parameters of the Component Test

> **Tip:** You can save values to the data pool so that you do not have to enter the values again when you need to use them at a latter time. Instead, add the values to the pool, and then later specify that you want to use a value from the pool.
>
> To save the values to the data pool right-click the **customer** object → **Add value to Pool**. Then, accept the default name, then click **OK**.

10.Click **Continue** to execute this activity with the values that you specified.

Note: Notice the check mark next to the first emulate, as shown in Figure 3-51.

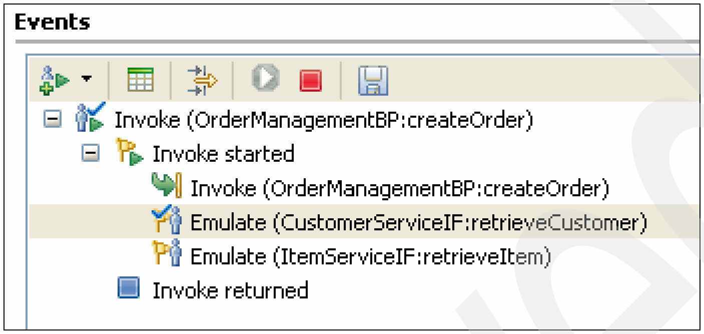


Figure 3-51 Component Test after executing first emulation activity

11. Select the second emulation point (ItemServiceIF:retrieveItem). Enter the second emulation output parameters:

   a. In the output parameters, expand item. Right-click **warehouses** and select **Add Elements**.

   b. Enter the number of elements to add (2). Then, click **OK**.

   c. Use item001.xml to set the item output parameters and set returnCode.RC to 0. See Figure 3-52.

   d. Add the item to the data pool and click **Continue**.

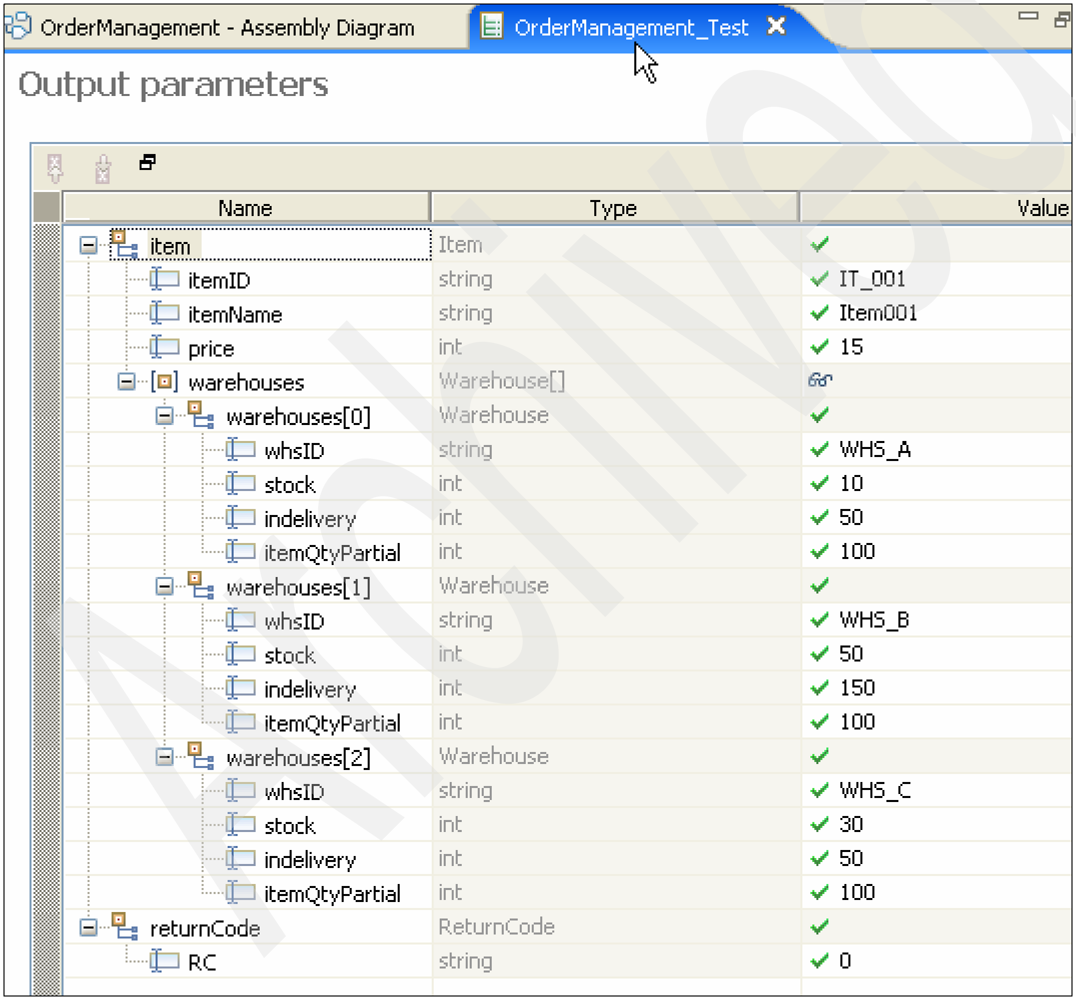


Figure 3-52 Setting the item and returnCode output parameters in the integration test client

The process continues executing and stops at the third emulation point.

12. When the emulation stops at **order**, use the order input value to set the order output value and then set returnCode.RC to `0` (Figure 3-53), as follows:

   a. Right-click the **order** *input* value and select **Copy Value**.

   b. Right-click the **order** *output* value and select **Paste Value**.

   c. Set returnCode.RC to `0`.

   d. Click **Continue**.

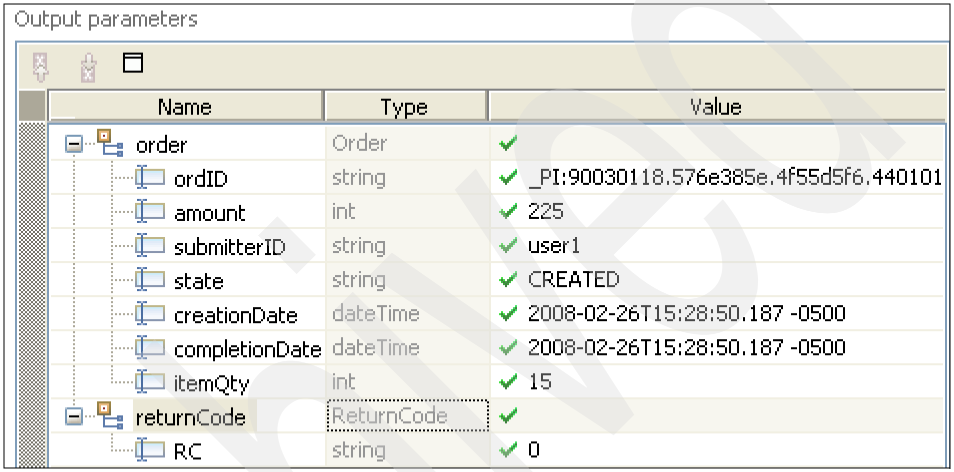


Figure 3-53 Order and returnCode output parameters

Note: The dates reflect the date and time that you ran this application.

13. Switch to the Console view to look at the SystemOut messages. You see the log statements shown in Figure 3-54.

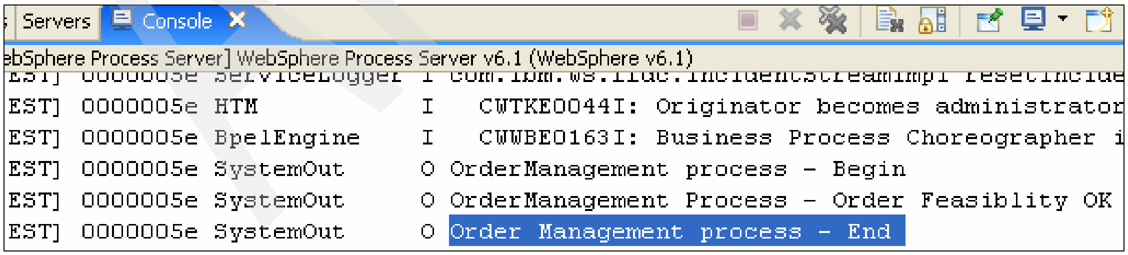


Figure 3-54 Console log showing process' log statements

14. Add the **order** values to the data pool.
15. Use the instructions in step 5 to load the initial request parameter values for orderManagementInput and add these to the data pool.
16. Save this execution trace for either running this again or for documentation purposes:
    a. Create a folder to hold the execution trace file. From the Business Integration view, right-click **OrderManagement**.
    b. Select **New** → **Other...** → **Folder**. Click **Next**.
    c. Select **OrderManagement** as the parent folder. Enter the name of the folder as `TestCases`.
    d. Click **Finish**.
17. Close the execution trace file and, when prompted, if you want to save it, click **Yes** and specify the location as OrderManagement/TestCases/*filename*.

> **Note:** Now is probably a good time to check in projects to CVS.

3.4 Adding business rules to the process

There are two business rules that you need to add to the process:

- A Financial Rule
- A Warehouse Rule

In keeping with iterative development, we first describe the add logic to the OrderFeasibility activity to process the return values of the rules. Next, we explain how to implement and test the Financial Rule while keeping the Warehouse Rule stubbed out. Finally, we describe how to implement and test the Warehouse Rule.

Figure 3-55 shows the activity that we focus on in our BPEL is the OrderFeasibility activity.

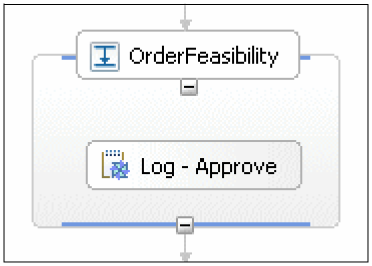


Figure 3-55 OrderFeasibility activity stubbed out

The logic for these parallel activities is as follows:

- The rules are evaluated in parallel. The rules can return three different values:
  - ACCEPT
  - REJECT
  - HUMAN
- Verify the return value of the rule:
  - If returnCode equals ACCEPT, create an entry in a log.
  - Else if returnCode equals REJECT, create an entry in a log.
  - Otherwise, send the request it to an officer for a decision. In our example, we stub out only the Human Activity (log statement), and we add the human tasks later.

We begin development by implementing the Financial Rule.

3.4.1 Adding logic around the rules

The following steps show how to add activities that invoke a rule in the business process and then some logic to process the return values of the rule. After you add this logic, we describe how to build the rules.

To add logic around the rules, follow these steps:

1. Note that the process currently has three reference partners:
   - CustomerServiceIF
   - ItemServiceIF
   - OrderServiceIF
2. Add the FinancialOfficeBRIF and WarehouseOfficeBRIF to the process, as shown in Figure 3-56.

Figure 3-56 Adding the rule reference partners to OrderManagementBP

3. Add a variable to the process for the return code of the Financial Rule:
   – name: `financialRuleRC`
   – type: `ReturnCode`
4. Add an Invoke to the OrderFeasibility activity. Name it `InvokeFinancialOfficeBR`, and set it with the properties shown in Figure 3-57.

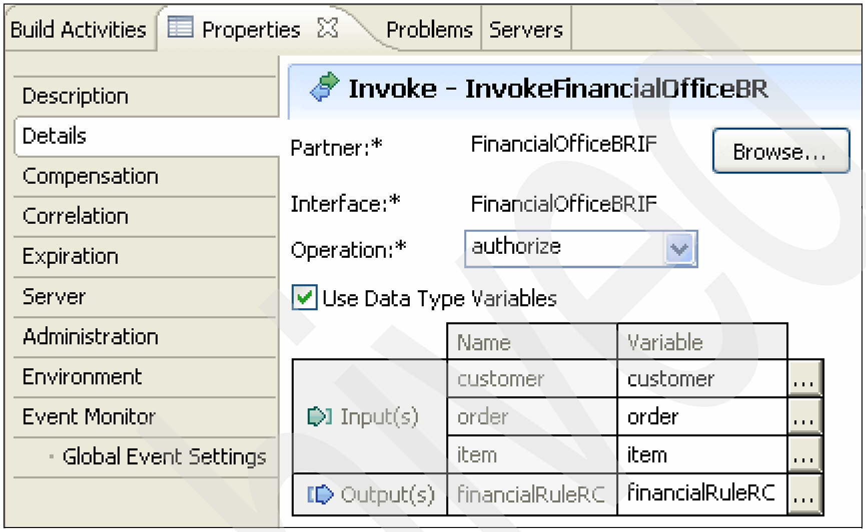


Figure 3-57 InvokeFinancialOfficeBR properties

5. Add a variable to the process for the return code of the Warehouse Rule:
   – name: `warehouseRuleRC`
   – type: `ReturnCode`
6. Add an Invoke activity to the OrderFeasibility activity. Name it `InvokeWarehouseOfficeBR` and set it with the properties shown in Figure 3-58.

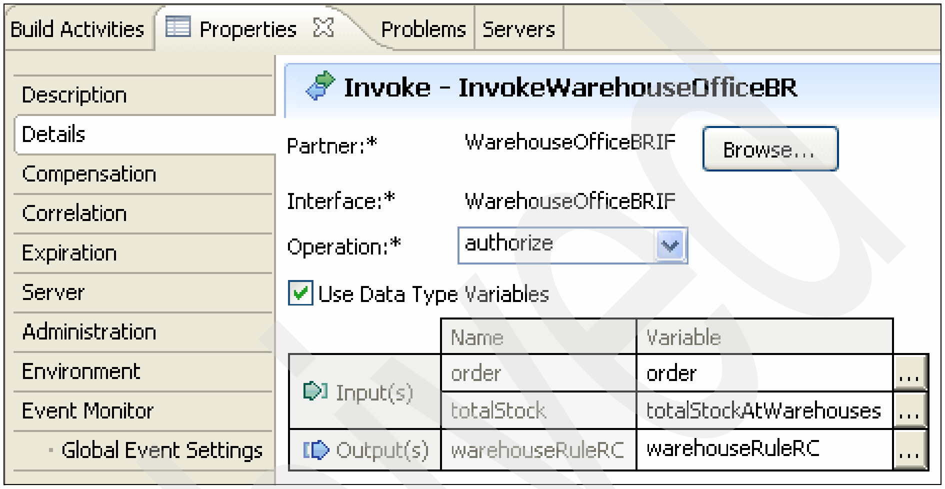


Figure 3-58 InvokeWarehouseOfficeBR properties

7. Right-click the OrderFeasibility activity and select **Align Parallel Activities Contents Automatically**.
8. Add a Choice activity to the OrderFeasibility activity and name it `FinancialCheck` as shown in Figure 3-59.

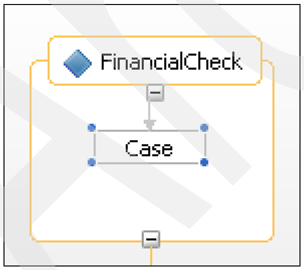


Figure 3-59 FinancialCheck activity

Then, follow these steps:

a. Wire the InvokeFinancialOfficeBR to the FinancialCheck.

b. Make the Log - Approve snippet the first activity of the Case element as shown in Figure 3-60.

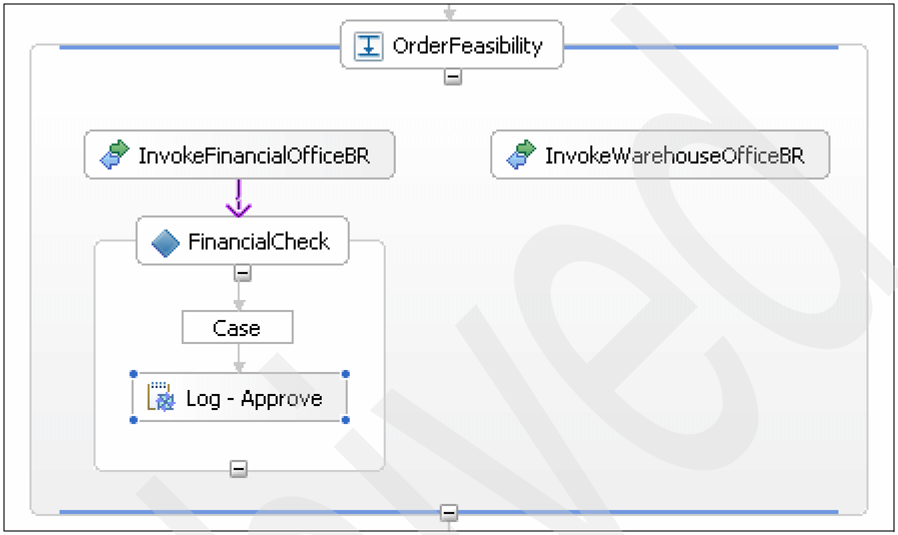


Figure 3-60 OrderFeasibility activity

c. Set the logic for the **Log - Approve** snippet as shown in Figure 3-61.

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\FinancialCheck_LogApprove.txt. See Appendix B, “Additional material” on page 461.

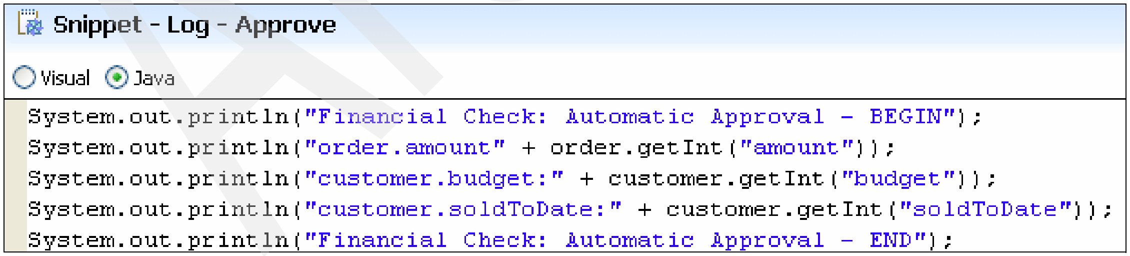


Figure 3-61 Log - Approve on the financial path

9. Repeat the previous step to add a Choice activity in the warehouse path. Name it `WarehouseCheck`.

10. Add a new snippet called `Log - Approve`. See Figure 3-62.

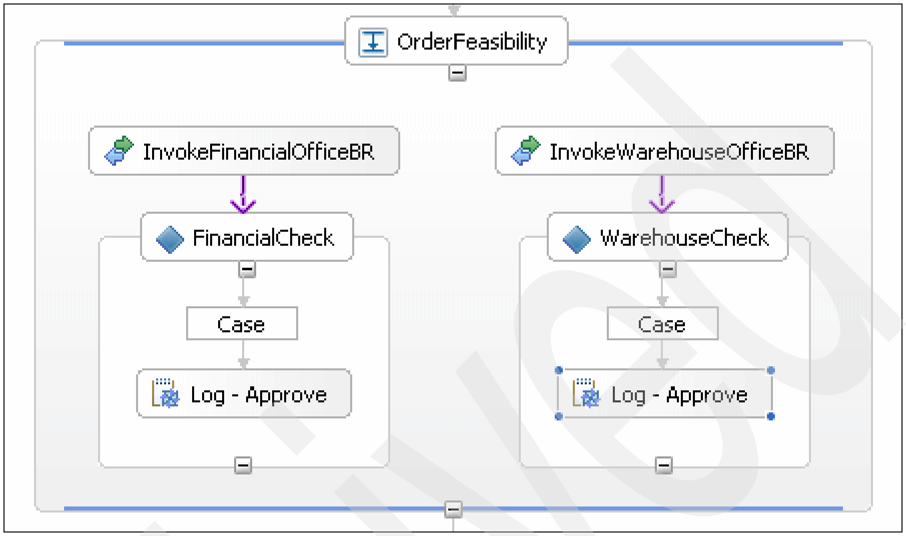


Figure 3-62 OrderFeasibility with two paths

d. Define the Log - Approve snippet as shown in Figure 3-63.

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\WarehouseCheck_LogApprove.txt. See Appendix B, “Additional material” on page 461.

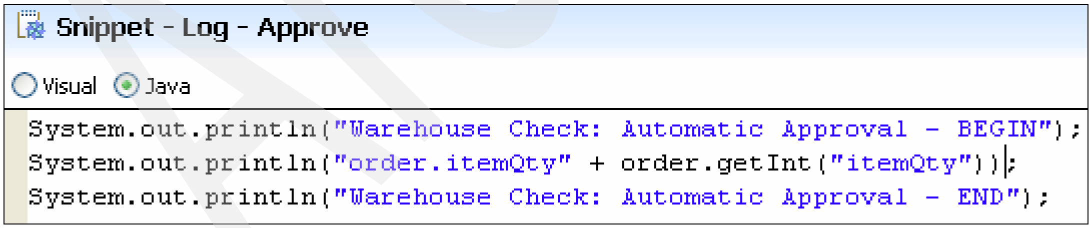


Figure 3-63 Log - Approve on the warehouse path

11.Right-click **FinancialCheck** → **Add Otherwise**. Then:

a. Below the Otherwise element, add a Snippet and name it `Log - HumanDecision`.

b. Build the snippet as shown in Figure 3-64.

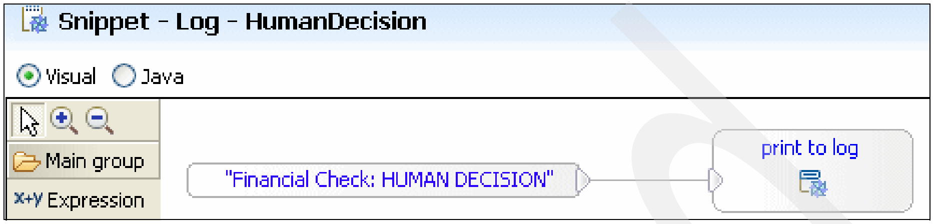


Figure 3-64 Log - Human Decision on financial path

12.Repeat the previous step for the warehouse path, as shown in Figure 3-65.

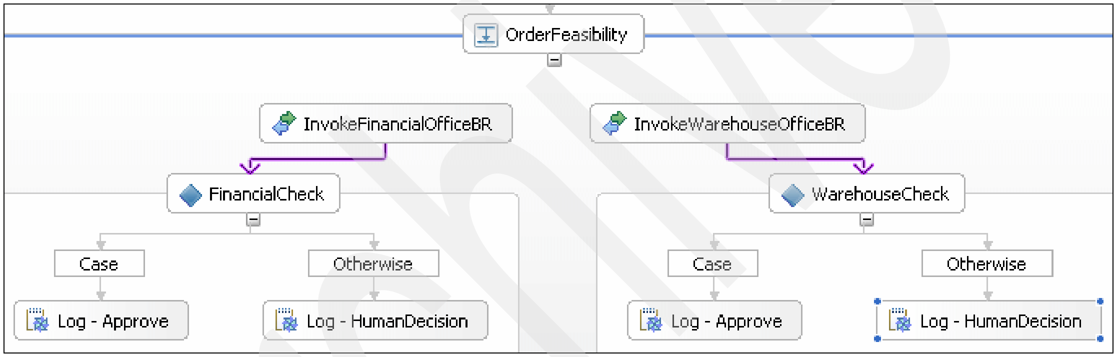


Figure 3-65 OrderFeasibility showing two paths with choice after invoke

The Log - HumanDecision snippet looks as shown in Figure 3-66.

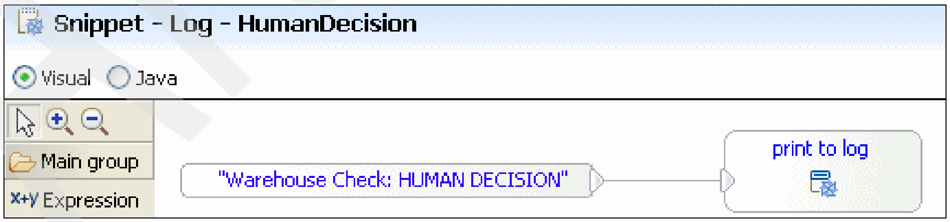


Figure 3-66 Log - Human Decision on warehouse path

13. Implement the logic for FinancialCheck's Case element as follows:

a. Go to the Case element's properties and switch to the Description tab. Set the Display Name as `Approve`.

b. Switch to the Details tab for Case and do the following steps:

i. Set the expression language to `Same as Process (Java)`.

ii. Add the **text equal to (ignore case)** expression to the canvas. You can find the this expression under the **Standard Visual Snippets** → **text** → **text equal to (ignore case).**

c. Complete the Case element's logic as follows:

i. Delete True.

ii. Drag the financialRuleRC variable from the right to the canvas. Click on the new element and select **financialRuleRC** → **RC**.

iii. Add an expression and enter `APPROVE`.

iv. Wire as shown in Figure 3-67.

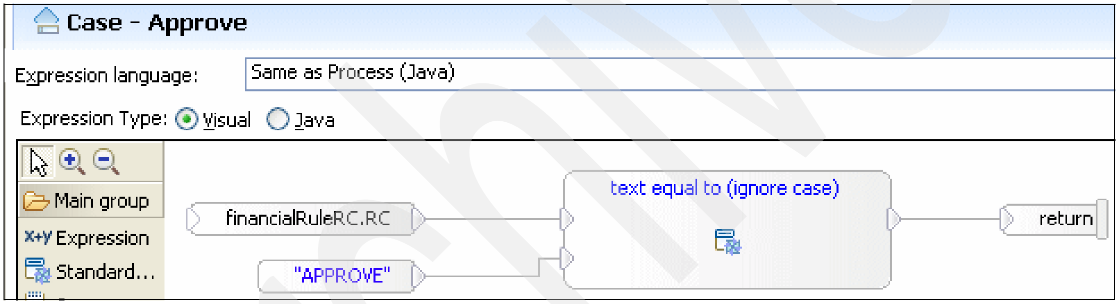


Figure 3-67 Case - Approve logic on financial path

14. Repeat the previous step to set the Case - Approve logic on the warehouse path, using warehouseRuleRC this time, as shown in Figure 3-68.

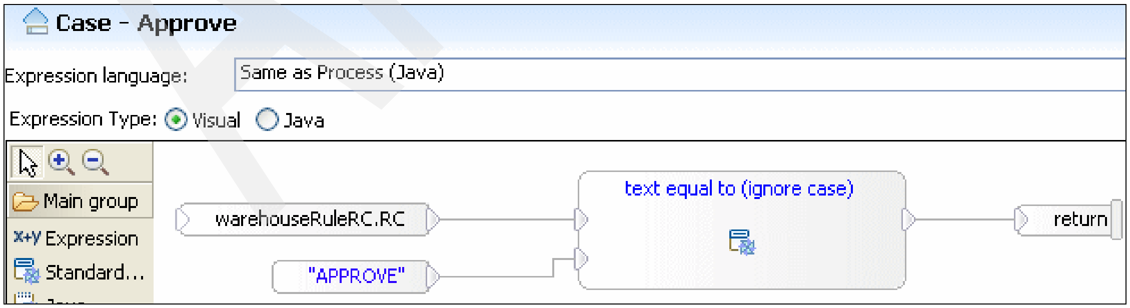


Figure 3-68 Case - Approve logic on warehouse path

15. Right-click **FinancialCheck** → **Add Case**. Then:

   a. Name the Case `Reject`.

   b. Below the Case, add a Snippet and name it `Log - Reject`.

   c. Set the logic for Log - Reject on financial path as shown in Figure 3-69.

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\FinancialCheck_LogReject.txt. See Appendix B, “Additional material” on page 461.

Snippet - Log - Reject

○ Visual ◉ Java

```java
System.out.println("Financial Check: Automatic Rejection - BEGIN");
System.out.println("order.amount" + order.getInt("amount"));
System.out.println("customer.budget:" + customer.getInt("budget"));
System.out.println("customer.soldToDate:" + customer.getInt("soldToDate"));
System.out.println("Financial Check: Automatic Rejection - END");
```

Figure 3-69 Log - Reject on financial path

16. Repeat the previous step to add a case and a snippet for logging on the warehouse path. Set the logic for Log - Reject on warehouse path as shown in Figure 3-70.

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\WarehouseCheck_LogReject.txt. See Appendix B, “Additional material” on page 461.

Snippet - Log - Reject

○ Visual ◉ Java

```java
System.out.println("Warehouse Check: Automatic Rejection - BEGIN");
System.out.println("order.itemQty:" + order.getInt("itemQty"));
System.out.println("totalStockAtWarehouses:" + totalStockAtWarehouses);
System.out.println("Warehouse Check: Automatic Rejection - END");
```

Figure 3-70 Log - Reject on warehouse path

17. Set the Reject Case logic on financial path as follows:

a. Go to Case element's properties. Switch to the Description tab, and set the Display Name to `Reject`.

b. Switch to the Details tab and set the expression language to `Same as Process (Java)`.

c. Complete the logic as shown in Figure 3-71.

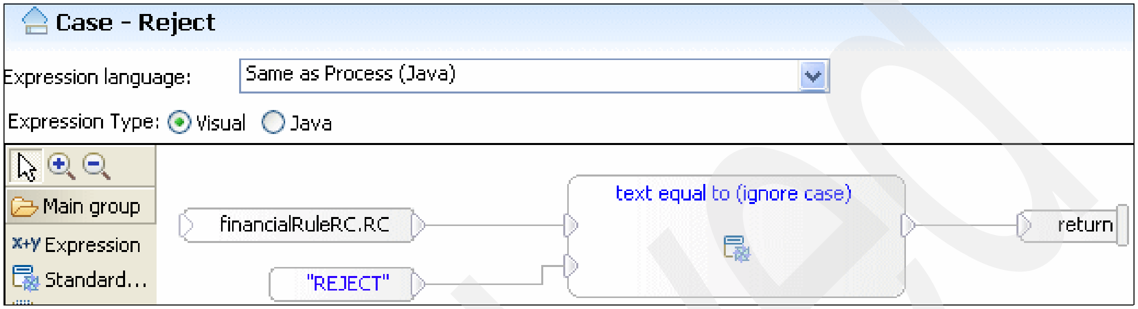


Figure 3-71 Case - Reject logic on financial path

18. Set the Reject Case logic on the warehouse path as follows:

a. Go to the Case element's properties. Switch to the Description tab, and set the Display Name to `Reject`.

b. Switch to the Details tab, and set the expression language to `Same as Process (Java)`.

c. Complete the Case element's logic as shown in Figure 3-72.

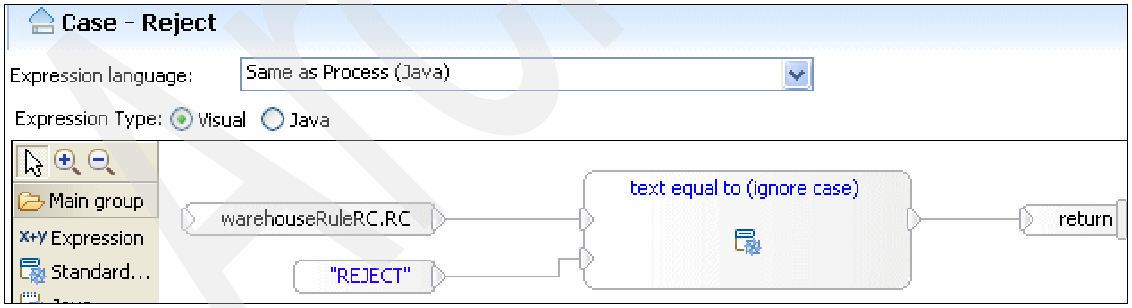


Figure 3-72 Case - Reject logic on warehouse path

19. Ensure the financial path looks as shown in Figure 3-73.

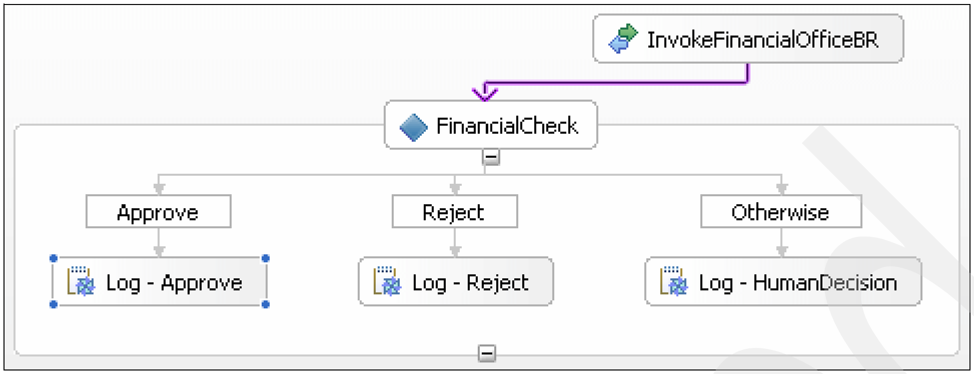


Figure 3-73 Financial path

20. Ensure the warehouse path looks as shown in Figure 3-74.

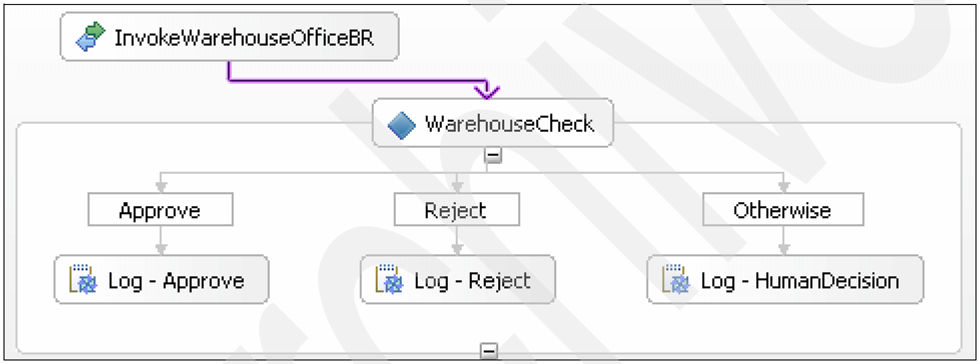


Figure 3-74 Warehouse path

21. Save the business process. A couple of errors displays on the Problems view, as shown in Figure 3-75.

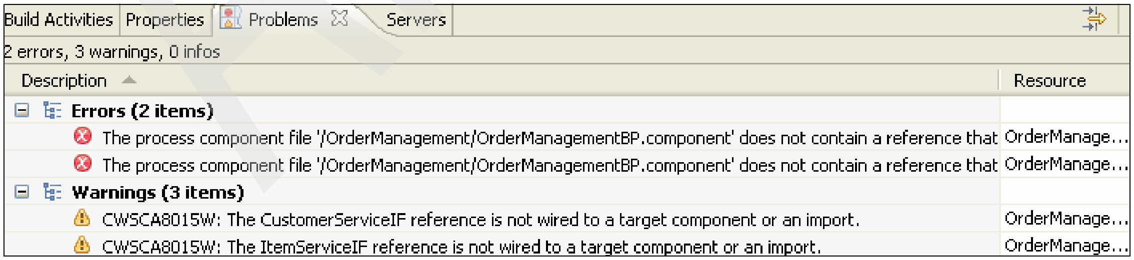


Figure 3-75 Errors in Problems View after adding rules to business process

22. If you recall, when you added the rule logic, you added a couple of reference partners. Therefore, the OrderManagementBP's representation on the assembly diagram is out-of-sync with the actual OrderManagementBP implementation. To fix the errors:

   a. Open the OrderManagement assembly diagram.

   b. Right-click **OrderManagementBP**.

   c. Select **Synchronize Interfaces and References** → **from Implementation**.

   The updated assembly diagram now contains an OrderManagementBP with five references.

   No errors display in your Problems view, but you should see five warnings, one for each reference partner.

3.4.2 Implementing the Financial Rule

In this section, we explain the logic of the rules that we implement, and then we actually develop the rules.

Financial Rule logic

For each customer that places an order, a financial check runs based on a customer's exposure. Exposure is defined as:

exposure = order amount - (customer budget - customer's sold to date)

> **Example:** Say a customer has a budget of $5000 and the customer ordered previously $3000 worth of merchandise sold-to-date. That leaves $2000 in the budget. If the new order is for $1000, the exposure is:
>
> -1000 = 1000 - (5000-3000). (ACCEPT)
>
> Using the same example, if the customer orders $13000:
>
> 11000 = 13000 - (5000-3000). (REJECT)
>
> If the customer orders $4000:
>
> 2000 = 4000 - (5000-3000). (HUMAN DECISION)
>
> We allow a customer to go $1000 over budget without flagging the order for human decision or rejection.

The financial check logic is summarized in Table 3-2.

Table 3-2 Financial check logic

Condition	Result
Exposure <= 1000	Automatically accept
Exposure > 10000	Automatically reject
Otherwise	Human Decision

> **Tip:** If we look at the conditions in Table 3-2, we see static values for an acceptance threshold equal 1000, while the rejection threshold is 10000. These two values are prime candidates to use as parameters that can be changed during runtime, giving a business flexibility to change business logic at runtime without having to change the underlying implementation. If you set these values as parameters, then you can change them at runtime using the Business Rules Manager.

Creating the rule group and rule set

To create the financial rule group and rule set:

1. Expand the OrderManagement module in the Business Integration view.
2. Right-click **Business Logic**, select **New** → **Rule Group**, and set the Name to `FinancialOfficeRG`, as shown in Figure 3-76. Click **Next**.

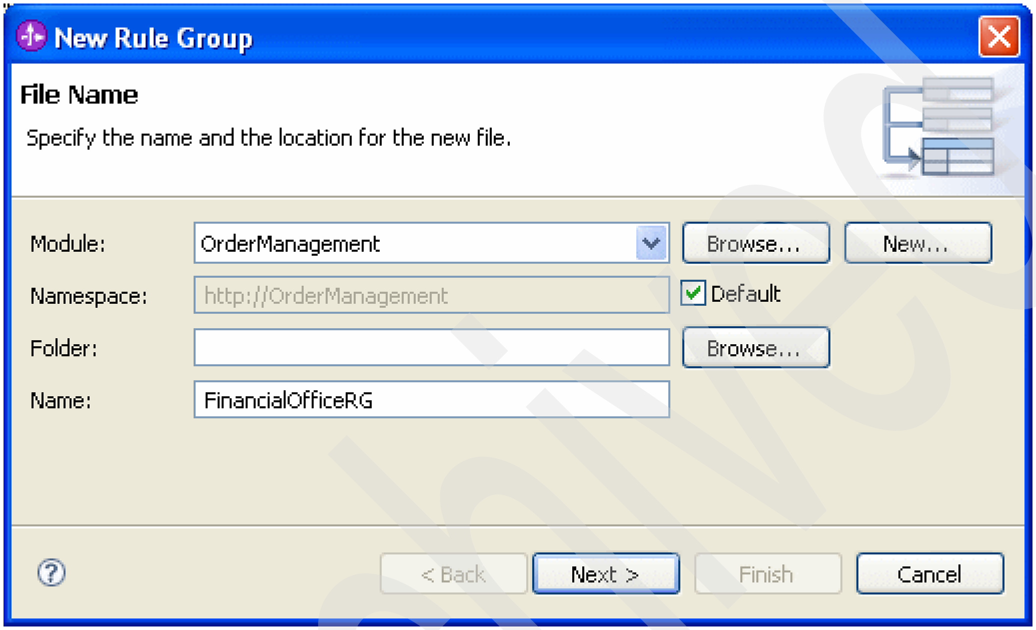


Figure 3-76 Create new rule group FinancialOfficeRG

3. Select the **FinancialOfficeBRIF** interface, and then click **Finish**.

   The FinancialOfficeRG opens up with the following error (as shown in Figure 3-77):

   ```
   The definition of the authorize operation is missing the rule
   destination.
   ```

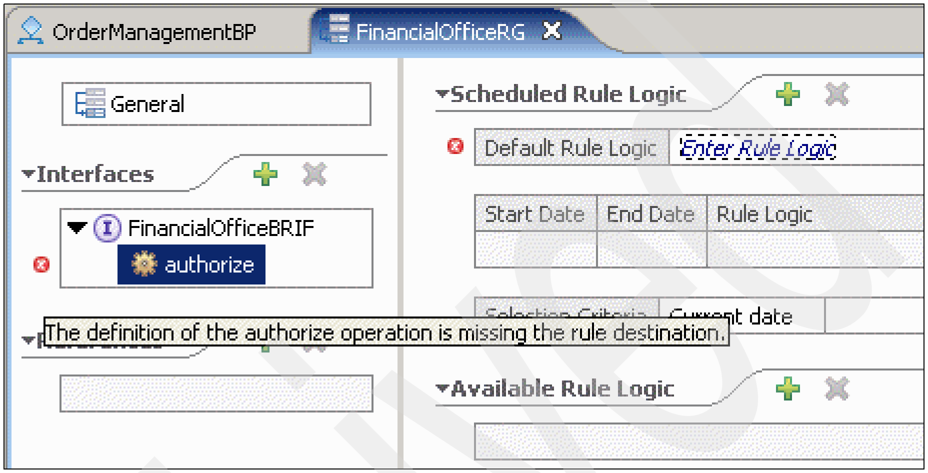


Figure 3-77 Rule group error

> **Note:** A rule group needs one default destination defined, which is why you see the error. In the next step, we create a rule set.

4. Click **Enter Rule Logic** and select **New rule set**. Then, enter the rule name as `defaultFinancialOfficeRS` and accept the other defaults. Click **Finish.**

   > **Tip:** A best practice when naming rule sets is to prepend the name of the rule set with *default*. Then, you can recognize during runtime which of your rule set is your default rule set.

   The defaultFinancialOfficeRS opens and shows an error because a rule set must implement at least one rule.

5. Switch to the FinancialOfficeRG rule group and save it. The error on the rule group should go away.

Creating the rules

To create the rules in the rule set:

1. Go back to the defaultFinancialOfficeRS.
2. In the Rules section, click **Add Action Rule**.

> **Tip:** When you have non-overlapping rule conditions in your rule set, you can set your rules for the “otherwise” condition (see Table 3-2 on page 125) as the first rule in the set followed by your if cases. In our scenario, we test for accept, then reject. Otherwise, it is a human decision. The rule set sets the return code to HUMAN automatically. Then, the if-then rules check conditions to determine if the return code needs to be changed to APPROVE or REJECT.

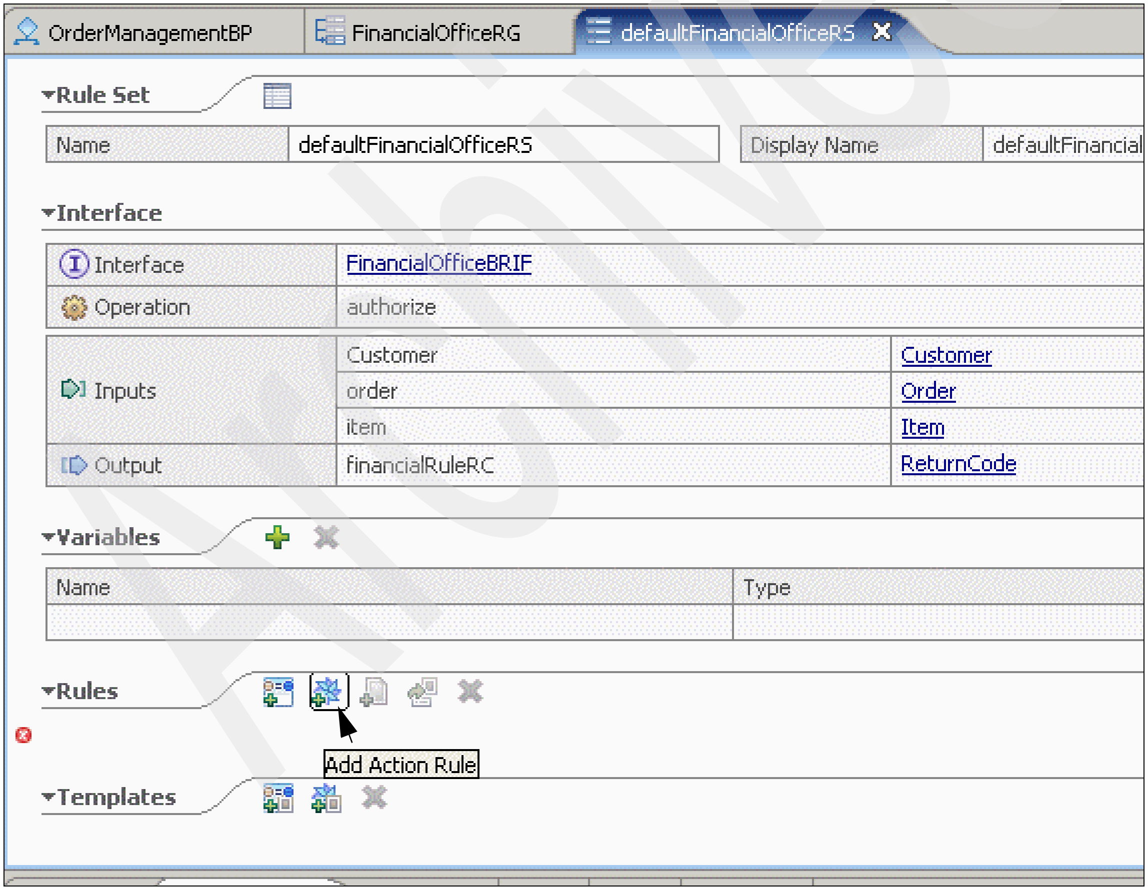


Figure 3-78 Add an action rule to the rule set

3. Add the action rule with the settings shown in Figure 3-79.

   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.

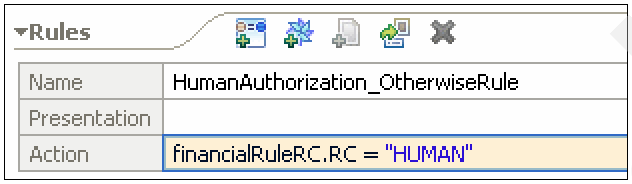


Figure 3-79 Human authorization Otherwise rule

4. Add an if-then rule for the *accept* case with the settings as shown in Figure 3-80.

   An if-then rule determines what action is performed based on the condition of the incoming message.

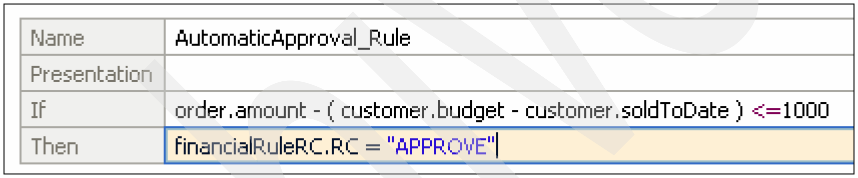


Figure 3-80 Financial approval rule

5. Save the rule set. No errors display in your Problems view, and you have the same five warnings from earlier.
6. Now that the rule set has no errors, convert the rule to a template.

   > **Tip:** You can only change parameters included in a template during runtime using the Business Rules Manager.

   Right-click **AutomaticApproval_Rule** and select **Convert Rule to Template**.

7. Set the approval template to the values shown in Figure 3-81.

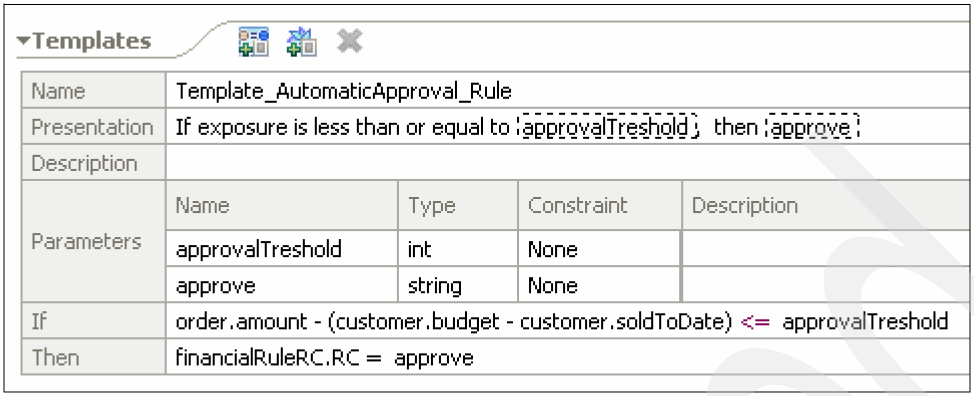


Figure 3-81 Financial approval template

Notice how the associated approval rule changes to reflect the changes in the template, as shown in Figure 3-82.

Figure 3-82 Financial approval rule from template

8. Save the project. No errors display in the Problems view, and you have the same five warnings from earlier.
9. Add an if-then rule for the REJECT case with the settings shown in Figure 3-83.

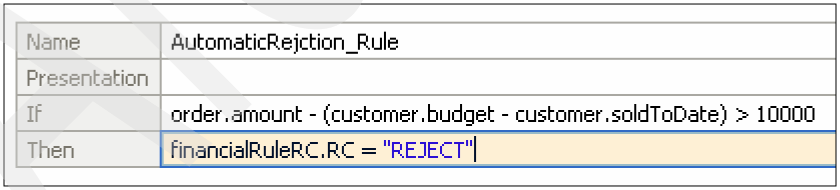


Figure 3-83 Financial rejection rule

10. Save the project. No errors display in the Problems view, and you have the same five warnings from earlier.

11. Convert this rule to a template with the settings shown in Figure 3-84.

Name	Template_AutomaticRejction_Rule			
Presentation	If exposure is greater than {rejectionTreshold}, then {reject}			
Description				
Parameters	Name	Type	Constraint	Description
	rejectionTreshold	int	None	
	reject	string	None	
If	order.amount - (customer.budget - customer.soldToDate) > rejectionTreshold			
Then	financialRuleRC.RC = reject			

Figure 3-84 Financial rejection template

Verify that the rule reflects the template as shown in Figure 3-85.

Name	AutomaticRejction_Rule
Template	Template_AutomaticRejction_Rule
Presentation	If exposure is greater than 10000, then REJECT

Figure 3-85 Financial rejection rule from template

12. Save the Project. No errors display in the Problems view, and you have the same five warnings from earlier.

13. Now, add the new rule to the assembly diagram (Figure 3-86):

    a. Open the OrderManagement assembly diagram, and drop the **FinancialOfficeRG** on the canvas.

    b. Right-click **OrderManagementBP** and select **Wire to Existing**.

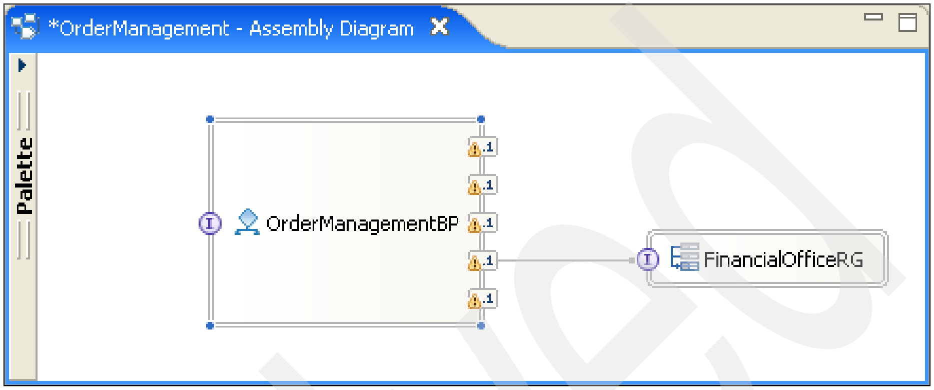


Figure 3-86 Assembly diagram with FinancialOfficeRG

14. Save the Project.

> **Note:** The number of warnings in the Problems View decreases by one. It goes down to four because one out of five of the OrderManagementBP's references is wired.

You have now added the logic around the rules and added implementation of the Financial Rule Group and Rule Set. Now, you can test the financial rules.

Testing the rules

In this section, we explain how to test the financial rules using sample data for the components that you are emulating. Follow these steps:

1. Start the WebSphere Process Server server.
2. Deploy OrderManagementApp to the server (if not already deployed) and make sure that the server is synchronized.
3. Open the OrderManagement assembly diagram.
4. Right-click an open spot in the canvas and select **Test Module**.

5. Verify that you are using the properties shown in Figure 3-87.

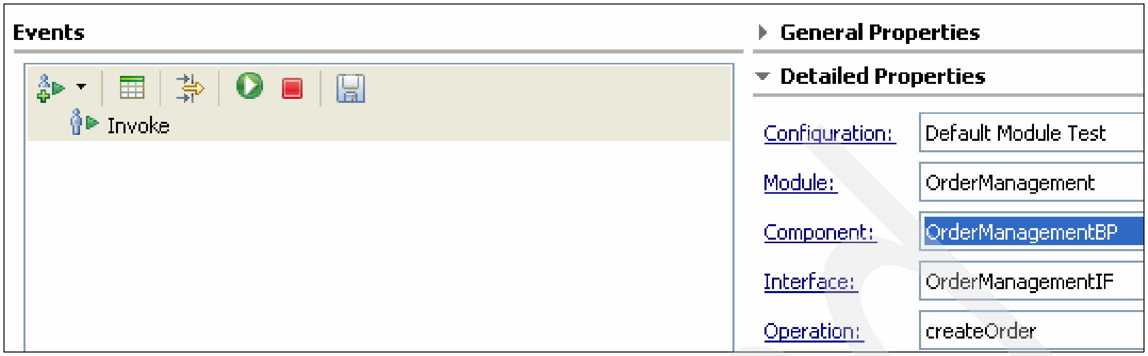


Figure 3-87 Integration test client settings for testing financial rules

6. Verify the emulators in the Configurations tab look as shown in Figure 3-88.

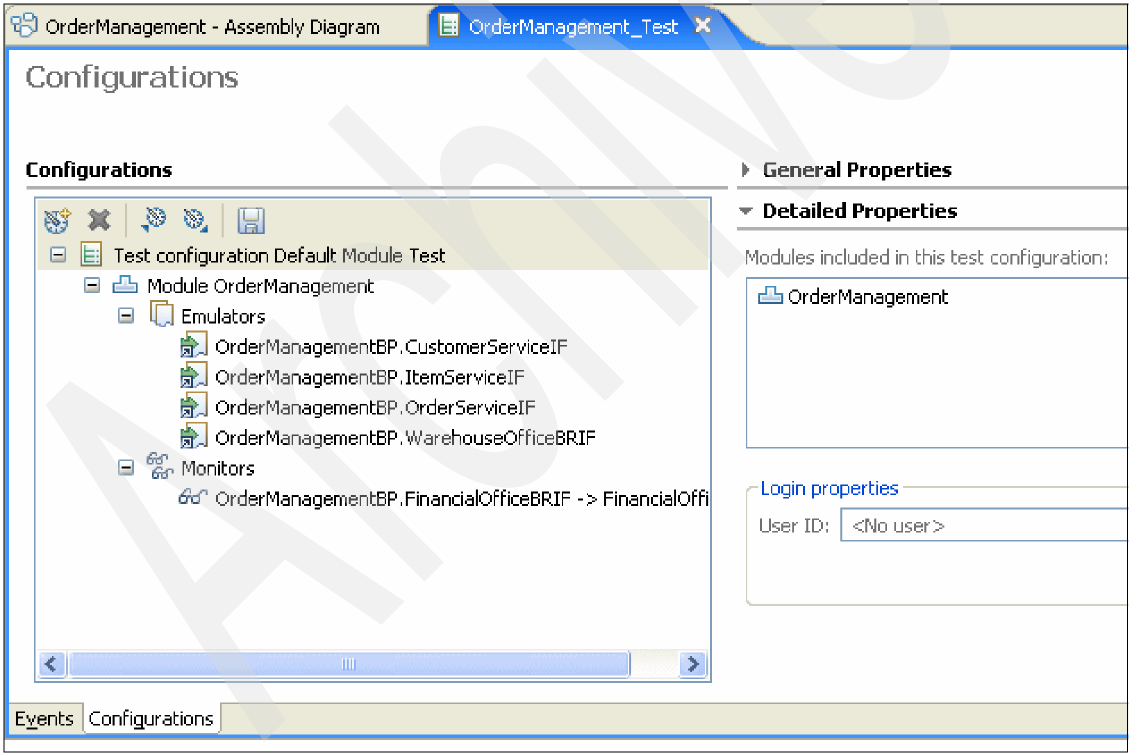


Figure 3-88 Integration test client's configuration settings for testing financial rules

Testing the APPROVE use case

The first test uses values that result in an automatic approval. To run this test:

1. Go back to the Events tab. Use the sample data file orderManagementInput_APPROVE.xml to set the input values as follows:

   a. Right-click **orderManagementInput** and select **Import from XML file**.

   b. Navigate to the sample data and select `orderManagementInput_APPROVE.xml`.Then, click **Open**. The values for the initial request parameters are populated as shown in Figure 3-89.

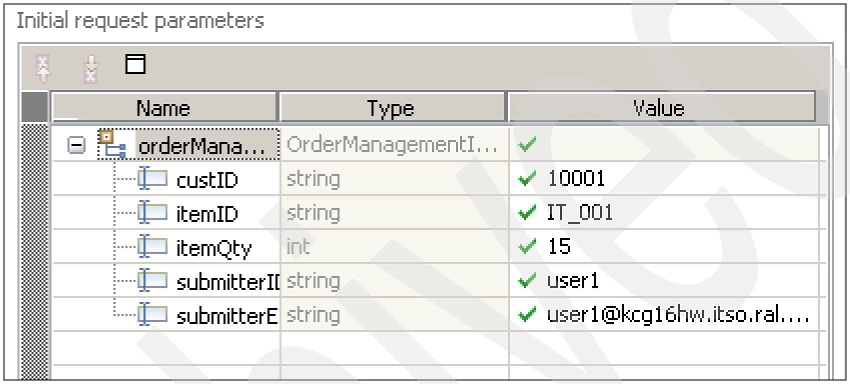


Figure 3-89 Initial request parameters

2. Click **Continue** to start the test.

3. The test stops at two emulation points for customer and item. For the item emulation point:

   a. Set the output value for item using the sample `item001.xml` file.

   b. Set returnCode.RC to `0`.

   c. Click **Continue**.

   Figure 3-90 shows the output values for item.

Name	Type	Value
item	Item	✔
itemID	string	✔ IT_001
itemName	string	✔ Item001
price	int	✔ 15
warehouses	Warehouse[]	✔
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_A
stock	int	✔ 100
indelivery	int	✔ 10
itemQtyPartial	int	✔ 5
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_B
stock	int	✔ 100
indelivery	int	✔ 50
itemQtyPartial	int	✔ 5
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_C
stock	int	✔ 100
indelivery	int	✔ 30
itemQtyPartial	int	✔ 5
returnCode	ReturnCode	✔
RC	string	✔ 0

Figure 3-90 Output values for emulated item and returnCode in APPROVE use case

Now, for customer emulation point:

a. Set the output for the customer using the sample `customer10001.xml` file.

b. Set returnCode.RC to `0`.

c. Click **Continue**.

Figure 3-91 shows the output values for customer.

Name	Type	Value
customer	Customer	
custID	string	10001
custName	string	John Smith Corp.
address	string	2455 South Road
zipCode	string	12601
city	string	Poughkeepsie
state	string	NY
budget	int	50000
soldToDate	int	1000
returnCode	ReturnCode	
RC	string	0

Figure 3-91 Output values for emulated customer and returnCode in APPROVE use case

4. When the emulation stops at order, use the order input value to set the order output value:
   a. Right-click the **order** input value and select **Copy Value**.
   b. Right-click the **order** output value and select **Paste Value**.
   c. Click **Continue**.

   Figure 3-92 shows the output values for order.

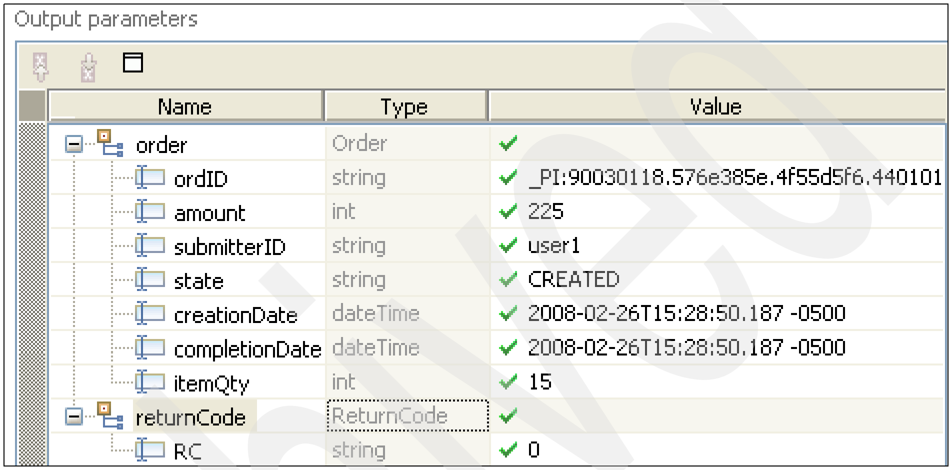


Figure 3-92 Output values for emulated order and returnCode in APPROVE use case

5. The next emulation point is the warehouse rule. (Recall that you have not yet implemented this rule.) You need to:
   a. Set warehouseRuleRC.RC to `APPROVE`.
   b. Click **Continue**.

   Figure 3-93 shows the output values for the warehouse rule.

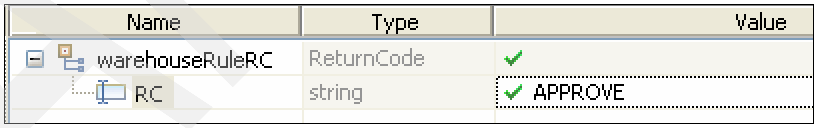


Figure 3-93 Output values for emulated warehouse rule in APPROVE use case

6. Verify the output of the log statements in the Console view, as shown in Figure 3-94.

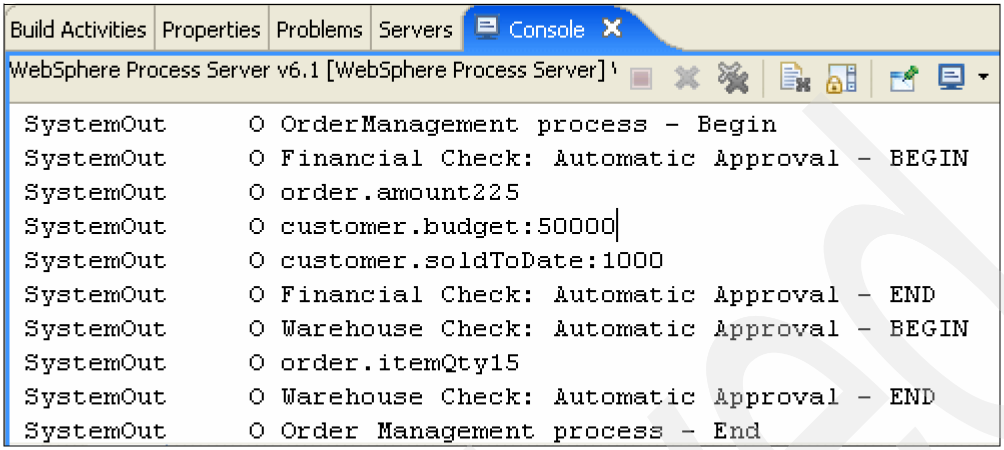


Figure 3-94 Logs of Financial Rule - APPROVE use case in Console

Testing the REJECT use case

Testing the REJECT use case is similar to testing an automatic approval use case. Follow these steps:

1. Use the sample data file orderManagementInput_REJECT.xml to set the input values.
2. Use the customer10002.xml and item002.xml to set the customer and item's output values. Set the return codes to 0.

Figure 3-95 shows the values using the item002.xml sample file.

Name	Type	
item	Item	✔
itemID	string	✔ IT_002
itemName	string	✔ Item002
price	int	✔ 10
warehouses	Warehouse[]	✔
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_A
stock	int	✔ -100
indelivery	int	✔ 45
itemQtyPartial	int	✔ 4
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_B
stock	int	✔ -100
indelivery	int	✔ 50
itemQtyPartial	int	✔ 3
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_C
stock	int	✔ -100
indelivery	int	✔ 5
itemQtyPartial	int	✔ 3
returnCode	ReturnCode	✔
RC	string	✔ 0

Figure 3-95 Output values for emulated item and returnCode in REJECT use case

Figure 3-96 shows the values using the customer10002.xml sample file.

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	✔ 15000
soldToDate	int	✔ 50000
returnCode	ReturnCode	✔
RC	string	✔ 0

Figure 3-96 Output values for emulated customer and returnCode in REJECT use case

3. Set the order output using the copy and paste method, and then set returnCode.RC to `0`.
4. For the warehouse rule, set warehouseRuleRC.RC to `REJECT`.

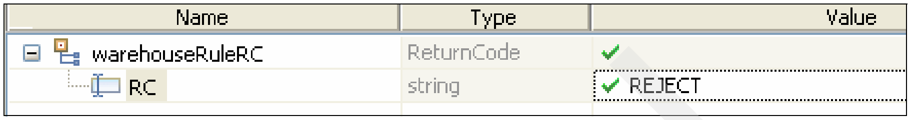


Figure 3-97 Output values for emulated warehouse rule in REJECT use case

5. Verify your output resembles that shown in Figure 3-98.

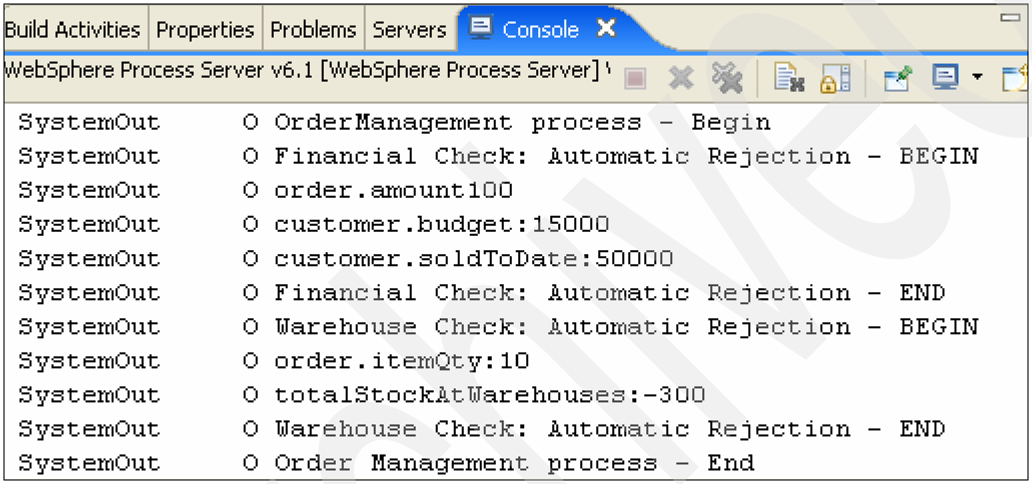


Figure 3-98 Logs of Financial Rule—REJECT use case in Console

Testing the HUMAN DECISION use case

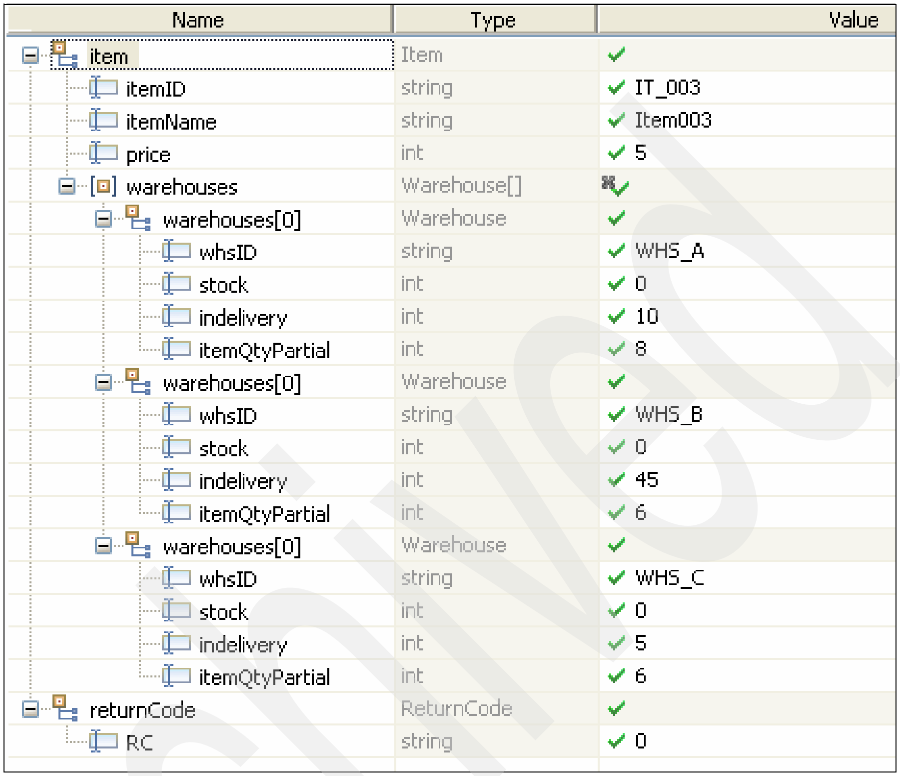
Testing the HUMAN DECISION use case is similar to testing the other use cases. Use these input files and set the order output using the copy and paste method:

1. Use the values in the orderManagementInput_HUMAN.xml sample file for the orderManagementInput (Figure 3-99).

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@itso.com

Figure 3-99 Initial request parameters for orderManagementInput in HUMAN use case

2. Use the values in the item003.xml sample file for the item output (Figure 3-100). Set RC to 0.



Name	Type	Value
item	Item	
itemID	string	IT_003
itemName	string	Item003
price	int	5
warehouses	Warehouse[]	
warehouses[0]	Warehouse	
whsID	string	WHS_A
stock	int	0
indelivery	int	10
itemQtyPartial	int	8
warehouses[0]	Warehouse	
whsID	string	WHS_B
stock	int	0
indelivery	int	45
itemQtyPartial	int	6
warehouses[0]	Warehouse	
whsID	string	WHS_C
stock	int	0
indelivery	int	5
itemQtyPartial	int	6
returnCode	ReturnCode	
RC	string	0

Figure 3-100 Output values for emulated item and returnCode in HUMAN use case

3. Use the values in the customer10003.xml sample file for the customer output (Figure 3-101). Set RC to 0.

Name	Type	Value
customer	Customer	✓
custID	string	✓ 10003
custName	string	✓ YXZ itso Corp.
address	string	✓ 4205 S. Miami Blvd
zipCode	string	✓ 27709
city	string	✓ RTP
state	string	✓ NC
budget	int	✓ 5000
soldToDate	int	✓ 10000
returnCode	ReturnCode	✓
RC	string	✓ 0

Figure 3-101 Output values for emulated customer and returnCode in HUMAN use case

4. Set the order output using the copy and paste method, and then set returnCode.RC to `0`.
5. For the warehouse rule, set warehouseRuleRC.RC to `HUMAN`, as shown in Figure 3-102.

Name	Type	Value
warehouseRuleRC	ReturnCode	✓
RC	string	✓ HUMAN

Figure 3-102 Output values for emulated warehouse rule in HUMAN use case

6. Verify that your output resembles that shown in Figure 3-103.

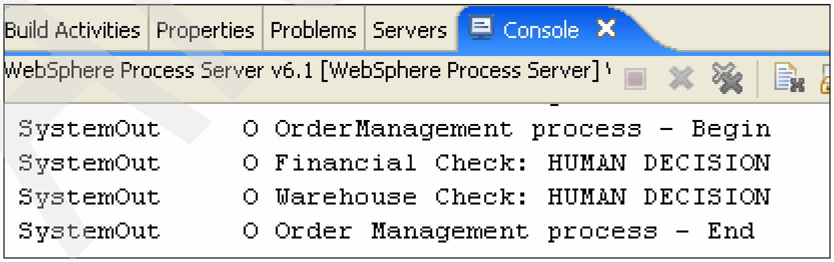


Figure 3-103 Logs of Financial Rule - HUMAN use case in Console

7. You can optionally save your trace file as shown in Figure 3-104.

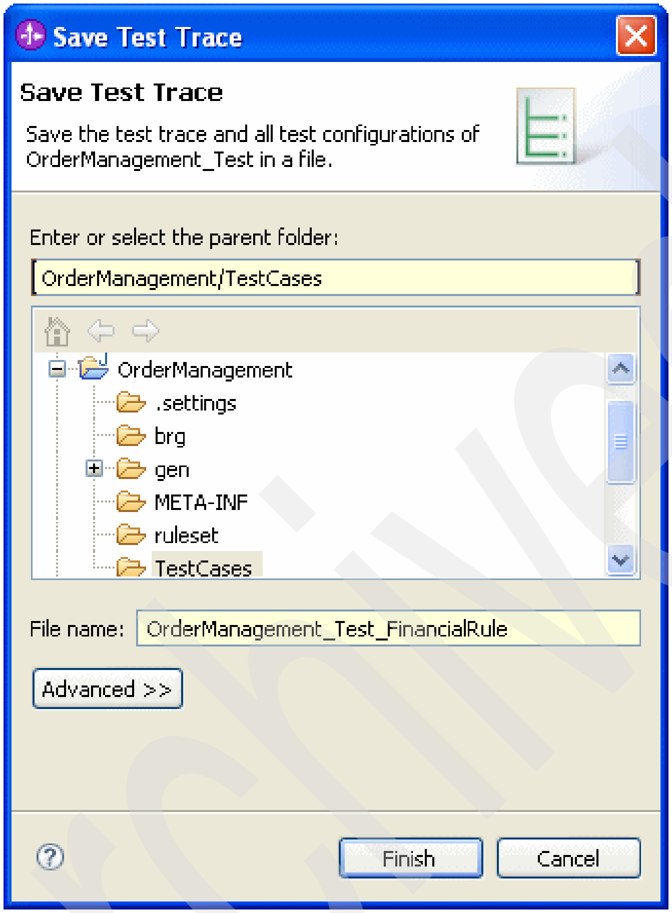


Figure 3-104 Saving financial rule trace file

3.4.3 Implementing the warehouse rule

In this section, we describe how to develop the warehouse rule and to test both rules.

Warehouse rule logic

For each warehouse that fulfills an item, a warehouse check runs based on an item's deficiency. Deficiency is defined as:

deficiency = amount ordered - warehouse's total stock for this item

Table 3-3 summarizes the warehouse check logic.

Table 3-3 Warehouse check logic

Condition	Result
Deficiency <= 10	Automatically accept
Deficiency > 100	Automatically reject
Otherwise	Human Decision

Creating the warehouse rule group and rule set

You begin by creating the Warehouse rule group and creating the associated rule set.

1. Create a new rule group and name the rule group `WarehouseOfficeRG`. Use the `WarehouseOfficeBRIF` interface.
2. Set WarehouseOfficeRG's default rule logic:
   a. Select the authorize operation in the WarehouseOfficeBRIF interface.
   b. Click **Enter Rule Logic** and select **New Rule Set**.
   c. Name the new rule set `defaultWarehouseOfficeRS`.
   d. Save **WarehouseOfficeRG**, as shown in Figure 3-105.

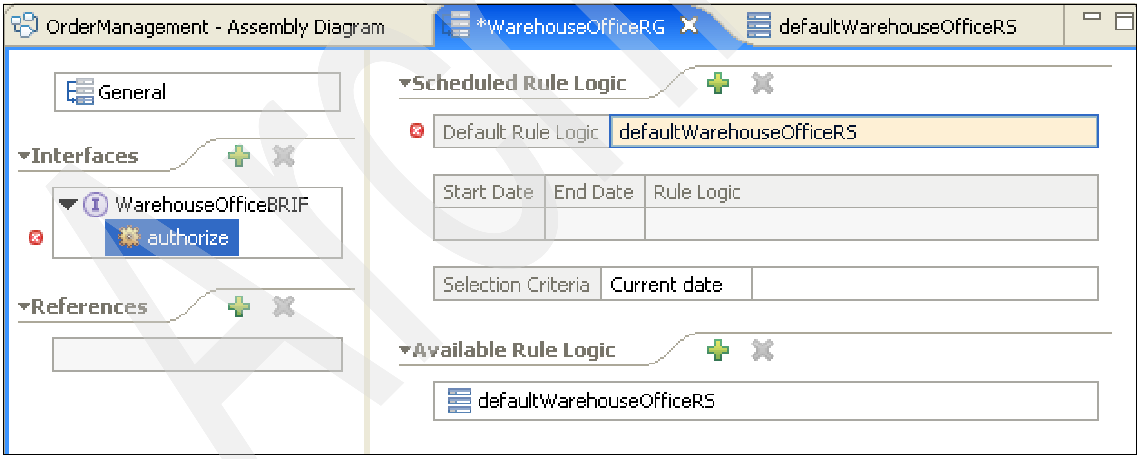


Figure 3-105 WarehouseOfficeRG default rule logic

3. The newly create defaultWarehouseOfficeRS opens and looks similar to that shown in Figure 3-106.

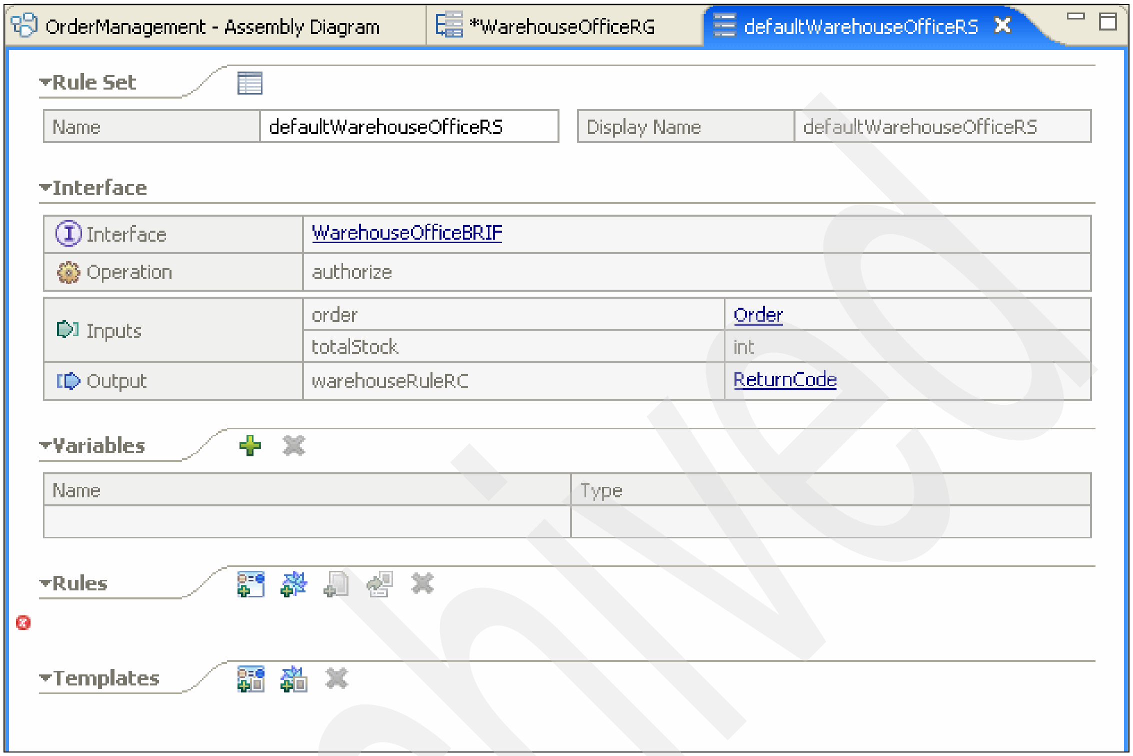


Figure 3-106 defaultWarehouseOfficeRS default values

4. Save the Project. No errors display in the Problems view, and you have the same warnings that you saw earlier.

Creating the rules

The rule set was created as part of the rule group. To add the rules to this rule set:

1. Add the action rule for the *otherwise* case (see Table 3-3 on page 145) with the settings shown in Figure 3-107.

Rules

Name	HumanAuthorization_OtherwiseRule
Presentation	
Action	warehouseRuleRC.RC = "HUMAN"

Figure 3-107 Warehouse otherwise rule

2. Add an if-then rule for the *accept* case with the settings shown in Figure 3-108.

Name	AutomaticApproval_Rule
Presentation	
If	order.itemQty - totalStock <=10
Then	warehouseRuleRC.RC = "APPROVE"

Figure 3-108 Warehouse approval rule

3. Save the Project. No errors display in the Problems view, and you have the same warnings that you saw earlier.
4. Now that the rule has no errors, convert the rule to a template. Right-click the **AutomaticApproval_Rule** → **Convert Rule to Template**.
5. Set the approval template to the values shown in Figure 3-109.

Name	Template_AutomaticApproval_Rule			
Presentation	If deficiency less than or equal to {approvalTreshold} then {approve}			
Description				
Parameters	Name	Type	Constraint	Description
	approvalTreshold	int	None	
	approve	string	None	
If	order.itemQty - totalStock <= approvalTreshold			
Then	warehouseRuleRC.RC = approve			

Figure 3-109 Warehouse approval template

6. Verify that the rule reflects the template, as shown in Figure 3-110.

Name	AutomaticApproval_Rule
Template	Template_AutomaticApproval_Rule
Presentation	If deficiency less than or equal to 10, then APPROVE

Figure 3-110 Warehouse approval rule from template

7. Save the Project. No errors display in the Problems view, and you have the same warnings that you saw earlier.
8. Add an if-then rule for the REJECT case with the settings shown in Figure 3-111.

Name	AutomaticRejection_Rule
Presentation	
If	order.itemQty - totalStock > 100
Then	warehouseRuleRC.RC = "REJECT"

Figure 3-111 Warehouse rejection rule

9. Save the Project. No errors display in the Problems view, and you have the same warnings that you saw earlier.
10. Now that the rule has no errors, convert the rule to a template. Right-click the **AutomaticApproval_Rule** → **Convert Rule to Template**.
11. Set the rejection template to the values shown in Figure 3-112.

Name	Template_AutomaticRejection_Rule			
Presentation	If deficiency is greater than rejectionTreshold, then reject			
Description				
Parameters	Name	Type	Constraint	Description
	rejectionTreshold	int	None	
	reject	string	None	
If	order.itemQty - totalStock > rejectionTreshold			
Then	warehouseRuleRC.RC = reject			

Figure 3-112 Warehouse rejection template

12. Save the Project. No errors display in the Problems view, and you have the same warnings that you saw earlier.

13. Verify your warehouse rules as shown in Figure 3-113.

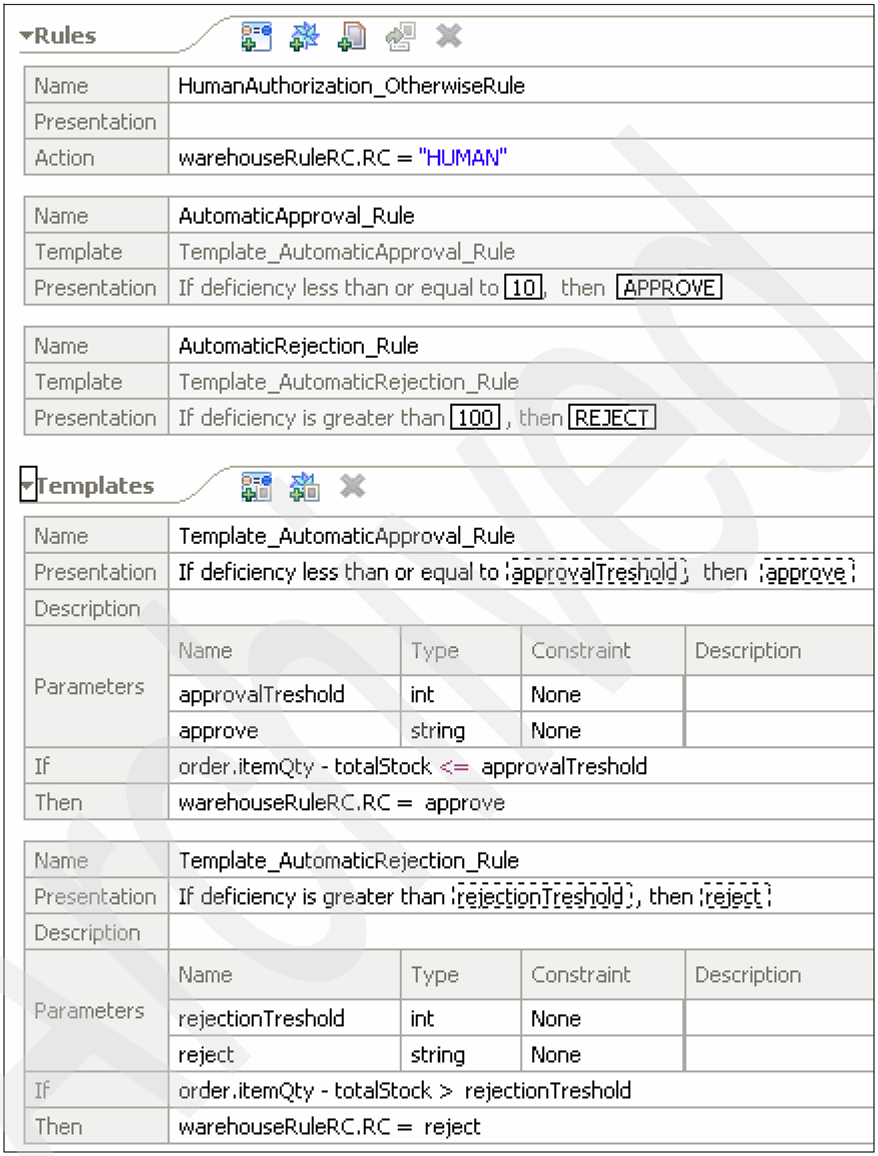


Figure 3-113 Warehouse rules

14. Now, add the WarehouseOfficeRG to the assembly diagram:

a. Open the OrderManagement assembly diagram and drag the WarehouseOfficeRG onto the assembly diagram.

b. Wire the OrderManagementBP and the WarehouseOfficeRG as shown in Figure 3-114.

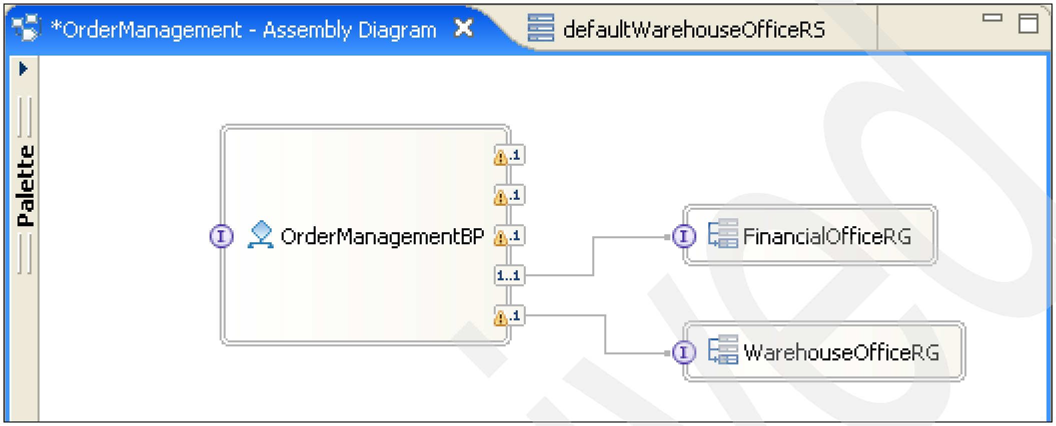


Figure 3-114 OrderManagement assembly diagram with OrderManagementBP and two rule groups

15. Save the Project. The number of warnings in the Problems view decreases by one, and you now have three warnings as shown in Figure 3-115.

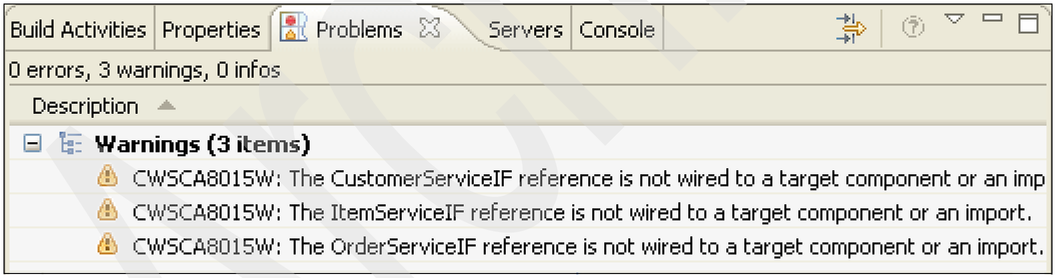


Figure 3-115 Problems view after adding warehouse rule to assembly diagram

Testing the warehouse rules

Now, test the warehouse rules along with the already implemented Financial Rules as follows:

1. Start the WebSphere Process Server server.
2. Deploy OrderManagementApp to the server (if not already deployed) and make sure the server is synchronized.

3. Open the OrderManagement assembly diagram. Right-click an open spot in the canvas and click **Test Module**.
4. Now that the two rules are implemented, your Configuration has only three emulators, as shown in Figure 3-116.

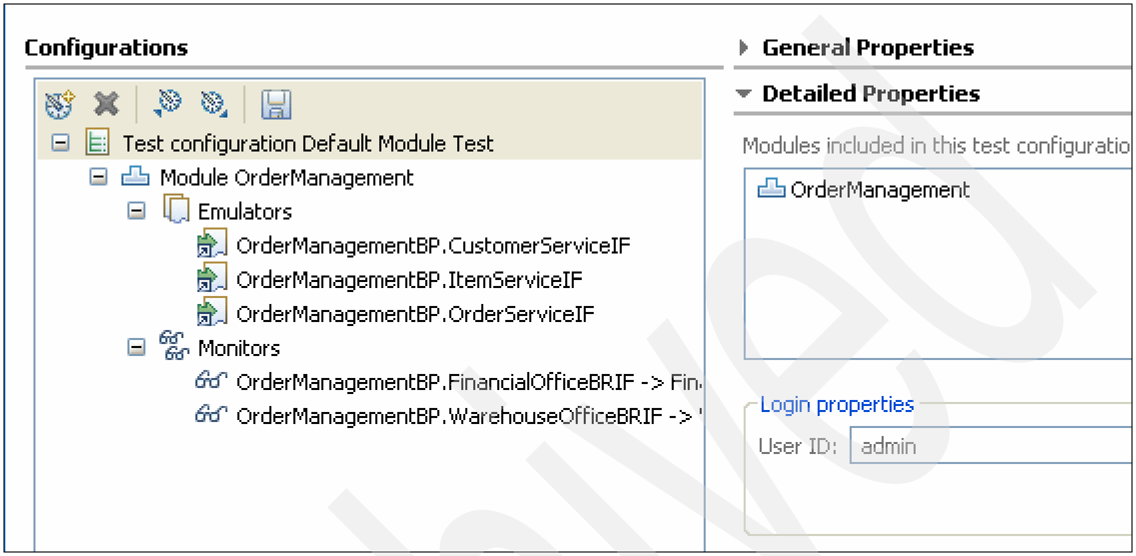


Figure 3-116 Integration test client's configuration settings for testing rules

5. Go back to the Events tab and make sure that you select the test properties shown in Figure 3-117.

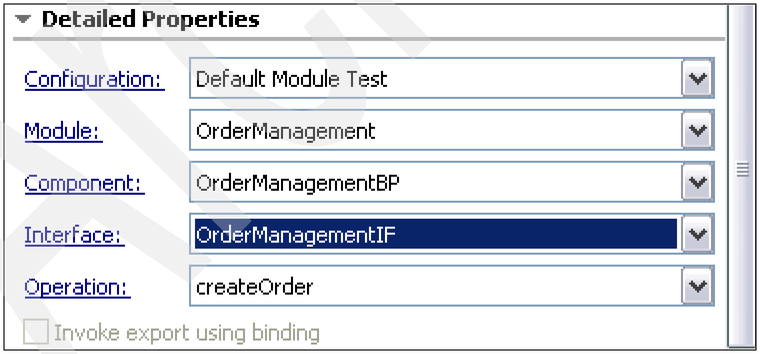


Figure 3-117 Test properties

6. Use the same process and sample data that we describe in “Testing the APPROVE use case” on page 134 for the APPROVE use case, with the following parameters:

   – Initial request parameters: Import from OrderManagementInput_APPROVE.xml.

   – Item output parameters: Import from item001.xml. Set RC=0.

   – Customer output parameters: customer10001.xml. Set RC=0.

   – Order output parameters: copied from Order input parameters. Set RC=0.

   The console output resembles that shown in Figure 3-118.

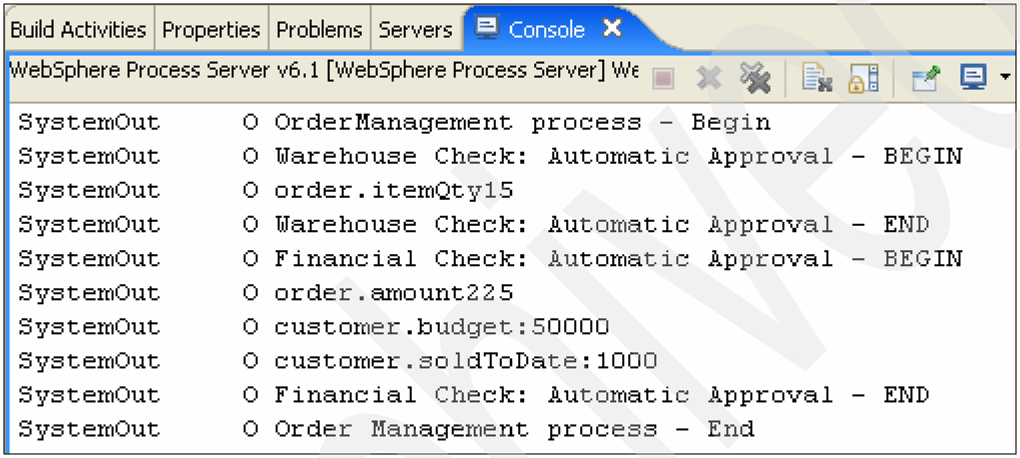


Figure 3-118 Approve use case with both rules implemented

7. Use the same sample data for the REJECT use case, with the following parameters:

   – Initial request parameters: Import from OrderManagementInput_REJECT.xml.

   – Item output parameters: Import from item002.xml. Set RC=0.

   – Customer output parameters: customer10002.xml. Set RC=0.

   – Order output parameters: copied from Order input parameters. Set RC=0.

The console output resembles that shown in Figure 3-119.

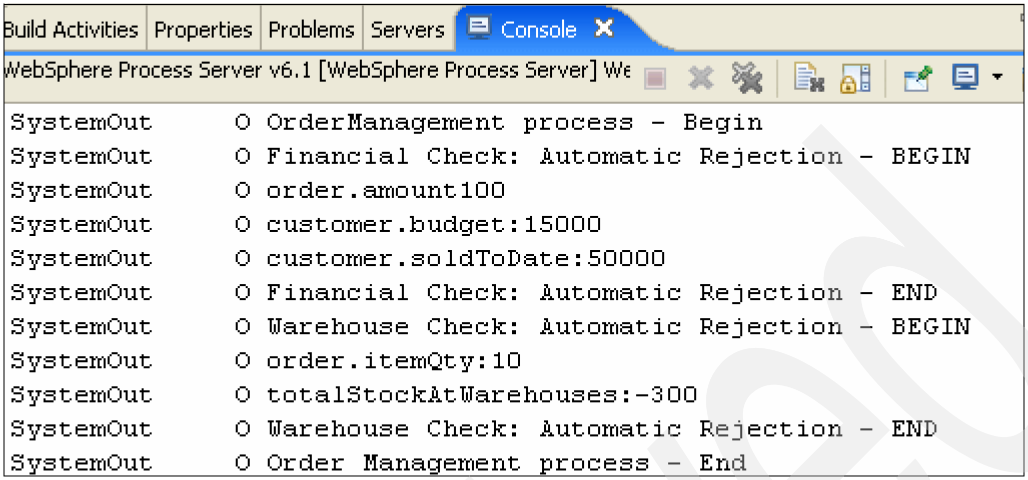


Figure 3-119 Reject use case with both rules implemented

8. Use the same sample data for the HUMAN DECISION use case, with the following parameters:
   - Initial request parameters: Import from OrderManagementInput_HUMAN.xml.
   - Item output parameters: Import from item003.xml. Set RC=0.
   - Customer output parameters: customer10003.xml. Set RC=0.
   - Order output parameters: copied from Order input parameters. Set RC=0.

   The console output resembles that shown in Figure 3-120.

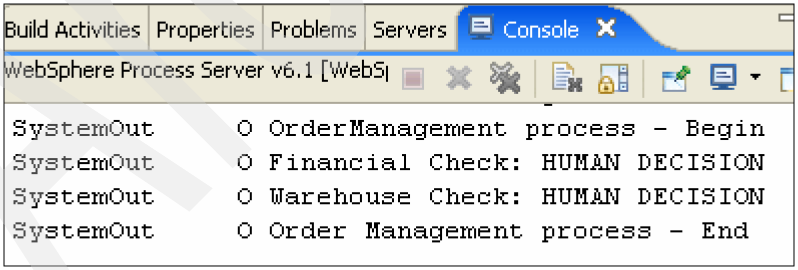


Figure 3-120 Human task use case with both rules implemented

9. You can also step into your Invoke Steps and click the response from the Warehouse Rule to verify your return code from the rule group as shown in Figure 3-121.

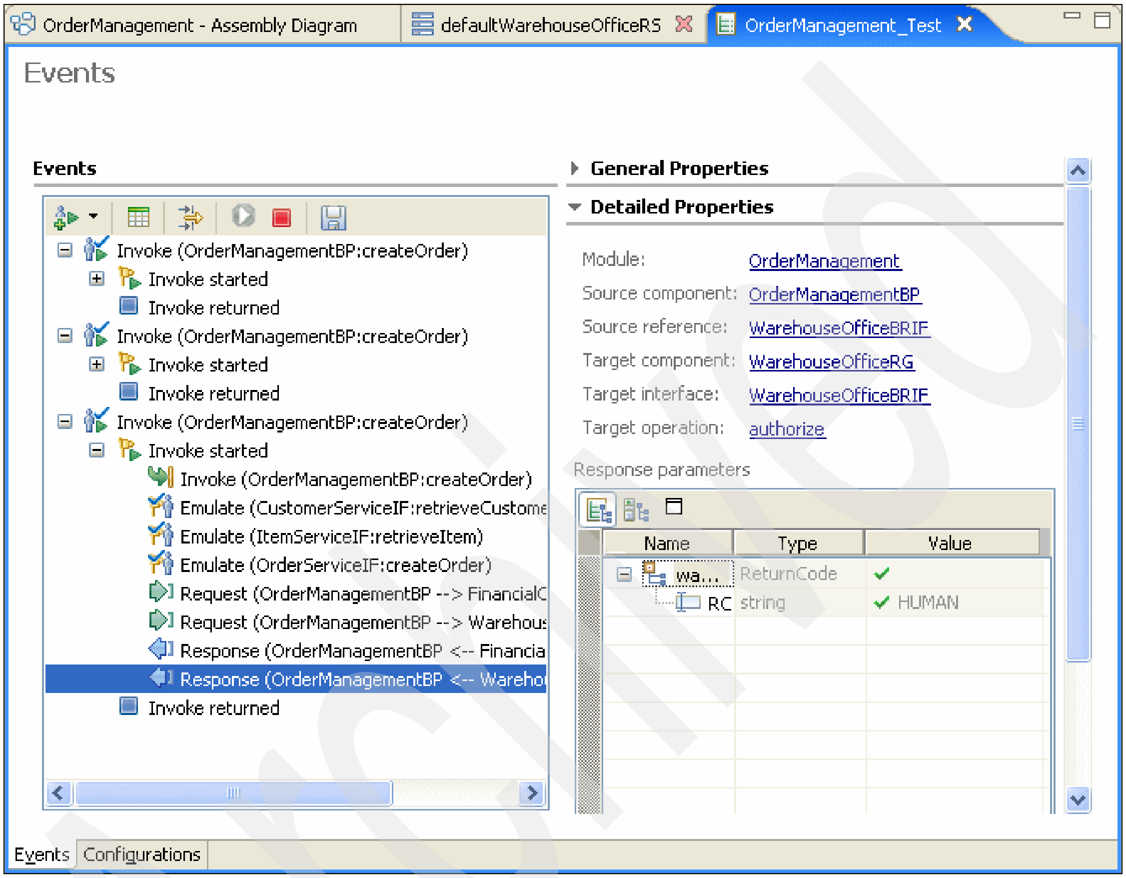


Figure 3-121 Human use case with both rules implemented

10. Optionally you can save the trace file to an execution trace file as shown in Figure 3-122.

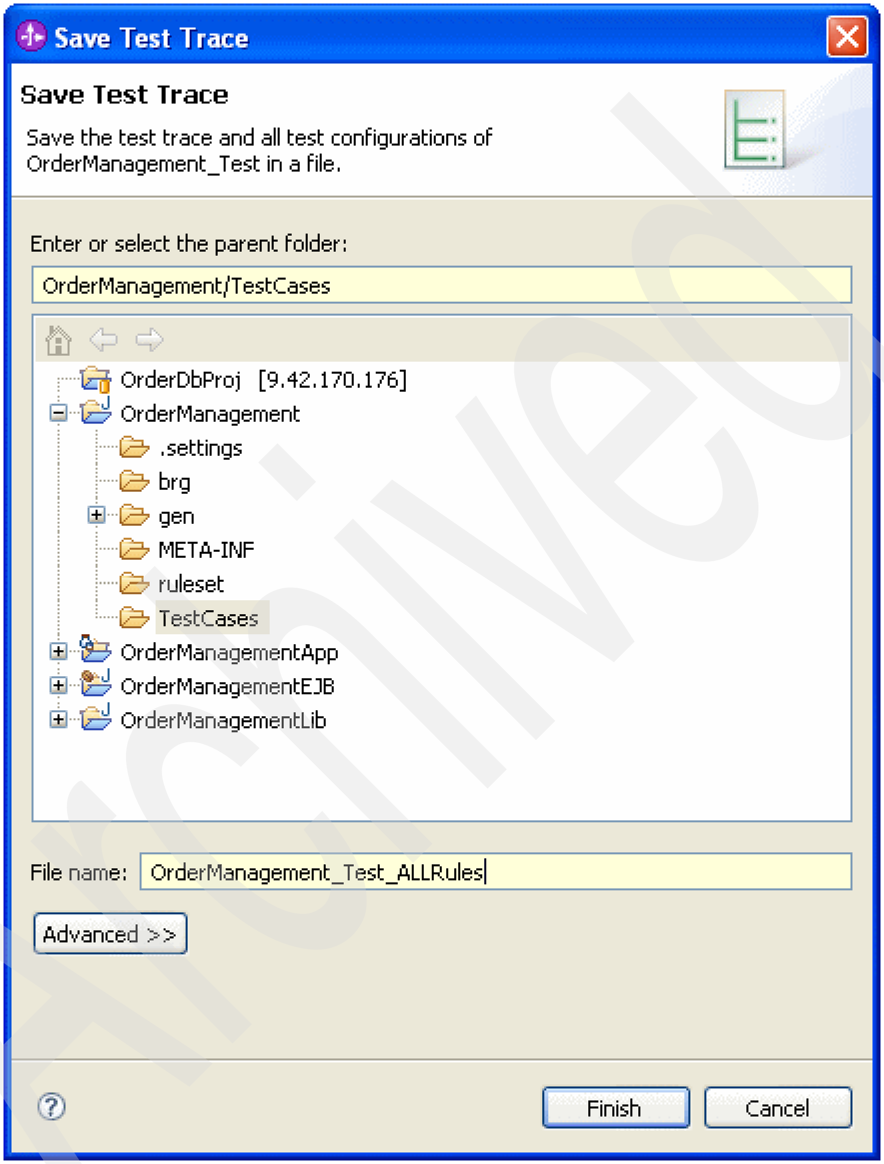


Figure 3-122 OrderManagement_Test_ALLRules execution trace file

3.5 Implementing and testing human tasks

In this section, we build in-line human tasks that allow members of the Financial Office and the Warehouse Office to decide whether to approve or reject orders that cannot be approved automatically.

The approvers need Customer, Order, and Item information to make a decision. This information is encapsulated in the FinancialAuthorizationIF (Figure 3-123) and WarehouseAuthorizationIF interfaces. Both interfaces implement one authorize operation and use the same input and output types.

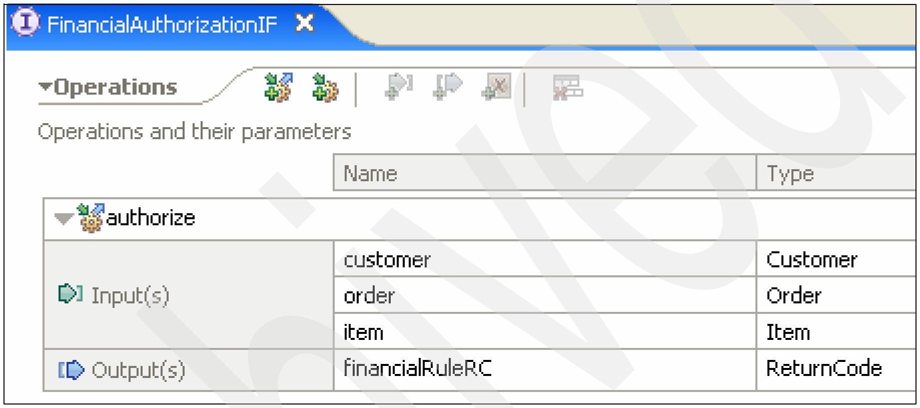


Figure 3-123 FinancialAuthorizationIF details

We discuss first how to replace the Java Snippet `Log - Human Decision` on the financial office path, shown in Figure 3-124, with a human task. We then describe how to make similar changes on the warehouse path.

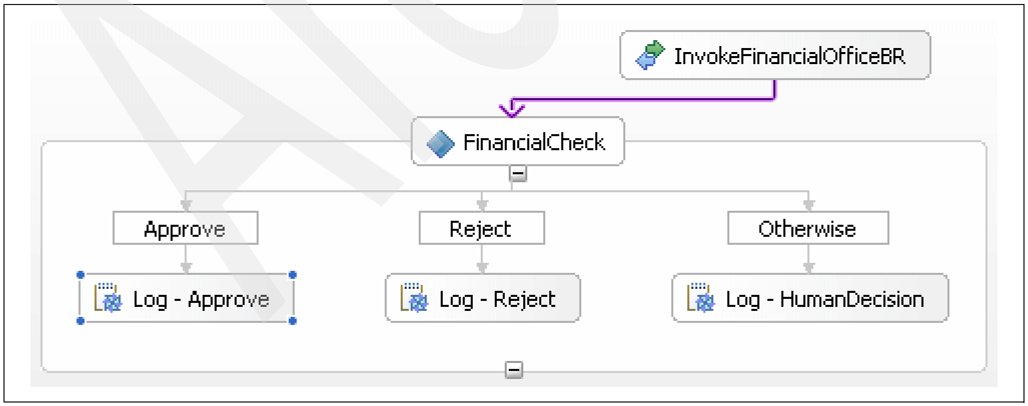


Figure 3-124 Financial path

We implement this step by developing the Financial Office human task and then test it with the pre-configured people directory in the test environment. When this basic test is working, we configure the financial task to use our own group. We then develop the warehouse office task, configure users and groups in the test environment, and finally test the human tasks with the users and groups within the test environment.

3.5.1 Developing the human tasks

To create the Financial Office human task, follow these steps:

1. Open OrderManagementBP. In the FinancialCheck Choice activity, right-click **Log - Human Decision**. Select **Change Type** → **Human Task**.
   a. Rename the human task: `Financial officer authorization`.
   b. Go to the Details tab in the Properties view and click **New** as shown in Figure 3-125.

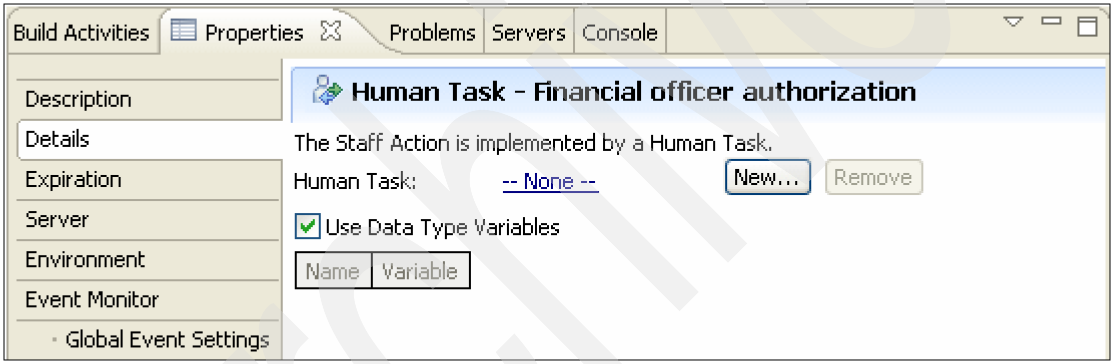


Figure 3-125 Financial officer authorization human task

   c. Select **FinancialAuthorizationIF** as the interface, then click **OK**.

The newly created human task opens. Notice that the People Assignment section lists the potential owners as *Everybody* (Figure 3-126).

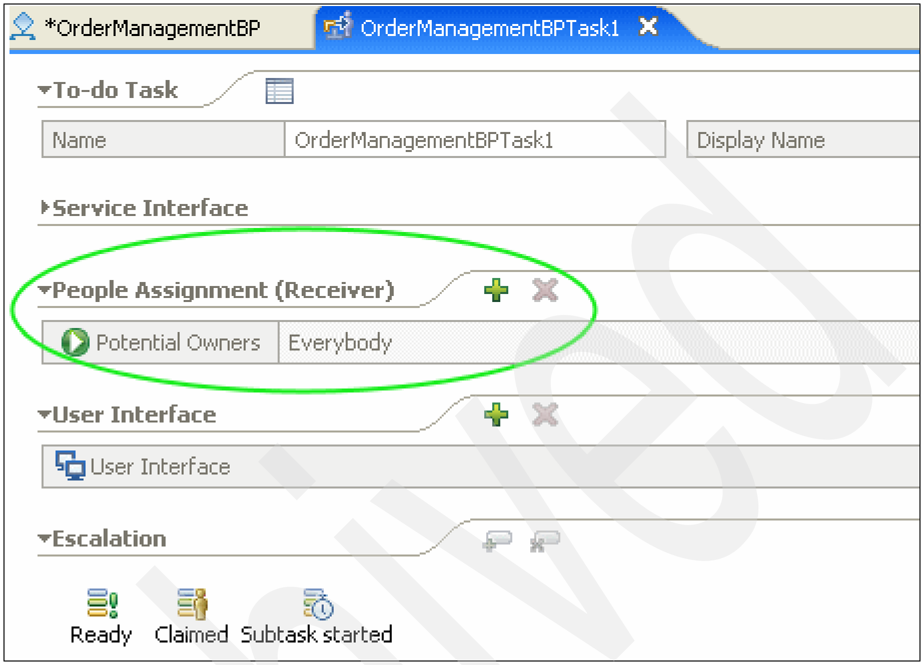


Figure 3-126 Financial task

2. To set the financial task group members:
   a. Click **Everybody**.
   b. In the Properties view, select **Group Members** in the People assignment criteria.

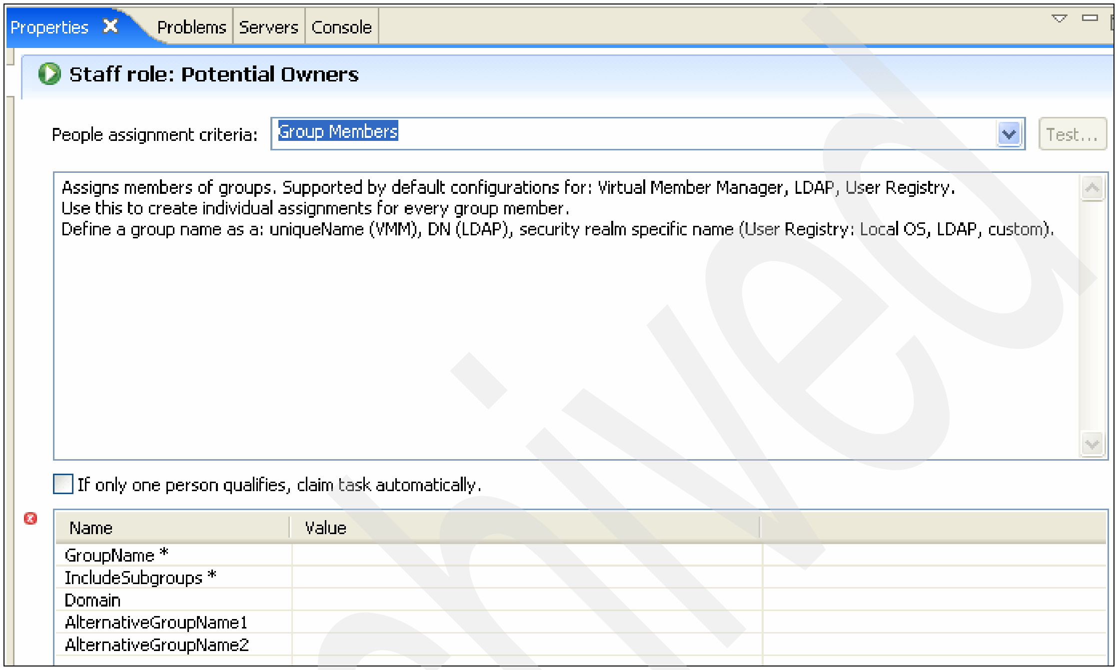
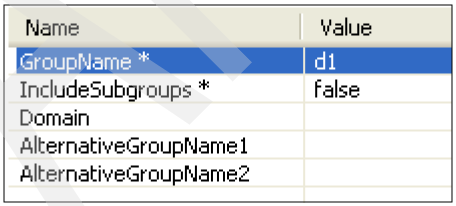


Figure 3-127 Financial task potential owners

3. Set the group name and include subgroups as shown in Figure 3-128. Some groups come pre-configured in the test environment. For this example, we use a pre-configured group called *d1* (department 1).



Name	Value
GroupName *	d1
IncludeSubgroups *	false
Domain	
AlternativeGroupName1	
AlternativeGroupName2	

Figure 3-128 Set financial task's potential owners

Important: If you skip steps, make sure that you come back and change the group name from `d1` to the appropriate financial office group.

4. Save the human task.
5. Go back to the business process and click **Financial officer authorization**.
6. Go to the Details tab of the Properties view and set the following inputs and outputs (as shown in Figure 3-129):
   – customer: `customer`
   – order: `order`
   – item: `item`
   – financialRuleRC: `financialRuleRC`

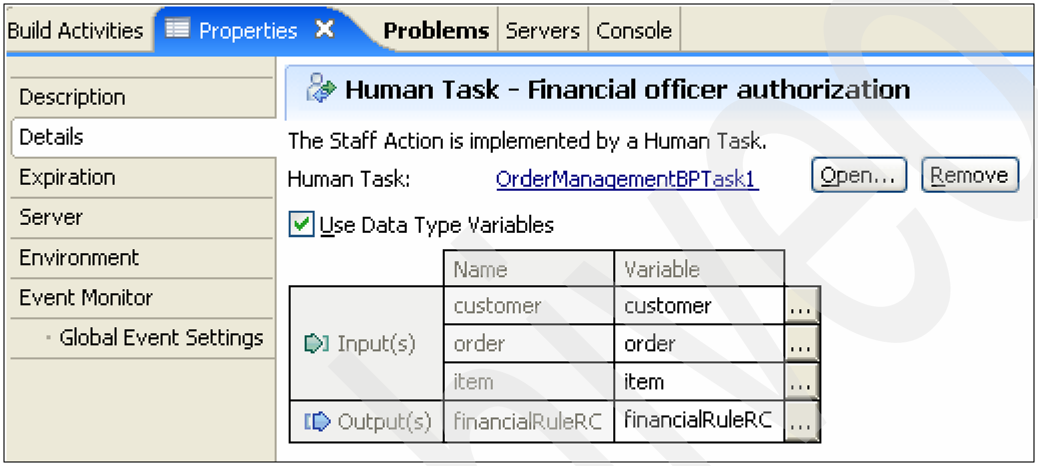


Figure 3-129 Set the financial task interface inputs and outputs

7. Save the business process. Ensure that you have no errors in the Problems view.

3.5.2 Using the pre-configured people directory for testing

To test the new human task using the pre-configured people directory:

1. Start the WebSphere Process Server test environment.
2. In the business process, double-click **Financial officer authorization** to open the OrderManagementBPTask1 human task.
3. Click **Potential Owners**, and then in the Properties view, click **Test**. See Figure 3-130.

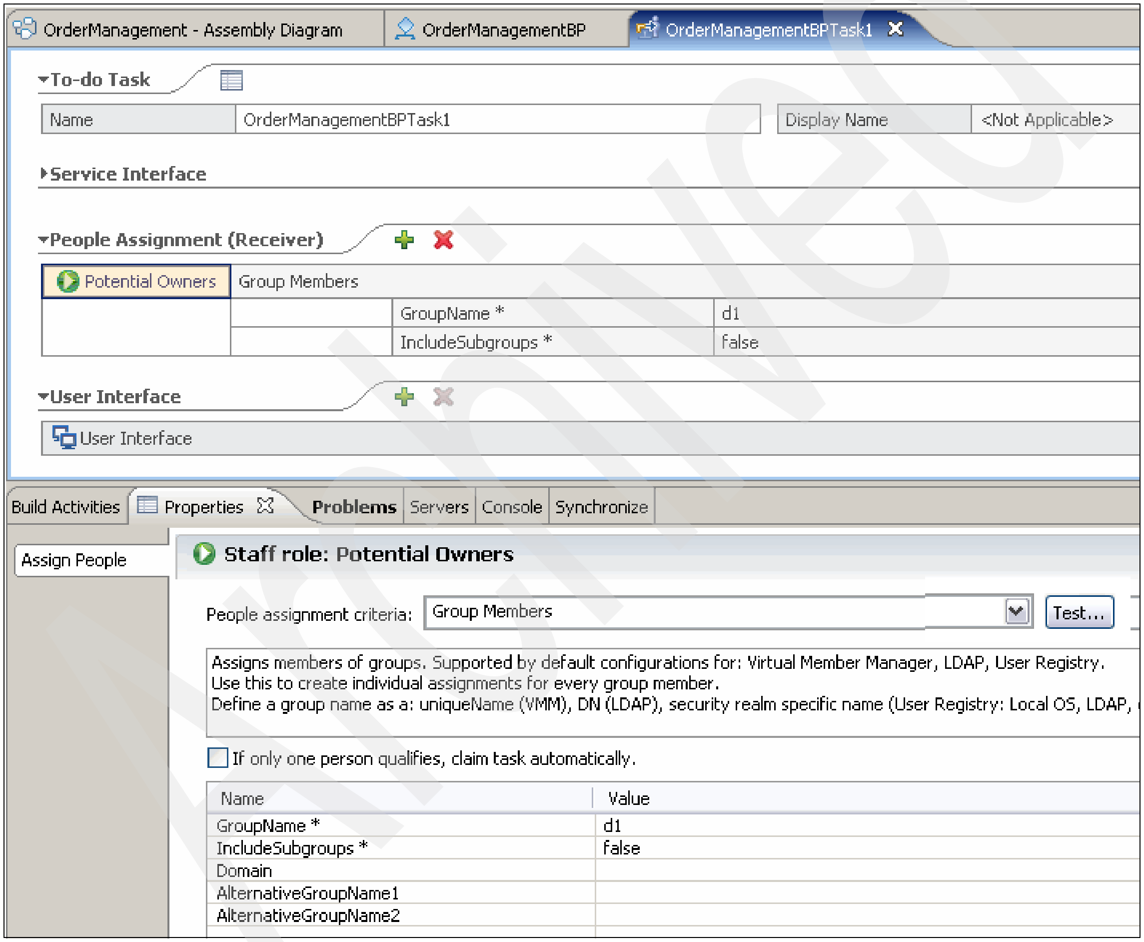


Figure 3-130 Financial task potential owners

4. Select **WebSphere Process Server v6.1** as the server with which to test and click **Submit**.

5. The test returns several users belonging to d1 group as shown in Figure 3-131.

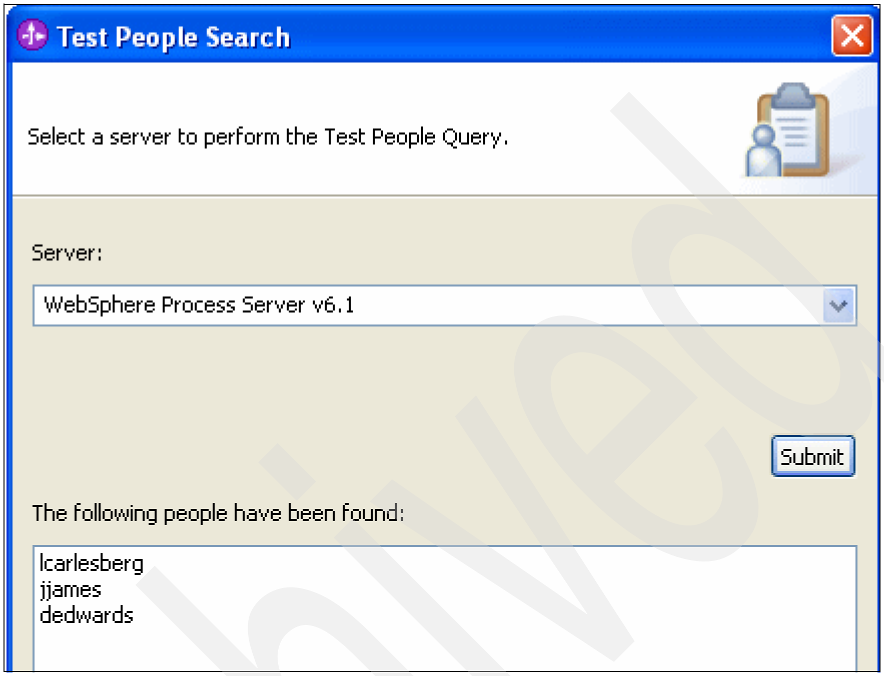


Figure 3-131 The d1 members of pre-configured people directory

Tip: These steps use the WebSphere Integration Developer test environment that is pre-configured with users and groups that you can use during development. You can replace the pre-configured people directory with your own.

6. Click **OK**.
7. Undeploy the module.

Pre-configured people directory

WebSphere Integration Developer's test environment now ships with a pre-configured people directory to allow developers to prototype human tasks quickly. Figure 3-132 shows the structure of this directory.

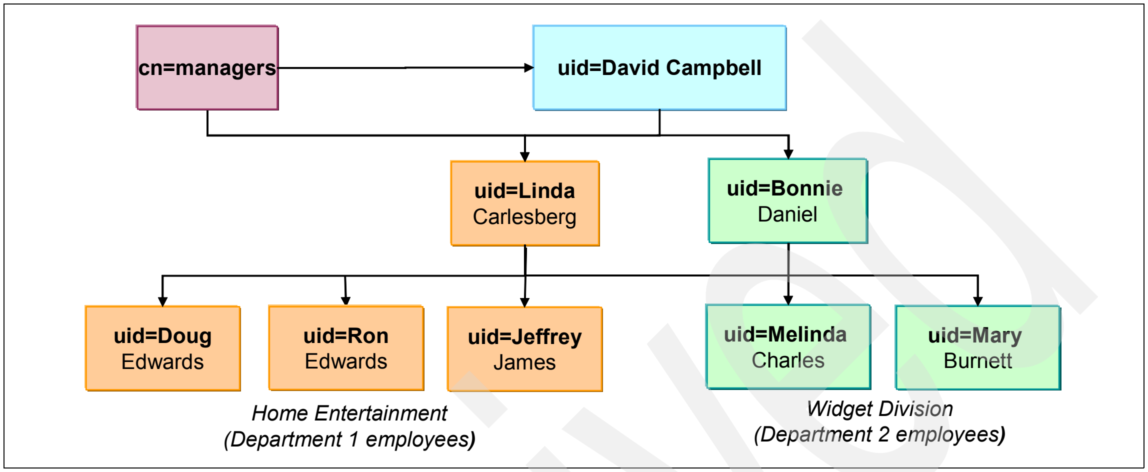


Figure 3-132 Pre-configured people directory

The directory allows developers to test people resolution in the test environment and is built upon Virtual Member Manager's file repository. You can find the file in the following directory:

WID_root\pf\wps\config\cells\widCell\fileRegistry.xml

> **Note:** Although Ron Edwards is depicted in Figure 3-132 as a member of the Home Entertainment group, he is technically not member of any group. He is set up as a substitute for Jeffrey James.

In 3.5.4, "Configuring users and groups in the integrated test environment" on page 168, we show how to create our own users and groups.

3.5.3 Completing the human tasks

Now that you have created the Financial Office human task and tested it with the pre-configured people directory, you need to point the Financial Office task to the correct group and create the Warehouse Office human task. This section describes the steps to complete the human tasks.

Pointing the Financial Office human task to the correct group

To assign the correct group:

1. Open the OrderManagementBPTask1 and select the Potential Owners field in the People Assignment area.
2. In the Properties view set the following values, as shown in Figure 3-133:
   - GroupName: Financial Office
   - IncludeSubgroups: false
3. Save the human task.

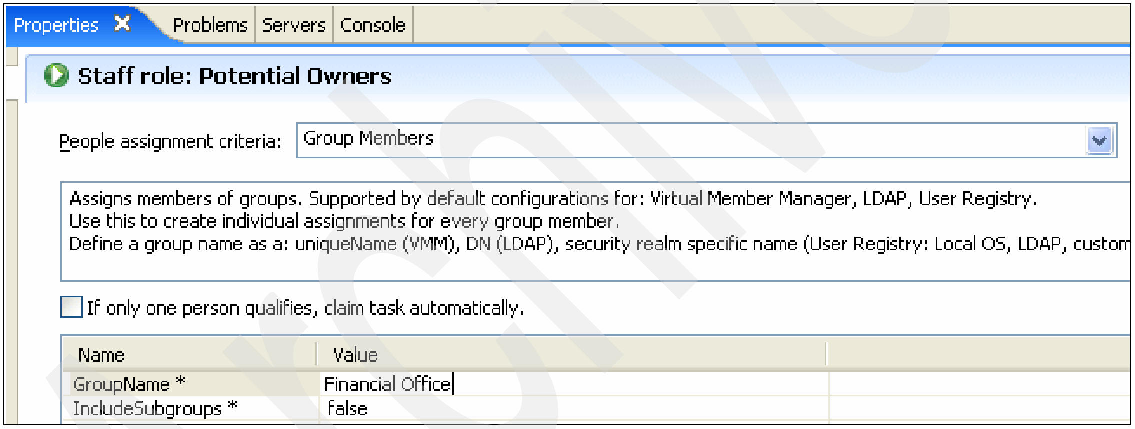


Figure 3-133 Financial task point to corrected group

Creating the warehouse office human task

Now, create the human task for the Warehouse Choice activity:

1. Open the OrderManagementBP business process.
2. Right-click **Log - Human Decision** and select **Change Type** → **Human Task**. Rename the human task: `Warehouse officer authorization`.
3. Go to the Details tab in the Properties view. Click **New** as shown in Figure 3-134.

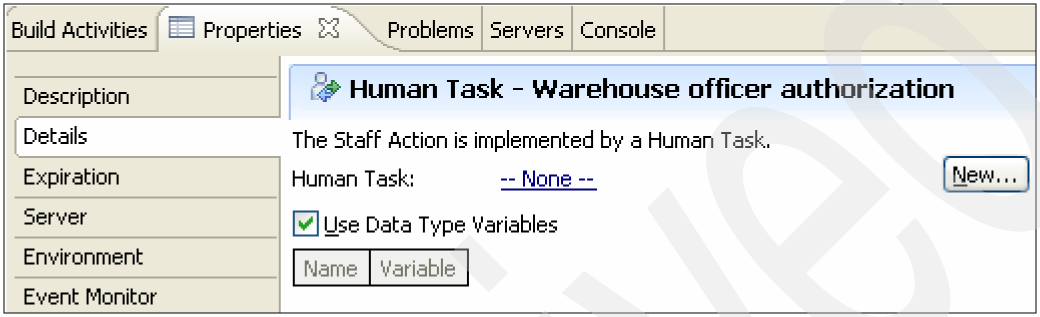


Figure 3-134 In-line warehouse task

4. Select **WarehouseAuthorizationIF**, then click **OK**.

   The new human task open, and as before, the People Assignment section lists the Potential Owners as *Everybody* as shown in Figure 3-135.

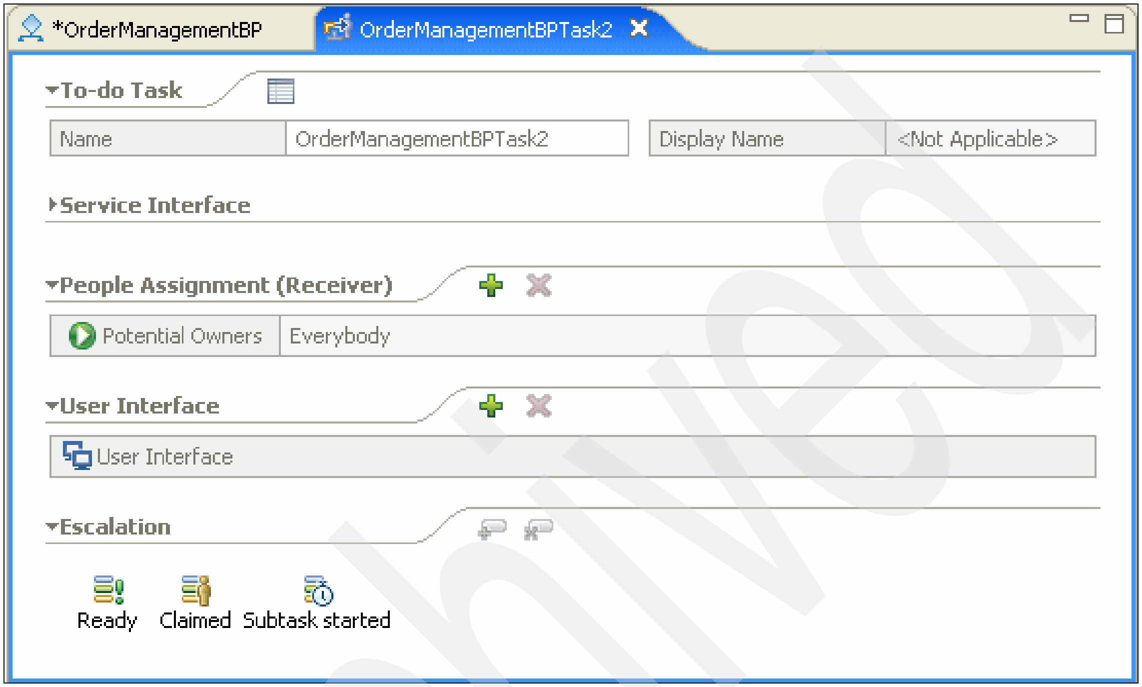


Figure 3-135 Warehouse task

5. Click **Everybody**.

6. Go to Properties view, and select **Group Members** in the People assignment criteria field. Set the group name and include subgroups as shown in Figure 3-136:

   – GroupName: Warehouse Office
   – IncludeSubgroups: `false`

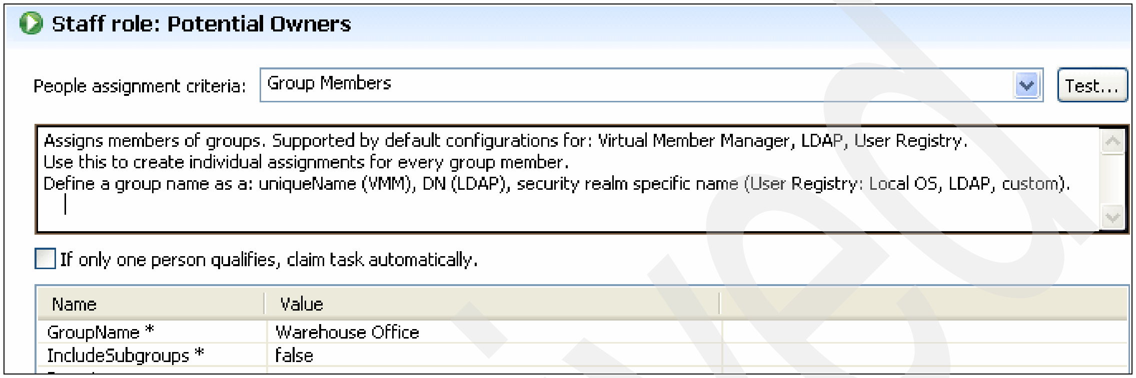


Figure 3-136 Warehouse task's potential owners

7. Save the human task.
8. Go back to the business process and click **Warehouse officer authorization**. In the Properties view and set the inputs and outputs (as shown in Figure 3-137):
   – customer: customer
   – order: order
   – item: item
   – warehouseRuleRC: warehouseRuleRC

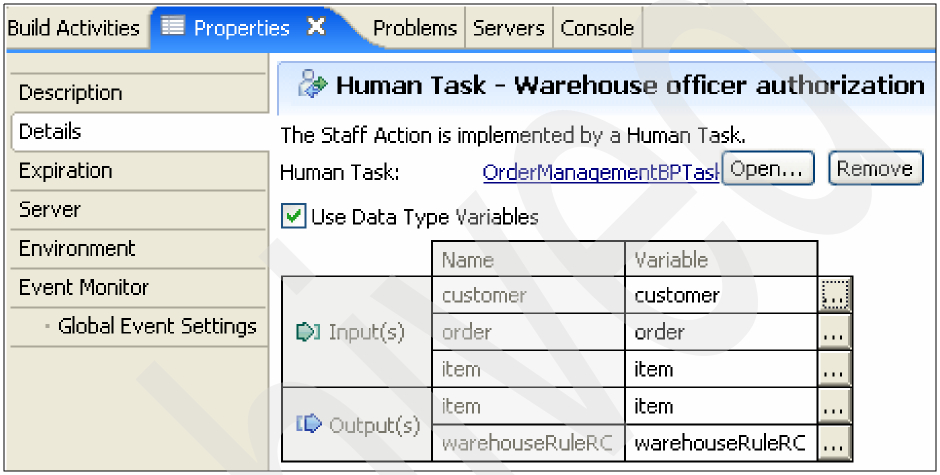


Figure 3-137 Warehouse office interface inputs and outputs

9. Save the business process.

Now that you have created both human tasks, you can configure your own users and groups to represent the Financial Office and Warehouse Office.

3.5.4 Configuring users and groups in the integrated test environment

You create users and groups from the administrative console to represent groups and users for the scenario (see Figure 3-138 on page 169). For this example, we create a main group called *Order Management* that has the following two groups as members:

- Financial Office
- Warehouse Office

We use an additional group called *managers*, to which all managers belong. This group is pre-configured in the test environment.

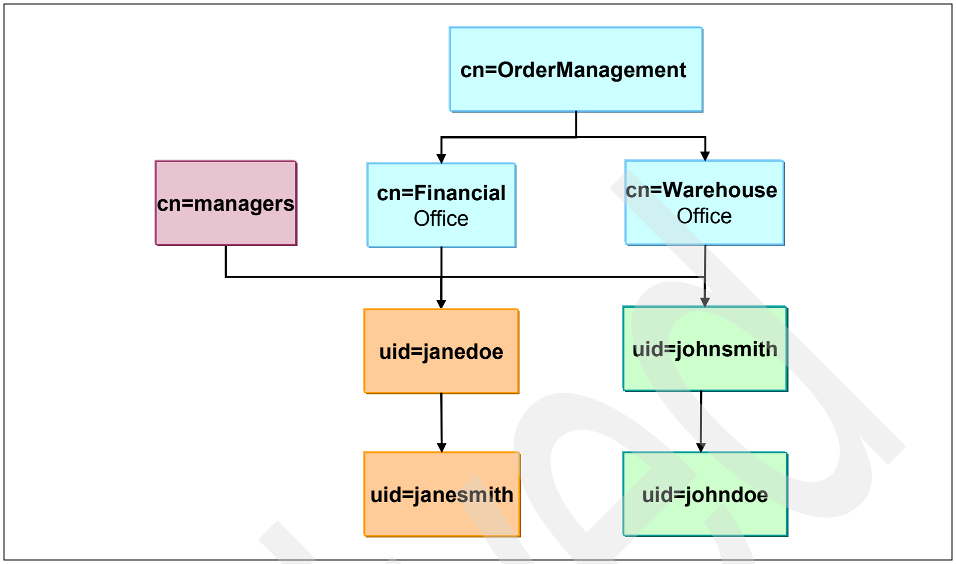


Figure 3-138 OrderManagement's users and groups

The Financial Office has a manager and a regular user who belongs to it. A similar structure exists for the Warehouse Office. It is up to you to add more users to these two groups.

Setting up the groups for the scenario

To set up the groups for the scenario, follow these steps:

1. Start the WebSphere Process Server test environment and open the administrative console. We first look at the existing groups and users that come pre-configured with the test environment.
2. Expand **Users and Groups** and click **Manage Groups**. Click **Search**. You see the three groups that are created already (Figure 3-139 on page 170).

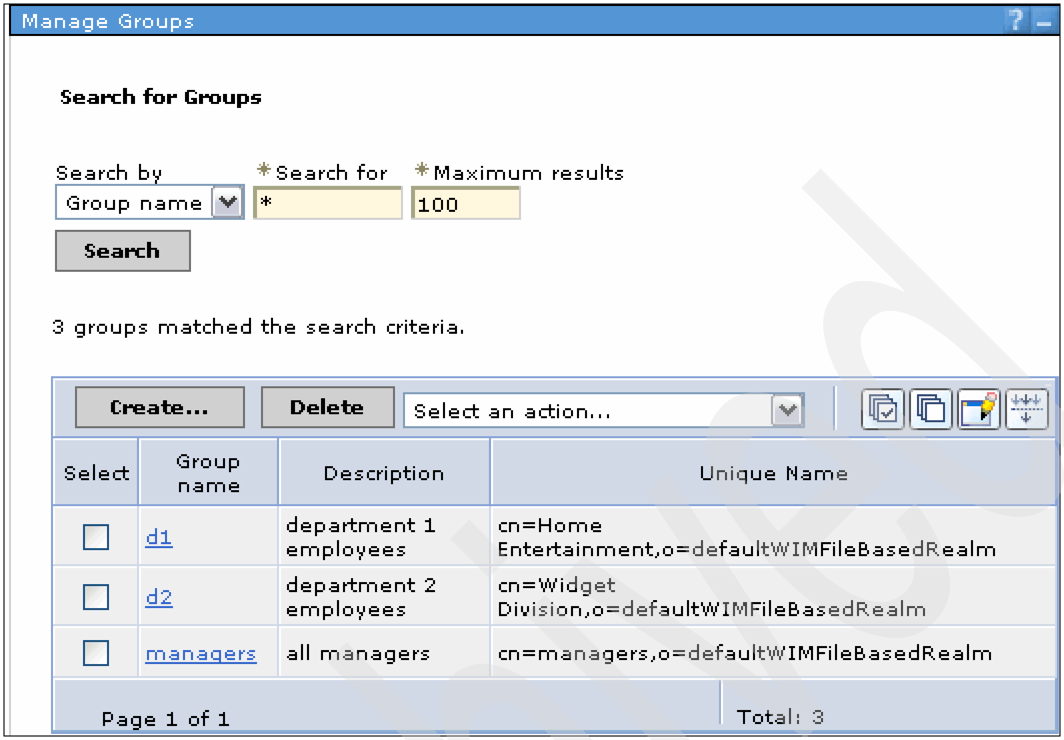


Figure 3-139 Test environment default groups

3. Expand **Users and Groups** and click **Manage Users**. Click **Search** to see the list of existing users (as shown in Figure 3-140).

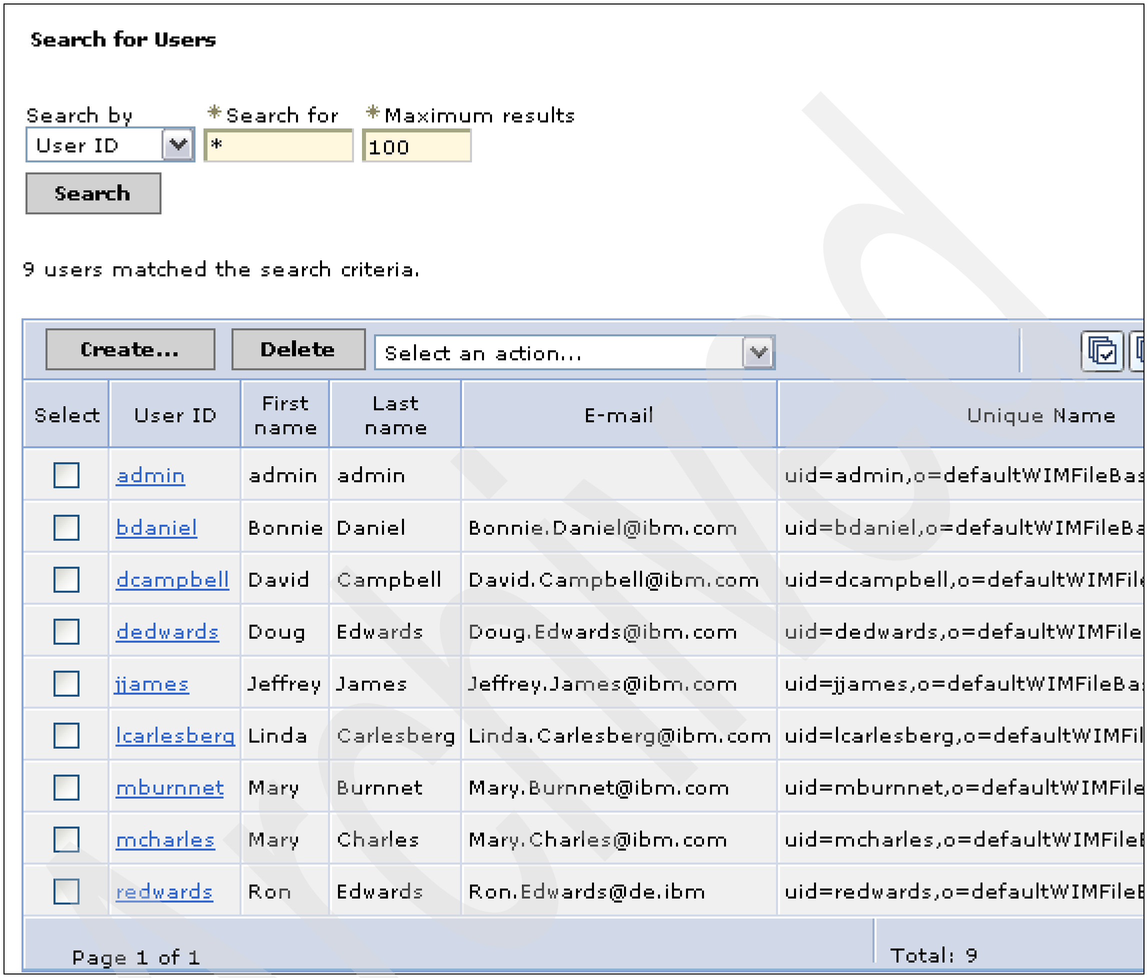


Figure 3-140 Manage Users view in the administrative console

4. Create a new group by going to **Manage Groups** and then click **Create**.
5. Enter the group name `Order Management`, and click **Create** (Figure 3-141).

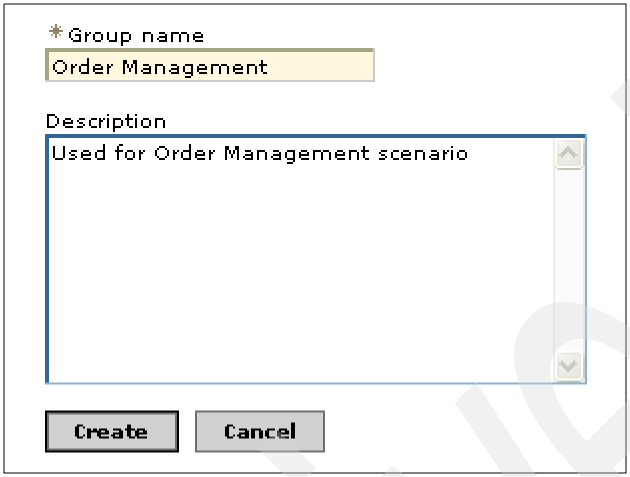
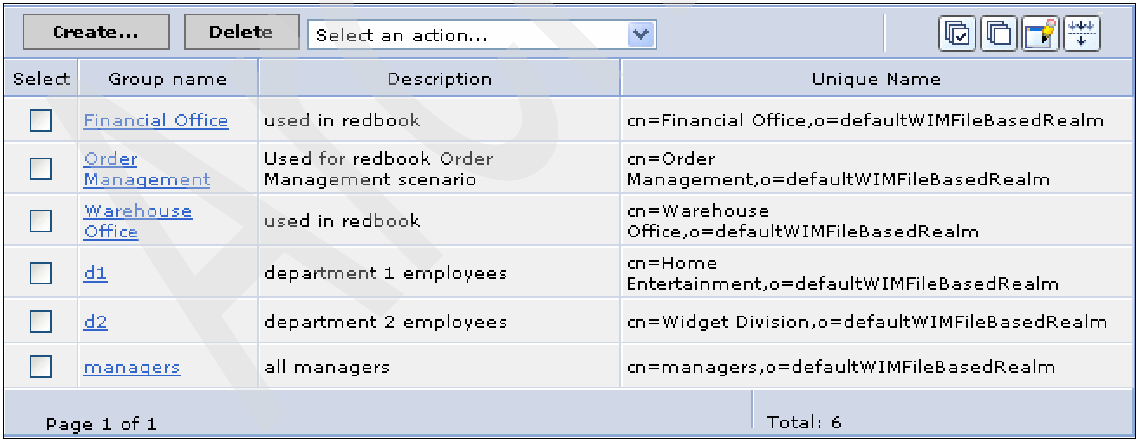


Figure 3-141 Create group Order Management

6. The group is created. A prompt displays that allows you to create another group (**Create Like**) or to close the configuration (**Close**). Use the Create Like option to create the Financial Office and Warehouse Office groups. (On a stand-alone server, you also need to create a *managers* group.).
7. Verify the groups that you created as shown in Figure 3-142.



Create... | Delete | Select an action...

Select	Group name	Description	Unique Name
☐	Financial Office	used in redbook	cn=Financial Office,o=defaultWIMFileBasedRealm
☐	Order Management	Used for redbook Order Management scenario	cn=Order Management,o=defaultWIMFileBasedRealm
☐	Warehouse Office	used in redbook	cn=Warehouse Office,o=defaultWIMFileBasedRealm
☐	d1	department 1 employees	cn=Home Entertainment,o=defaultWIMFileBasedRealm
☐	d2	department 2 employees	cn=Widget Division,o=defaultWIMFileBasedRealm
☐	managers	all managers	cn=managers,o=defaultWIMFileBasedRealm

Page 1 of 1 | Total: 6

Figure 3-142 Groups search results

8. Now, make the Financial Office and the Warehouse Office groups members of the Order Management group. Click the **Order Management** group, then switch to the Members tab. See Figure 3-143.

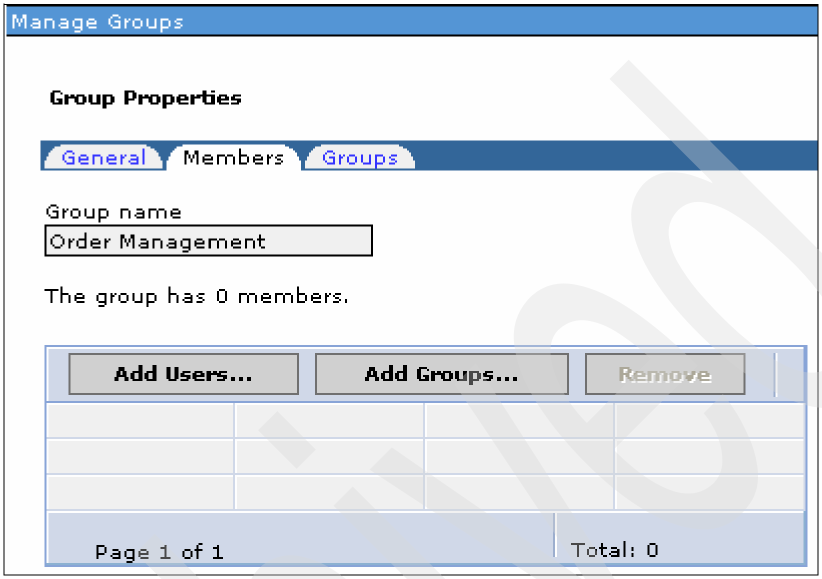


Figure 3-143 Order Management's members

9. Click **Add Groups**, and then click **Search** to see a list of the groups.
10. Select the Financial Office and Warehouse Office (use the Ctrl key to select both), and then click **Add**. (Figure 3-144).

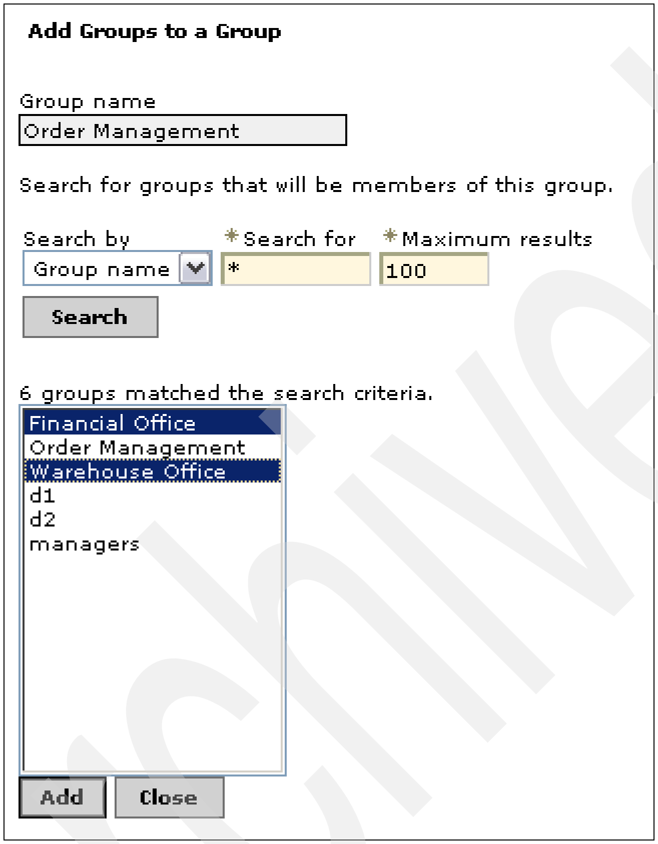


Figure 3-144 Adding groups to a group

11. Close the window and verify the members of the Order Management group as shown in Figure 3-145.

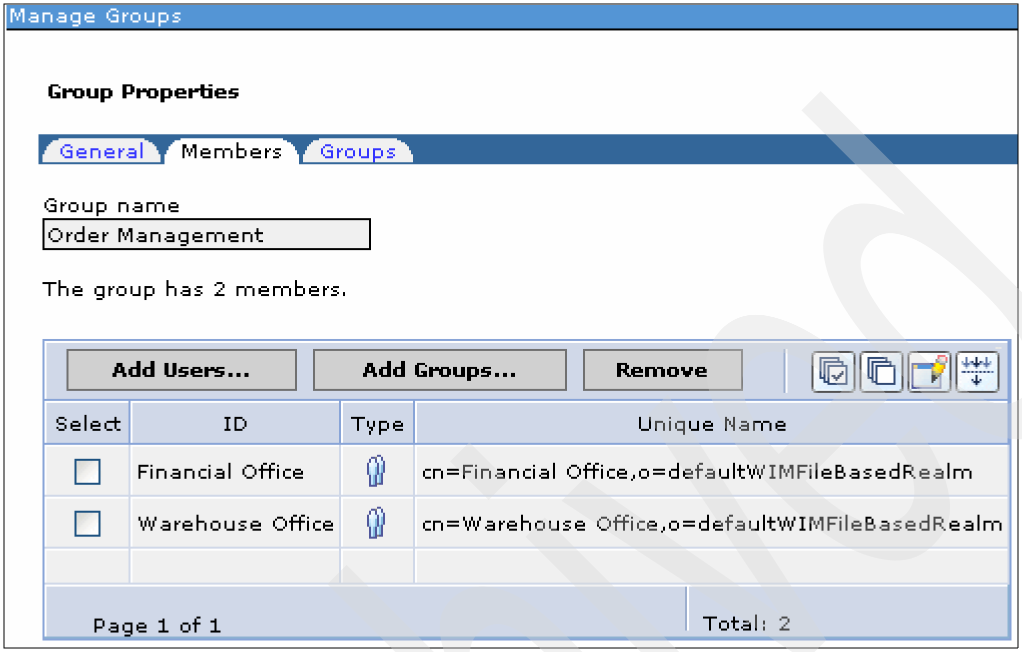


Figure 3-145 Order Management's members

You have finished adding the groups for the scenario and configuring the relationship among the groups.

Setting up the users for the scenario

Now, create four users:

- Two who belong to the Financial Office (a manager and a regular user)
- Two who belong to the Warehouse Office.

Follow these steps:

1. Go to **Manage Users** and then click **Create**.
2. Enter the User ID `janedoe`, and then click **Group Membership** as shown in Figure 3-146 on page 176.

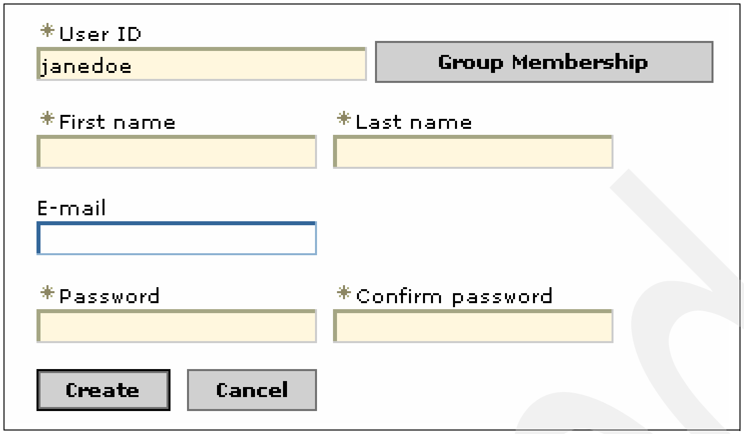


Figure 3-146 Create a user: janedoe

3. Click **Search**.
4. Select **Financial Office** and **managers**, then click **< Add** as shown in Figure 3-147. Then, click **Close**.

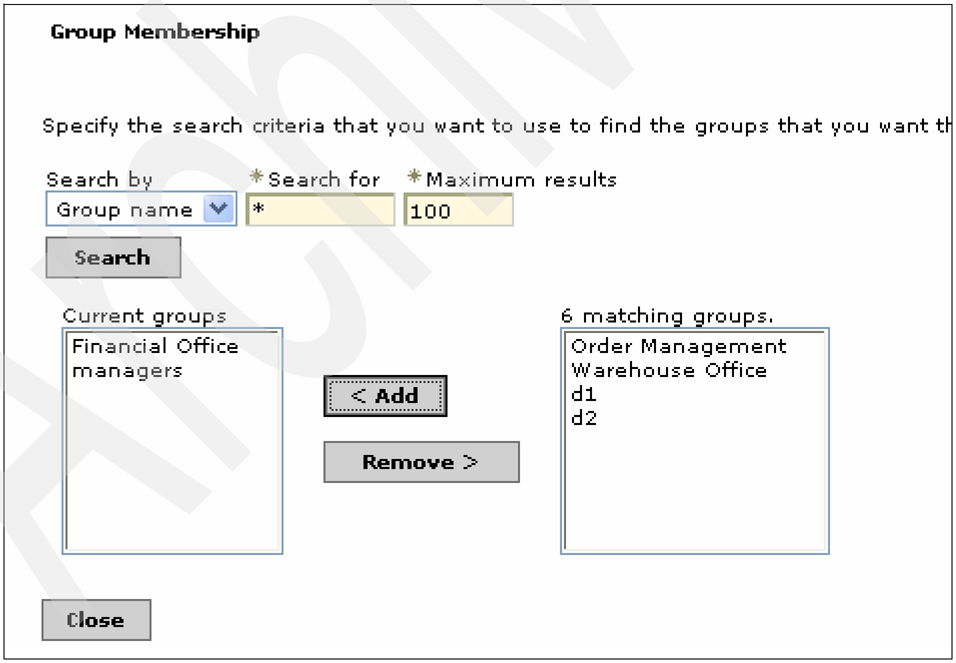


Figure 3-147 Adding janedoe to Financial Office and managers groups

5. Complete the information for user `janedoe`, then click **Create**. For testing, we made the user ID and password the same.

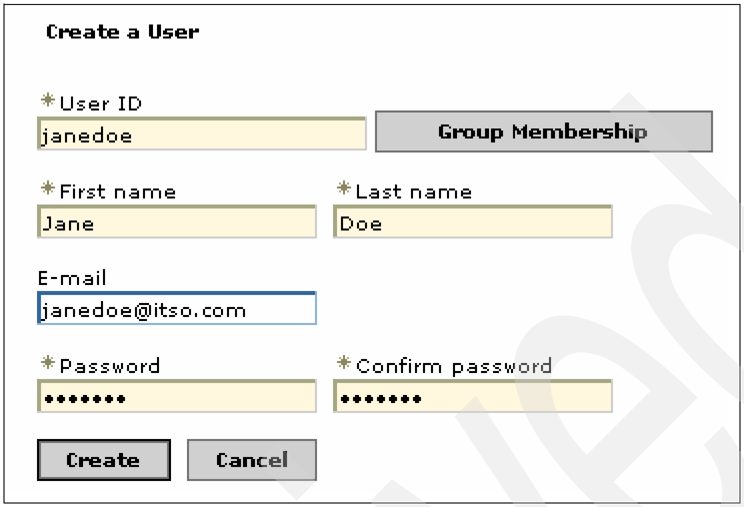


Figure 3-148 Create a User wizard

6. A message displays that indicates the user ID was created and gives you the option to close the window or to create a new ID. Click **Create Like** to create the following users:
   - `janesmith` in the Financial Office group
   - `johnsmith` in the Warehouse Office and managers groups
   - `johndoe` in the Warehouse Office group

   Verify the users in your repository look as shown in Figure 3-149 on page 178.

Create... | Delete | Select an action...

Select	User ID	First name	Last name	E-mail	Unique Name
☐	admin	admin	admin		uid=admin,o=defaultWIMFileBasedRealm
☐	bdaniel	Bonnie	Daniel	Bonnie.Daniel@ibm.com	uid=bdaniel,o=defaultWIMFileBasedRealm
☐	dcampbell	David	Campbell	David.Campbell@ibm.com	uid=dcampbell,o=defaultWIMFileBasedRealm
☐	dedwards	Doug	Edwards	Doug.Edwards@ibm.com	uid=dedwards,o=defaultWIMFileBasedRealm
☐	janedoe	Jane	Doe	janedoe@itso.com	uid=janedoe,o=defaultWIMFileBasedRealm
☐	janesmith	Jane	Smith	johnsmith@itso.com	uid=janesmith,o=defaultWIMFileBasedRealm
☐	jjames	Jeffrey	James	Jeffrey.James@ibm.com	uid=jjames,o=defaultWIMFileBasedRealm
☐	johndoe	John	Doe	johnsmith@itso.com	uid=johndoe,o=defaultWIMFileBasedRealm
☐	johnsmith	John	Smith	johnsmith@itso.com	uid=johnsmith,o=defaultWIMFileBasedRealm
☐	lcarlesberg	Linda	Carlesberg	Linda.Carlesberg@ibm.com	uid=lcarlesberg,o=defaultWIMFileBasedRealm
☐	mburnnet	Mary	Burnnet	Mary.Burnnet@ibm.com	uid=mburnnet,o=defaultWIMFileBasedRealm
☐	mcharles	Mary	Charles	Mary.Charles@ibm.com	uid=mcharles,o=defaultWIMFileBasedRealm
☐	redwards	Ron	Edwards	Ron.Edwards@de.ibm	uid=redwards,o=defaultWIMFileBasedRealm

Page 1 of 1 | Total: 13

Figure 3-149 Users in repository

This concludes the user configuration. You now test that the users and groups that you configured work as expected.

Testing the users and groups

In this section, we show how to test the users and groups configuration from WebSphere Integration Developer and from the administrative console.

Testing from WebSphere Integration Developer

To test from WebSphere Integration Developer:

1. Open the OrderManagementBP business process.
2. Open the in-line human task, OrderManagementBPTask1 (Figure 3-150). To open this task, select the Financial officer authorization human task. In the Details tab of the Properties view, click **Open**.

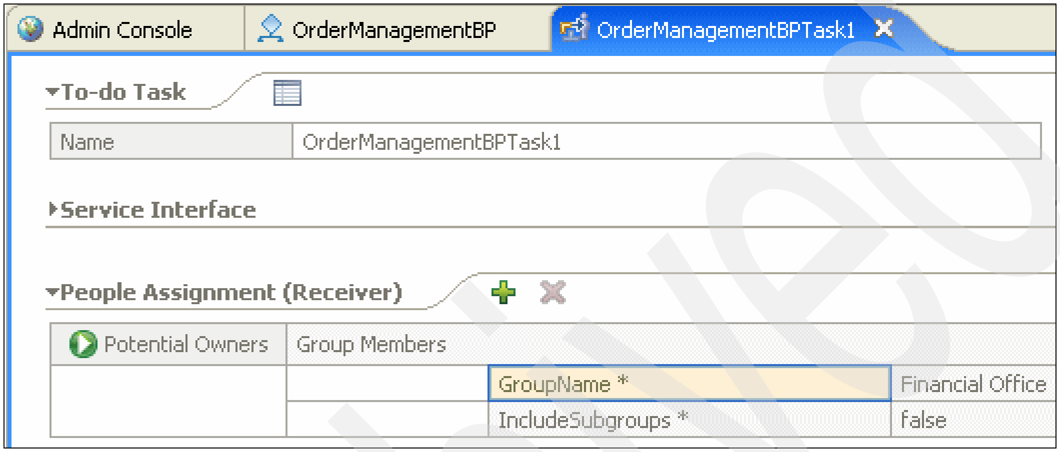


Figure 3-150 In-line Financial Office human task

3. Select **Potential Owners**, and go to the Properties view. Click **Test**, as shown in Figure 3-151.

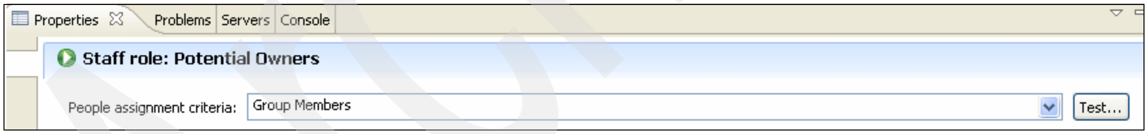


Figure 3-151 Financial Office Potential Owners

4. Select **WebSphere Process Server v6.1** and click **Submit** (Figure 3-152).

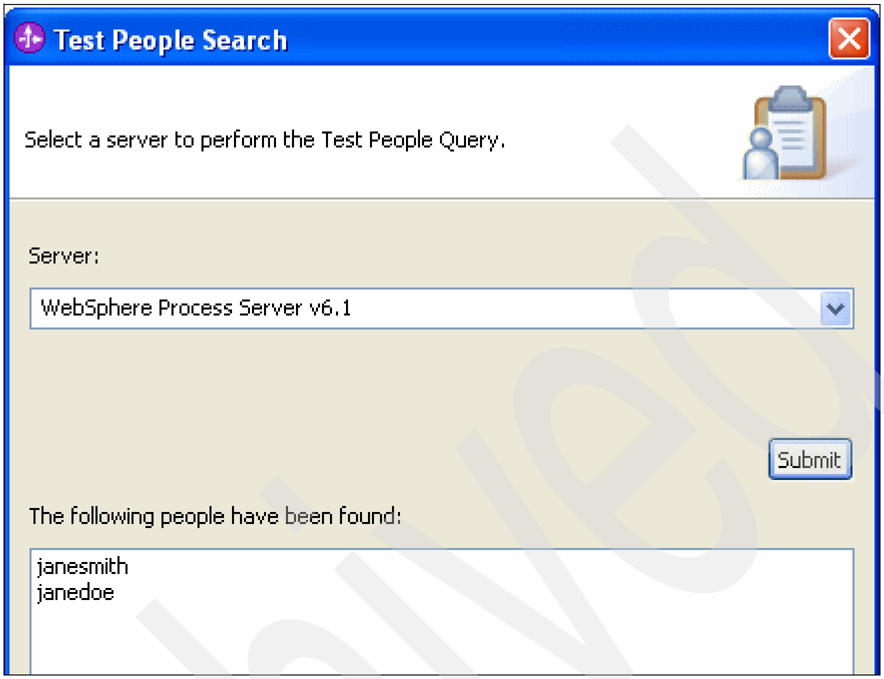


Figure 3-152 Test People Search results

5. Two users are found: `janesmith` and `janedoe`.
6. Repeat steps 2 through 5 to test the Warehouse Office (OrderManagementBPTask2) people query.

Testing from the administrative console

To test from the administrative console:

1. Go to **Manage Groups** and then click **Search**.
2. Click **Financial Office**. Then, click **Members**. Verify that you have two members, Jane Doe and Jane Smith.
3. Repeat these steps to check that the Warehouse Office has two members, John Smith and John Doe.

This concludes the users and groups setup and testing.

3.5.5 Testing the human tasks in the test environment

To test the human tasks in the test environment, follow these steps:

1. Start the WebSphere Process Server test environment.
2. Deploy the OrderManagementApp module.
3. From the assembly diagram, right-click the canvas and select **Test Module**.

   This test uses the same process and data that we describe in “Testing the rules” on page 132. Use the following parameters:

   – Initial request parameters: Import from OrderManagementInput_HUMAN.xml.

   – Item output parameters: Import from item003.xml. Set `RC=0`.

   – Customer output parameters: customer10003.xml. Set `RC=0`.

   – Order output parameters: copied from Order input parameters. Set `RC=0`.
4. Ensure that you see three emulators for Customer, Item, and Order.
5. Set the Detailed Properties and the initial request parameters as shown in Figure 3-153.

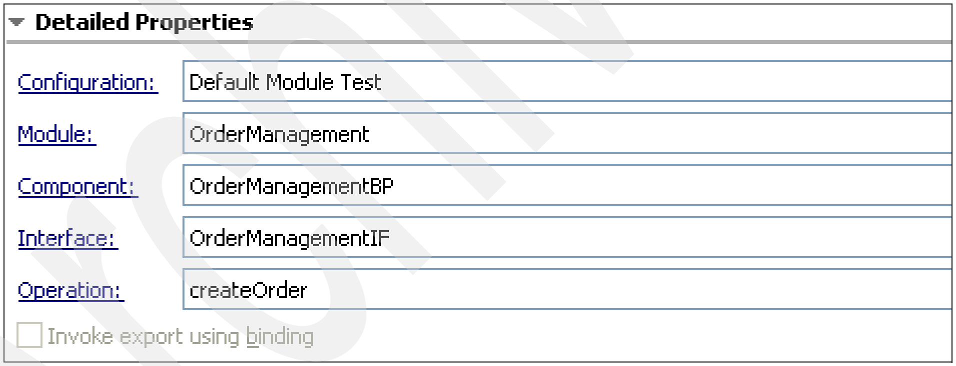


Figure 3-153 Detailed properties for human task test

6. Click **Continue** to start the test.
7. Set the emulation values for item using item003.xml, and click **Continue**.
8. Set the emulation values for customer using customer10003.xml, and click **Continue**.

9. The test continues and then pauses after both rules have been run (see Figure 3-154).

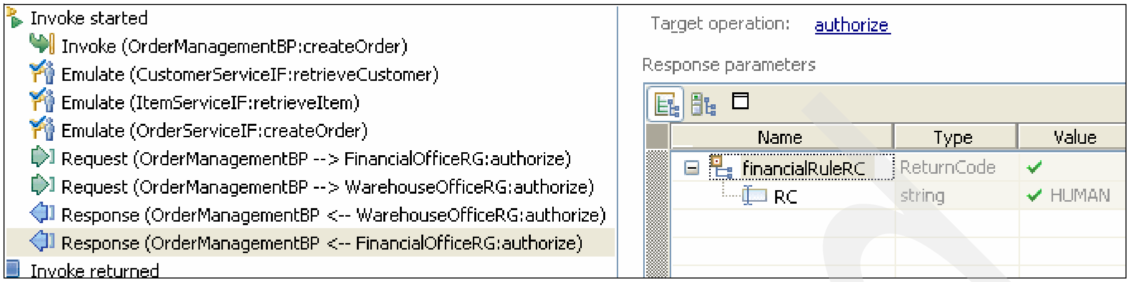


Figure 3-154 Human Task test pausing for tasks

10. Launch the Business Process Choreographer Explorer. Go to the Servers view. Right-click the WebSphere Process Server and select **Launch → Business Process Choreographer Explorer**. Enter:

   – User Name: `janedoe`
   – Password: `janedoe`

   Click **Login**.

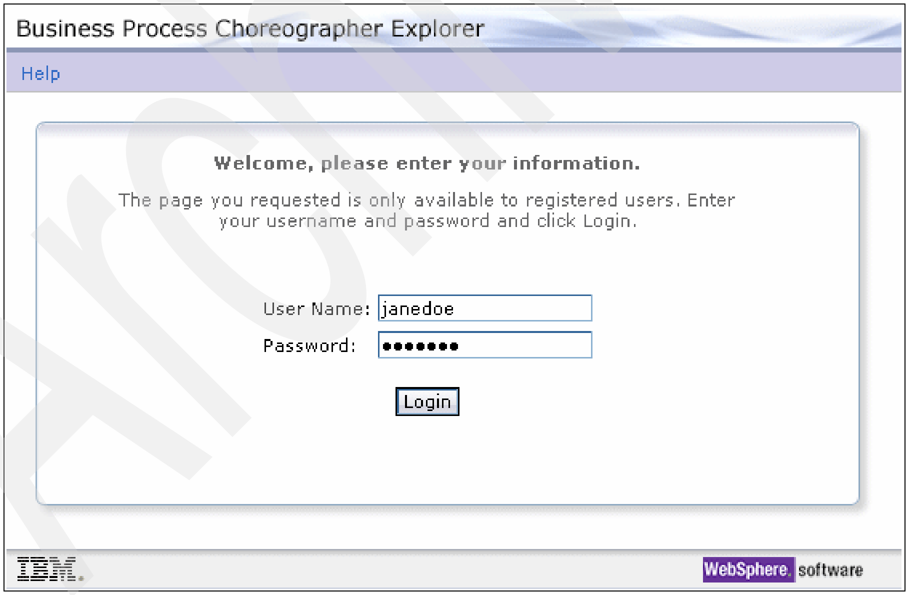


Figure 3-155 Business Process Choreographer Explorer login

The Business Process Choreographer Explorer opens up to janedoe's to-dos, as shown in Figure 3-156.

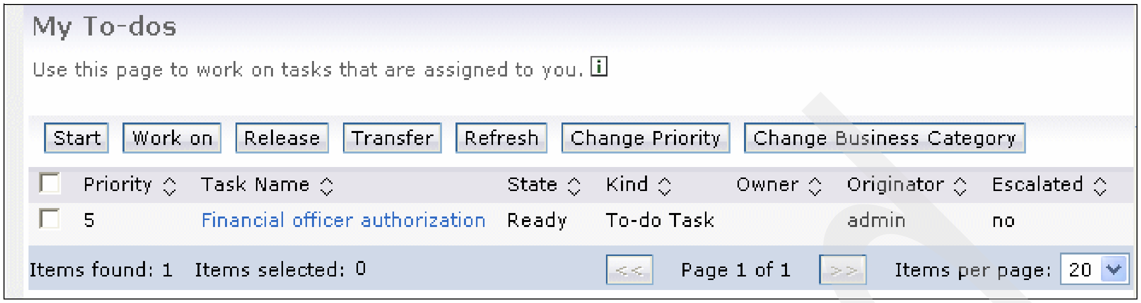


Figure 3-156 janedoe's To-dos

11.Click **Financial officer authorization**. Then, click **Work on** (Figure 3-157).

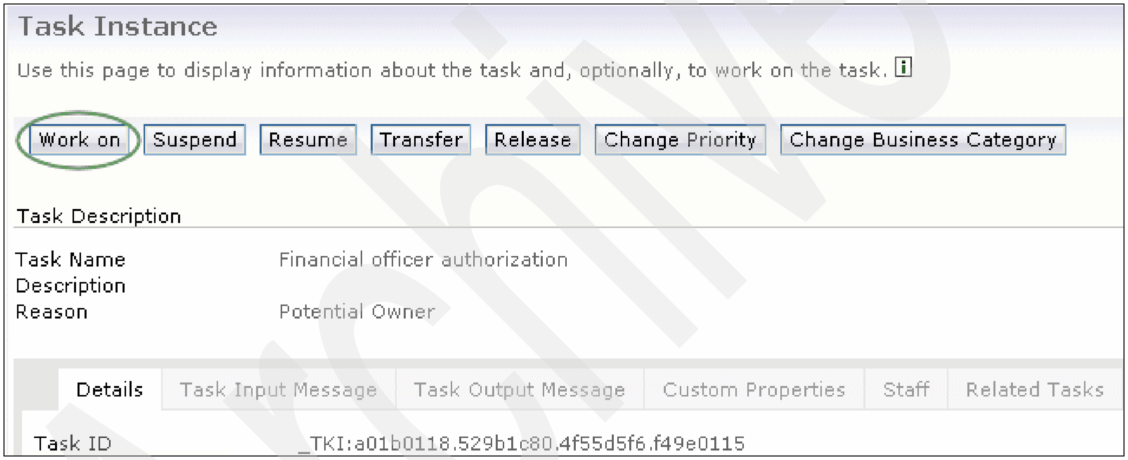


Figure 3-157 janedoe's Task Instance

The Task Message open as shown in Figure 3-158.

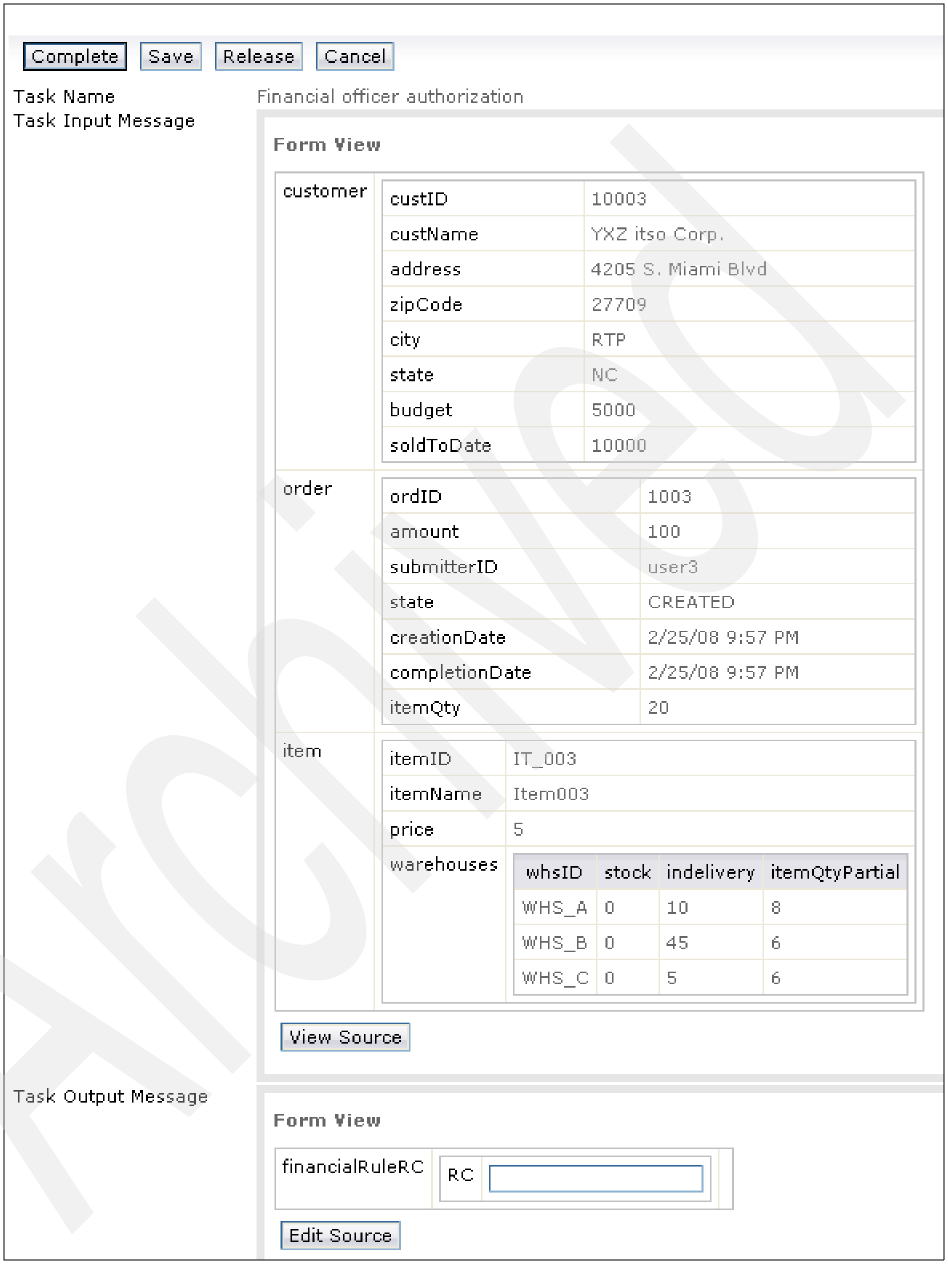


Figure 3-158 janedoe's Task Message

12. Enter `APPROVE` in the **Task Output Message** as shown in Figure 3-159.

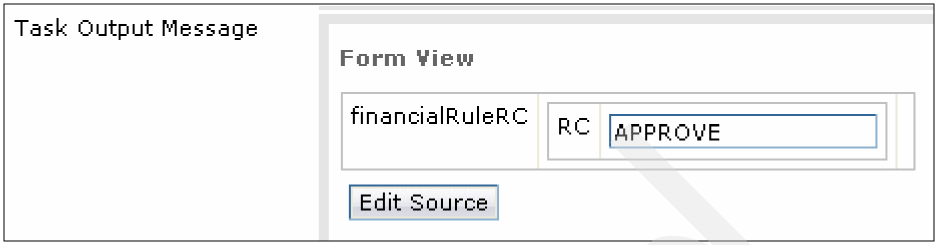


Figure 3-159 janedoe's Task Output Message

13. Click **Complete**.
14. Log out and then login as `johnsmith`.
15. Click **Warehouse officer authorization**. Then, click **Work on**.
16. Click the Task Input Message's **View Source**.

> **Tip:** In V6.1, the Business Process Choreographer Explorer lets you view the input and output as XML. This view makes it easier to copy the input value and populate the output value by pasting or to have some predefined XML to use as either an input or output.

17.Select the contents of the item tag as shown in Figure 3-160.

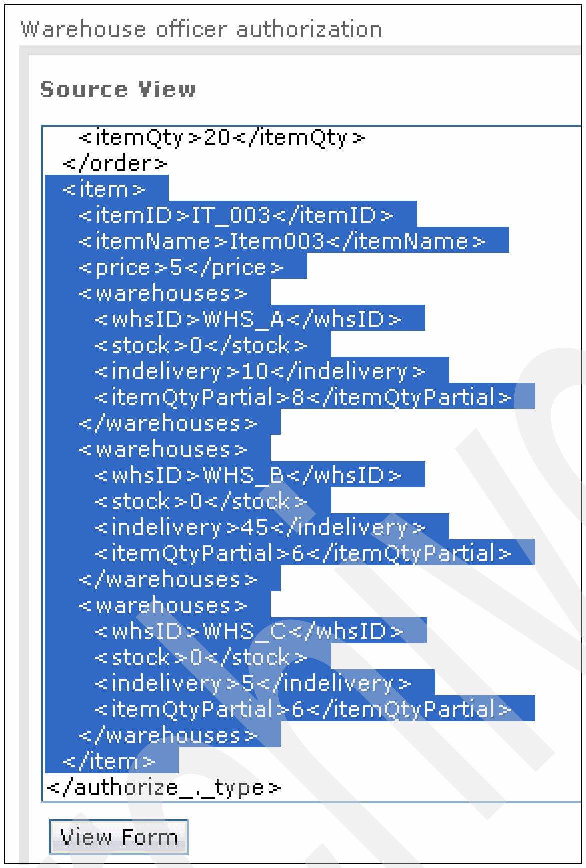


Figure 3-160 Source view of input message

18. Right-click **Copy**. Then, set the Task Output Message warehouseRuleRC.RC value to `ACCEPT`, and click **Edit Source** as shown in Figure 3-161.

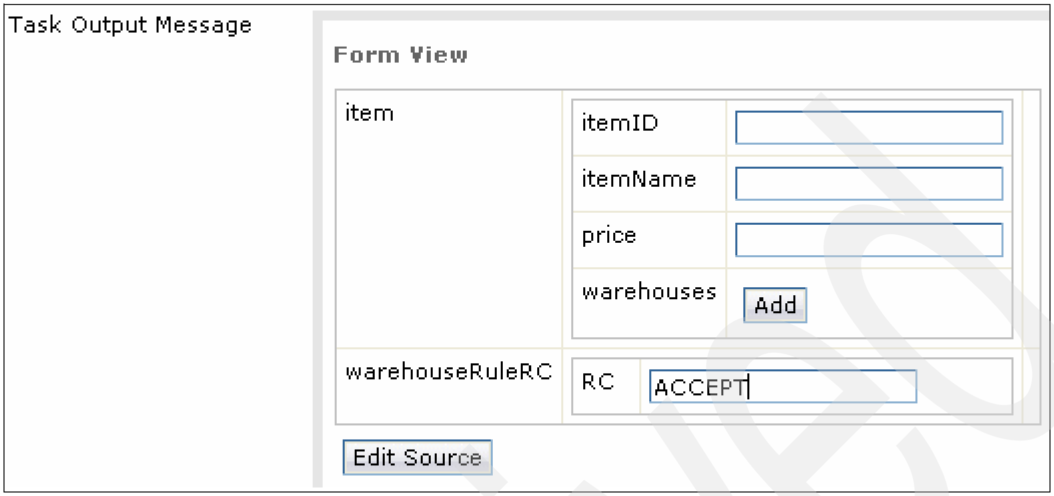


Figure 3-161 johnsmith's Task Output Message

19. Select the item tag in the task output message (Figure 3-162).

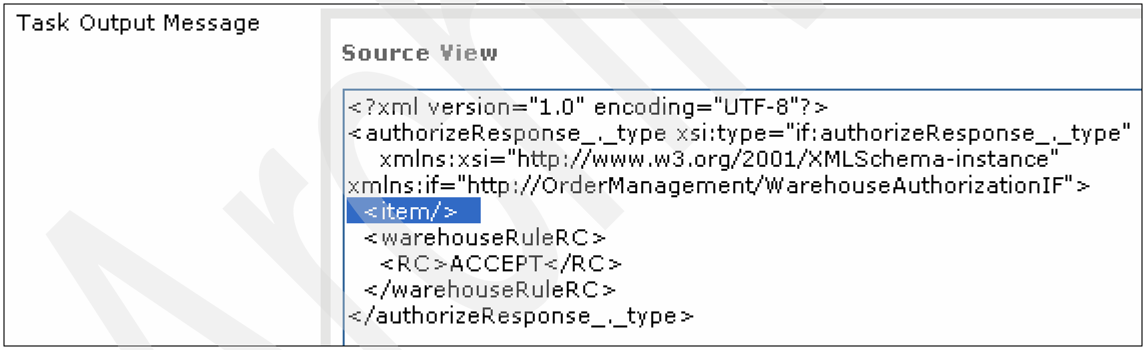


Figure 3-162 Default item tag in Task Output Message

20. Right-click the selected item tag, and then select **Paste**. Replace the itemQtyPartial for WHS_A and WHS_C. Set the itemQtyPartial for WHS_A to 9 as shown in Figure 3-163.

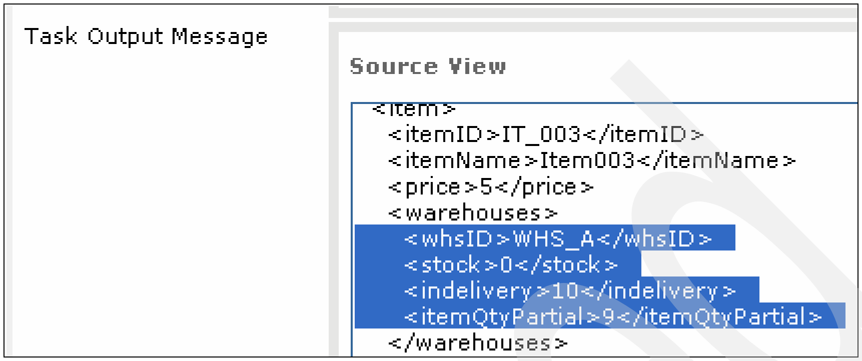


Figure 3-163 WHS_A's values

Set the itemQtyPartial for WHS_C to 5, as shown in Figure 3-164.

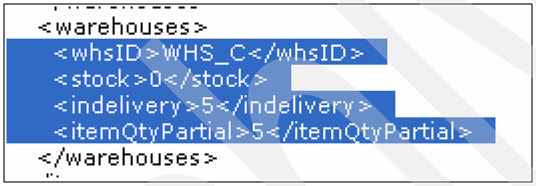


Figure 3-164 WHS_C's values

21. Click **Complete** and log out.

22. Go to the Console view and look for the message that tells you that the process is completed, as shown in Figure 3-165.

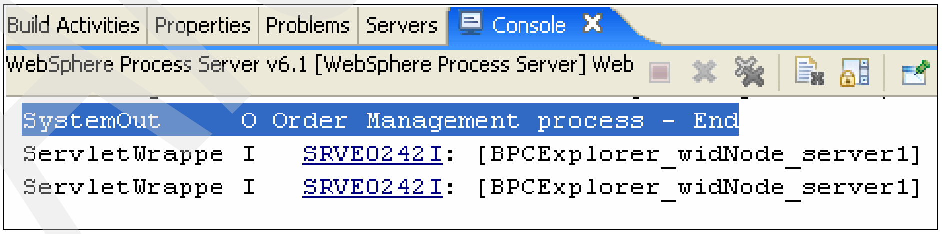


Figure 3-165 Console view after running human tasks

Tip: If you check in your code to CVS, check in the changes.

3.6 Implementing and testing the final verification steps

In this section, we explain how to implement the final verification steps, encapsulated in a Choice activity in the OrderManagementBP, as shown in Figure 3-166.

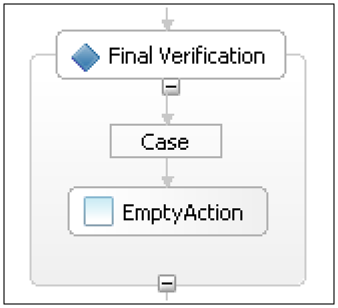


Figure 3-166 Unimplemented final verification Choice activity

If the financial and warehouse tasks are APPROVED, go through the Confirm Order steps. Otherwise, execute the Suspend Order steps.

The *Confirm Order* steps are:

1. Initialize the order for update.
2. Update the order.
3. Update the item.
4. Initialize the customer for update.
5. Update the customer.

The *Suspend Order* steps are:

1. Initialize the order for update.
2. Update the order.

3.6.1 Implementing the final verification steps

You implement the logic that we have defined by defining the Case logic in the final verification activity and an Otherwise element. Below each, you add Java snippets and Invoke activities to the Choice activity.

Setting the Case logic and Otherwise element

Set the Case logic by following these steps:

1. Click the Case element. Go to **Properties** → **Description** and set the **Display Name** to `ConfirmOrder`, as shown in Figure 3-167.

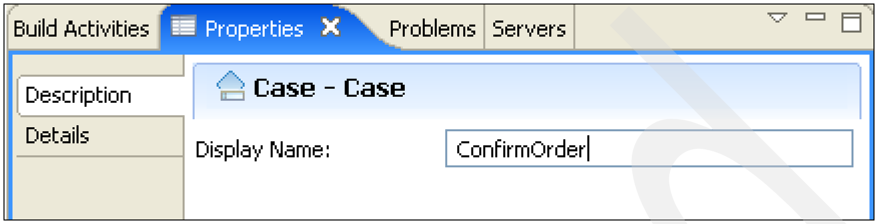


Figure 3-167 Case Display Name

2. Go to **Properties** → **Details** and build the Case as shown in Figure 3-168. (The process is similar to that described in "Adding a Choice activity to verify the customer and item exist" on page 86.)

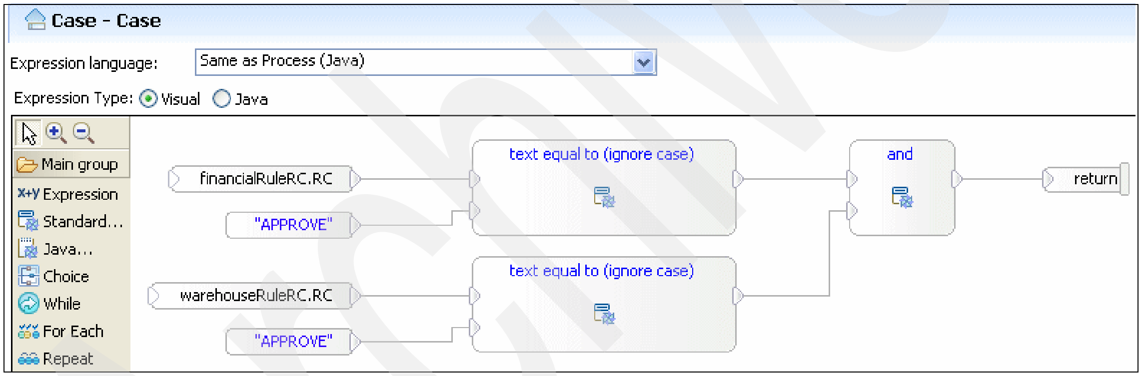


Figure 3-168 Case Details

3. Right-click **Final Verification** and select **Add Otherwise** (Figure 3-169).

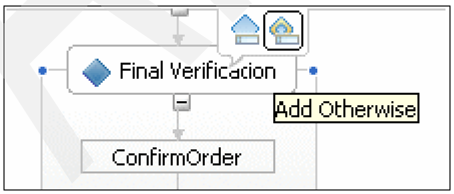


Figure 3-169 Add Otherwise context menu

The Final Verification activity now has two branches as shown in Figure 3-170.

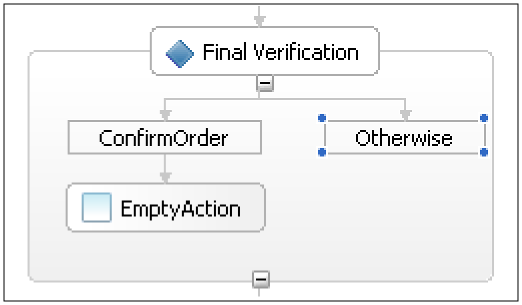


Figure 3-170 Final Verification Choice activity with Case and Otherwise

Implementing ConfirmOrder

The Confirm Order steps are a series of Java snippets and Invoke activities. Follow these steps to implement:

1. Right-click **Empty Action**. Select **Change Type** → **Snippet**. Then, rename the snippet `Init Update Order`, as shown in Figure 3-171.

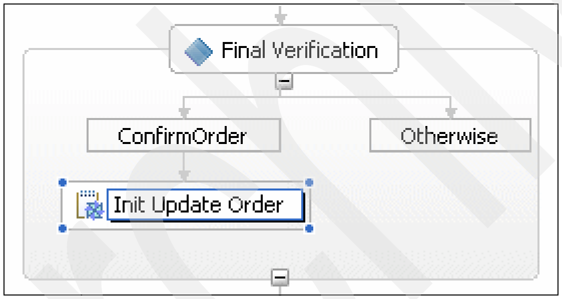


Figure 3-171 Snippet Init Update Order

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\ConfirmOrder_InitUpdateOrder.txt. See Appendix B, “Additional material” on page 461.

2. Set the details as shown in Figure 3-172.

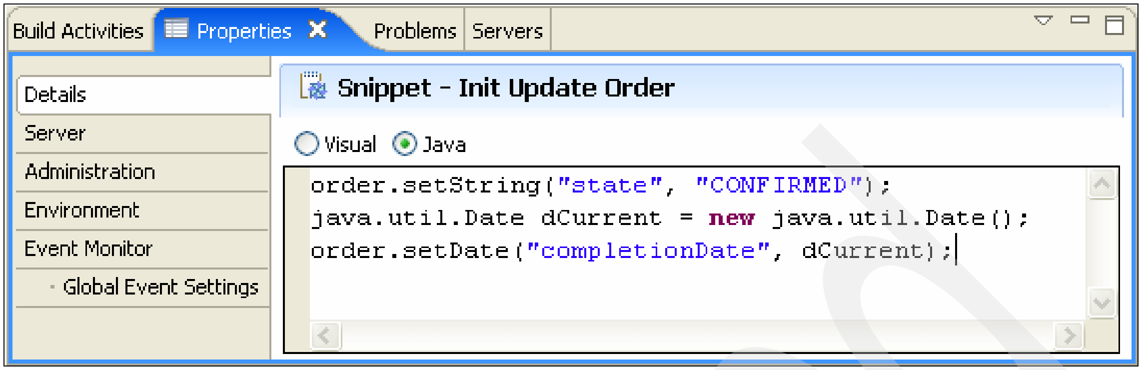


Figure 3-172 Init Update Order details

3. Right-click **ConfirmOrder**. Select **Add** → **Invoke**. Then, name the Invoke `Update Order -- CONFIRMED`, and set the Invoke's details as shown in Figure 3-173.

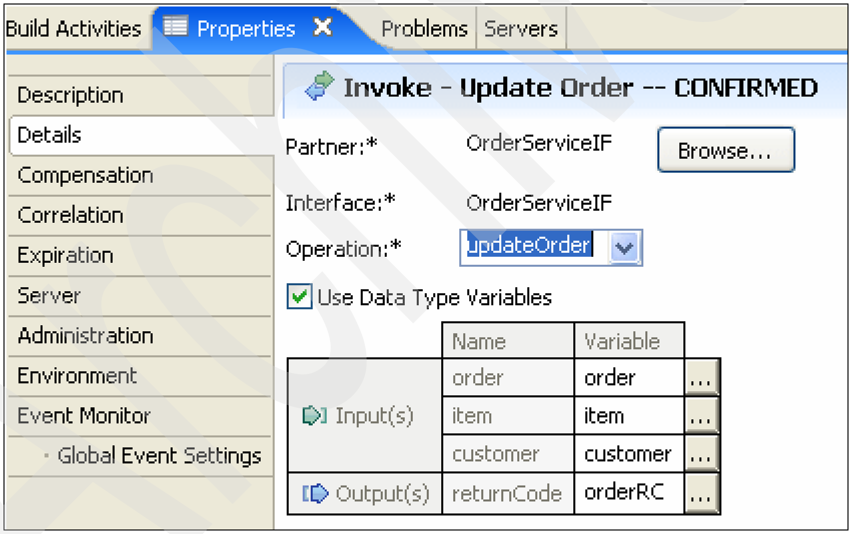


Figure 3-173 Invoke Update Order -- CONFIRMED

> **Note:** Make sure that you select the UpdateOrder operation. The default operation is createOrder.

4. Add another Invoke activity Update Order -- CONFIRMED. Name the activity `Update Item`, and set the Invoke Details as shown in Figure 3-174.

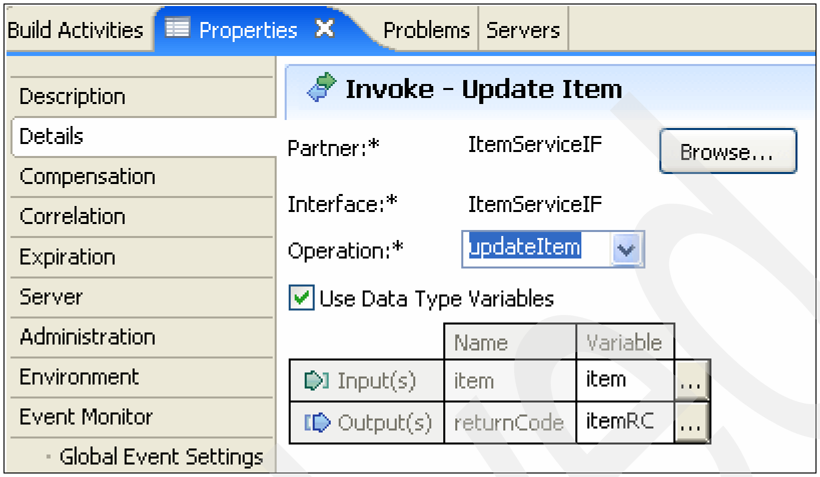


Figure 3-174 Invoke Update Item

5. Right-click **ConfirmOrder**. Select **Add** → **Snippet**. Then, name the snippet `Init Update Customer`, and define the snippet as shown in Figure 3-175.

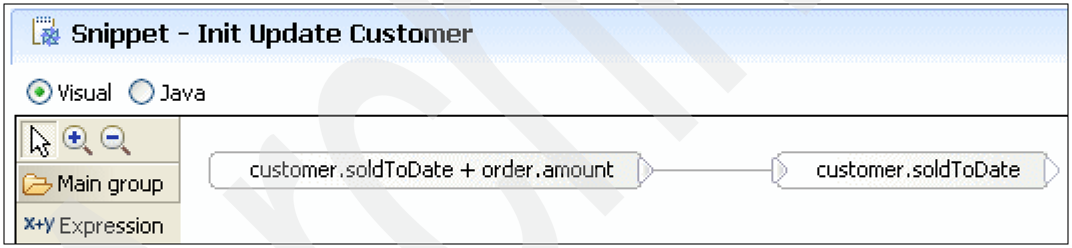


Figure 3-175 Init Update Customer snippet

6. Right-click **ConfirmOrder**. Select **Add** → **Invoke**. Then, name the Invoke activity `Update Customer`, and configure the detail properties for the Invoke activity as shown in Figure 3-176.

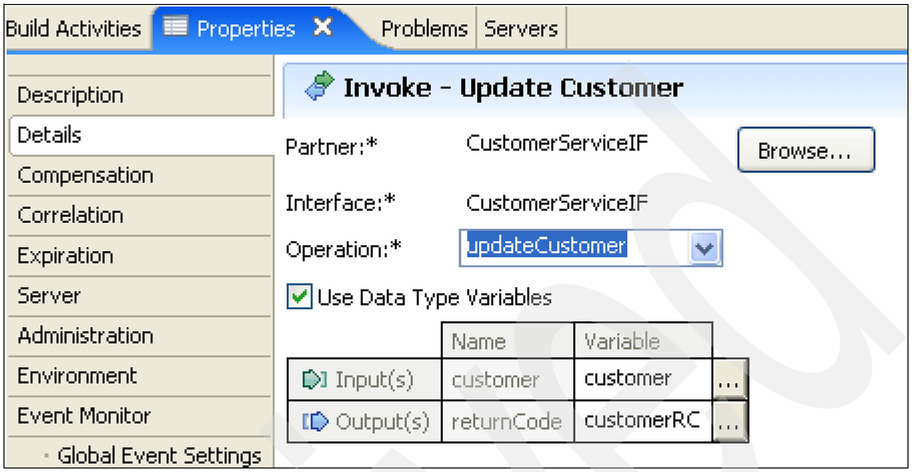


Figure 3-176 Invoke Update Customer

7. Verify the Confirm Order design as shown in Figure 3-177.

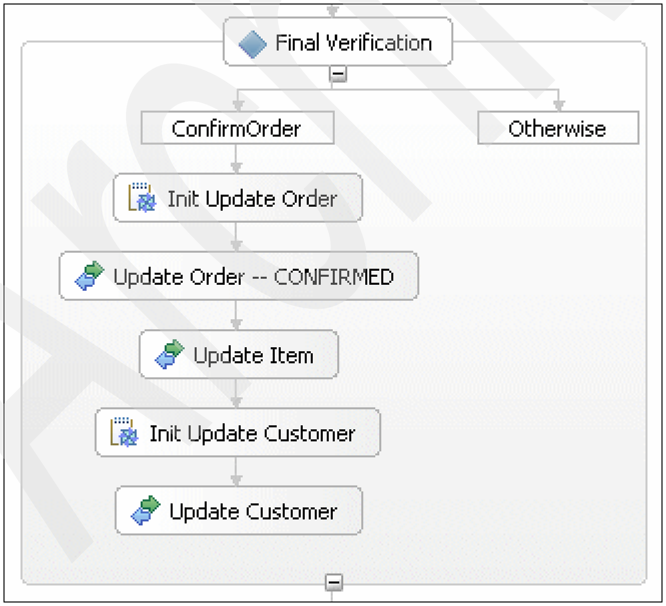


Figure 3-177 Confirm Order design

Implementing Suspend Order steps

The Suspend Order steps are also a series of Java snippets and Invoke activities. Follow these steps to implement:

1. Right-click **Otherwise**. Select **Add** → **Snippet**. Name the snippet `Init Update Order`, and define the snippet as shown in Figure 3-178.

> **Additional material:** The Java code for this snippet is contained in BusExampleSnippets\Otherwise_InitUpdateOrder.txt. See Appendix B, “Additional material” on page 461.

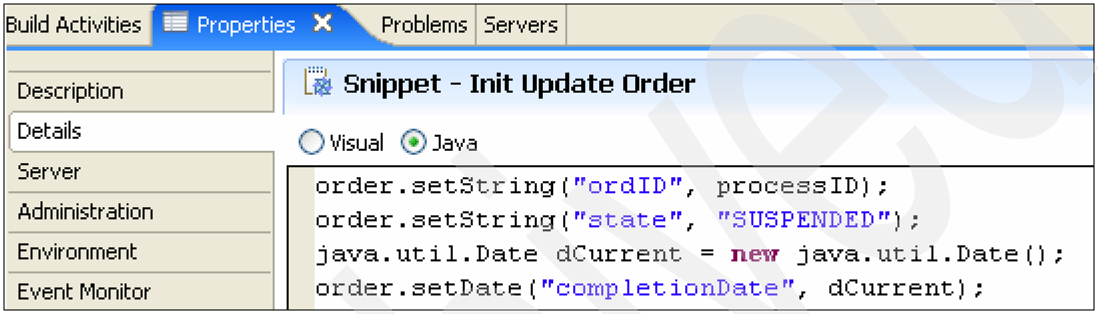


Figure 3-178 Init Update Order snippet

2. Right-click Otherwise. Select **Add** → **Invoke**. Then, name the Invoke, `Update Order -- SUSPENDED`, and define the property details as shown in Figure 3-179.

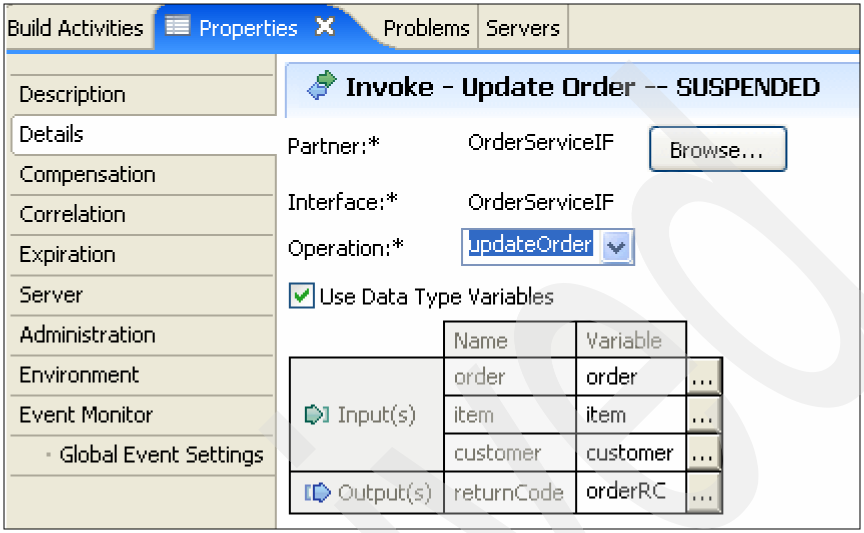


Figure 3-179 Invoke Update Customer

3. Verify the Final Verification design looks as shown in Figure 3-180.

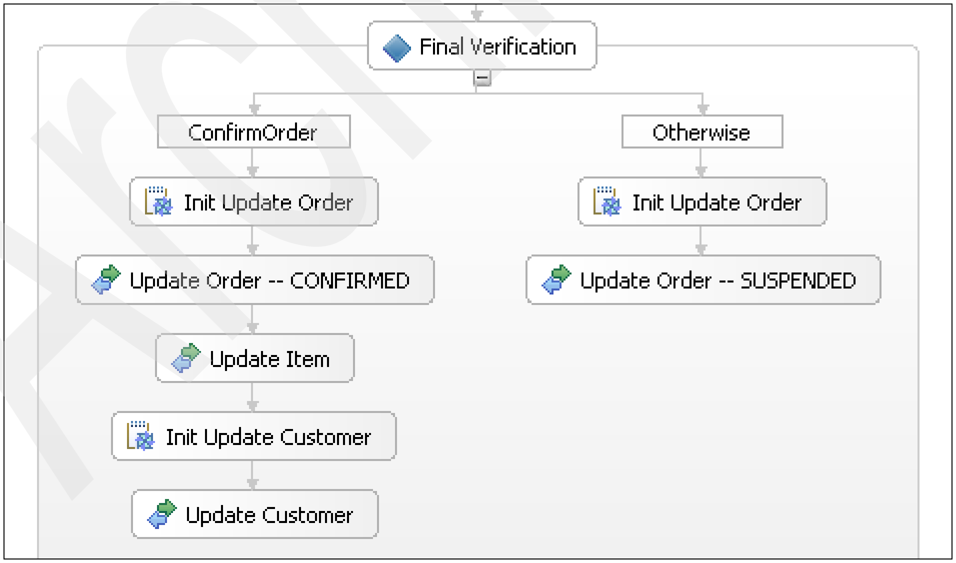


Figure 3-180 Final Verification design

4. Save the Projects. There are no errors in the Problems view.

3.6.2 Testing the final verification steps

You now test the BPEL with the two pertinent test cases: APPROVED and REJECTED. For the APPROVED case, the process goes through the ConfirmOrder path, while for the REJECTED case it goes through the Otherwise path.

Confirming Order test

Test the module using the Approve input data and the settings shown in Figure 3-181. Use the following parameters:

- Initial request parameters: Import from OrderManagementInput_APPROVE.xml.
- Item output parameters: Import from item001.xml. Set `RC=0`.
- Customer output parameters: customer10001.xml. Set `RC=0`.
- Order output parameters: copied from Order input parameters. Set `RC=0`.

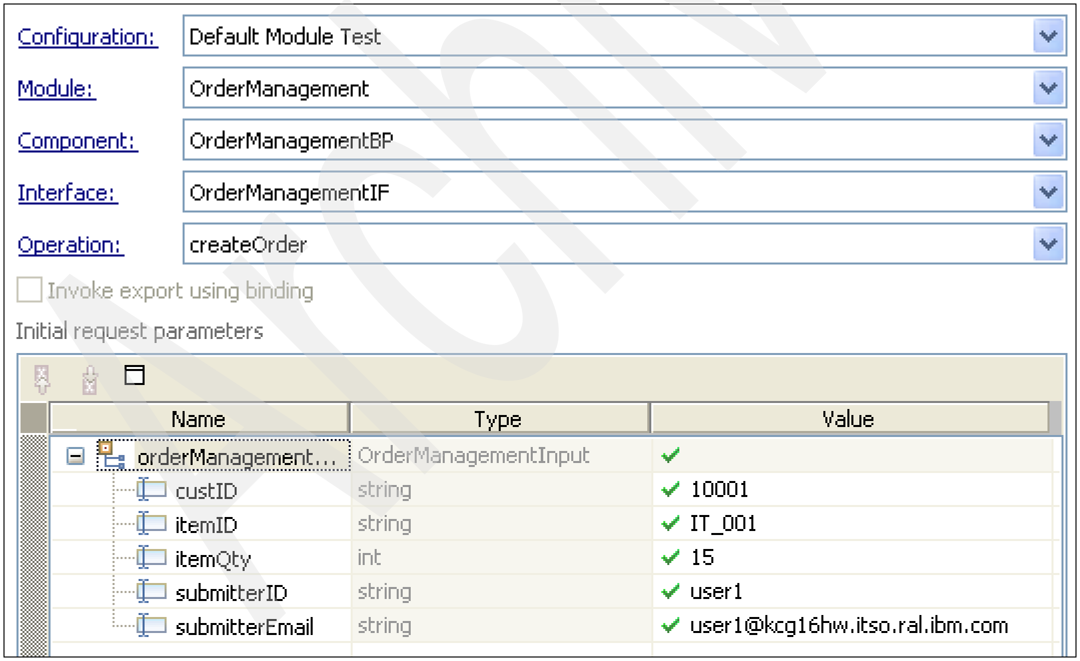


Figure 3-181 Approve input data

After running the scenario, note the emulation points for updateOrder, updateItem, and updateCustomer, as shown in Figure 3-182.

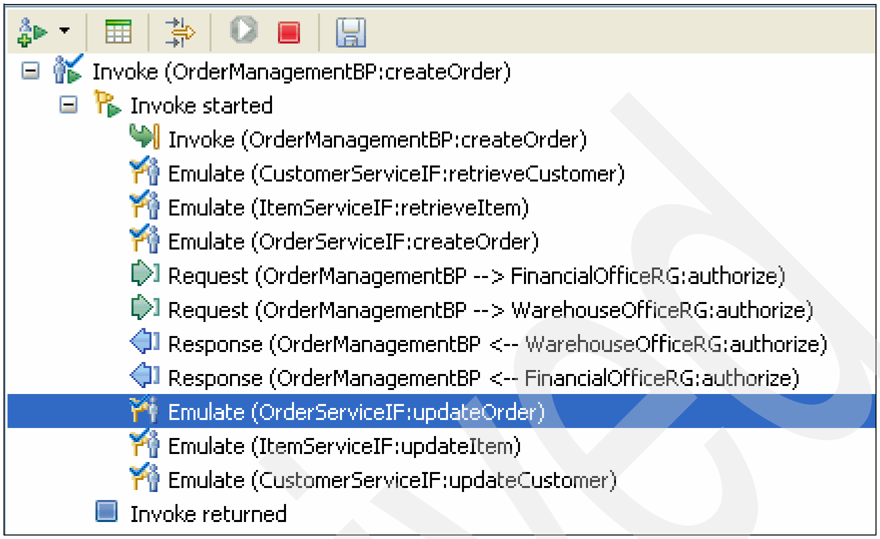


Figure 3-182 Confirm order execution steps

Suspending Order test

Test the module using the Reject input data and the settings shown in Figure 3-183 on page 199. Use the following parameters:

- Initial request parameters: Import from OrderManagementInput_REJECT.xml.
- Item output parameters: Import from item002.xml. Set `RC=0`.
- Customer output parameters: customer10002.xml. Set `RC=0`.
- Order output parameters: copied from Order input parameters. Set `RC=0`.

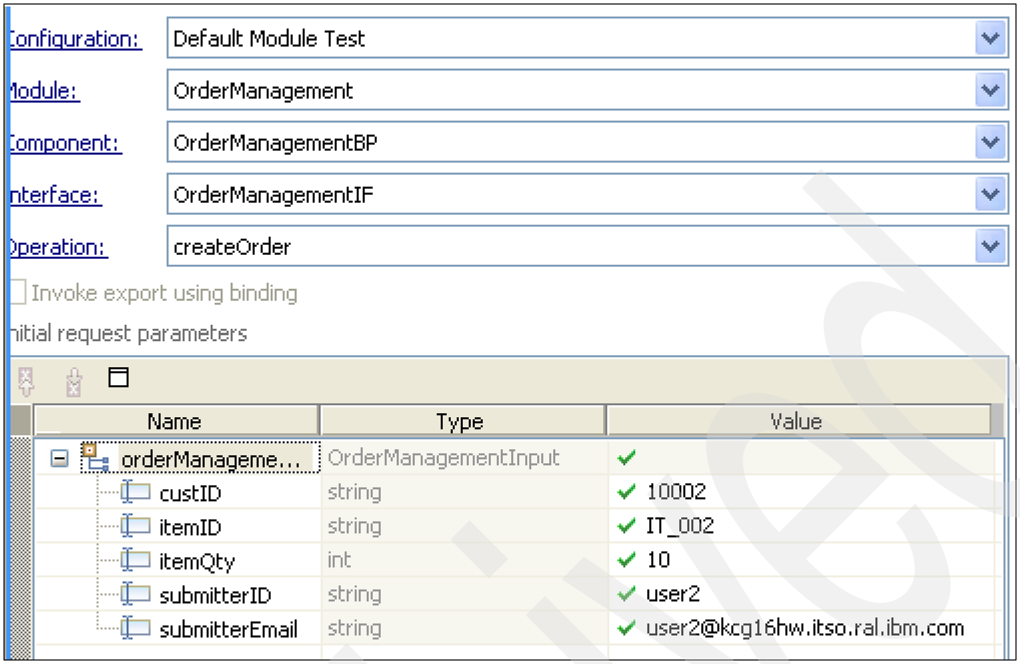


Figure 3-183 Reject input data

After running the scenario, note the emulation points for updateOrder *only*, as shown in Figure 3-184.

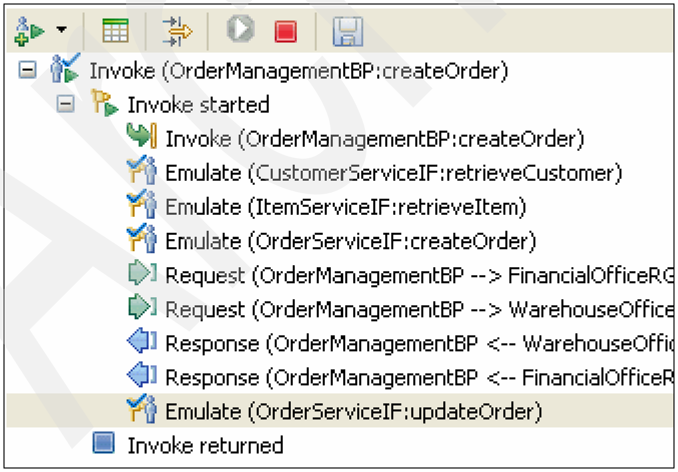


Figure 3-184 Suspend order execution steps

This concludes the final verification implementation.

Tip: If you check in your code to CVS, check in changes.

3.7 Adding the ForEach activity and test

This section explains how to fulfill an order that is placed (as shown in Figure 3-185). When the order is created, it has an associated item and quantity of that item that is requested. You need to add a call to the Warehouse Availability Mediation to tell you how this order will be fulfilled across the warehouses. After you know which warehouses will fill the order, you also need to calculate the total stock that is available throughout the warehouses. Total stock is used as an input to the InvokeWarehouseOfficeBR. In the testing that you have done so far, you have used the initial value for total stock of 0 and passing it to the rule.

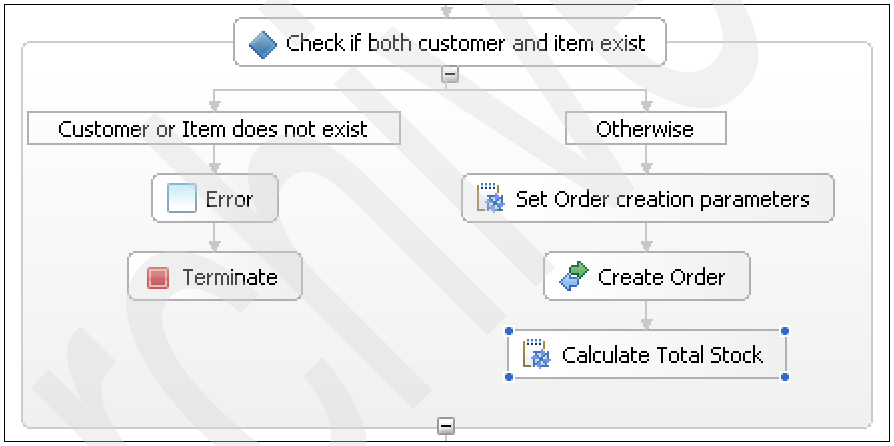


Figure 3-185 Create order steps stubbed out

To invoke the Warehouse Availability Mediation, you add an Invoke activity after Create Order, followed with a ForEach activity that you use to iterate over the array of warehouses to calculate total stock. You then implement these activities and then test them.

3.7.1 Implementing ForEach activity and warehouse splitting steps

The first step is to add an invoke of the Warehouse Availability Mediation followed by the ForEach activity.

Invoking the Warehouse Availability Mediation

To invoke the Warehouse Availability Mediation:

1. Open the OrderManagementBP and add the WarehouseItemSplitIF as a reference partner, as shown in Figure 3-186.

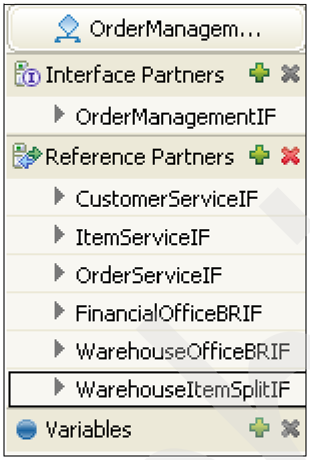


Figure 3-186 OrderManagementBP reference partners after adding warehouse split

2. Add an Invoke activity below Create Order and name it `Split Items Across Warehouses` as shown in Figure 3-187.

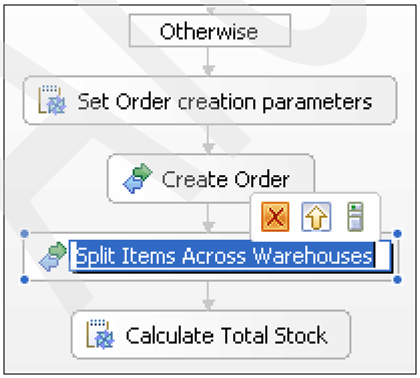


Figure 3-187 Adding invoke Split Items Across Warehouses

3. Define the detail properties for Split Items Across Warehouses as shown in Figure 3-188.

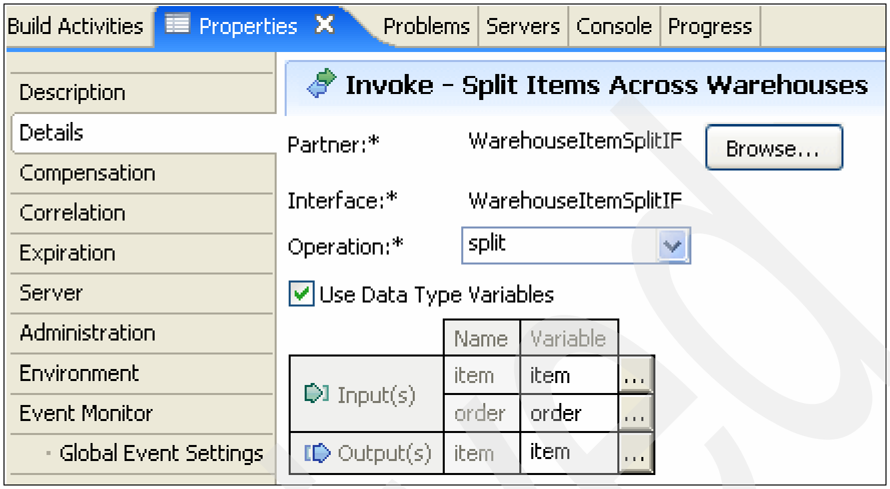


Figure 3-188 Details of Split Items Across Warehouses

These steps are all that you need to do to invoke the Warehouse Availability Mediation.

Using the ForEach activity to calculate total stock

To calculate the total stock, use the ForEach activity and iterate over an item's warehouse array. You can obtain this updated item as a return value from the Warehouse Availability Mediation. Follow these steps:

1. Find the ForEach activity in the Palette in the Structures list. Drag the ForEach activity below Split Items Across Warehouses.
2. Move Calculate Total Stock within the ForEach activity a shown in Figure 3-189.

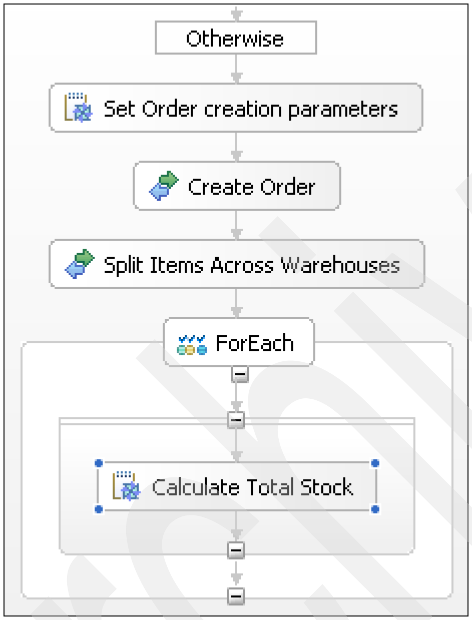


Figure 3-189 ForEach with snippet

3. Click the ForEach activity. Then, go to the Details tab in the Properties view.

   Notice the defaults for execution of iterations (sequential) and early exit criterion (none). You do not need to make changes here because you want to loop sequentially through the values of the array and because you do not have any early exit criterion. (You want to go over each array value.) Set the iteration variable to a more consumable value, and specify the array over which you want to iterate.

Tip: While we show a very simple example here, this ForEach activity can have many activities within it (including invoke a sub-process). For example, it can execute in parallel, it can have early exit criterion, and so on.

4. Change the Index-Variable Name from the default (Index) to w as shown in Figure 3-190.

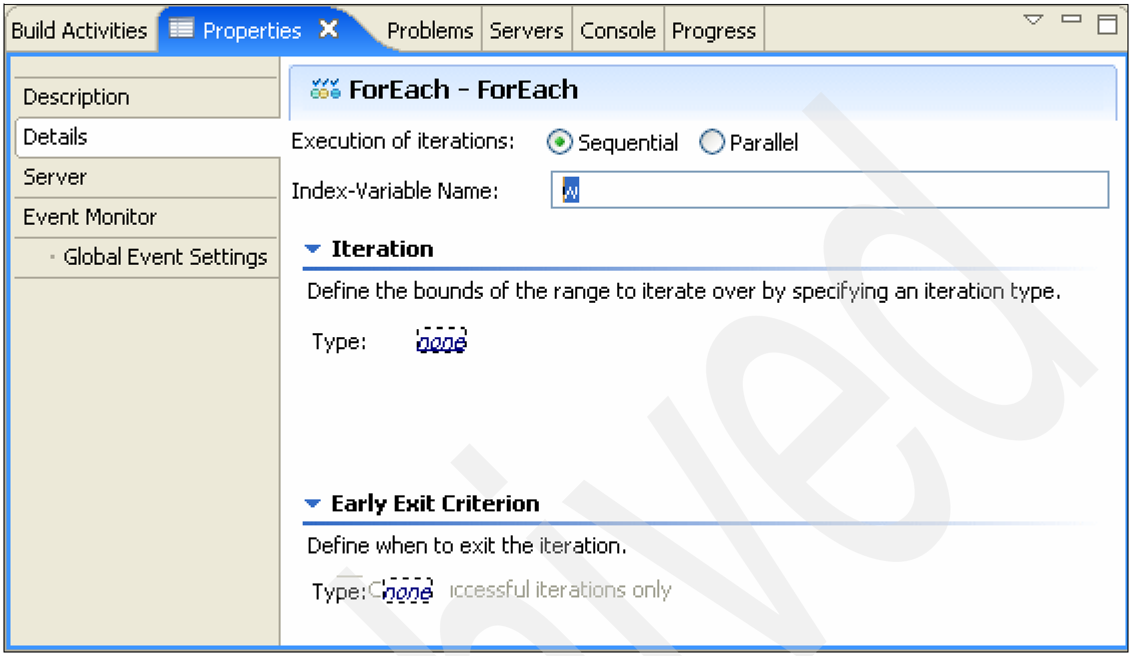


Figure 3-190 Set the Index-Variable Name of a ForEach

5. Set the iteration type by clicking **none** to open a context menu. Expand **Array** and select the **warehouses** array as shown in Figure 3-191.

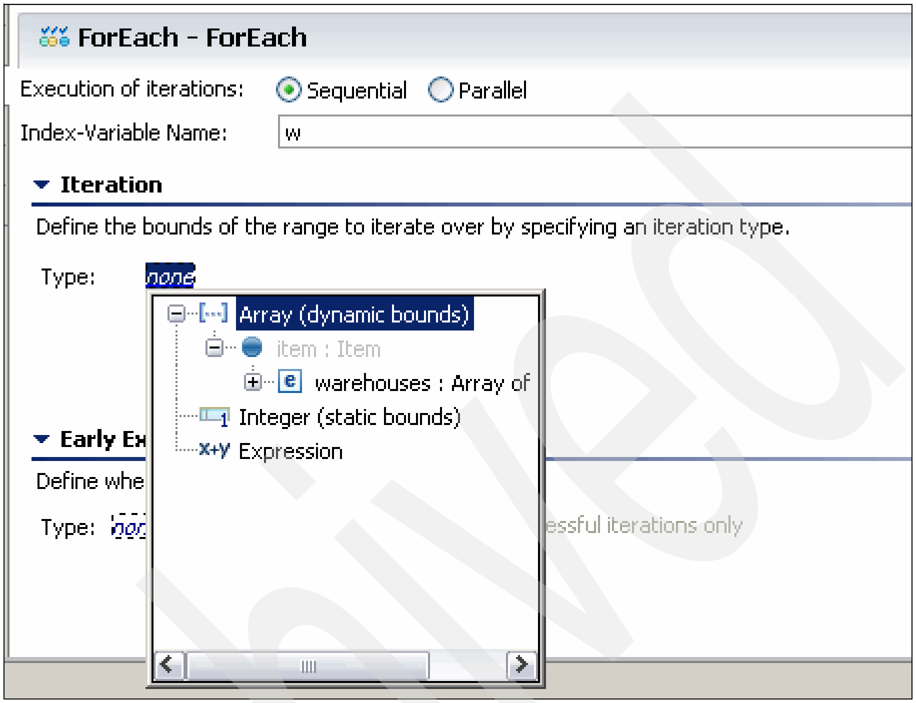


Figure 3-191 ForEach iteration context menu

6. Verify the ForEach activity details as shown in Figure 3-192.

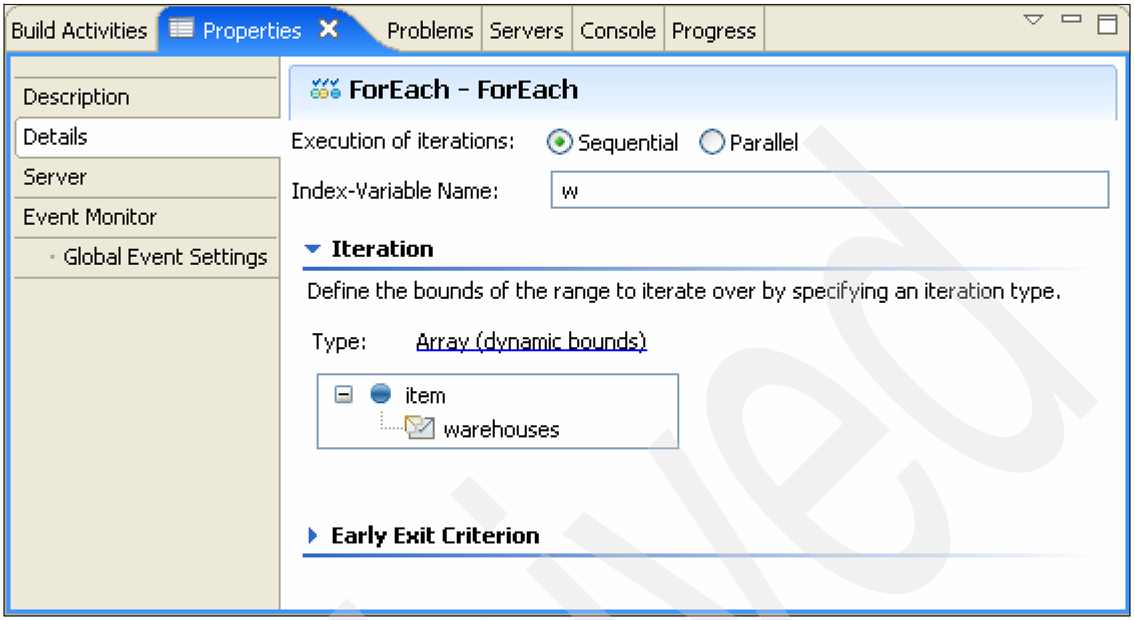


Figure 3-192 ForEach details

Calculating Total Stock snippet

The following steps show how to add logic to the snippet within the ForEach activity to calculate the total stock. The logic is to increment the Total Stock variable by the current warehouse's stock. Key learning steps are how to use the index to access the current warehouse from the array and create a temporary variable within a Java snippet.

To calculate the Total Stock snippet:

1. Rename the snippet Calculate Total Stock to `Add to Total Stock` (as shown in Figure 3-193) to be more reflective of the step that you are performing.

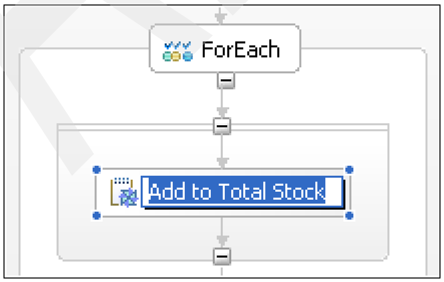


Figure 3-193 Rename snippet Calculate Total Stock, Add to Total Stock

2. Open the Details tab of the Properties view. Create a temporary variable called currentWarehouse, of type Warehouse as follows:

   a. Find the Standard visual snippet **create specific BO** under SCA services and click **OK** (Figure 3-194).

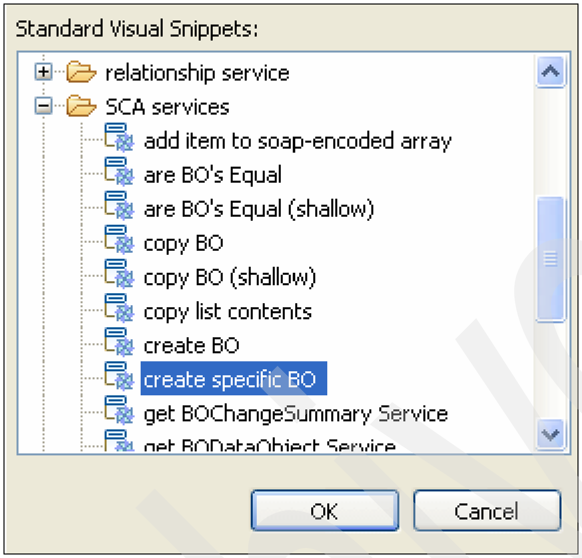


Figure 3-194 create specific BO standard snippet

   b. Select **Warehouse** as the data type and click **OK**.

   c. Click the canvas to add the create Warehouse snippet as shown in Figure 3-195.

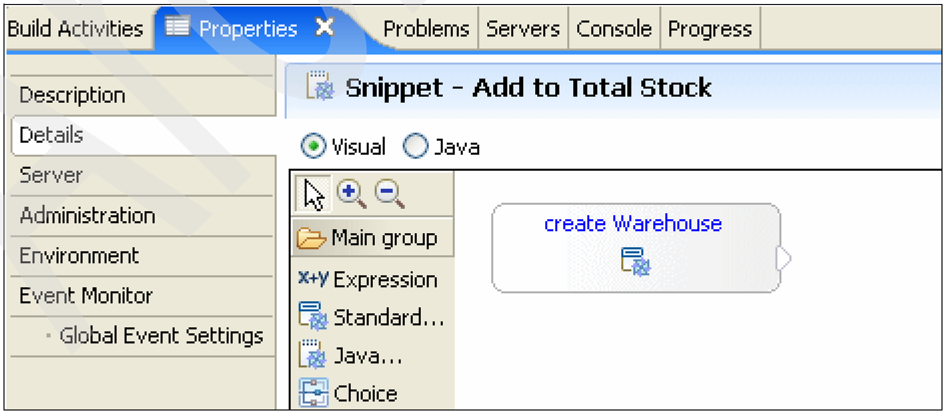


Figure 3-195 create Warehouse snippet on canvas

3. Add an expression onto the canvas. This expression is your temporary variable, currentWarehouse. Follow these steps:

   a. Click the expression and type `currentWarehouse`.

   b. Link `create Warehouse` to the currentWarehouse variable. This link sets the currentVariable's type to Warehouse. See Figure 3-196.

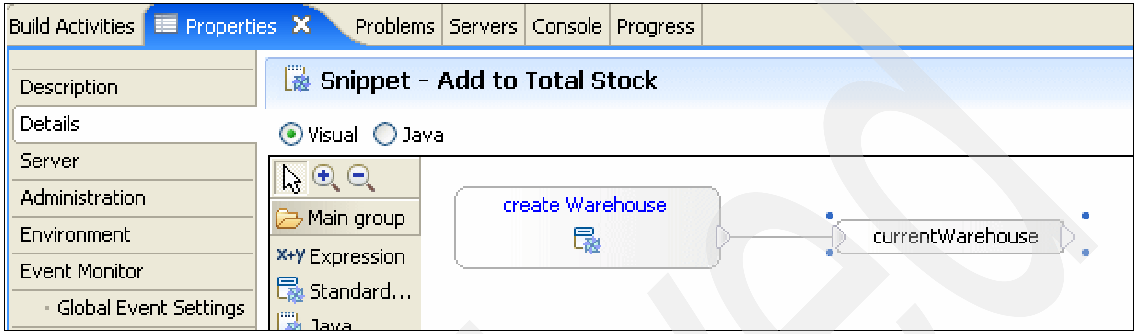


Figure 3-196 The create Warehouse temporary variable of type Warehouse

4. Add a snippet to access the current array value as follows:

   a. Find the Standard Visual Snippet list, select **list** → **get item at index,** and drag it onto the canvas as shown in Figure 3-197.

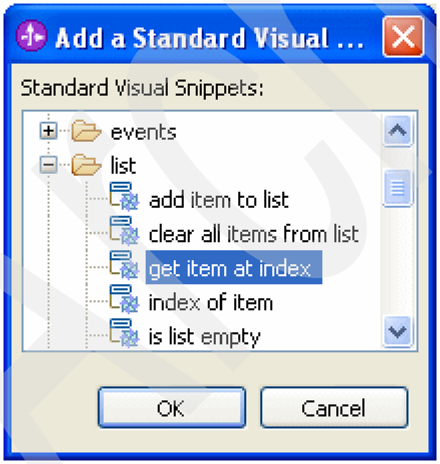


Figure 3-197 get item at index standard snippet

If you hover over **get item at index** it provides a brief description of its functionality, the input that it expects, and the output that it provides, as shown in Figure 3-198.

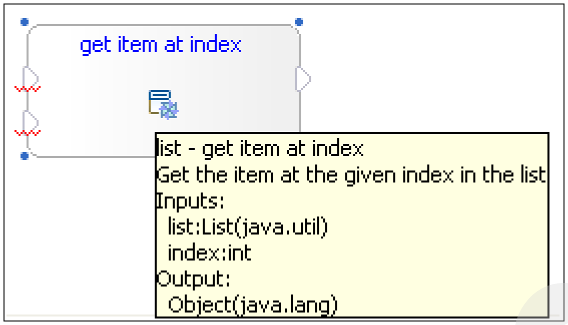


Figure 3-198 get item at index parameters

5. Set the index input value for the array access:

   a. Drag the w variable onto the canvas as shown in Figure 3-199.

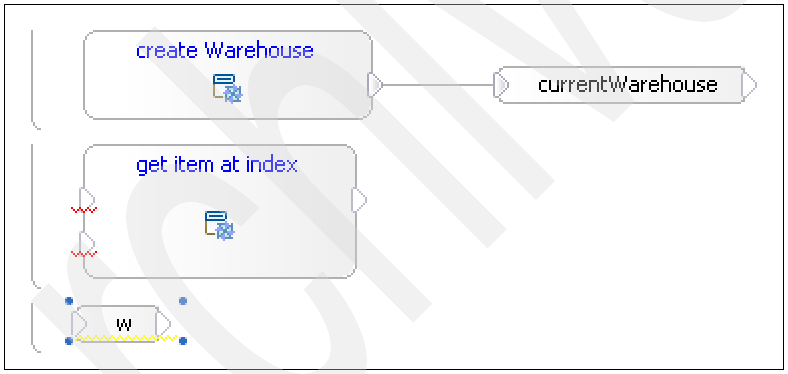


Figure 3-199 w variable onto canvas

   b. Click **w** to open up its context menu and select “-” (Figure 3-200).

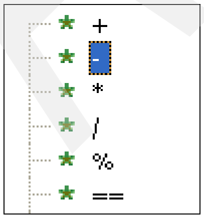


Figure 3-200 w’s context menu

c. From the same context menu, select **Number** as shown in Figure 3-201.

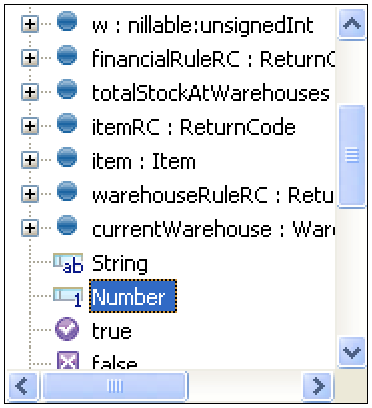


Figure 3-201 Select Number from context menu

d. At the **Type a number prompt**, enter 1 (as shown in Figure 3-202), and then press Enter.

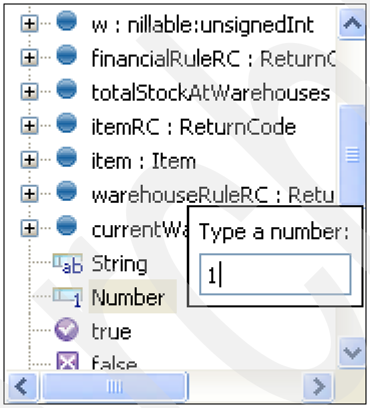


Figure 3-202 Set numeric value from context menu

e. Link **w - 1** to **get item at index** as shown in Figure 3-203.

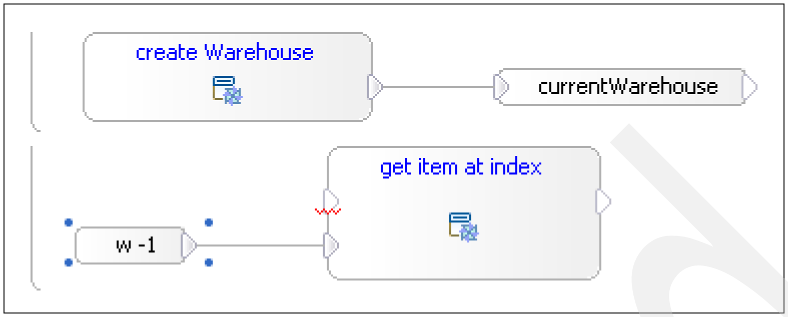


Figure 3-203 Link w - 1 to get item at index

6. Set the output of **get item at index** to go to the temporary variable **currentWarehouse** (Figure 3-204):

   a. Right-click **currentWarehouse**, and then select **Copy**.

   b. Click any where on the canvas, and select **Paste**.

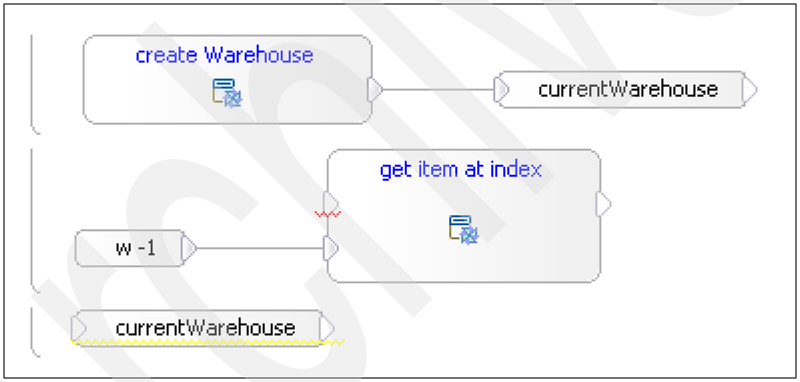


Figure 3-204 Adding another copy of currentWarehouse to the canvas

c. Wire **get item at index** to **currentWarehouse** a shown in Figure 3-205.

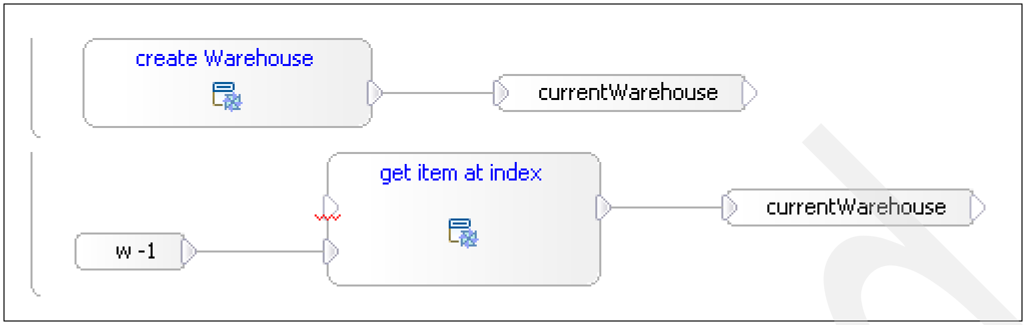


Figure 3-205 Set the output of get item at index

7. You also need to set the other input (the list) to **get item at index** (as shown in Figure 3-206). The list is available from item.warehouse. Follow these steps:

   a. Drag **item** onto the canvas.

   b. Using the context menu, set it to `item.warehouses`.

   c. Link **item.warehouses** to **get item at index**.

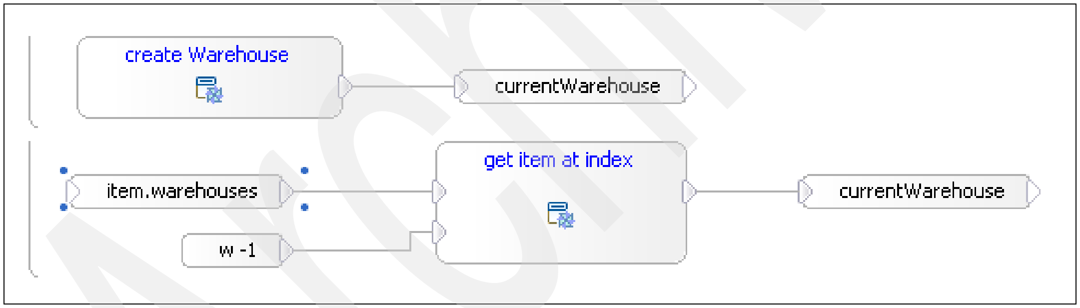


Figure 3-206 get item at index with all inputs and outputs set

8. The last thing that you need to do is to increment the totalStockAtWarehouses by the stock amount at the current warehouse (as shown in Figure 3-207 on page 213). Follow these steps:

   a. Add an expression to the canvas, and set it to `currentWarehouse.stock + totalStockAtWarehouses`.

   b. Add totalStockAtWarehouses to the canvas.

   c. Wire the two expressions.

9. Save the Project.

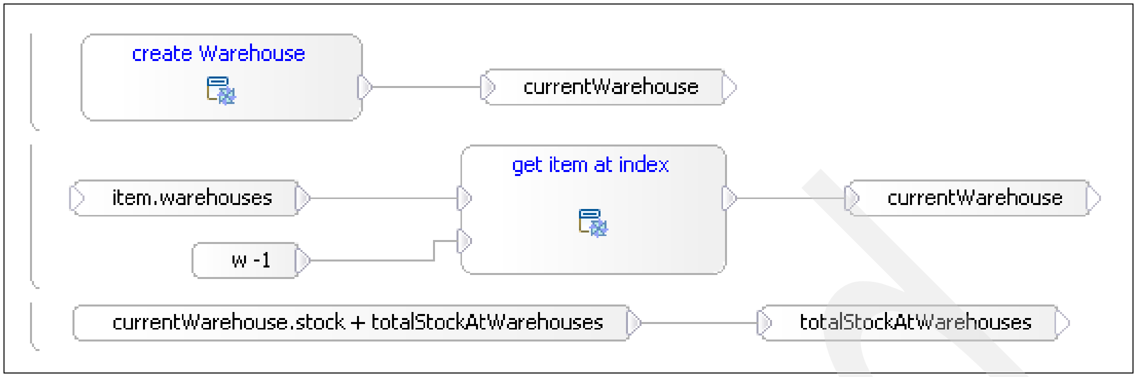


Figure 3-207 currentWarehouse.stock + totalStockAtWarehouses

At this point, there is one error in the assembly diagram. Recall that you added a reference partner to the process, but the assembly diagram of the project has the old version.

You can fix the error by following these steps:

1. Right-click **OrderManagementBP** in the assembly diagram and select **Synchronize Interfaces and References** → **from Implementation**.
2. Click **Yes**.

The assembly diagram now has an OrderManagementBP with six references and no errors, as shown in Figure 3-208.

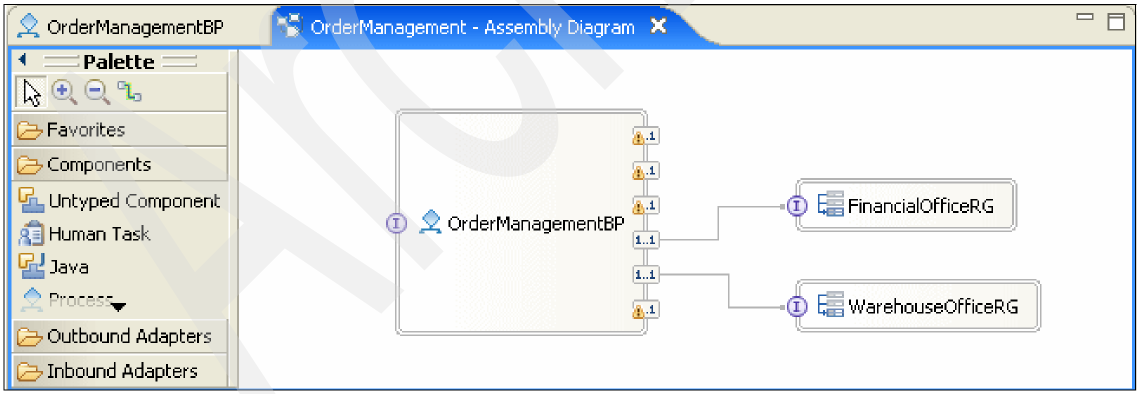


Figure 3-208 OrderManagement assembly diagram after adding warehouse split reference

This concludes the development steps for warehouse splitting and the ForEach activity. You can now test the process, which is described in the next section.

3.7.2 Testing the ForEach activity and warehouse split steps

In all previous tests, we concentrated on the approve, reject, and human decision use cases and their associated sample data. For this test, you test the approve use case and add another test (warehouse split), where the Warehouse Availability Mediation determines that a warehouse is down and cannot be used to fulfill the order. For this case, the warehouse might receive as an input an item with three warehouses, but the response has only an item with two warehouses.

Testing with the Approve input data

Test the module using the Approve input data and the settings shown in Figure 3-209. Use the following parameters:

- Initial request parameters: Import from OrderManagementInput_APPROVE.xml.
- Item output parameters: Import from item001.xml. Set `RC=0`.
- Customer output parameters: customer10001.xml. Set `RC=0`.
- Order output parameters: copied from Order input parameters. Set `RC=0`.

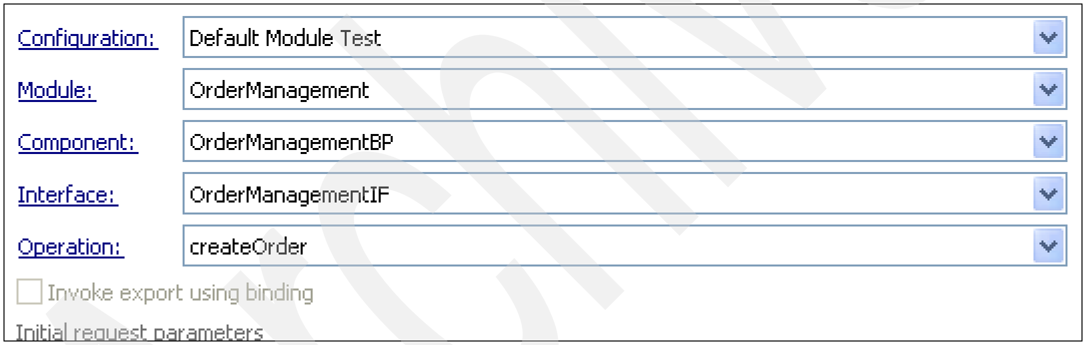


Figure 3-209 Approve input data

Before starting the test, make sure you have four emulation points as shown in Figure 3-210.

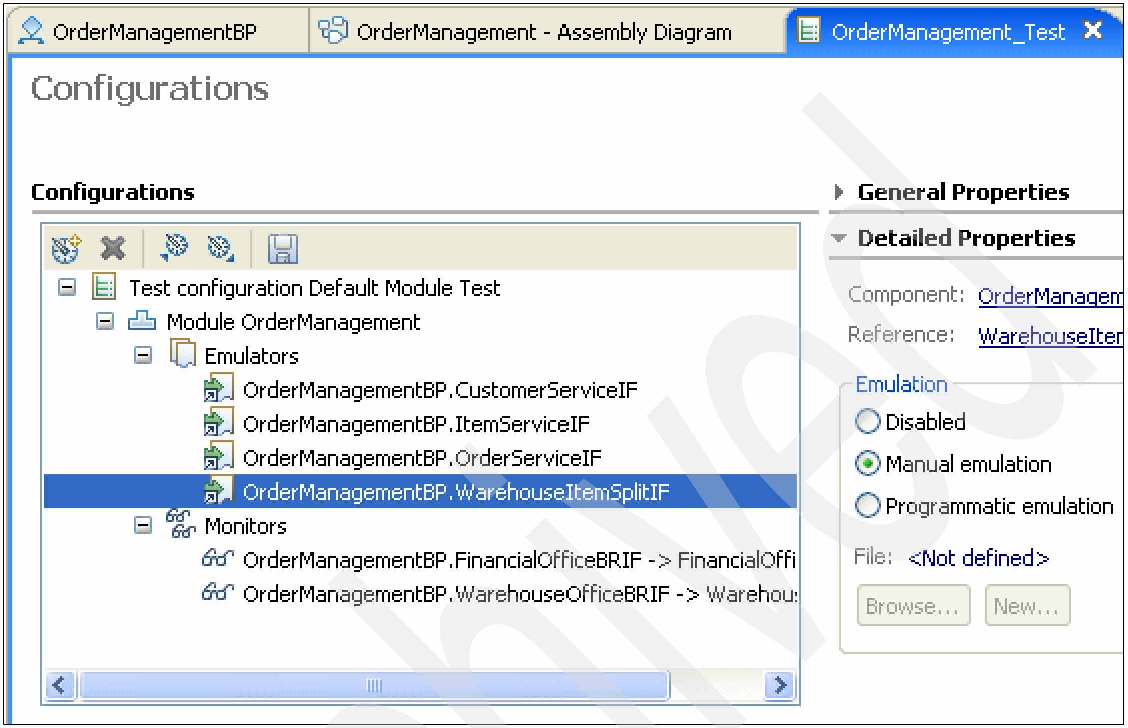


Figure 3-210 Configurations showing emulators for use case

To run the test:

1. Set the emulation values for Customer, Item, and Order. Then, the next emulation point is WarehouseItemSplitIF, as shown in Figure 3-211.

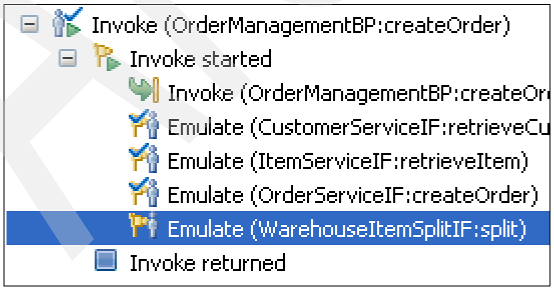
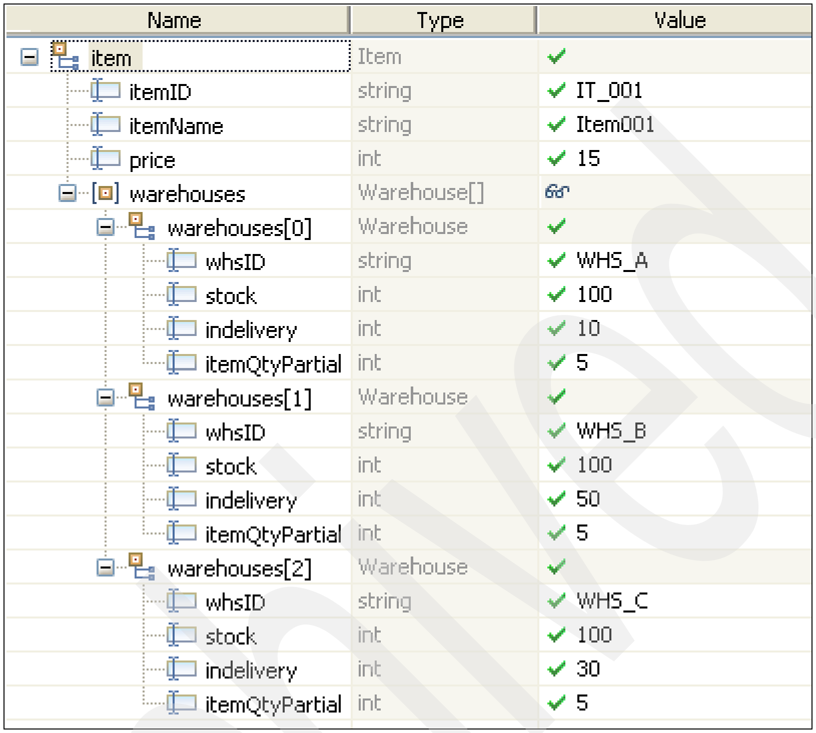


Figure 3-211 WarehouseItemSplitIF emulation point

2. Use the copy and paste method to fill the output parameter using the input parameter for item. See Figure 3-212.



Name	Type	Value
item	Item	✔
itemID	string	✔ IT_001
itemName	string	✔ Item001
price	int	✔ 15
warehouses	Warehouse[]	
warehouses[0]	Warehouse	✔
whsID	string	✔ WHS_A
stock	int	✔ 100
indelivery	int	✔ 10
itemQtyPartial	int	✔ 5
warehouses[1]	Warehouse	✔
whsID	string	✔ WHS_B
stock	int	✔ 100
indelivery	int	✔ 50
itemQtyPartial	int	✔ 5
warehouses[2]	Warehouse	✔
whsID	string	✔ WHS_C
stock	int	✔ 100
indelivery	int	✔ 30
itemQtyPartial	int	✔ 5

Figure 3-212 Item output parameters for approval case

3. The next emulation point is updateOrder. Click the **WarehouseOfficeRG** request to see that the totalStock was calculated and sent in as an input parameter, as shown in Figure 3-213.

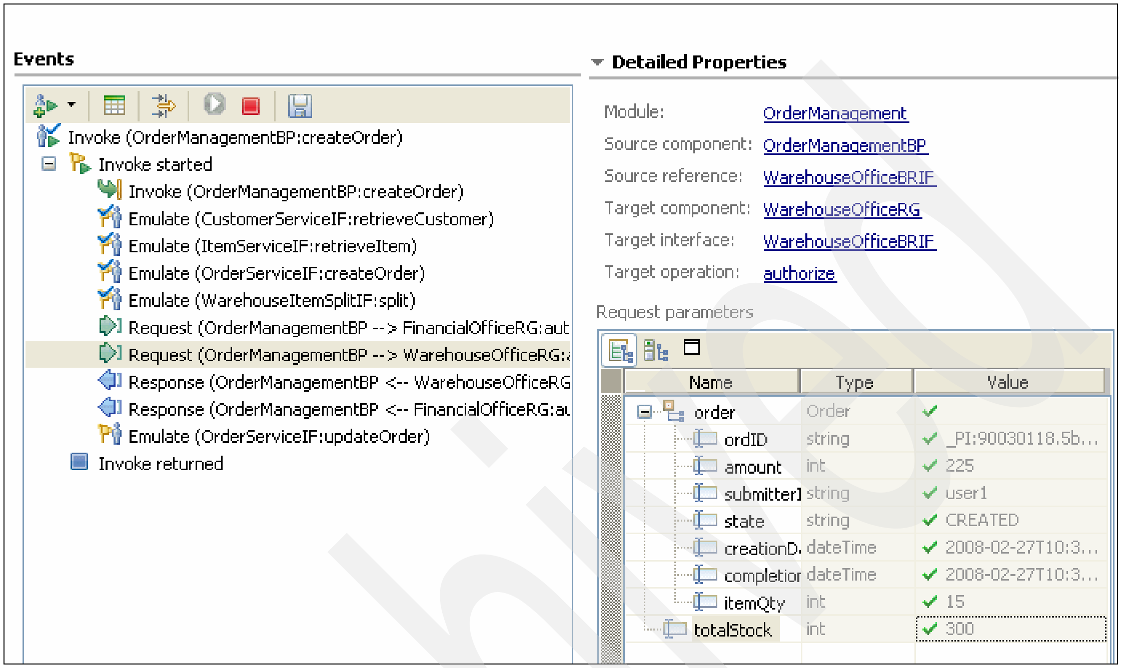


Figure 3-213 WarehouseOfficeRG request showing input parameters with totalStock value

After you run the scenario, you have completed the steps shown in Figure 3-214.

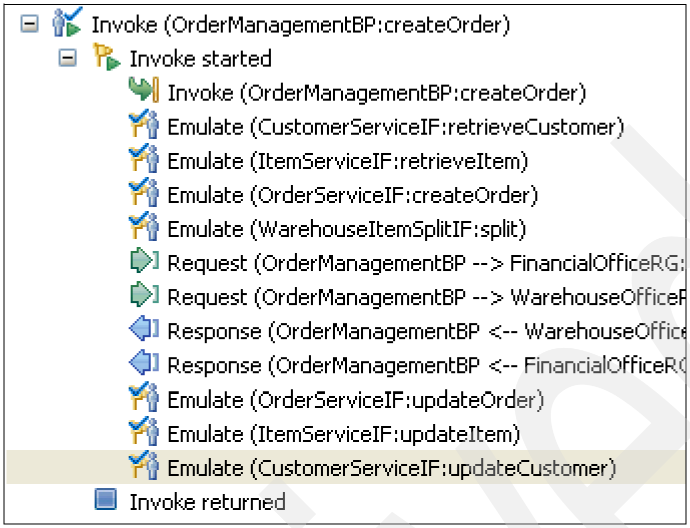


Figure 3-214 Approval execution steps

Testing with the warehouse split test

Run the next test using the orderMangementInput_SPLIT.xml input data, shown in Figure 3-215.

Figure 3-215 Warehouse split input data

To test with this data, follow these steps:

1. Make sure that you have the appropriate emulation points. Set the Approve case test emulation values for Customer, Item, and Order.
2. At the WarehouseItemSplitIF emulation point, modify the input parameters to show an unavailable warehouse with the following steps:
   a. Use the copy and paste method to fill the output parameter using the input parameter for item.
   b. Remove warehouses[0], which has values for WHS_A, by right-clicking **warehouses[0]** and selecting **Remove**.

c. Update itemQtyPartial for both WHS_B and WHS_C (Figure 3-216):

- set itemQtyPartial for WHS_B to 10.
- set itemQtyPartial for WHS_C to 9.

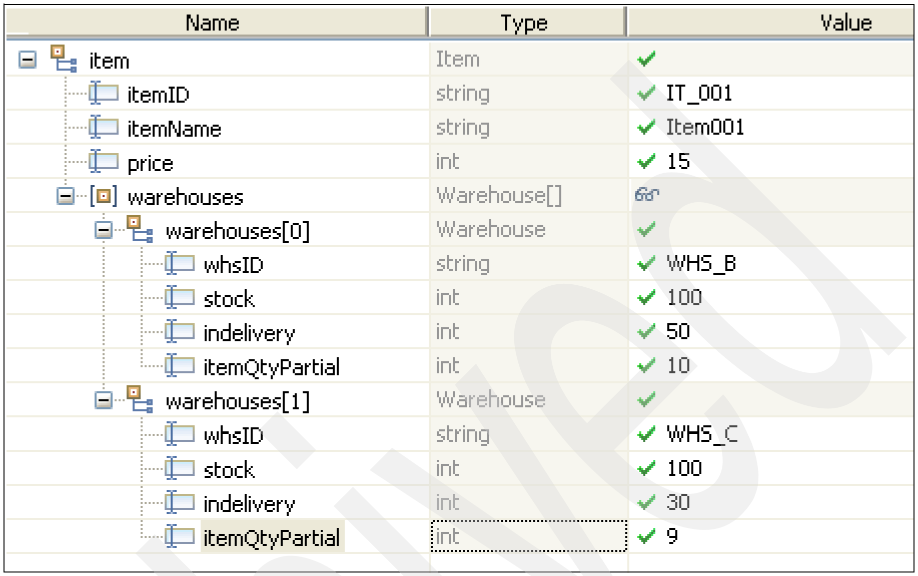


Figure 3-216 Item output parameters for warehouse split case

3. The next emulation point is updateOrder. Click the **FinancialOfficeRG** request to see that the item sent in as input parameter has only two warehouses, as shown in Figure 3-217.

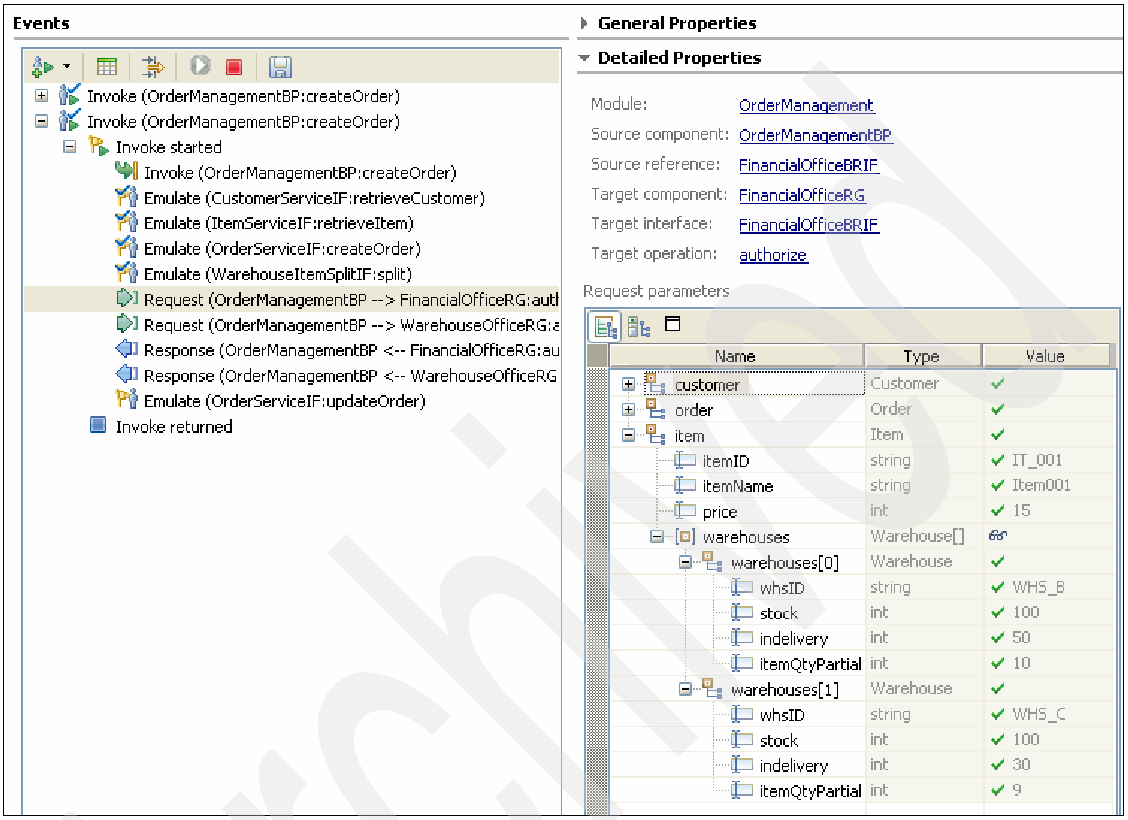


Figure 3-217 FinancialOfficeRG request showing input parameters with only two warehouses

4. After you run the scenario, you have completed the sequence shown in Figure 3-218.

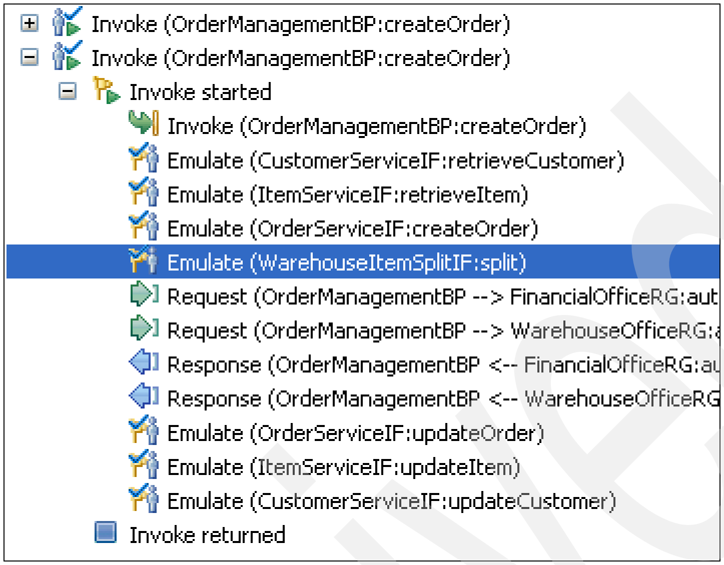


Figure 3-218 Warehouse split execution steps

You now move to the final development step in which you add e-mail notification.

Tip: If you check in your code to CVS, check in changes.

3.8 Implementing and testing the e-mail

The last step of the business process is to send an e-mail. This step is in between the **Final Verification** and **Log - End**, as shown in Figure 3-219.

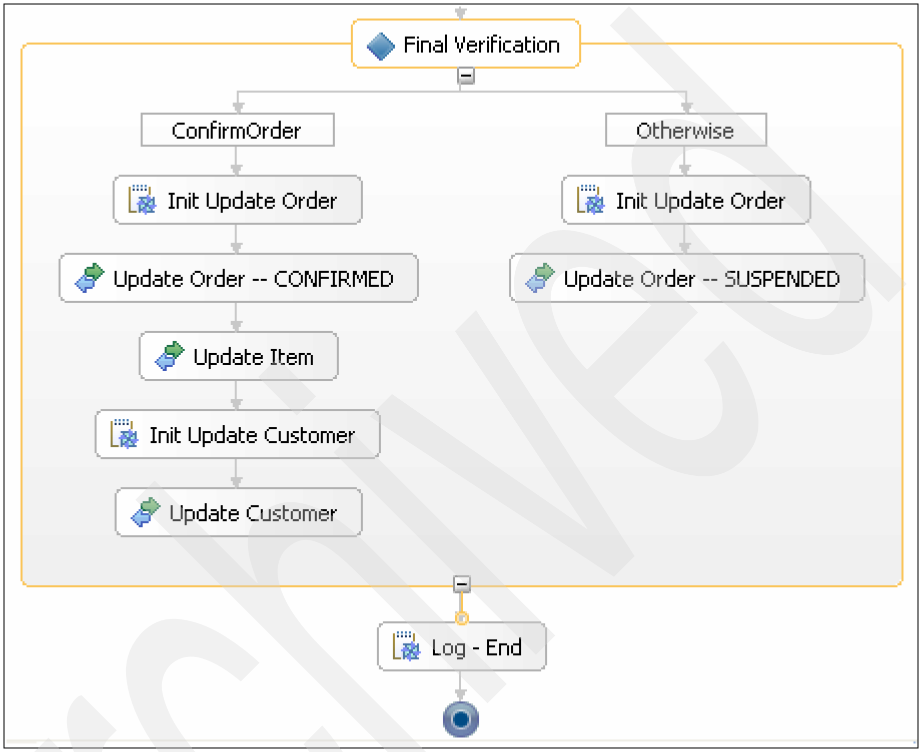


Figure 3-219 Last steps of the process

In previous steps, all the development was done within the OrderManagement module. However, in this test, you add another module for sending e-mail. You use the IBM WebSphere Adapter for Email to send e-mail, and you implement it within a module named *EmailOutbound*. To bridge any gaps between the EmailServiceIF in the library and the interface of the e-mail service, you use an interface map. Finally, you export this interface map so that it can be used by the OrderManagement module.

Within the OrderManagement module, you add a couple of activities in between **Final Verification** and **Log - End**:

- A Java snippet to set the value of the e-mail parameters (for example to, from, subject, and body)
- An Invoke activity that calls the EmailOutbound module using the parameters set in the snippet

The steps to follow include:

1. Develop the e-mail module
2. Integrate the e-mail module with the OrderManagementBP
3. Test the OrderManagementBP with the E-mail module

3.8.1 Developing the e-mail module

You develop the e-mail module in three parts:

1. First, you make use of a new feature in V6.1 that allows you to create a new outbound e-mail service from the Email pattern (New From Pattern).
2. Then, you develop an interface map to bridge the gap between the newly discovered e-mail service and the EmailServiceIF from the common library.
3. Finally, you create an export component for this interface.

> **Tip:** If you deploy your modules in the same cluster, exports and imports with an SCA binding are the least expensive in terms of performance.

Building a new e-mail service from a pattern

To build the new e-mail service:

1. Create a new module named `EmailOutbound` with OrderManagementLib as a dependency.
2. Use the New From Pattern wizard (shown in Figure 3-220 on page 224) to create the service:
   a. Select **File** → **New** → **From Patterns**.
   b. In the wizard, select from the list of available patterns **Integration** → **Adapters** → **Email** → **Create an outbound Email service to send mail.**
   c. Click **Next**.

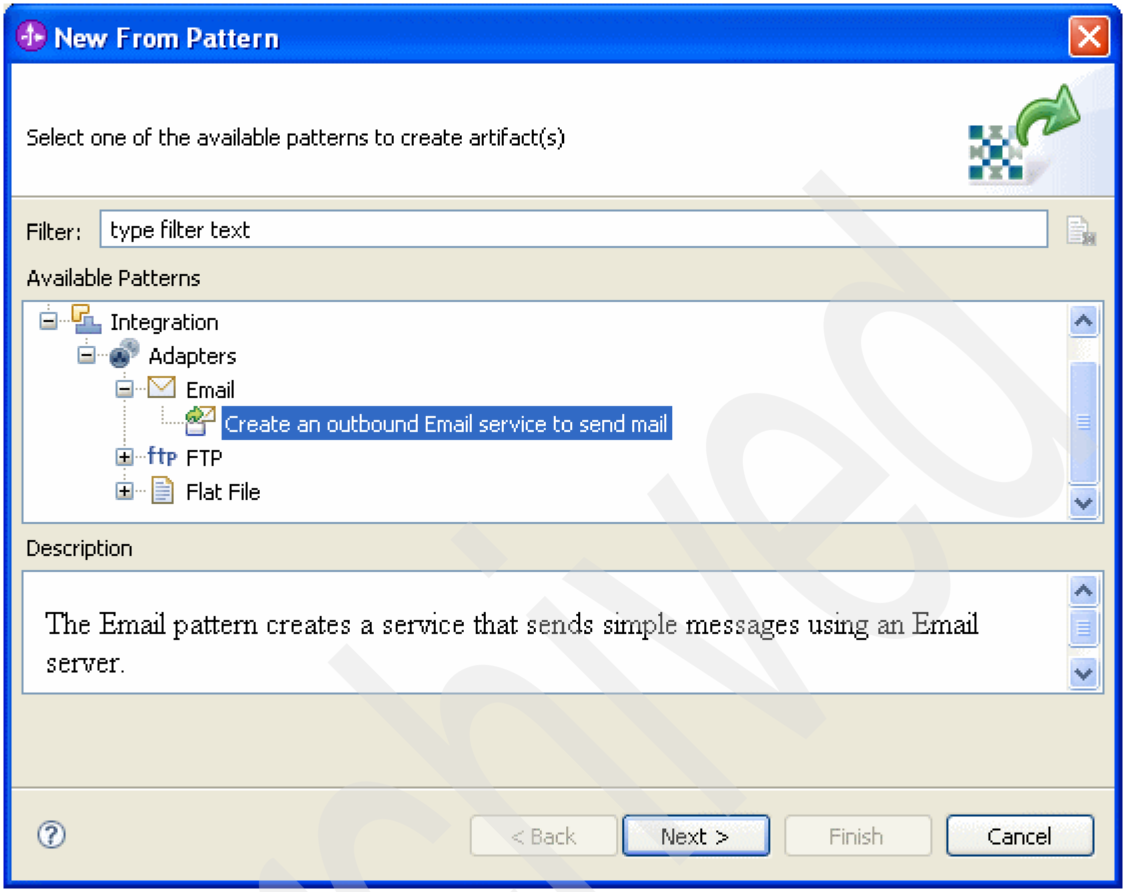


Figure 3-220 Email outbound service in New From Pattern

d. Select the EmailOutbound module, name the service `EmailImport`, then click **Next** as shown in Figure 3-221.

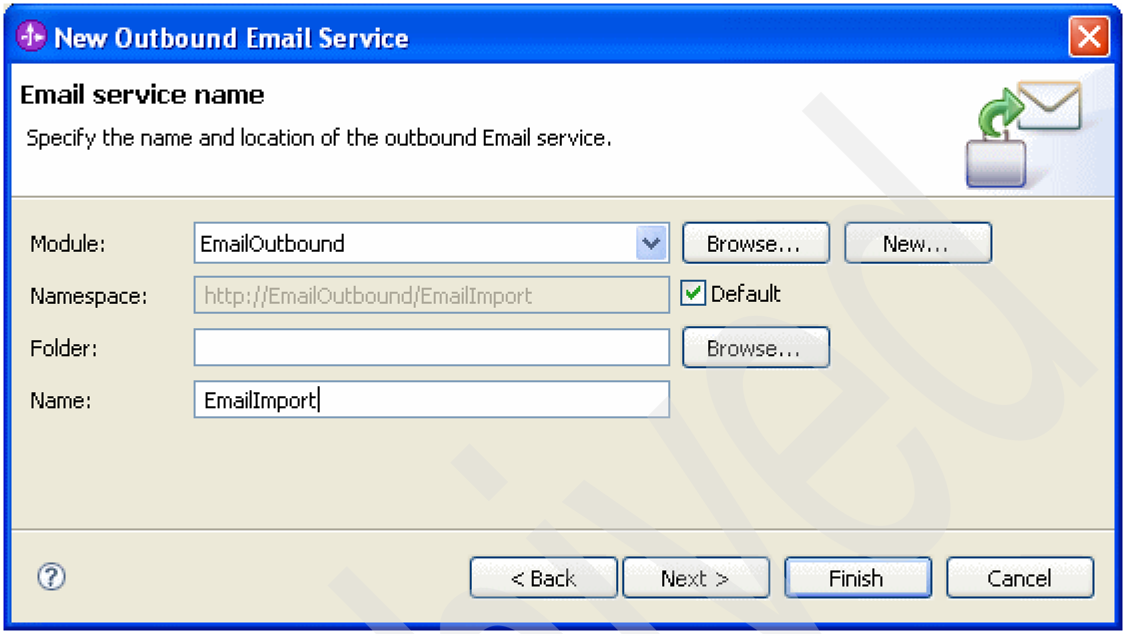


Figure 3-221 New Outbound Email Service wizard

e. Set the connection properties in order to test the connection (shown in Figure 3-222) as follows:

- For **Email server host name** enter the host name of your e-mail service.
- For **Port** enter `25` or the port of your e-mail service.

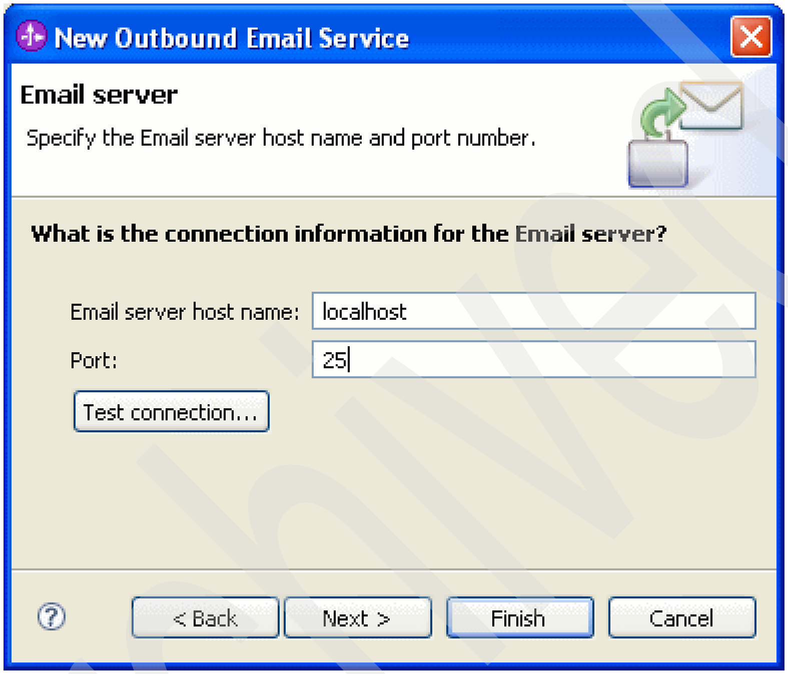


Figure 3-222 Email server information

f. Click **Test connection**. Assuming that the connection is successful, click **OK**. Otherwise, check the connection properties and re-test the connection.

> **Important:** In this example, we show how to create an e-mail without using an existing JAAS alias, but note that this method is not the recommended practice due to passwords that are stored in clear text. The best practice is to create a JAAS alias before starting this New From Pattern and then use that alias instead of the following steps that we show here.

3. Set security **At the Email server security credential** wizard as follows:
   a. Select **Using user name and password**.
   b. Enter `admin` for the user name and `admin` for the password.
   c. Click **Finish**.
4. When the wizard completes, you have an e-mail adapter project, CWYEM_EMail, and the EmailOutbound has additional artifacts from the discovery, as shown in Figure 3-223.

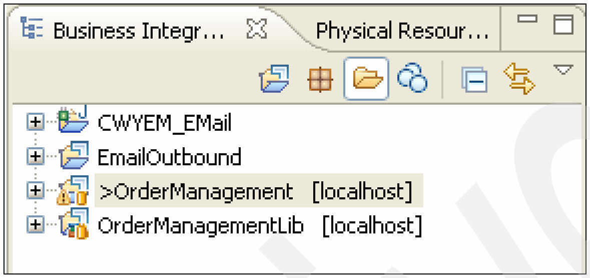


Figure 3-223 Business Integration perspective after New from Pattern

5. Open the EmailOutbound assembly diagram and note the EmailImport (shown in Figure 3-224).

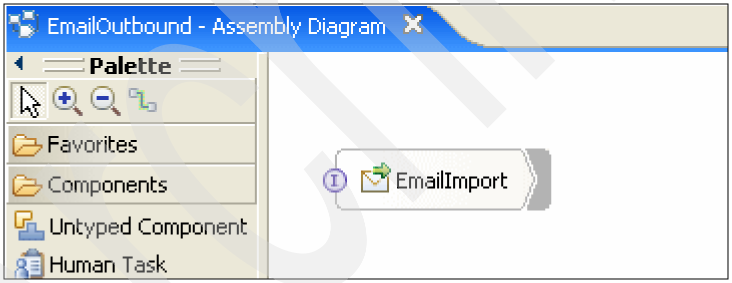


Figure 3-224 EmailOutbound assembly diagram

6. To see the business object that the adapter uses to send e-mail, expand **EmailOutbound** → **Data Types** → **Common Schemas**. Double-click **SimpleAlertEmail** (Figure 3-225).

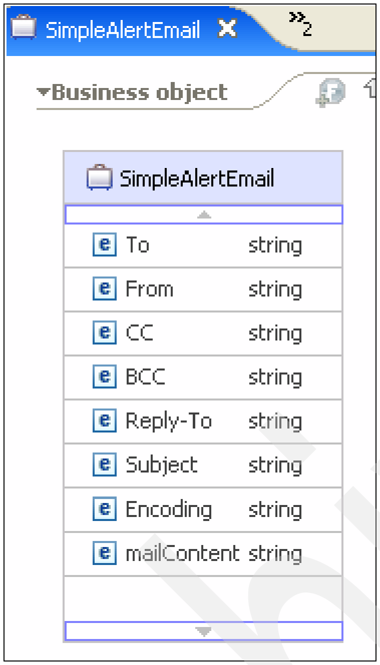


Figure 3-225 SimpleAlertEmail business object

7. Likewise, if you expand the **EmailOutbound** → **Interfaces** and double-click **EmailImport**, you can see the interface that the adapter uses to send e-mail, as shown in Figure 3-226.

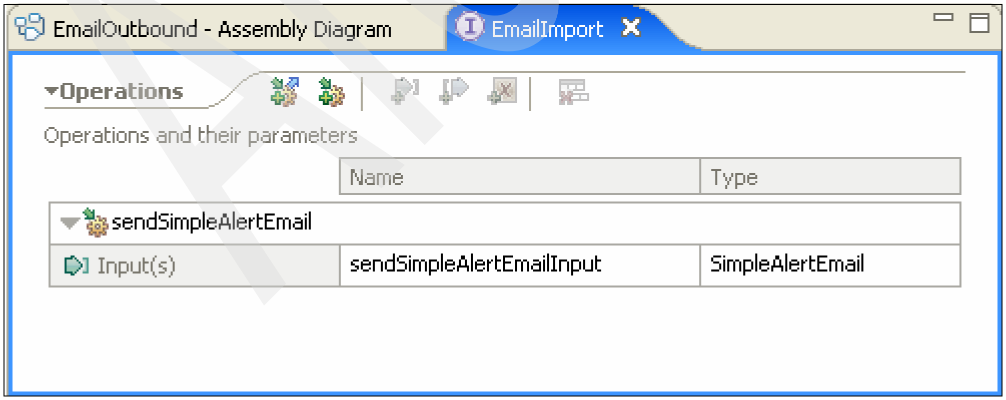


Figure 3-226 EmailImport interface

This concludes creating an e-mail service. To allow other modules to use this service, you need to create an interface map and an export the interface map. This export allows other modules with access to the common library to use this e-mail service.

Implementing the interface map

In this section, we explain how to create the interface map. In the library, you have the EmailOrder business object (shown in Figure 3-227).

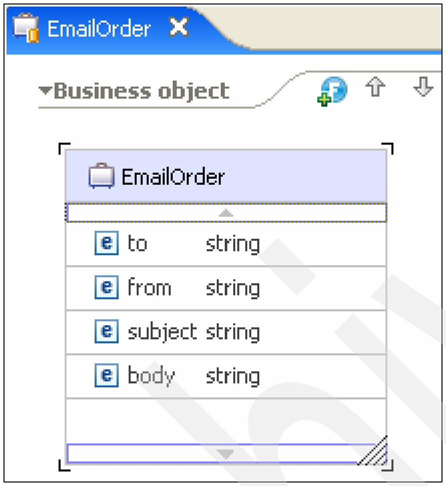


Figure 3-227 EmailOrder business object

You also see the EmailServiceIF interface (Figure 3-228).

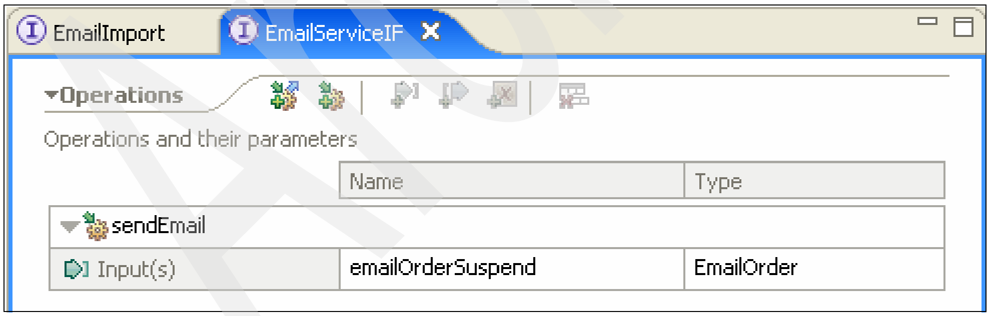


Figure 3-228 EmailService interface

If you compare the two e-mail business objects, they have similar fields. Likewise the interfaces are both one-way operations with one input. Creating an interface map from EmailServiceIF (source) to EmailImport (target) allows the OrderManagementBP to send e-mails using the e-mail service.

To create the interface map:

1. Create an interface map EmailServiceIF_to_EmailImport_IMap following these steps:

   a. Expand the EmailOutbound module and right-click **Mapping**. Select **New** → **Interface Map**.

   b. Enter for Name `EmailServiceIF_to_EmailImport_IMap` as shown in Figure 3-229, then click **Next**.

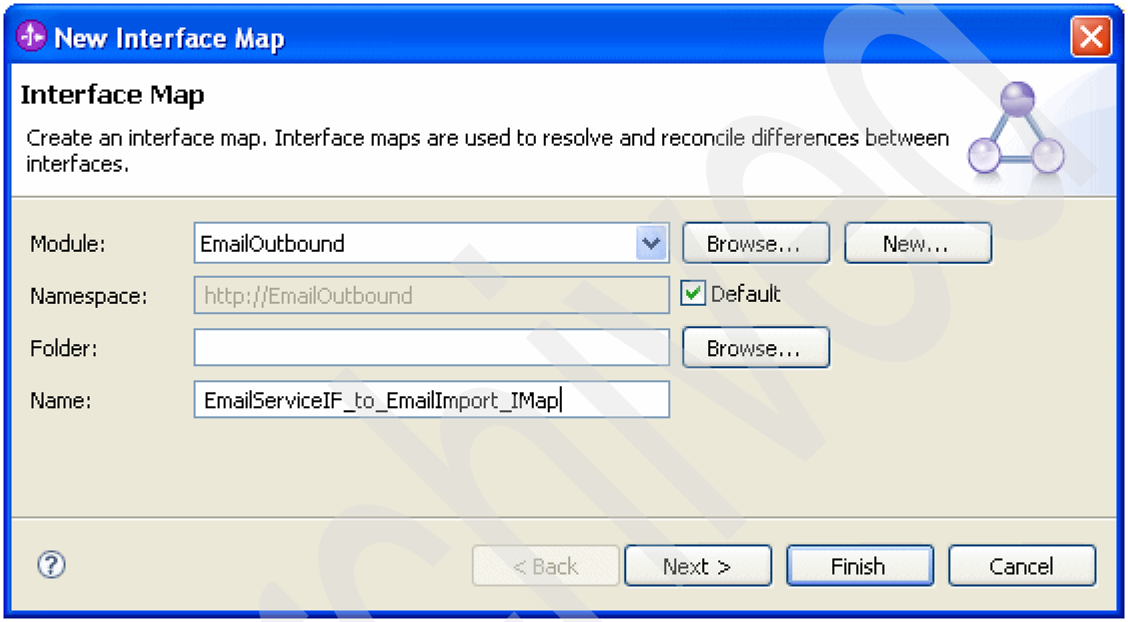


Figure 3-229 New Interface Map wizard

c. Set the source and target interfaces. Use **Browse** to select `EmailServiceIF` as the source interface and to select `EmailImport` as the target interface. Then, click **Finish**. See Figure 3-230.

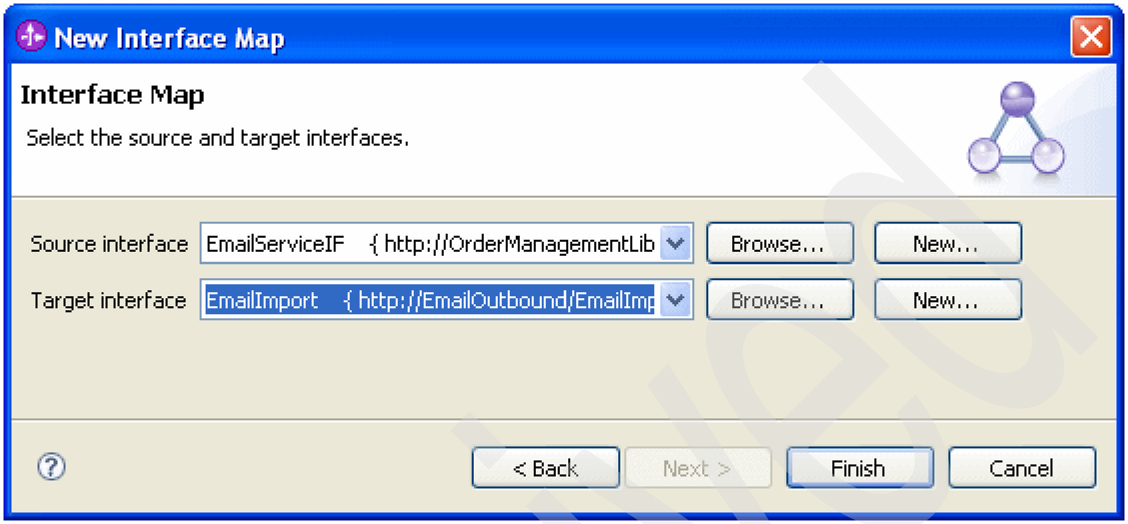


Figure 3-230 New Interface Map wizard source and target selection

2. Define the operation mapping by adding a link from `EmailServiceIF`'s sendEmail to `EmailImport`'s sendSimpleAlertEmail as shown in Figure 3-231.

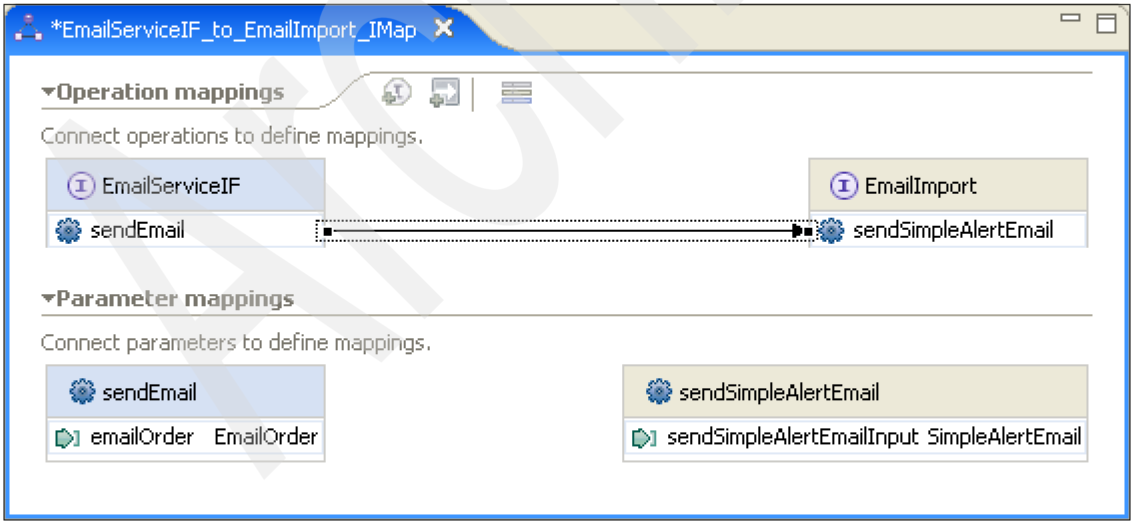


Figure 3-231 sendEmail to sendSimpleAlertEmail mapping

3. Define the parameter mapping between the input parameters of the operations by adding a link from `emailOrder` to `sendSimpleAlertEmailInput` as shown in Figure 3-232.

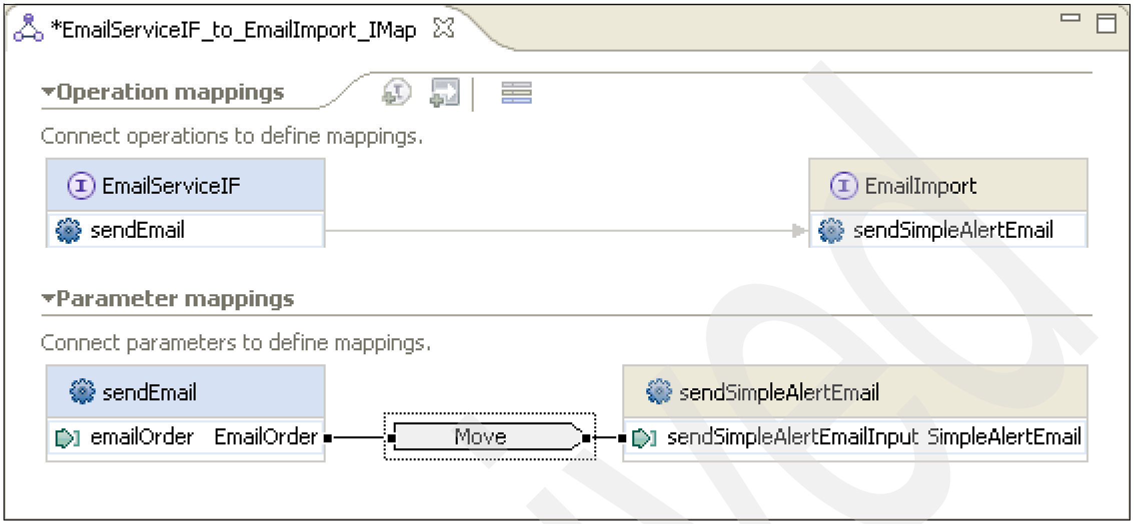


Figure 3-232 emailOrder to sendSimpleAlertEmailInput parameter mapping

> **Note:** The default operation mapping is *Move*, which is appropriate if the parameter type is of the same type. In this example, the parameter types are different business objects, so you need to create a business object map.

4. Go to the Properties view and change the Parameter Mapping Type to Map as shown in Figure 3-233.

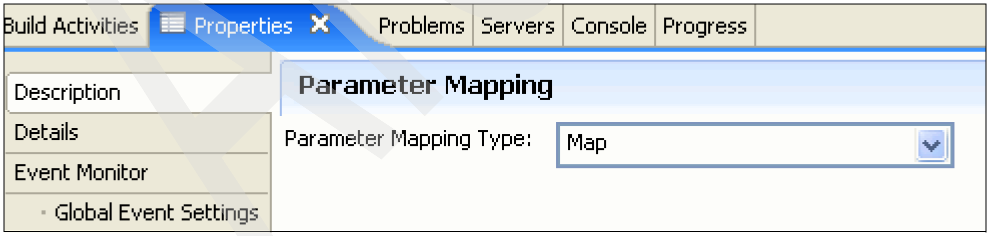


Figure 3-233 Parameter Mapping of type Map

5. In the Details tab, click **New** (Figure 3-234).

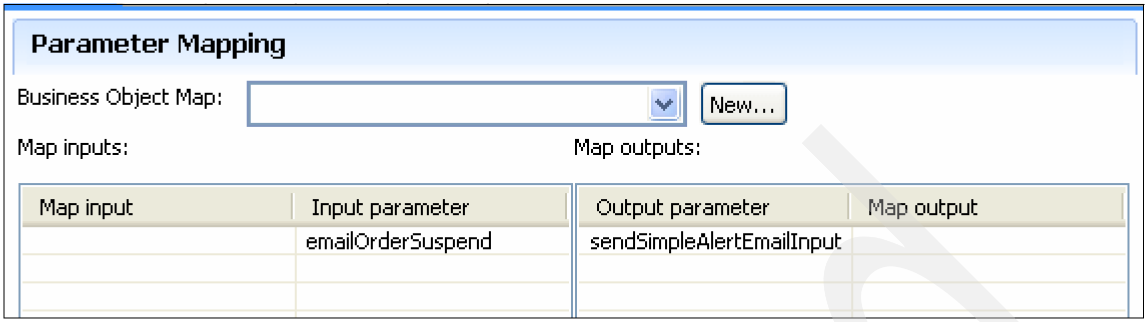


Figure 3-234 Details Parameter Mapping

6. At the New Business Object map wizard, enter `EmailOrder_to_SimpleAlertEmail_BOMap` for the name, and then click **Finish** as shown in Figure 3-235.

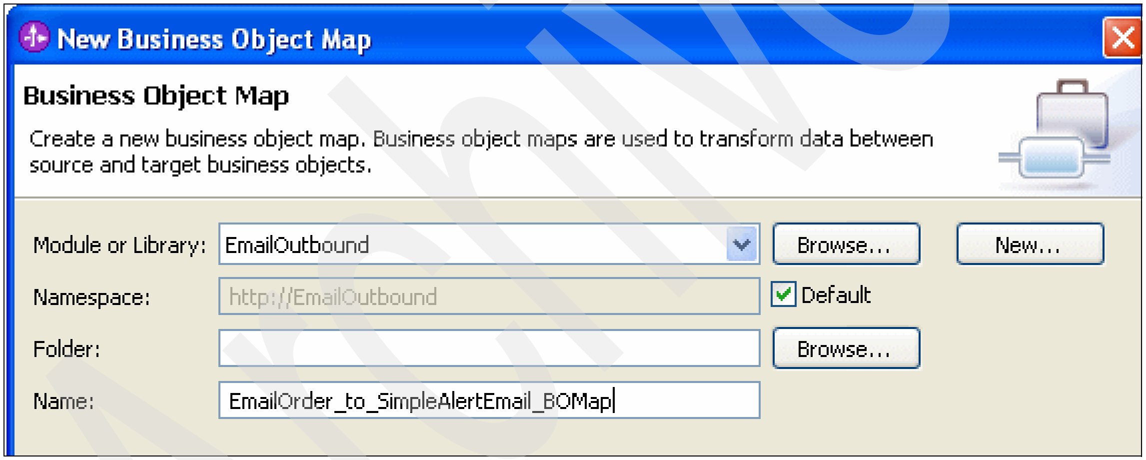


Figure 3-235 New Business Object Map

7. Verify that the interface map detail properties look as shown in Figure 3-236.

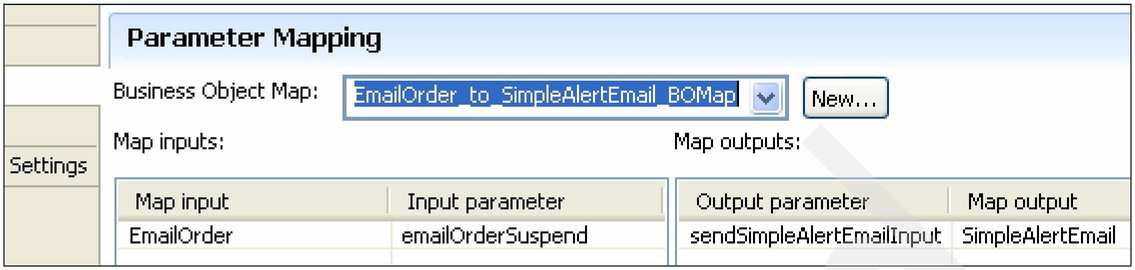


Figure 3-236 Interface Map's Details

8. Verify the mappings look as shown in Figure 3-237, and then save the interface map.

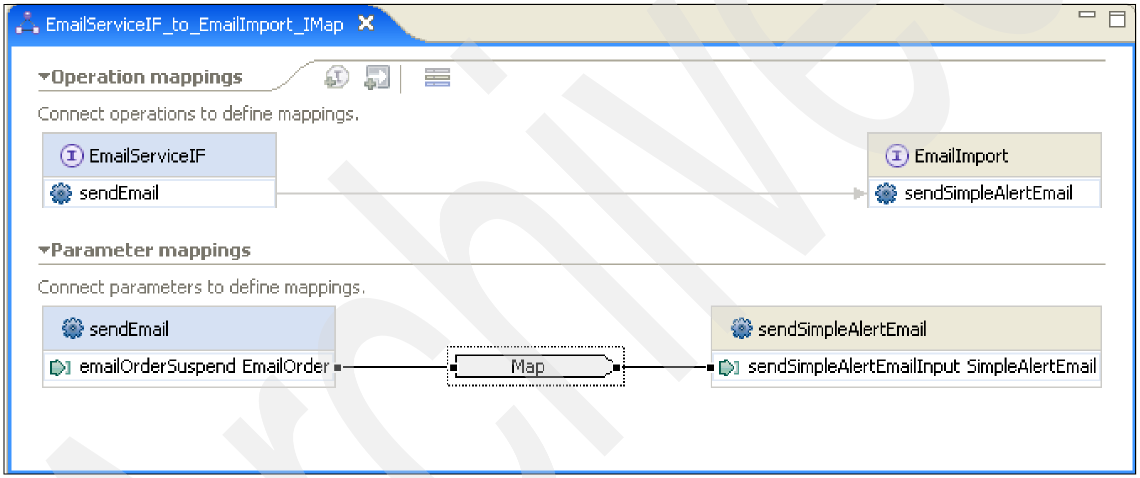


Figure 3-237 EmailServiceIF_to_EmailImport_IMap

9. Complete the business object map as follows:

   a. Click **EmailOrder**, and then click **Map similar fields.**

   b. Click **OK** to continue with the default mapping settings. Mappings are created between the three fields with the same name. See Figure 3-238.

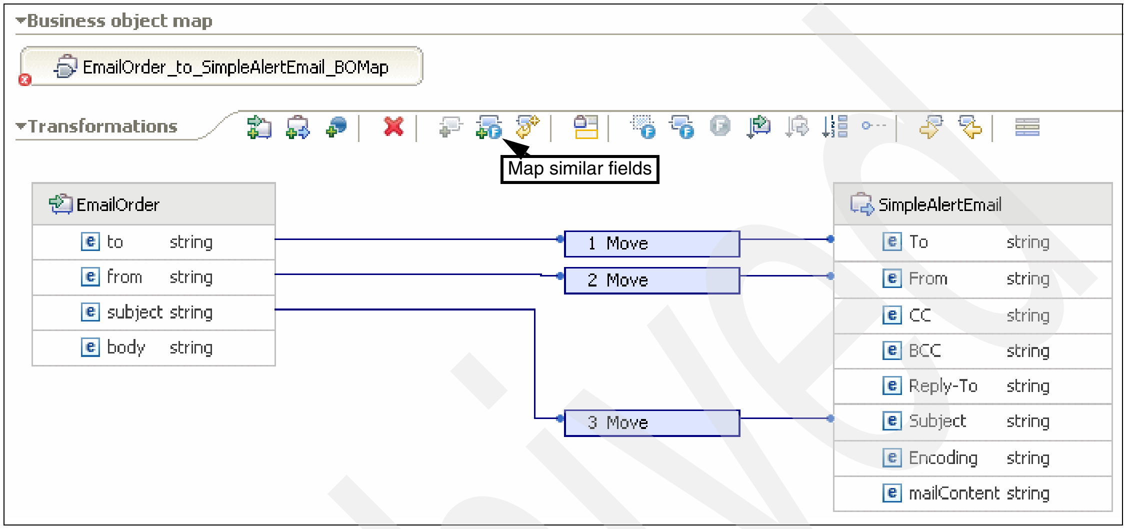


Figure 3-238 Similar fields mapping

   c. Map **body** to **mailContent** using a Move action.

   d. Set SimpleAlertEmail Encoding's field to UTF-8 by right-clicking **Encoding** and selecting **Create Transform** → **Assign**. Then, go to Properties and select **User defined value**, and enter `UTF-8`. See Figure 3-239.

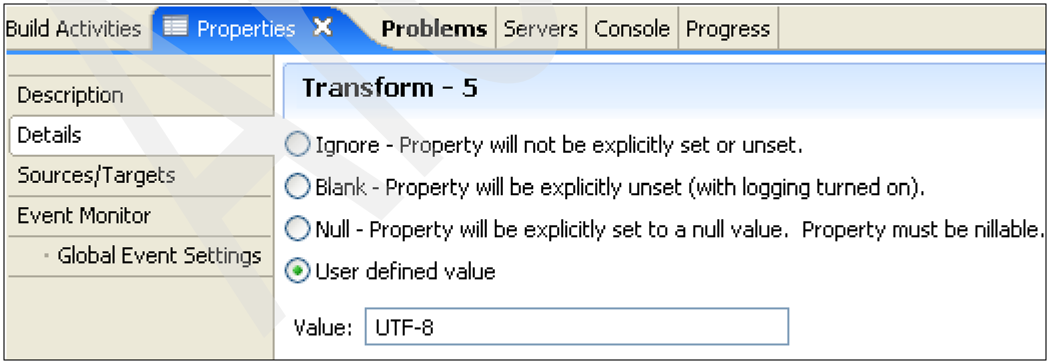


Figure 3-239 Business object assign user-defined value

   e. Save the business object map. There are no errors in the Problems view.

10. Now that you have completed the interface map and its associated business object map, you add the interface map to the assembly diagram as follows:

   a. Open the EmailOutbound assembly diagram.

   b. Drag the EmailServiceIF_to_EmailImport_IMap onto the assembly diagram.

   c. Right-click EmailImport and select **Wire to Existing**. See Figure 3-240.

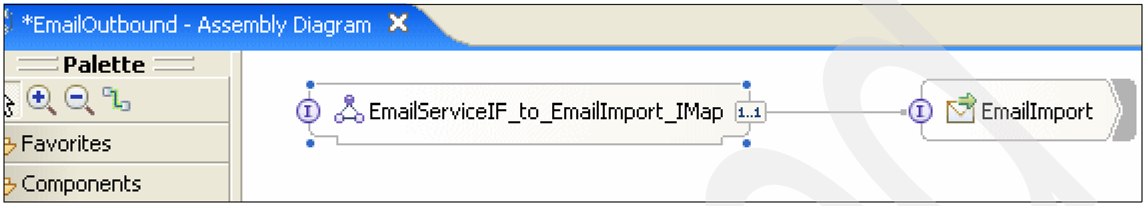


Figure 3-240 EmailOutbound assembly diagram with interface map

Generating an export with an SCA binding

Create an export for the interface map as follows:

1. Right-click **EmailServiceIF_to_EmailImport_IMap** and select **Generate Export → SCA Binding** as shown in Figure 3-241.

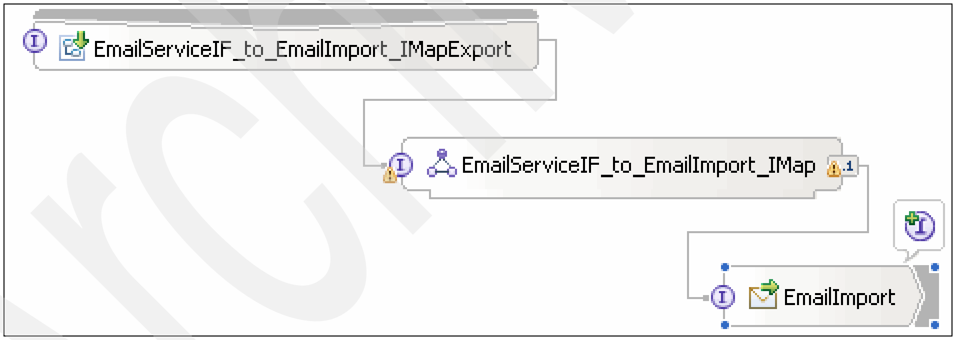


Figure 3-241 EmailOutbound assembly diagram with interface map and export

2. Save the assembly diagram. There are no errors in the Problems view.

With the export of the interface map, other modules can now use the e-mail service. We explain how to do this in the next section.

3.8.2 Integrating the E-mail Module with the OrderManagementBP

To use the e-mail service in the OrderManagementBP, you need to create an import component using the e-mail service's export component. You also need to add logic in the main process to set the e-mail parameters prior to invoking the e-mail service.

Adding an e-mail import to the Order Management assembly diagram

To add an e-mail import to the Order Management assembly diagram, follow these steps:

1. Open the OrderManagementBP assembly diagram.
2. Drag EmailServiceIF_to_EmailImport_IMapExport onto the OrderManagementBP assembly diagram.
3. At the Component Creation panel, select Import with SCA Binding, then click **OK**. See Figure 3-242.

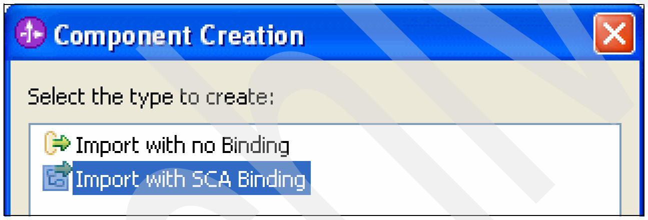


Figure 3-242 Component Creation prompt

4. The import component displays on the assembly diagram. Change the name, prior to saving, to `ImportEmail`, as shown in Figure 3-243. Otherwise, you need to refactor to change the name.

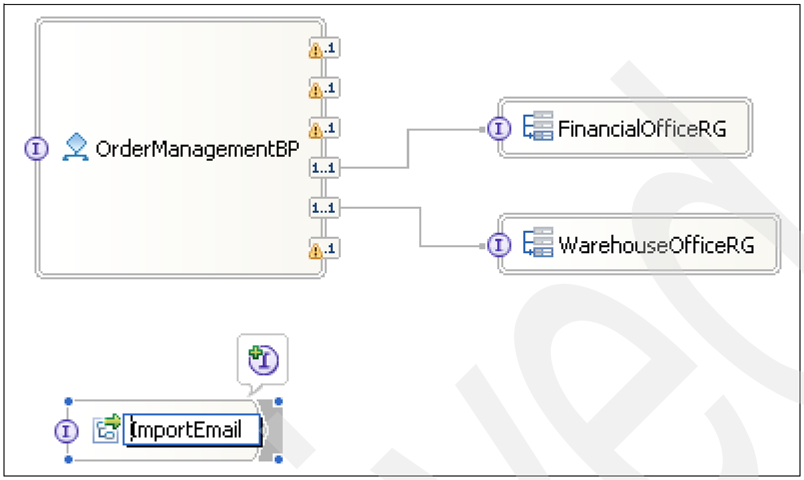


Figure 3-243 Assembly diagram with e-mail import

5. Save the assembly diagram. There are no errors in the Problems view.

Adding e-mail logic to OrderManagementBP

In terms of development, the final step is to set the e-mail parameters and then invoke the e-mail service in the business process as follows:

1. Add a reference partner to the OrderManagementBP by dragging EmailServiceIF in the OrderManagementLib to the list of reference partners as shown in Figure 3-244.

Figure 3-244 OrderManagementBP's reference partners

2. Set the e-mail parameters in the business process by adding a variable to the process and naming it `emailOrder`. Set its type to EmailOrder.

3. Then, add a Java snippet above Log - End and name it `Init e-mail` as shown in Figure 3-245.

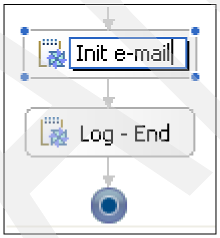


Figure 3-245 Init e-mail

4. Set the code for the `Init e-mail` using the snippets that are provided in the additional materials (see Figure 3-246 on page 241). The code mainly tailors the e-mail's subject and body based on the financial and warehouse approval results.

Additional material: The Java code for this snippet is included in BusExampleSnippets\InitEmail.txt. See Appendix B, "Additional material" on page 461.

```
/*************** INITIALIZE EmailOrder BEFORE FIRST USE ***************/
commonj.sdo.DataObject __result__1;
{// create EmailOrder
   com.ibm.websphere.bo.BOFactory factory =
      (com.ibm.websphere.bo.BOFactory) new
com.ibm.websphere.sca.ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");
    __result__1 = factory.create("http://OrderManagementLib","EmailOrder");
}
emailOrder = __result__1;

/*************** SET THE FROM & TO ***************/
emailOrder.setString("to", orderManagementInput.getString("submitterEmail") );
//change the from email to your own sender's e-mail address
emailOrder.setString("from", "admin@kcg16hw.itso.ral.ibm.com");

/*************** SET THE SUBJECT ***************/
String ordID = order.getString("ordID");
String subjectType = ( financialRuleRC.getString("RC").equalsIgnoreCase("REJECT")
   && warehouseRuleRC.getString("RC").equalsIgnoreCase("REJECT") ) ? "rejected" : "accepted";
emailOrder.setString("subject", new String("Order: "+ordID+" has been "+subjectType));

/*************** SET THE EMAIL CONTENT ***************/
StringBuffer sb=new StringBuffer();
String fResult = financialRuleRC.getString("RC").equalsIgnoreCase("REJECT") ? "rejected" : "accepted";
sb.append("Order has been "+fResult+" by Financial Officer \n");
sb.append("---- Customer & Order information ---- \n");
sb.append("Customer ID: " + customer.getString("custID") + " \n");
sb.append("Customer name: " + customer.getString("custName") + " \n");
sb.append("Order amount: " + order.getInt("amount") + " \n");
sb.append("Budget: " + customer.getInt("budget") + " \n");
sb.append("Sold To Date: " + customer.getInt("soldToDate")+ " \n");
sb.append(" \n");
sb.append(" \n");
String wResult = warehouseRuleRC.getString("RC").equalsIgnoreCase("REJECT") ? "rejected" : "accepted";
sb.append("Order has been "+wResult+" by Warehouse Officer \n");
sb.append("---- Order & Item information ---- \n");
sb.append("Item ID: " + item.getString("itemID") + " \n");
sb.append("Item name: " + item.getString("itemName")+" \n");
sb.append("Order itemQty: " + order.getInt("itemQty")+" \n");
sb.append("Item total stock: " + totalStockAtWarehouses.intValue() );
sb.append(" \n");
sb.append(" \n");
sb.append(" \n");
sb.append(" \n");
sb.append(" \n");
sb.append(" \n");
sb.append("#### Email sent by a system id. Don't reply to it ####");
emailOrder.setString("body", new String(sb.toString()) );
```

Figure 3-246 Init e-mail logic

5. Save the business process. There are no errors in the Problems view. If you do have errors, make sure that you created the variable and then set the type correctly.

6. Add the e-mail invocation to the business process as follows:

   a. Add an Invoke activity between **Init e-mail** and **Log - End** and name it `Send e-mail to submitter`.

   b. Set Send e-mail to submitter's Properties as shown in Figure 3-247.

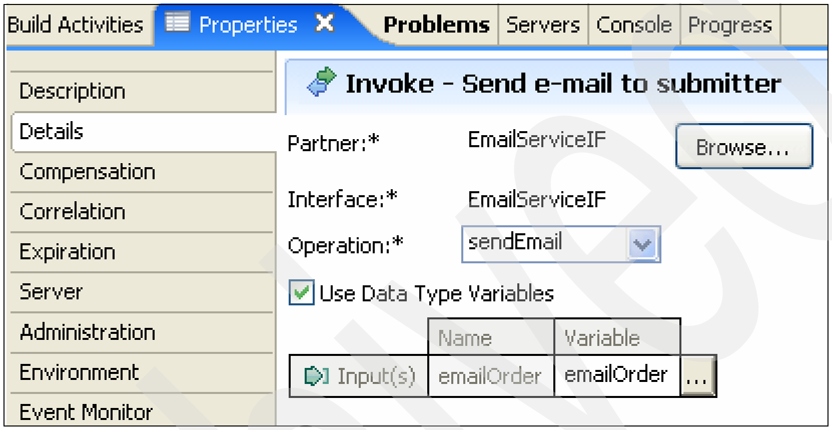


Figure 3-247 Send e-mail to submitter's Properties

   c. Verify the process logic as shown in Figure 3-248.

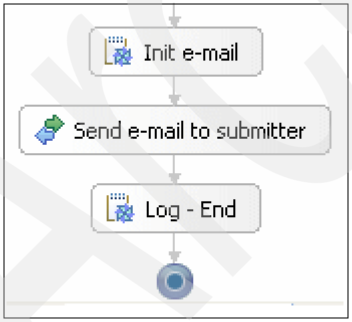


Figure 3-248 OrderManagementBP with e-mail logic

   d. Save the business process. You see one error in the Problems view that is related to the assembly diagram having an old version of the business process.

e. Fix the assembly diagram's error by right-clicking **ImportEmail** and selecting **Wire to Existing** as shown in Figure 3-249.

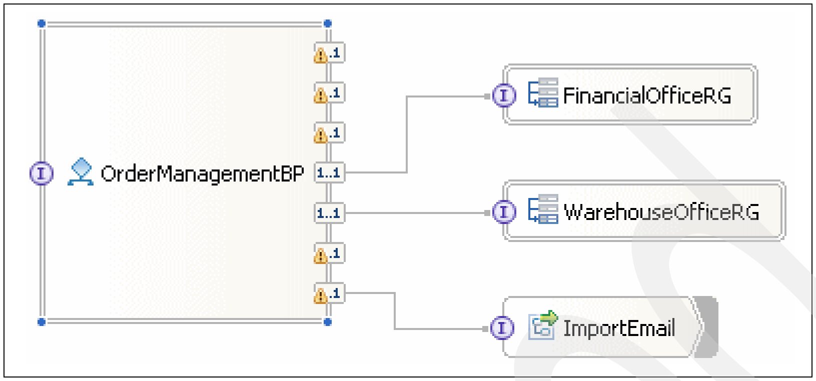


Figure 3-249 OrderManagement assembly diagram wired to e-mail import

7. Save the assembly diagram. Three are no errors in the Problems view.

You now test the integration solution that you developed.

3.8.3 Testing OrderManagementBP with e-mail

You can now test the business process with the two pertinent test cases, APPROVED and REJECTED. For the APPROVED test case, the process sends an e-mail confirmation, and for the REJECTED, an e-mail rejection.

> **Important:** When testing more than one component, it is important to use the **Add and Remove Projects** to make sure that all your related modules are deployed prior to testing using the integration test client. The integration test client will not deploy any undeployed modules that you might have forgotten to deploy prior to testing.

Using the Approve Order test

Test the module using the Approve input data and the settings shown in Figure 3-250 on page 244. Use the following parameters:

- Initial request parameters: Import from OrderManagementInput_APPROVE.xml.
- Item output parameters: Import from item001.xml. Set `RC=0`.
- Customer output parameters: customer10001.xml. Set `RC=0`.
- Order output parameters: copied from Order input parameters. Set `RC=0`.

Figure 3-250 Approve input data

After running the scenario, note the emulation points for updateOrder, updateItem, and UpdateCustomer as shown in Figure 3-251.

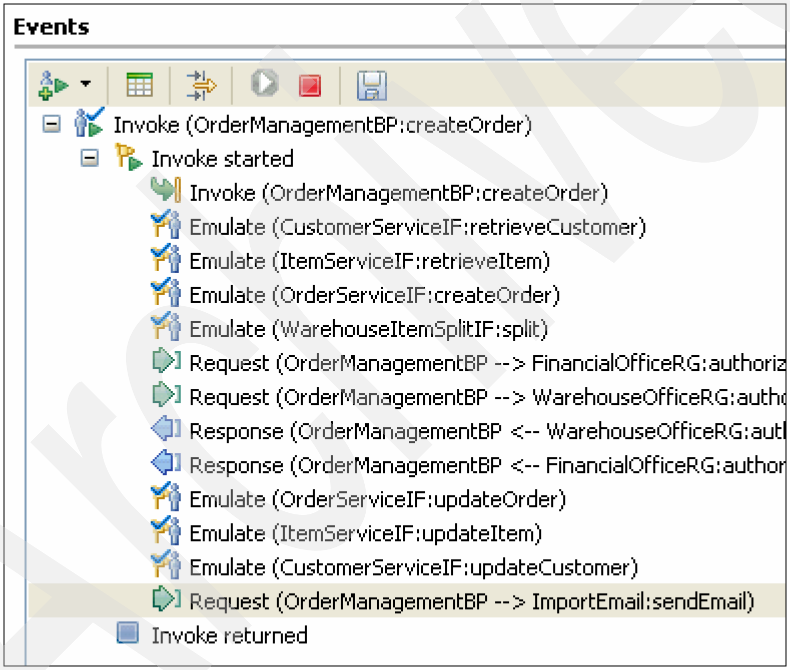


Figure 3-251 Approve order execution steps

If you click the request for ImportEmail you can also see the request inputs sent to the e-mail service, shown in Figure 3-252.

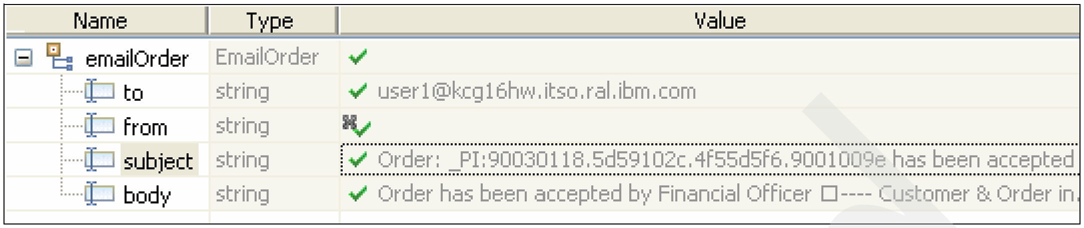


Figure 3-252 ImportEmail request parameter values

You can also see the usual log statements, including the e-mail adapter preparing and sending the e-mail on the console.

Using the Reject Order test

Test the module using the Reject input data. Use the following parameters:

- Initial request parameters: Import from OrderManagementInput_REJECT.xml.
- Item output parameters: Import from item002.xml. Set RC=0.
- Customer output parameters: customer10002.xml. Set RC=0.
- Order output parameters: copied from Order input parameters. Set RC=0.

After running the scenario, note the emulation points for updateOrder *only* as shown in Figure 3-253.

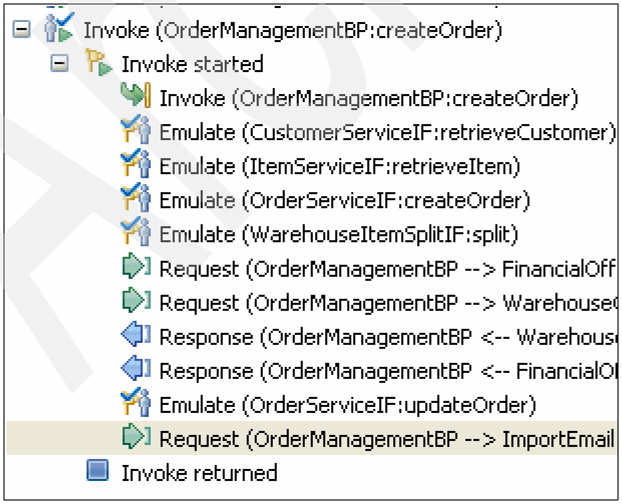


Figure 3-253 Reject order execution steps

If you click the request for ImportEmail, you can also see the request inputs sent to the e-mail service (Figure 3-254).

Figure 3-254 ImportEmail request parameter values

The process development is now complete.

Tip: If you check in your code to CVS, check in changes.

3.9 Testing the module components end-to-end

The last step before you test the WebSphere Process Server components with the WebSphere ESB components is to make a final test of the WebSphere Process Server components in isolation using the four test cases. The fifth test case (special orders) is only appropriate for when you are testing the WebSphere ESB components only or both WebSphere Process Server and WebSphere ESB components.

To test the module components from end-to-end, follow these steps:

1. Deploy the OrderManagement module and the EmailOutbound module.
2. Open the integration test client and make sure that you have the appropriate emulation points on.
3. Test the four test cases that we list in this section and use the usual emulation strategy and sample data for the emulation points.

Regular Order: Approve

Figure 3-255 shows the pre-processing input for the Regular Order: Approve test case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 15
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 3-255 Sample pre-processing input for Regular Order: Approve

Regular Order: Warehouse split

Figure 3-256 shows the pre-processing input for the Regular Order: Warehouse split test case.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 19
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 3-256 Sample pre-processing input for Regular Order: Warehouse split

Regular Order: Reject

Figure 3-257 shows the pre-processing input for the Regular Order: Reject test case.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10002
itemID	string	✔ IT_002
itemQty	int	✔ 10
submitterID	string	✔ user2
submitterEmail	string	✔ user2@kcg16hw.itso.ral.ibm.com

Figure 3-257 Sample pre-processing input for Regular Order: Reject

Regular Order: Human

Figure 3-258 shows the pre-processing input for the Regular Order: Human test case.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com

Figure 3-258 Sample pre-processing input for Regular Order: Human

You need to send an e-mail in each case.

4

OrderPreProcessingMediation module

This chapter guides you through the development and testing of the OrderPreProcessingMediation module. This testing involves the setup and use of the IBM WebSphere Adapter for Flat File (hereafter referred to as *the flat file adapter*).

This chapter includes the following topics:

- Building the mediation module and artifacts
- Using the flat file adapter
- Developing the mediation flow
- Testing the mediation

4.1 Building the mediation module and artifacts

As described in 1.2, “Scenario high-level design” on page 9, the OrderPreProcessingMediation module is invoked to:

1. Get an order from the client application
2. Determine the type of order and process accordingly:
   a. If the order is a regular order, the module forwards the request to the OrderManagement module, which implements the OrderManagementIF defined in the application high level design
   b. If the order is special, the module logs the order detail to a file using the flat file adapter

Figure 4-1 shows the processing flow for the mediation.

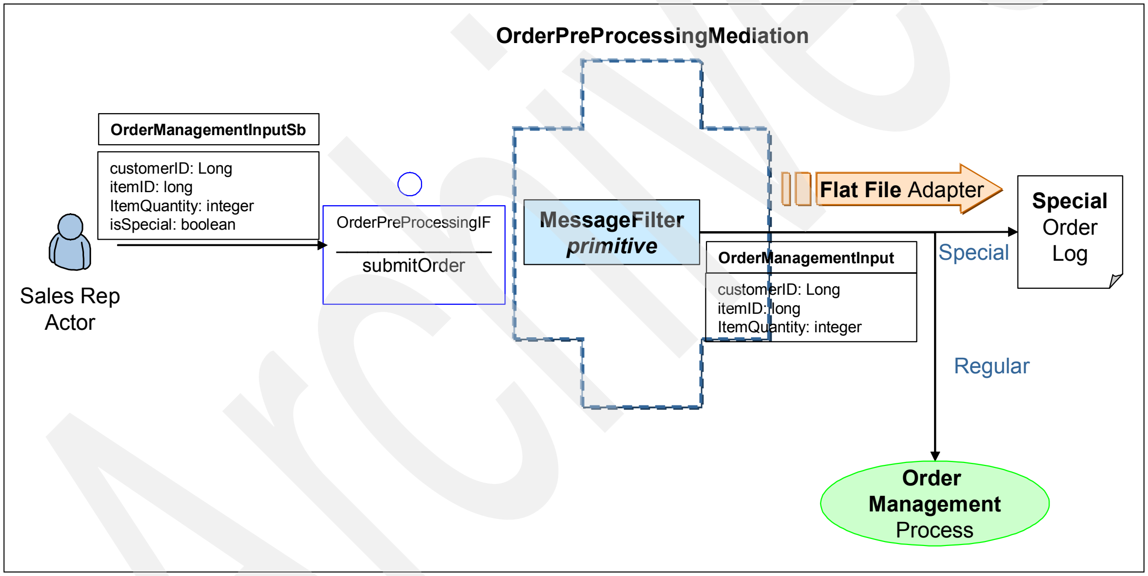


Figure 4-1 OrderPreProcessingMediation module flow

The first step in implementing the mediation flow is to create the module structure and base artifacts, which are:

- The mediation module: OrderPreProcessingMediation
- The incoming business object: OrderManagementInputSb

  This object extends the Order business object in the common library by appending an `isSpecial` flag. The flag is used to determine if the order is regular or special.

- The interface the mediation module must implement: OrderPreProcessingIF

  This interface has a one-way operation with OrderManagementInputSb as the input.

4.1.1 Creating the OrderPreProcessingMediation module

The first step in creating the mediation is to create the mediation module. To build a new mediation module, open the Business Integration perspective. Then, follow these steps:

1. Select **File** → **New** → **Mediation Module** from the top menu bar.
2. In the first panel (shown in Figure 4-2):
   a. Enter `OrderPreProcessingMediation` as the name for the module.
   b. Ensure **WebSphere ESB Server v6.1** is selected as target runtime.
   c. Click **Next**.

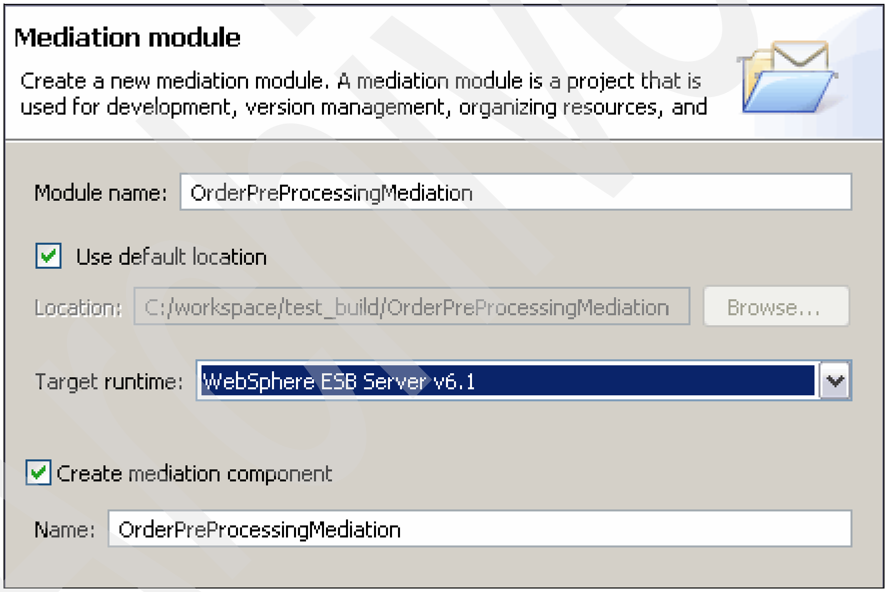


Figure 4-2 WebSphere ESB must be the target runtime for the module

3. Next, add OrderManagementLib as a required library and click **Finish**, as shown in Figure 4-3.

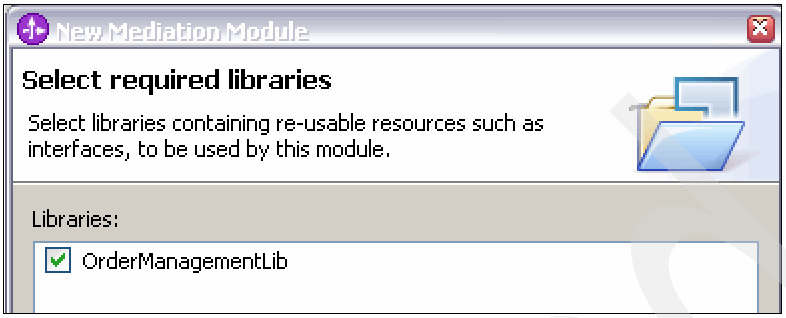


Figure 4-3 Select OrderManagementLib as a required library

The resulting module then displays in the Business Integration view (Figure 4-4).

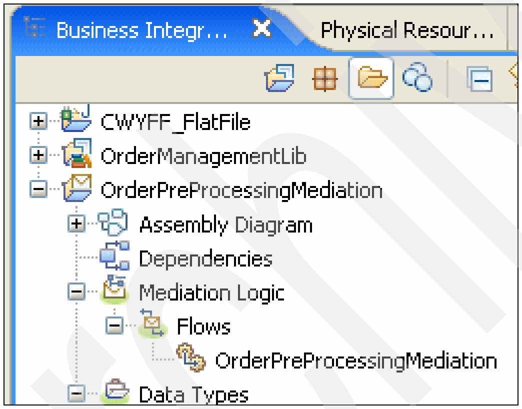


Figure 4-4 OrderPreProcessingMediation module in the Business Integration view

4.1.2 Creating the business objects

As shown in Figure 4-1 on page 250, an OrderManagementInputSb business object is required as input to the module. This object is not part of the common library, because it is not needed by any other module. Thus, you now need to define it within the mediation module.

This business object actually is a superset of the OrderManagementInput business object (defined in 1.4.2, “Interfaces and business objects definition” on page 17) which is passed eventually as a regular order business object to the OrderManagement module. You use the inheritance mechanism to define the business object (see Figure 4-5).

To define the business object:

1. In the Business Integration view, right-click the Data Types folder in OrderPreProcessingMediation and select **New** → **Business Object** from the context menu. In the New Business Object dialog box:
   a. Enter `OrderManagementInputSb` for the name for the new business object.
   b. Specify the OrderManagementInput object from which to inherit. You get all the fields in OrderManagementInput object by default. Do not select any additional fields in the wizard.
   c. Click **Finish**.

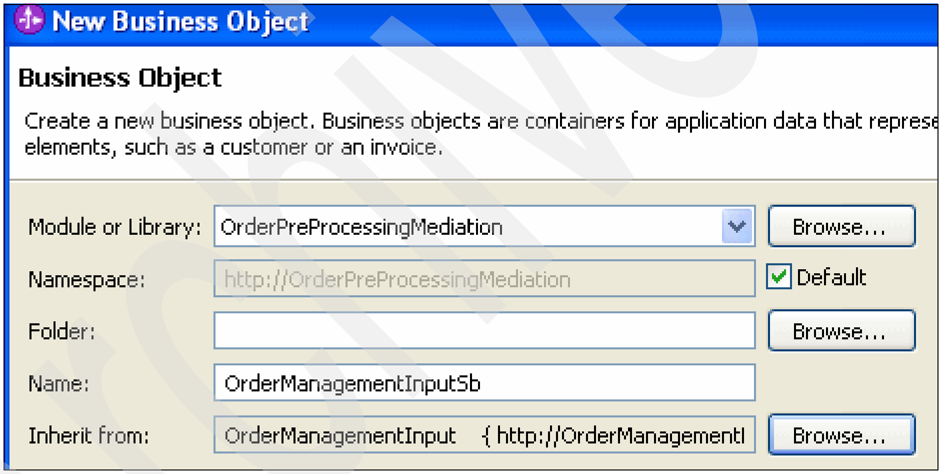


Figure 4-5 OrderManagementInputSb inherits from OrderManagementInput

2. The new business object is created in the location that you specified and opens in the business object editor. Add a new boolean field called `isSpecial` to the business object as shown in Figure 4-6.
3. Save and close the business object.

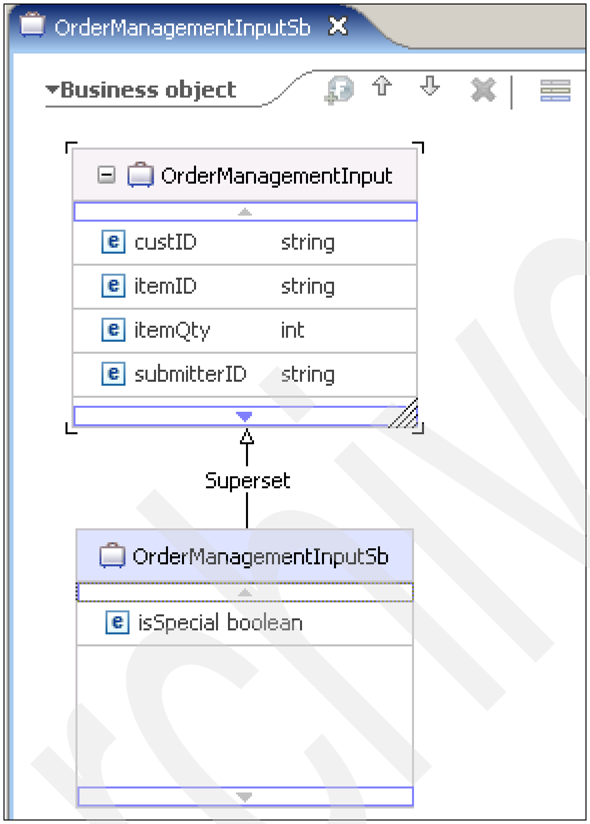


Figure 4-6 OrderManagementInput business object

The mediation uses this field to determine whether the order is a special or regular order. Depending on the value of the field, the business object is forwarded to the flat file adapter (special orders) for outbound processing to the file system or to the OrderManagement module (regular orders).

4.1.3 Creating the interfaces

As shown in Figure 4-1 on page 250, the module must implement the OrderPreProcessingIF interface. This interface is not part of the library, because it is not implemented by any other module. Thus, you need to define it now in the mediation module.

The interface must have a single one-way operation called `createOrder` that takes a single input of type OrderManagementInputSb. To create the interface:

1. In the Business Integration view, expand the OrderPreProcessingMediation folder. Right-click **Interfaces**, and select **New** → **Interface** from the context menu.
2. Enter `OrderPreProcessingIF` as the name for the interface and click **Finish**.

   The new interface is added to the folder in the Business Integration view, and the interface opens for editing.
3. In the interface editor, click the **Add One Way Operation** icon, as shown in Figure 4-7.

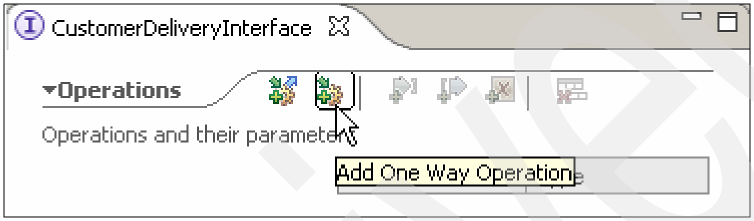


Figure 4-7 Add a one-way operation

4. Change the operation name to `createOrder`. Then, change the Input name to `orderManagementInputSb`, and select the **OrderManagementInputSb** business object as the type. See Figure 4-8.

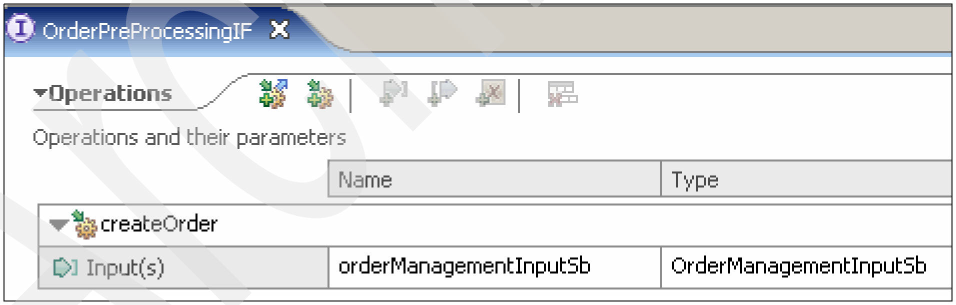


Figure 4-8 OrderPreProcessingIF Interface

5. Save and close the new interface.

4.2 Using the flat file adapter

As defined in the high-level design (1.2, “Scenario high-level design” on page 9), the flat file adapter is used to put special order requests on the file system.

Configuring an adapter for use in a mediation module occurs in several phases, as shown in Figure 4-9. This section explains how to perform these configuration steps for the scenario.

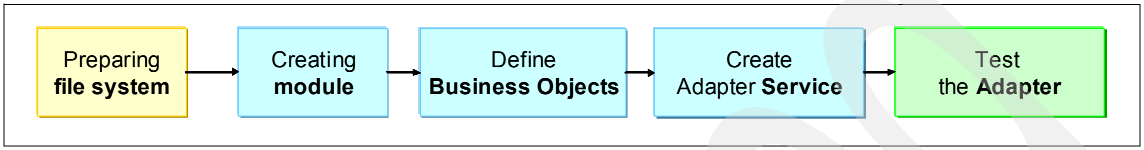


Figure 4-9 Overall picture of the Adapter configuration phases

The configuration steps include:

1. Prepare the file system to host outbound files from the adapter.
2. Create the module that accesses the adapter. In our case, this module is the OrderPreProcessingMediation module that was created earlier.
3. Define the business objects that are sent to the file system.
4. Create the adapter connector module (CWYFF_FlatFile), the service for accessing it, and the service import in the OrderPreProcessingMediation.
5. Test the adapter to ensure that it is working as expected.

4.2.1 Preparing the file system

In this step, you create the folders on the file system that the adapter uses. In our case, we send data to file system, so we need an *output* directory to store the files.

You can also create a *staging* directory for the adapter to store files temporarily while creating or overwriting them. Staging directories are used when you want to avoid writing conflicts with external processes. In our scenario, we dictate that a new file is created for every special order request. We do not expect conflicts and do not use any staging directory in our scenario.

To prepare the file system:

1. Create a C:\ITSOSpecialOrders folder in the file system as shown in Figure 4-10.

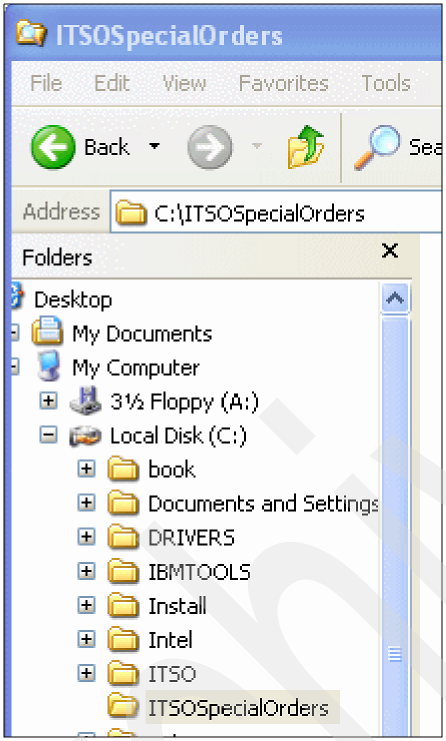


Figure 4-10 Our Adapter output directory

4.2.2 Defining the business objects for the adapter

Next, you need business objects to represent the files to be placed on the file system. You can create these objects using the New Business Object wizard or using the external services wizard for the adapter.

According to our high-level design, when an order is determined to be special, it must be sent to the file system through this adapter. All of the OrderManagementInput properties are included in the object to be sent to the file. In addition, we add a *specialNote* field that is available for possible future implementations. (You might want to add notes about how the order must be handled.) So, we make the outbound processing object an extension of the OrderManagementInput.

To define the business object for the adapter:

1. Select **File** → **New** → **Business Object** to start the New Business Object wizard, shown in Figure 4-11.
   a. Specify `OrderManagementSpecialInput` as the name of the object.
   b. Specify the OrderManagementInput as the object from which to inherit. You get all the fields in the OrderManagementInput object by default. Do not select any additional fields in the wizard.

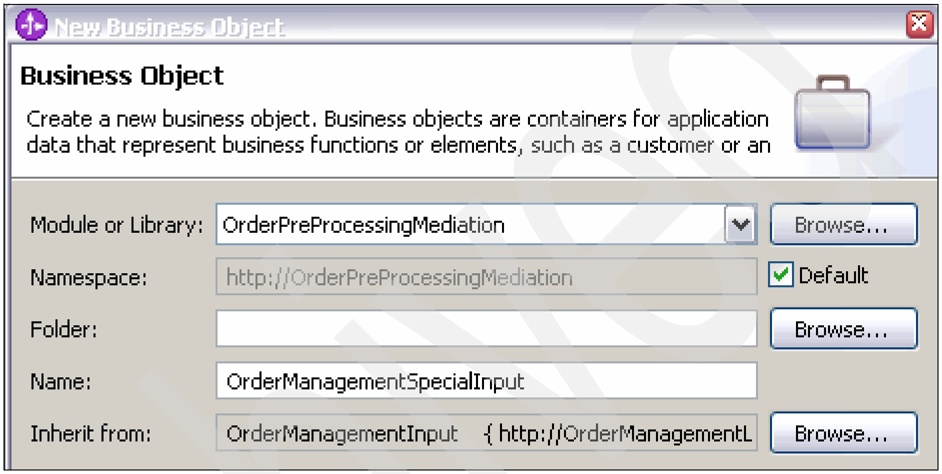


Figure 4-11 OrderManagementSpecialInput inherits from OrderManagementInput

2. After the object is created, add a new string field called `specialNote`, as shown in Figure 4-6.

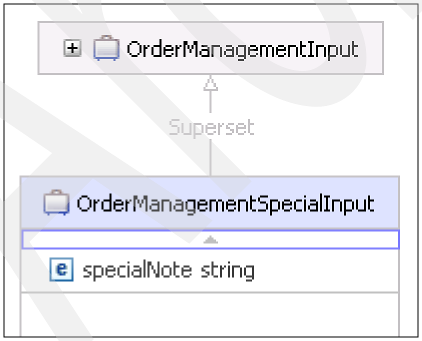


Figure 4-12 OrderManagementSpecialnput business object

4.2.3 Creating a service for the adapter using the wizard

The next step is to create a service for the adapter. You create the service using an adapter pattern, which is the simplest way to create a service. Using a pattern gives you immediate access to simple scenarios such as the one that you need for this scenario. In the event that you cannot use one of the patterns, you can use the external services wizard instead.

In our scenario, we use an outbound flat file pattern. The output file format is XML by default. If this is not sufficient, you can use a data handler for transforming the XML to suit your needs.

To create a service for the adapter using the wizard:

1. Right-click the OrderPreProcessingModule in the Business Integration View and select **New** → **From Patterns**, as shown in Figure 4-13.

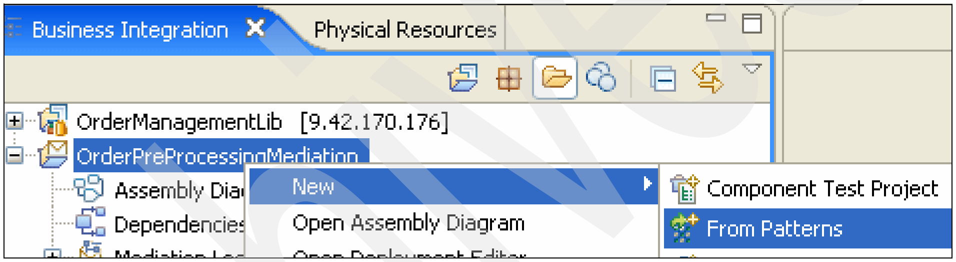


Figure 4-13 Creating service from patterns

2. Expand **Integration** → **Adapters** → **Flat File**. Then, select **Create an Outbound Flat File service to write to a local file**, and click **Next** (Figure 4-14).

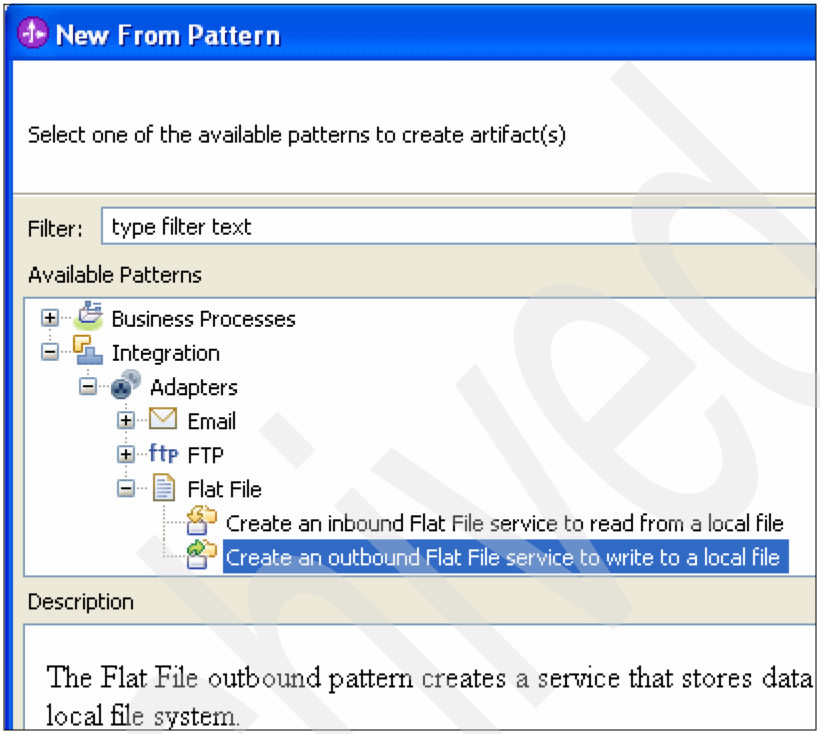


Figure 4-14 Selecting outbound flat file adapter usage

3. Name the service `SpecialOrderFileImport`, as shown in Figure 4-15, and click **Next**.

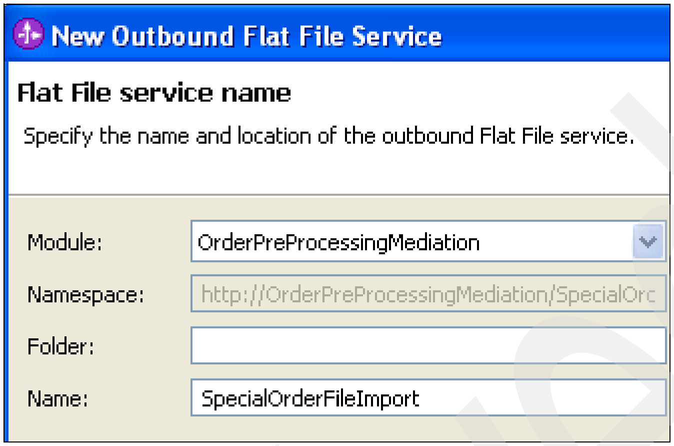


Figure 4-15 The outbound service name will be SpecialOrderFileImport

4. In the Business object and output directory panel (Figure 4-16):
   a. Specify **OrderManagementSpecialInput** as the business object to be sent to a file.
   b. Specify C:\ITSOSpecialOrders as output folder (which was created in 4.2.1, “Preparing the file system” on page 256). Click **Next**.

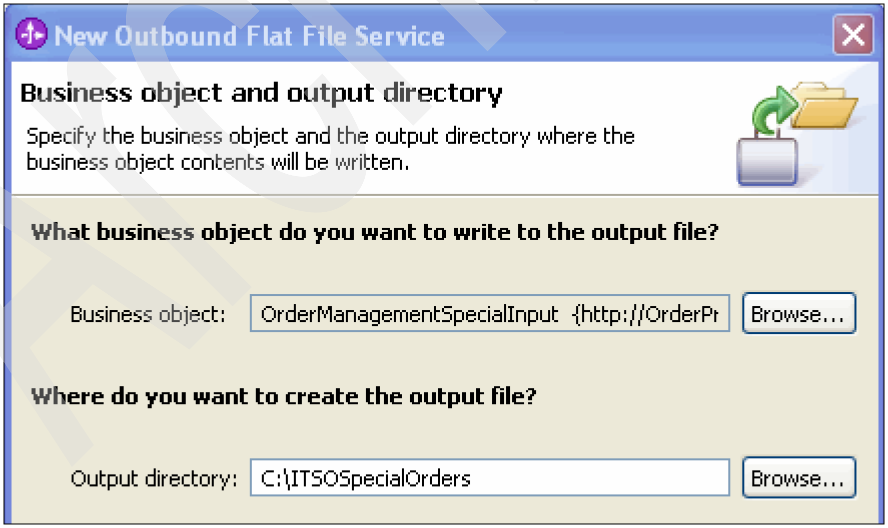


Figure 4-16 Business object and output folder selection

5. In the Output file name properties panel (Figure 4-17):

   a. Keep the **Generate a file name with an appended sequence number** setting. This option creates a new file using a sequence number for every special order request.

   b. Use `SpecialOrderRequest.txt` as the file name and `SpecialOrderRequest.seq` as sequence file name.

      The adapter uses the sequence file to determine the number of the file to be created next in the sequence.

   c. Click **Next**.

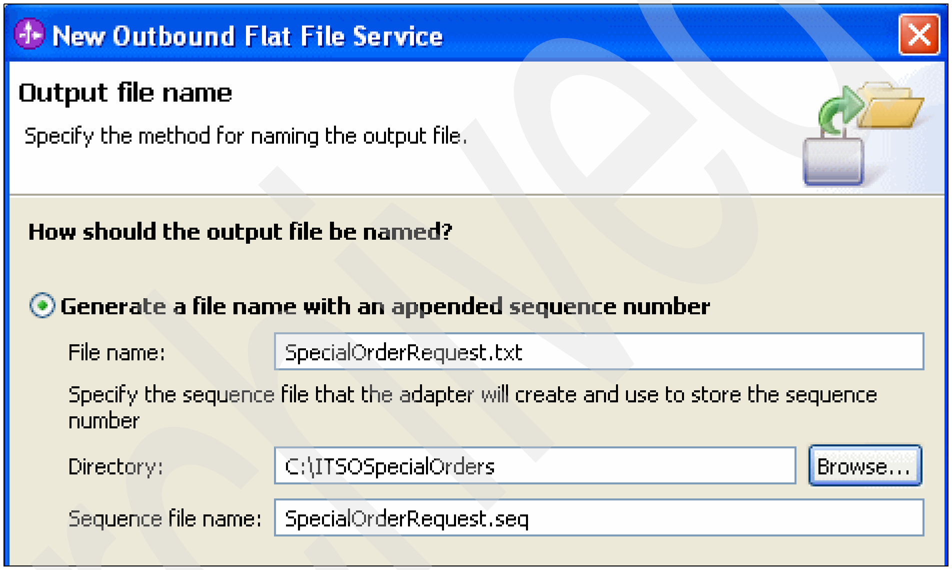


Figure 4-17 Sequencing files configuration

6. In the next panel (Figure 4-18), leave the file type as **XML** (the default type).

   To use another type of file, you can specify `Other` and identify a data handler that would transform the business object to the desired format. This assumes that the data handler was created previously (after the business object definition is created for the adapter).

   Click **Finish**.

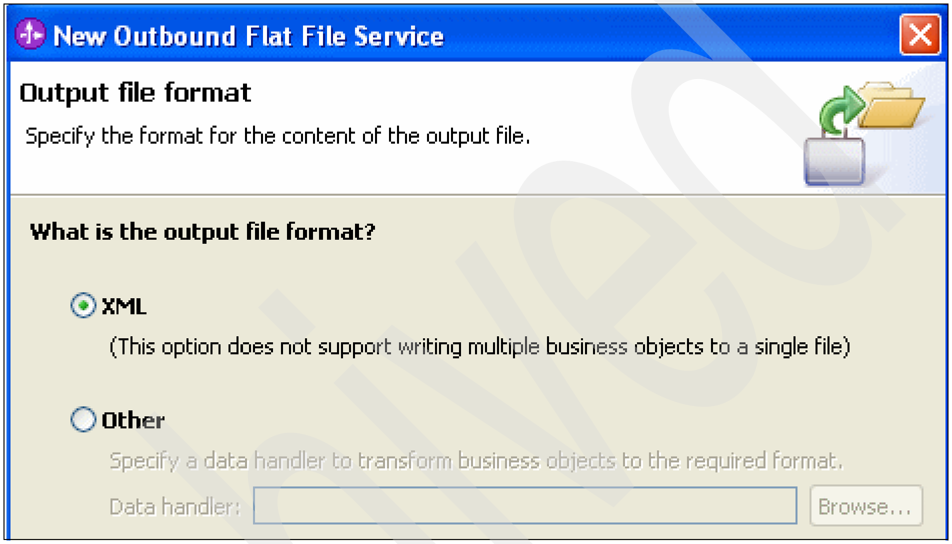


Figure 4-18 File type selection will be the default

The mediation module assembly diagram now looks similar to that shown in Figure 4-19.

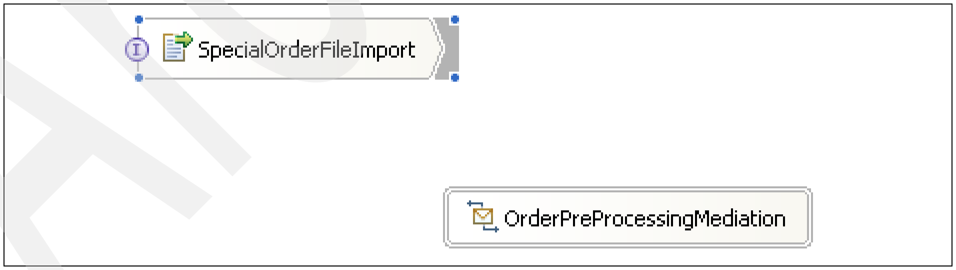


Figure 4-19 OrderPreProcessingMediation assembly diagram

> **Note:** The wizard created an Import component to access the adapter.

7. **Save** the assembly diagram.

The wizard also created other artifacts (shown in Figure 4-20), including:

- An adapter connector project (CWYFF_FlatFile**)** that handles this technology connector.
- Additional business objects in the CommonSchemas folder.
- A data binding and data handler to convert the OrderManagementInputSb business object into the generic FlatFileUnstructuredRecord object the adapter uses to outbound to the file system.

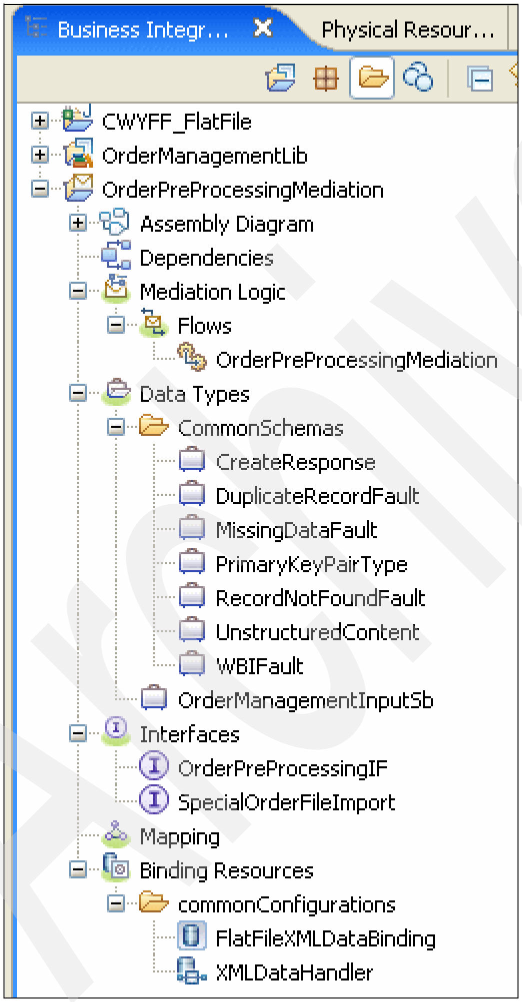


Figure 4-20   Adapter pattern wizard generated artifacts

4.2.4 Testing the flat file adapter through the mediation module

The last adapter configuration phase is testing. You test the adapter using its client module, OrderPreProcessingMediation. This method ensures that the adapter is working before you wire the mediation flow (to be developed later) to the connector SpecialOrderFileImport.

To test the flat file adapter:

1. Start the WebSphere ESB Server v6.1 integrated test environment.
2. Deploy the OrderPreProcessingMediation using the **Add and Remove Projects** utility on the server, shown in Figure 4-21.

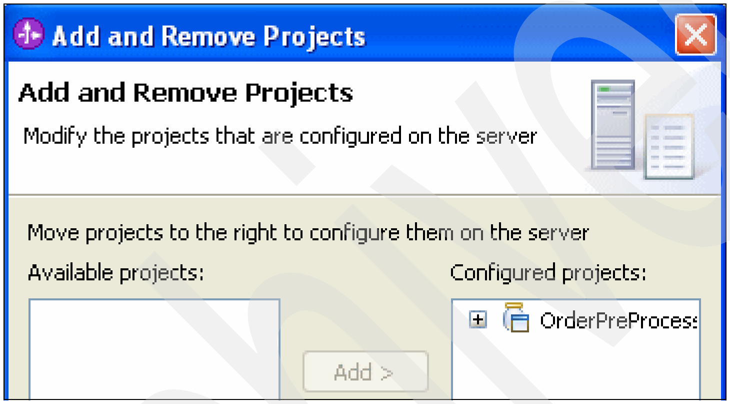


Figure 4-21 Mediation deployment over the test environment

   Wait for the publishing process to complete (the server status is `Started` and the state is `Synchronized`).

3. Right-click the **OrderPreProcessingMediation** module in the Business Integration view and select **Test** → **Test Module**.

4. A new view opens for the test properties (Figure 4-22).

   a. Verify that the test properties are as shown in Figure 4-22. Ensure that you are testing the SpecialOrderFileImport component.

   b. Enter test data to use as the initial request parameters.

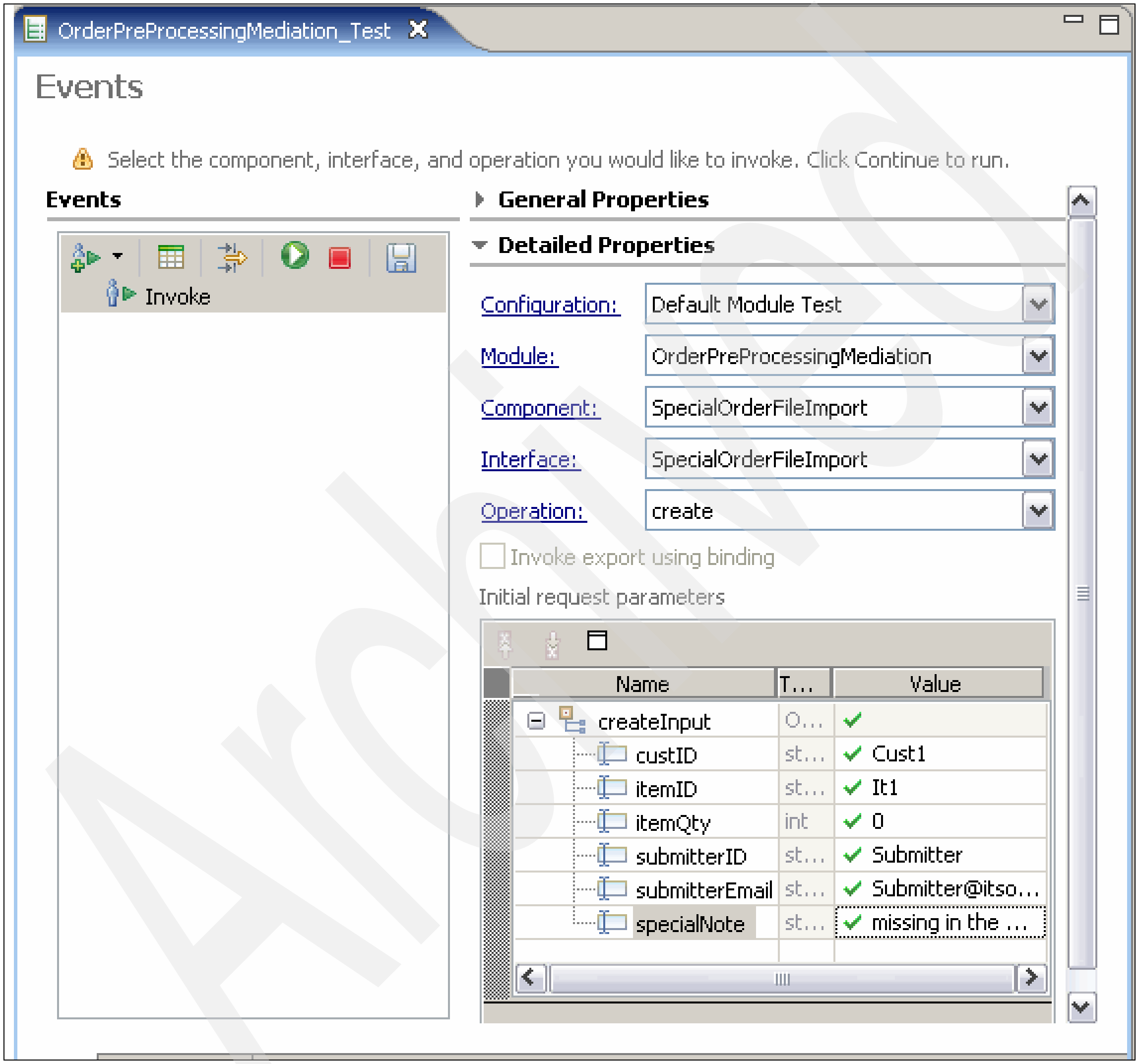


Figure 4-22 Test panel

5. Launch the invocation by clicking the **Continue** icon ( ). Select **WebSphere ESB Server v6.1** as the test environment (Figure 4-23). If the server has administrative security enabled, provide the user ID and password when requested.

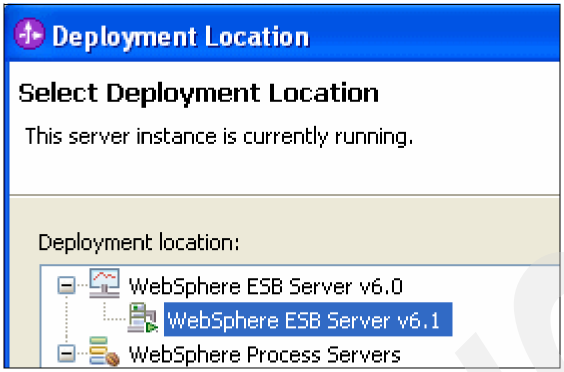


Figure 4-23 Select WebSphere ESB as the deployment location

6. Wait for the test to complete. The request and response looks similar to Figure 4-24.

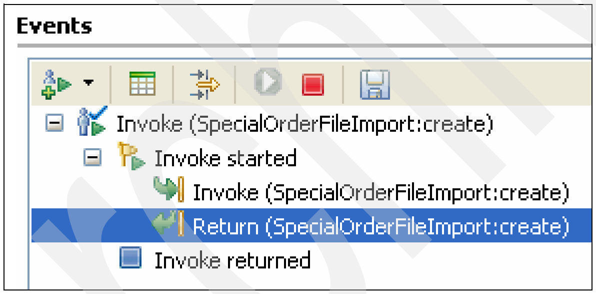


Figure 4-24 Adapter Import component test completed

7. Verify two files are created in C:\ITSOSpecialOrders, as shown in Figure 4-25.

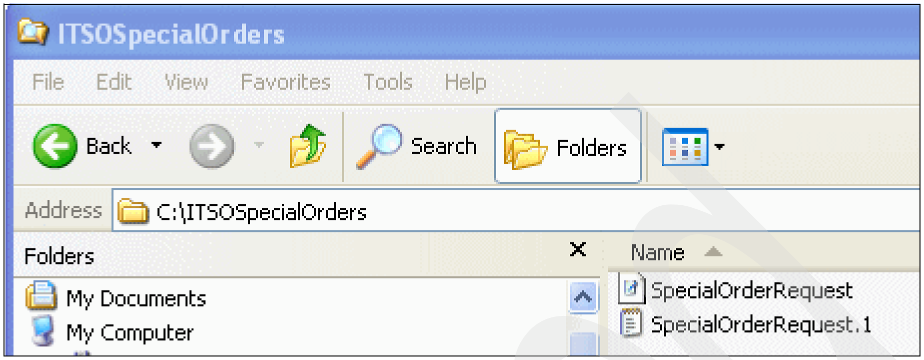


Figure 4-25 The flat file adapter sent two files as expected

The SpecialOrderRequest.1 file (with *1* being the sequence number) contains the business object data that corresponds to the request parameters that you specified when invoking the mediation. See Figure 4-26.

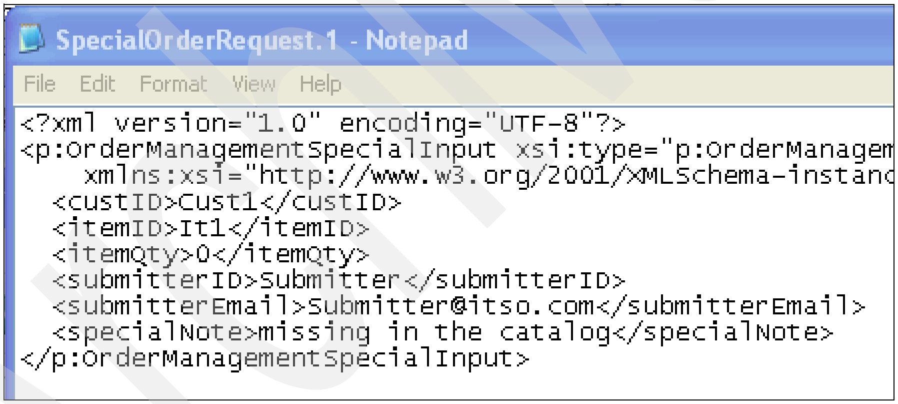


Figure 4-26 OrderManagementSpecialInput sent to file system

The SpecialOrderRequest.seq file contains the next sequence number that the adapter uses when requested for a new file creation, as shown in Figure 4-27.

Figure 4-27 Sequence number handling file

Now that you know the flat file adapter is working, along with the binding in the mediation module, you can develop the mediation flow.

4.3 Developing the mediation flow

According to the design described in 4.1, "Building the mediation module and artifacts" on page 250, the mediation flow has the following import and export dependencies:

- The mediation is accessed by an external client through the OrderPreProcessingIF interface. So, the mediation flow must implement and export the interface.
- The mediation forwards an OrderManagementInput business object to the following systems:
  - The flat file adapter when the order is determined to be special. You must wire the flat file adapter SpecialOrderFileImport to the mediation flow and add a reference to it.
  - The OrderManagement module when the order is determined to be regular. You must create a simple module to emulate the OrderManagement module, create an import of the emulation module, and wire it to the mediation flow through another reference.

4.3.1 Implementing and exporting the OrderPreProcessingIF interface

You first implement and export the OrderPreProcessingIF interface as follows:

1. Open the OrderPreProcessingMediation assembly diagram.
2. Right-click over the **OrderPreProcessingMediation** flow and select **Add** → **Interface**.
3. Select the **OrderPreProcessingIF** interface and click **OK**.

   As a result, an Interface icon is added to the left of the OrderPreProcessingMediation, as shown in Figure 4-28.

Figure 4-28 OrderPreProcessingMediation implementing the OrderPreProcessingIF interface

4. You want to export this interface as a Web service by right-clicking the mediation flow and selecting **Generate Export** → **Web service binding**. Then, accept **soap/http** as transport.

   The Export component is added in the assembly diagram as shown in Figure 4-29.

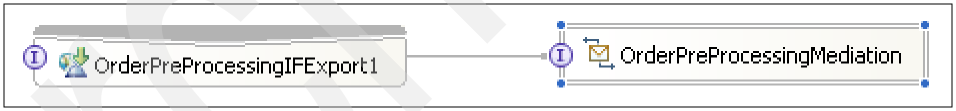


Figure 4-29 Mediation interface exported as Web service

5. Save the assembly diagram.

For now, do not worry about errors that display in the Problems view. The errors indicate that the mediation flow is not compliant to the interface that you implemented. We addressed these errors in 4.3.4, “Developing the mediation flow logic” on page 278.

4.3.2 Wiring the mediation flow to the SpecialOrderFileImport

You must wire the mediation flow to the flat file adapter SpecialOrderFileImport to forward a request to it. Follow these steps:

1. Open the OrderPreProcessingMediation module assembly diagram.
2. Pass the mouse over the right border of OrderPreProcessingMediation flow, wait for the yellow "lollipop" to appear, and drag it to the SpecialOrderImport Interface.

   As you release the mouse, you are prompted to allow automatic matching reference creation. Accept it, and save the assembly diagram. (Again, do not worry about any errors that display.)

   The resulting assembly diagram looks similar to that shown in Figure 4-30.

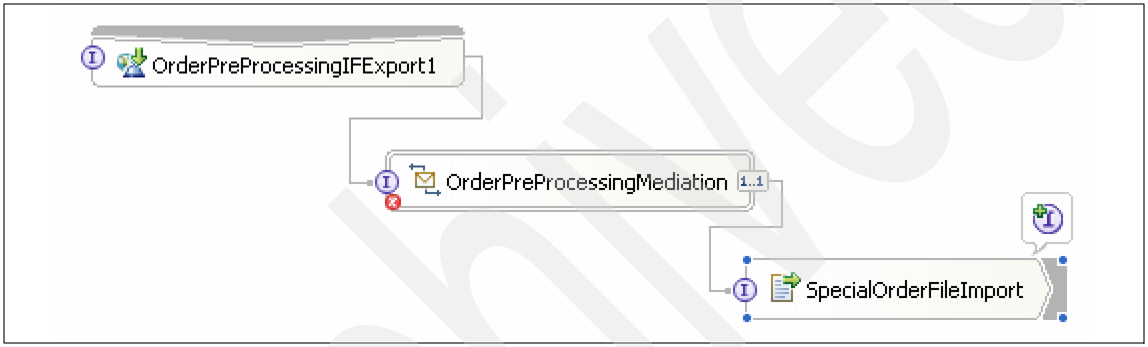


Figure 4-30 Assembly diagram after wiring the flat file adapter import

4.3.3 Emulating, importing, and wiring the OrderManagement module

The OrderPreProcessingMediation module must be able to forward requests to the OrderManagement module. For the purposes of this discussion, we are not interested in understanding how the OrderManagement module behaves, as long as it implements the OrderManagementIF interface and can be accessed through an export component.

You create an emulation of the OrderManagement module so that you can test the OrderPreProcessingMediation module quickly. Then, you import this emulation into the mediation module. Follow these steps:

1. Create a new mediation module:

   a. Name the module `OrderManagementProcessEm`.
   b. Select **WebSphere ESB Server v6.1** as the target runtime.
   c. Add OrderManagementLib as a required library.

2. In the assembly diagram for the new module, right-click the mediation flow and select **Add** → **Interface**, then select **OrderManagementIF**, as shown in Figure 4-31.

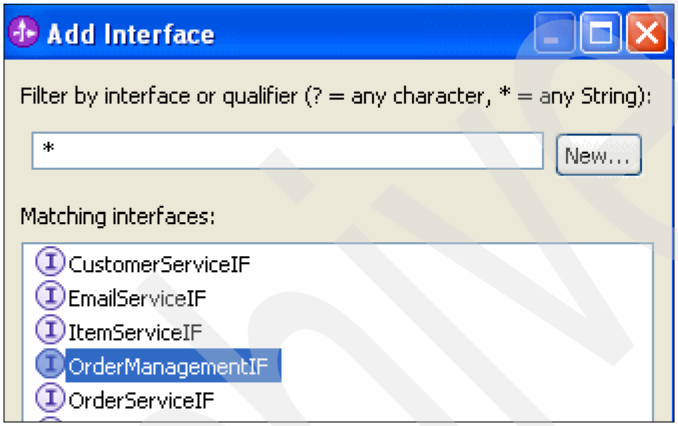


Figure 4-31 The OrderManagementProcessEm module must implement OrderManagementIF

3. Save the assembly diagram.

   When you save the assembly diagram, you receive errors because the flow implementation is not compatible with the interface.

4. Right-click the mediation flow, and select **Regenerate Implementation** as shown in Figure 4-32. Then, click **OK** to confirm replacement of the implementation.

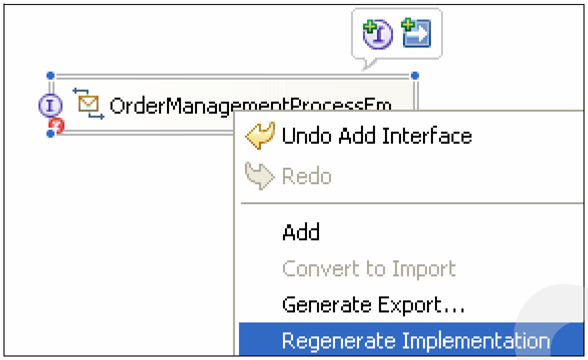


Figure 4-32 Regenerating implementation for the mediation module

The mediation flow editor opens.

5. Using the Palette, drop a Stop primitive on the canvas, as shown in Figure 4-33. The Stop primitive ends the mediation without throwing any exception.

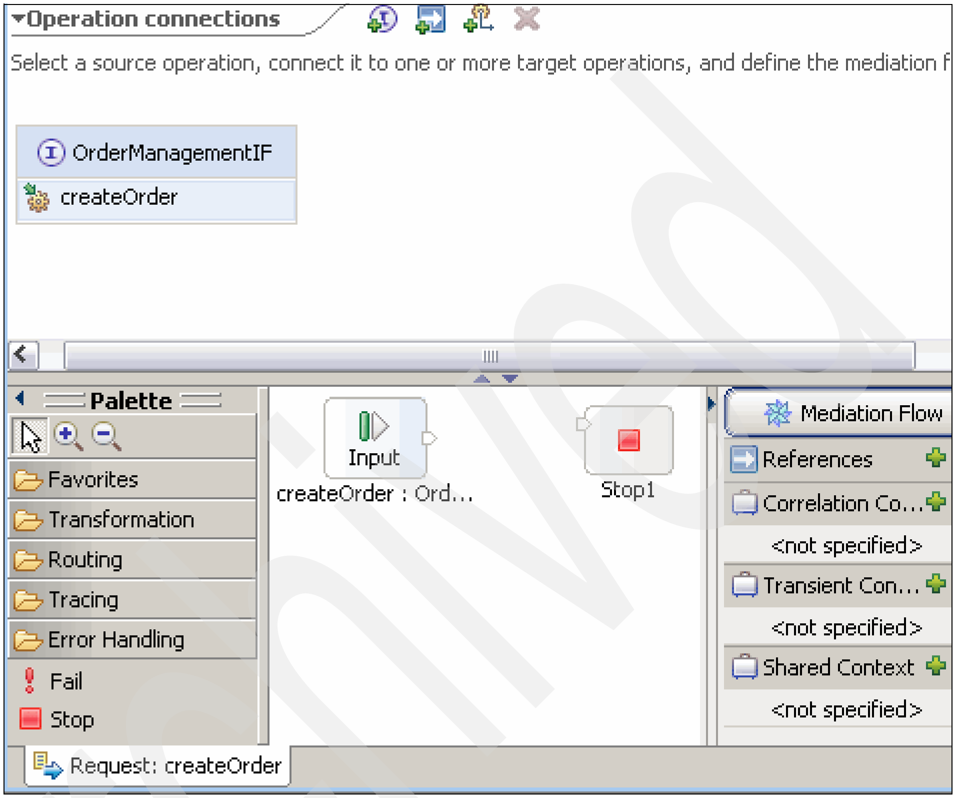


Figure 4-33   Dragging a Stop primitive to the mediation flow

6. Wire the output terminal of the Input primitive to the input terminal of the Stop primitive as shown in Figure 4-34.

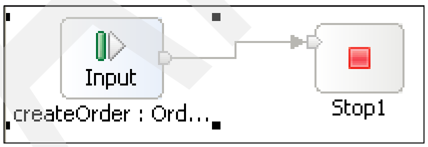


Figure 4-34   Connecting the Input to a Stop terminal

This mediation flow consumes the input without actually doing anything. It is just an emulation.

7. Save the mediation flow and close it.

8. Save the assembly diagram.
9. In the assembly diagram, right-click over the **OrderManagementProcessEm**, and select **Generate Export** → **SCA Binding**.
10. Save the assembly diagram again.

    Figure 4-35 shows the complete assembly diagram.

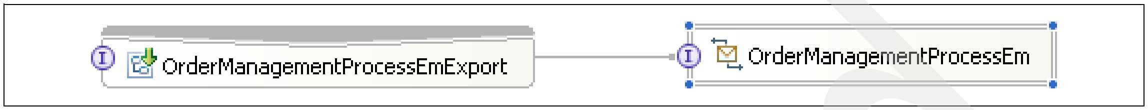


Figure 4-35 OrderManagementProcessEm complete assembly diagram

11. Close the assembly diagram.

    The OrderManagement module emulation is complete. Next, you import it to the OrderPreProcessingMediation module.
12. Open the OrderPreProcessingMediation module assembly diagram. Drag an Import component from the Component sub-menu in the Palette on the left, as shown in Figure 4-36.

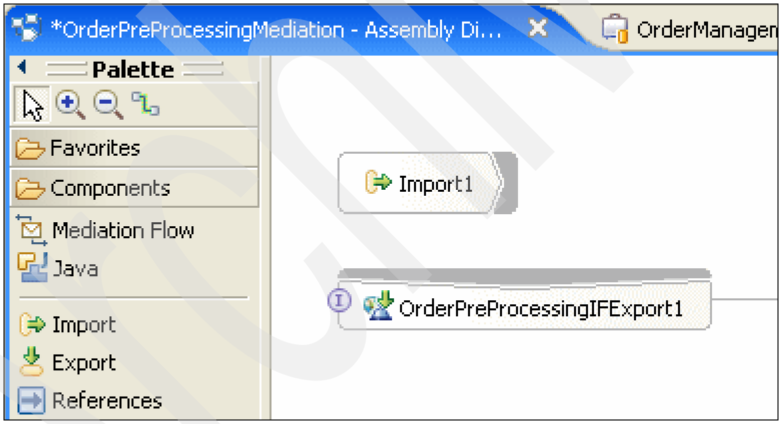


Figure 4-36 New Import for importing OrderManagementProcessEm

    Rename the new Import component `OrderManagementProcessImport`.
13. Right-click the Import component and select **Add** → **Interface**. Select **OrderManagementIF**, and click **OK**.
14. Right-click the Import component again, and select **Generate Binding** → **SCA Binding**.

> **Note:** You use the SCA binding because the emulation mediation runs in the same test server as the OrderPreProcessingMediation. This setup is for testing purposes only. Later, the emulation mediation is replaced by the OrderManagement module with a Web service binding. You might want to remove the binding when you complete the scenario in this chapter, when the mediation testing is complete.

15. Click the Import component in the assembly diagram, and select the Binding section in the Properties view. Then:

    a. Click **Browse** to the right of the “Export name” field to open a list of exports. Select **OrderProcessManagementEmExport**, and click **OK**. See Figure 4-37.

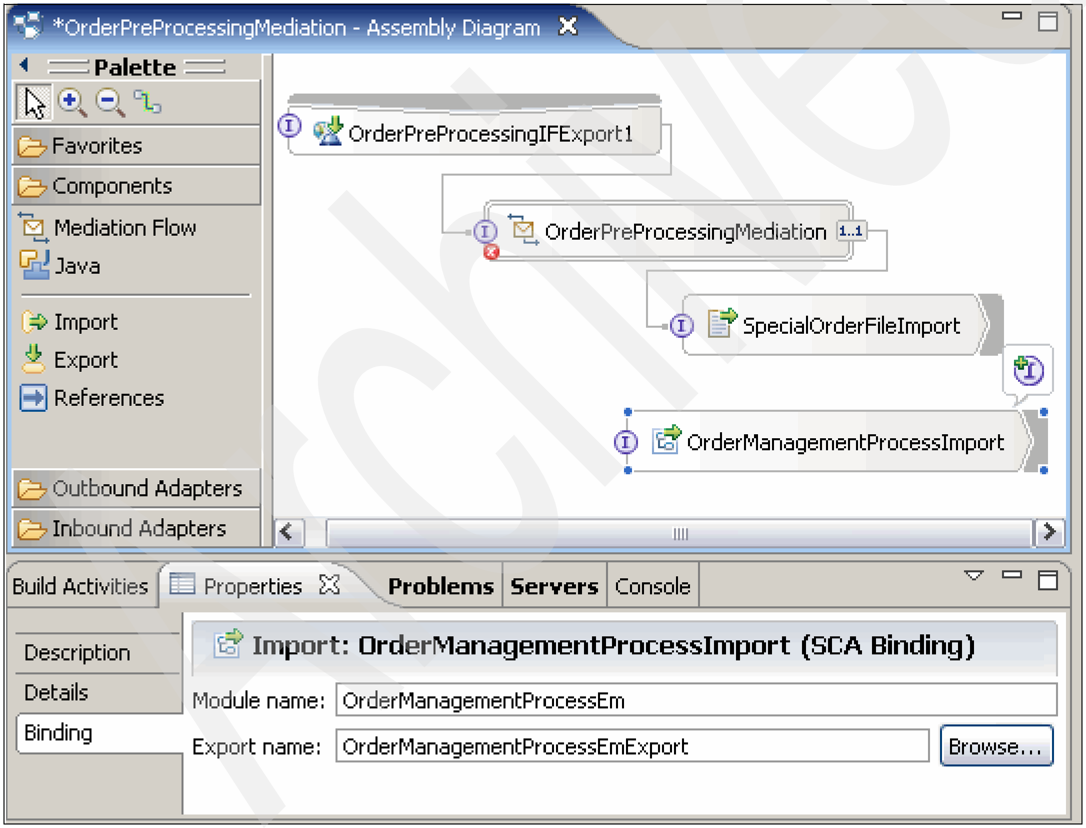


Figure 4-37 OrderManagementProcessImport SCA binding properties

16. Save the assembly diagram.

17. Wire OrderPreProcessingMediation to the OrderManagementProcessImport interface. Accept the automatic reference creation proposal.

18. Save the assembly diagram one more time.

Figure 4-38 shows the completed wirings.

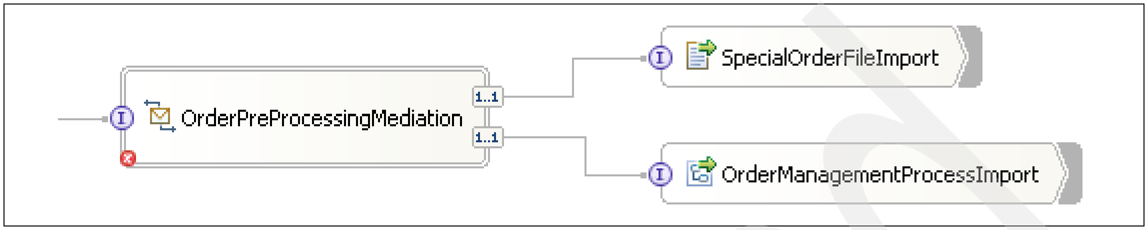


Figure 4-38 OrderPreProcessingMediation wirings

19. You still have errors on the mediation flow, because it does not comply with the implemented interface and references. To fix those errors, right-click over OrderPreProcessingMediation and select **Regenerate implementation**. Accept the replacement by clicking **OK**.

The mediation flow editor opens (Figure 4-39 on page 278). You use it for mediation development that is described in the next section.

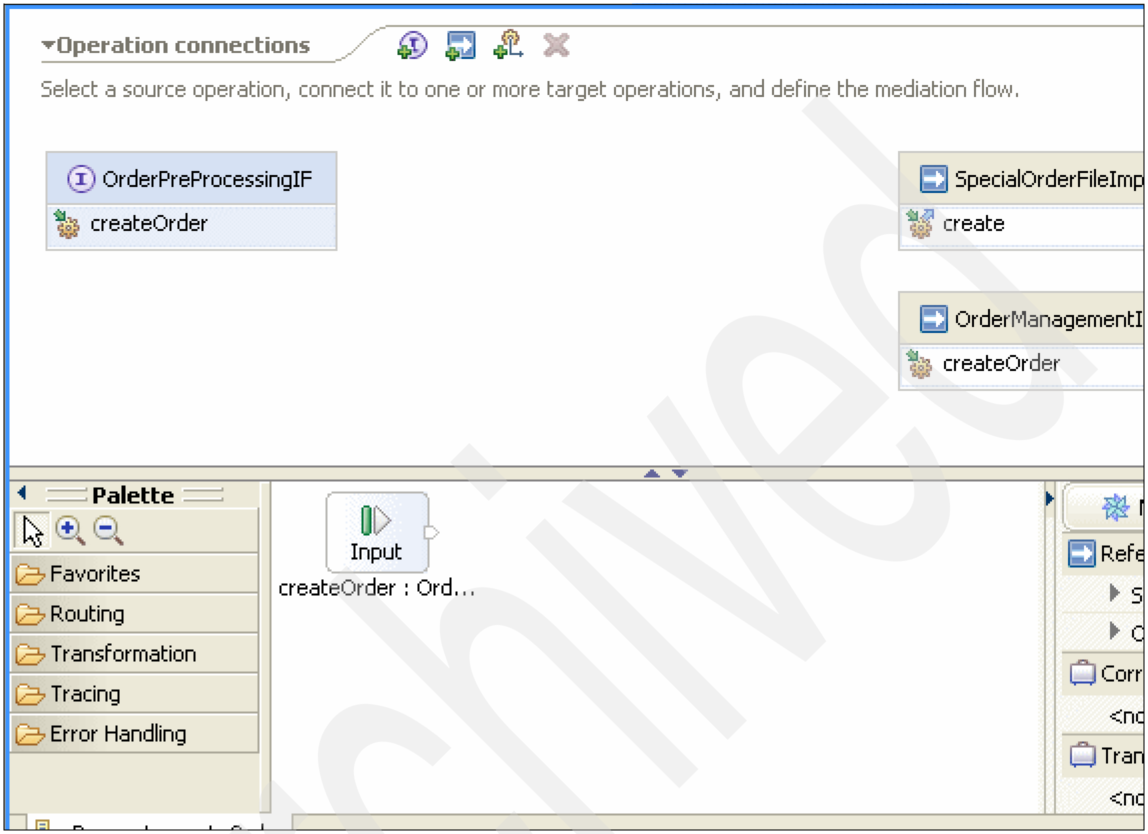


Figure 4-39 Mediation flow editor for the OrderPreProcessingMediation flow

4.3.4 Developing the mediation flow logic

At the top of the mediation flow editor (shown in Figure 4-39), the interface and two reference partners display. You must develop mediation flow logic that takes the input data that comes in through the createOrder operation and routes it to the appropriate reference partner. The routing is based on the value of the `isSpecial` field.

The input is an OrderManagementInputSb business object. You must route the data as an OrderManagementSpecialInput or OrderManagementInput business object.

The message flow logic uses the following mediation primitives:

- A *Message Filter* primitive for implementing the routing logic.
- Two specific *business object maps* for changing input format in the required output format.

To develop the mediation:

1. In the mediation flow editor, wire the OrderPreProcessingIF createOrder operation to the SpecialOrderFileImportPartner create operation.

   This is one mediation flow.

2. Wire the OrderPreProcessingIF createOrder operation to the OrderManagementIFPartner createOrder.

   This is the second mediation flow.

   Selecting a connection, for example the line from OrderPreProcessingIF createOrder to SpecialOrderFileImportPartner create, opens the flow in that flow in the Request and Response editor, as shown in Figure 4-40 on page 280.

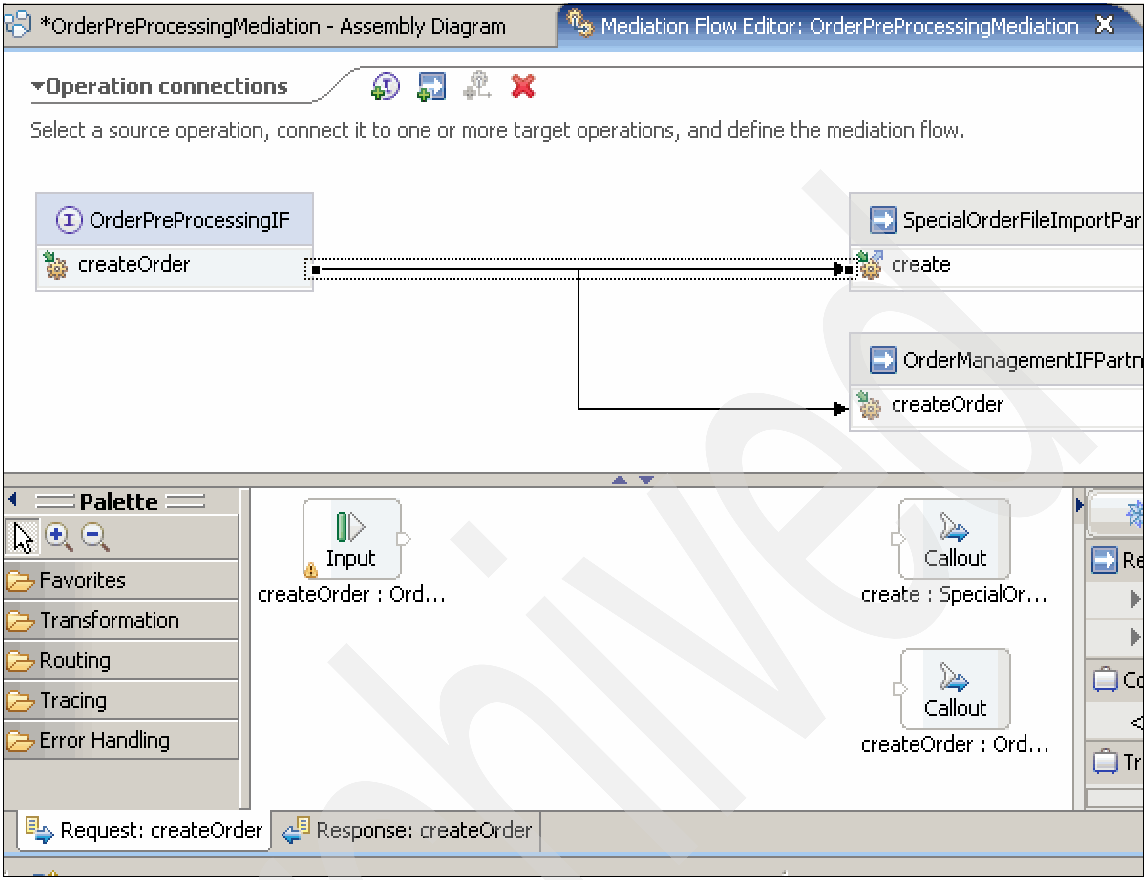


Figure 4-40 Mediation flow interface mapping

3. Save the mediation flow.

As shown in the Request:createOrder flow tab, you need to write the logic that connects the input message coming from the OrderPreProcessingIF createOrder request to the two Callouts.

Populating the mediation flow

Next, add the primitives to the mediation flow and provide meaningful names as follows:

1. In the Palette (Routing section), select a Message Filter primitive and drag it to the middle of the three existing primitives.
2. Then, again in the Palette (Transformation section), select and drop two Business Object Map primitives before the two callouts.

   Figure 4-41 shows the primitives in the OrderPreProcessingMediation flow.

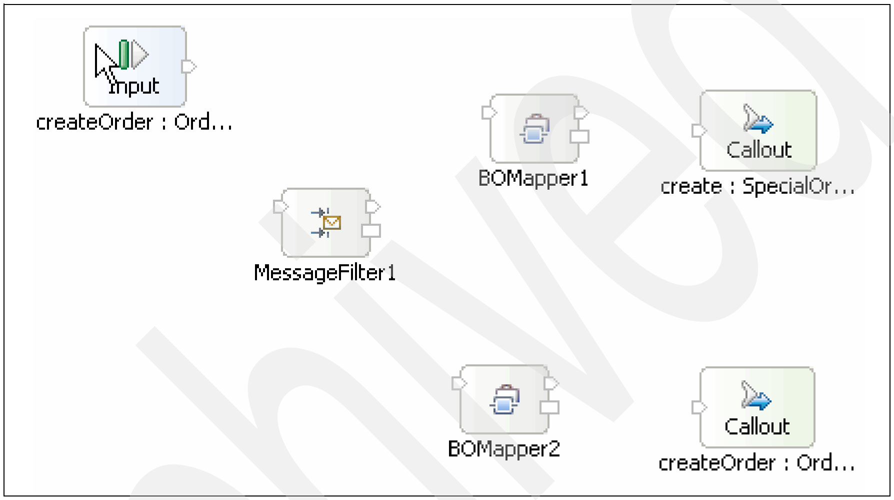


Figure 4-41 Primitives used in the OrderPreProcessingMediation flow

3. The MessageFilter1 primitive needs two output terminals because you route to two possible callouts:
   a. Right-click the primitive and select **Add Output Terminal**. Name the primitive `OrderRegularTerminal`.

   > **Note:** You cannot rename the default terminal. If the terminal name is important, you can add an extra terminal and name it appropriately.

   b. Change the display name of MessageFilter1 to `OrderKindOfRouter` (using the Properties view).
4. Rename the Business Object Map primitives to `GIBO2ADsbo` (representing generic to adapter specific) and `GIBO2BPBO` (representing generic to business process specific).

5. Save the mediation flow. Do no worry about errors now. You will resolve them as you complete the message flow development.

Configuring the Message Filter primitive

To configure the Message Filter primitive:

1. Wire the createOrder out terminal to the in terminal of the Message Filter primitive, OrderKindOfRouter as shown in Figure 4-42.

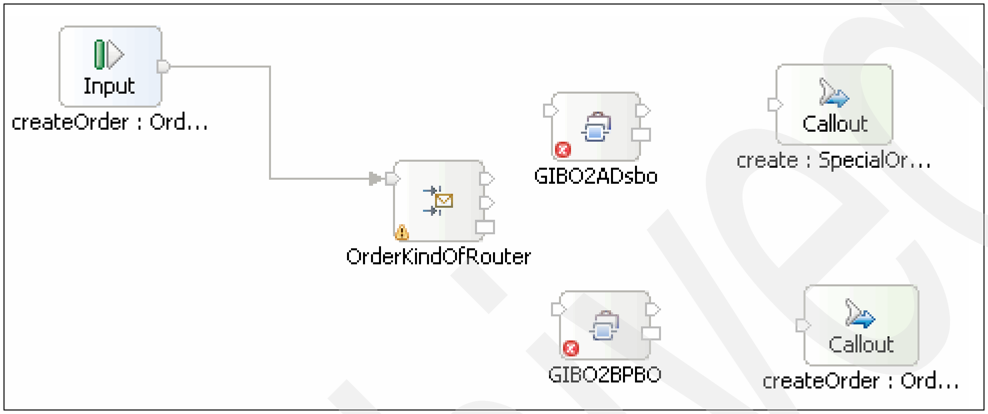


Figure 4-42 Primitives in the mediation flow renamed

2. Select the **OrderKindOfRouter** and in the Details pane in the Properties view (Figure 4-43):

   a. Leave `First` as distribution mode. With this mode, the first condition that is matched stops the filtering process and determines the routing.

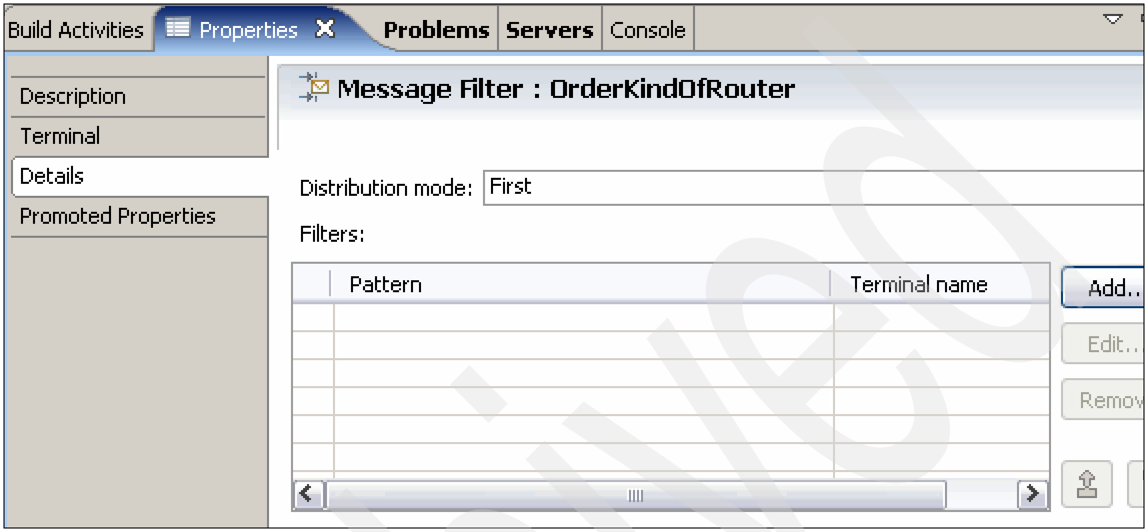


Figure 4-43 Distribution mode for OrderKindOfRouter

   b. Add a filter by clicking **Add**. Use the Edit button next to the Pattern field and follow these steps:

      i. Navigate the Data Types Viewer to **Data Types** → **ServiceMessageObject** → **body** → **createOrder** → **orderManagementInputSb: orderManagementInputSb** and double-click **isSpecial: boolean**.

      Ensure that the /body/createOrder/orderManagementInputSb/isSpecial XPath expression displays in the window.

      ii. In the Operators selection pane, double-click the equal sign (=) to append it to the expression.

      iii. Navigate the XPath functions to Boolean and double-click **false(): boolean**.

      Ensure `false()` is appended in the Xpath Expression window as shown in Figure 4-44 on page 284.

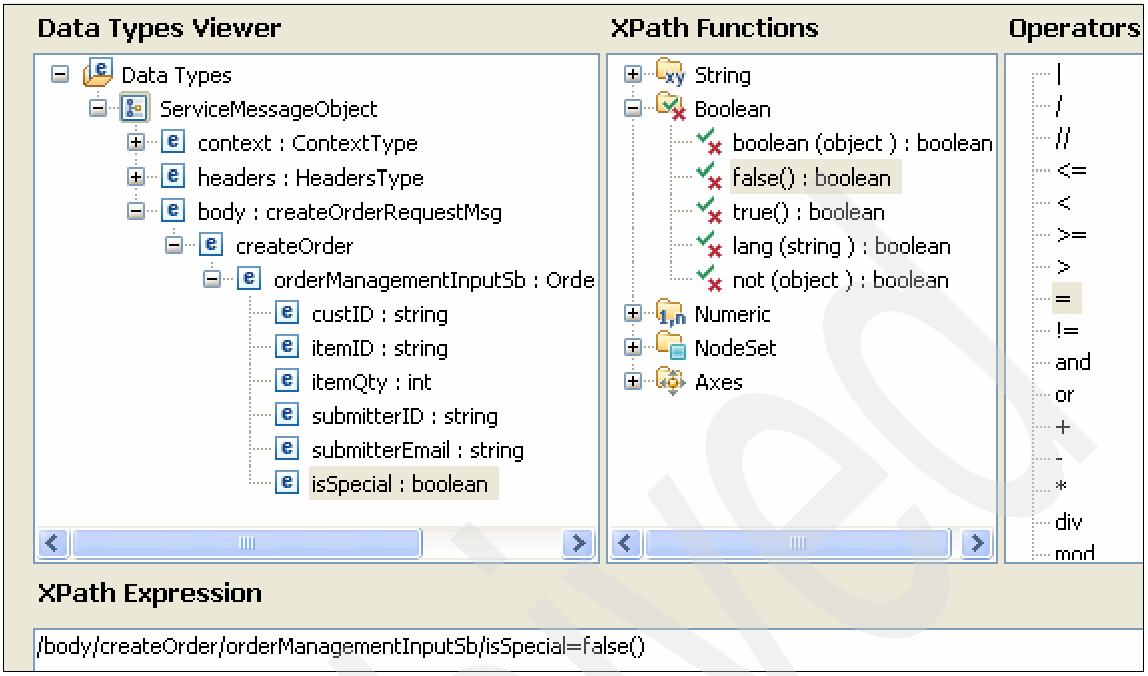


Figure 4-44 XPath expression

c. Click **Finish** in the XPath Expression Builder window and in the Add/Edit Properties dialog box.

3. Save the mediation flow.

   Warnings on the Input node and the Message Filter primitive do not display, but the Business Object Map errors are still there.

Configure the business object maps

To configure the business object maps:

1. Wire the first map as follows:

   a. Wire the OrderKindOfRouter default out terminal to the GIBO2ADsbo in terminal.

   b. Wire the GIBO2ADsbo out terminal to the create SpecialOrderFileImportPartner in terminal.

   Figure 4-45 on page 285 shows the mediation flow after wiring.

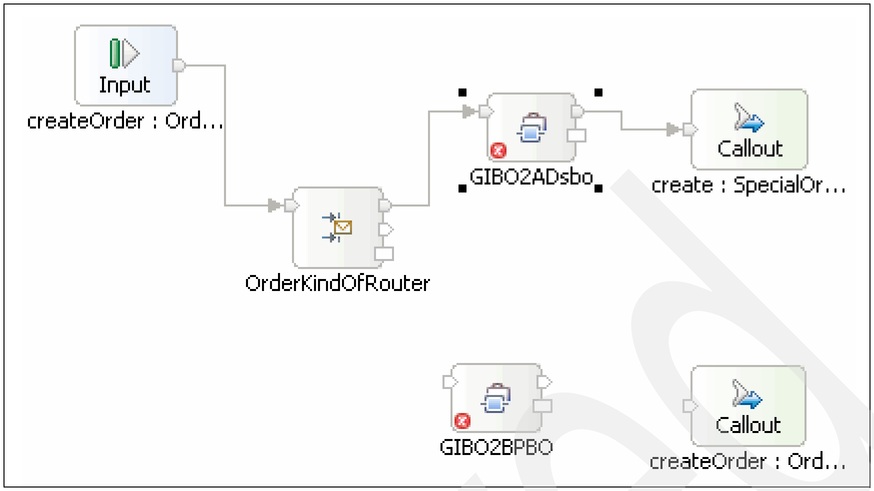


Figure 4-45 Mediation flow after wiring the GIBO2ADsbo

2. Save the mediation flow.
3. Double-click **GIBO2ADsbo** to go into the implementation. In the dialog box that displays, enter `Order2OrderSpecial` as the name, and click **Finish**.

   The Business Object Map editor opens.
4. Fully expand both the createOrderRequestMsg and the createRequestMsg elements. Then drag connections between all of the corresponding lower level elements (the elements are equal on both sides except for the isSpecial element in createOrderRequestMsg and the specialNote in createRequestMessage).

Next, follow these steps:

a. Keep the default Move operation for all connections, as shown in Figure 4-46 and Figure 4-47.

> **Note:** Figure 4-46 and Figure 4-47 represent the left and right halves of the business object map. The panel is too wide to represent it fully.

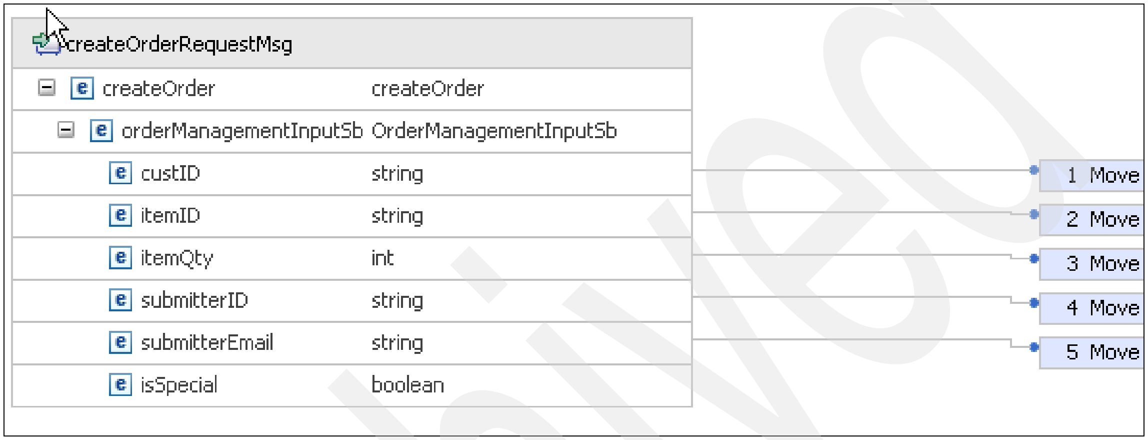


Figure 4-46 Order2OrderSpecial Mapping left side

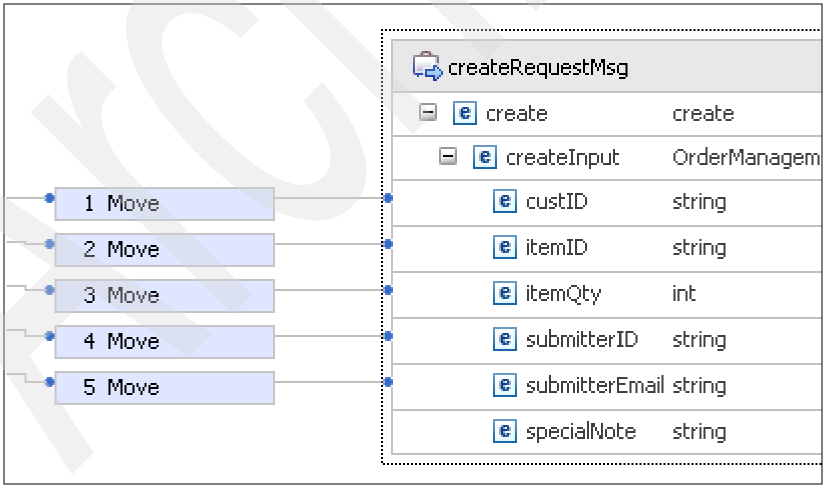


Figure 4-47 Order2OrderSpecial Mapping right side

b. Right-click createRequestMsg specialNote, and select **Create Transform** → **Assign**.

c. In the Details section of the Properties view, select **User defined value** and enter `For further implementation` (as shown in Figure 4-48).

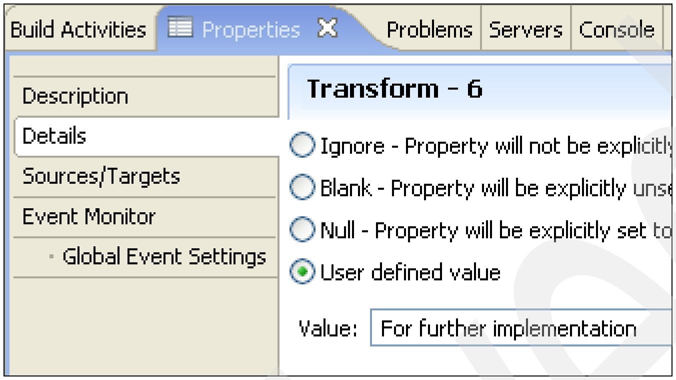


Figure 4-48 Assigning a fixed string to the special note

The map configuration is complete. Save and close it to return to the mediation editor.

5. Use the same technique to wire and implement the GIBO2BPBO map:

   a. Wire the OrderKindOfRouter OrderRegularTerminal out terminal to the GIBO2BPBO in terminal

   b. Wire the GIBO2BPBO out terminal to the createOrder OrderManagementIFPartner in terminal

6. Save the mediation flow.

Figure 4-49 shows the mediation flow completely wired.

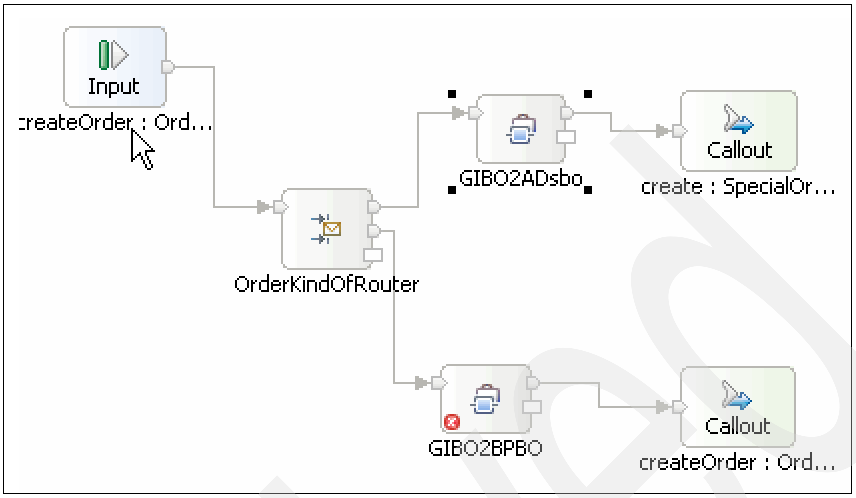


Figure 4-49 Mediation flow completely wired

7. Implement GIBO2BPBO. Follow these steps:

   a. Name it `Order2OrderRegular`.

   b. Fully expand the request message elements and make the corresponding connection, leaving the isSpecial field unconnected.

   c. Save and close the Map.

   d. Save the mediation flow.

   No more errors are expected on the mediation.

Figure 4-50 show the implementation complete.

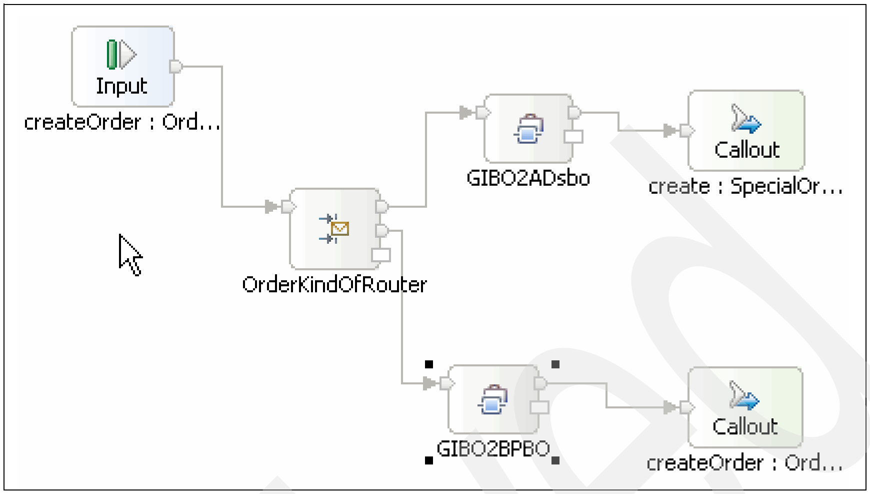


Figure 4-50 Mediation logic implementation completed

Error handling

Because all of the fail terminals on the primitives are not wired, now you need to define some basic error handling. Follow these steps:

1. Drag a Fail primitive from the Palette in the Error handling section.
2. Use the Details section of the Properties view to enter an error message, as shown in Figure 4-51.

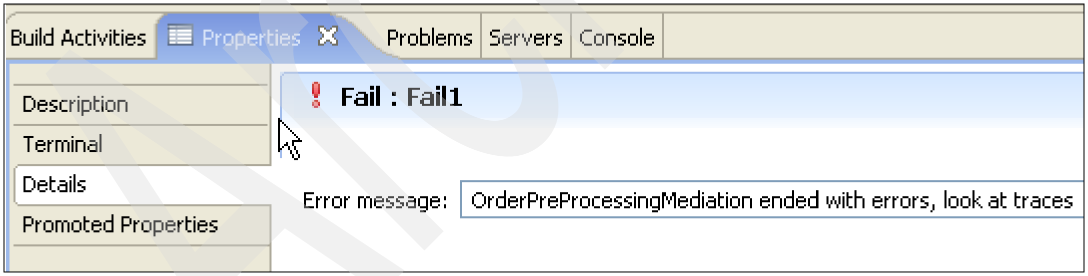


Figure 4-51 Configuring a fail primitive

3. Save the primitive and wire all of the unmatched fail terminals to this primitive.
4. Save the flow again.

Figure 4-52 shows the failure implementation.

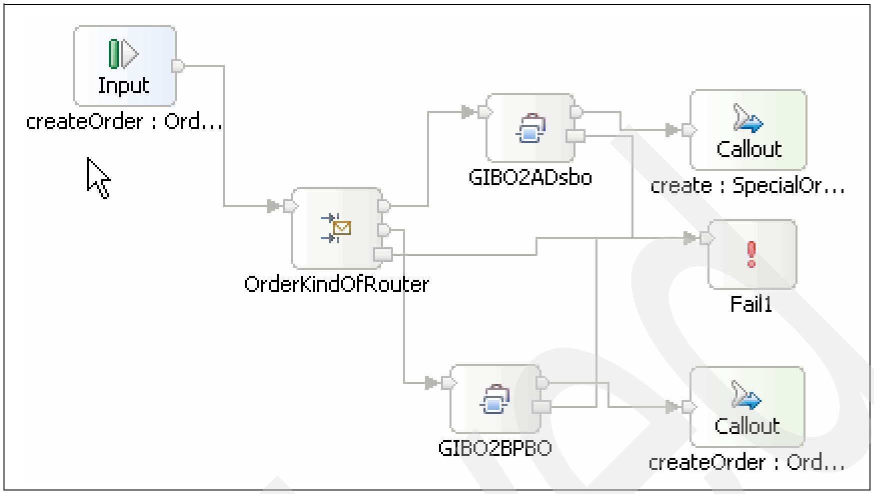


Figure 4-52 Mediation failure implementation

Building the response flow

Because the SpecialOrderFileInput Interface was created by the flat file adapter wizard (as described in 4.2.3, "Creating a service for the adapter using the wizard" on page 259) with a file name response, you need to implement the response flow. Follow these steps:

1. Switch to the Response tab at the bottom of the mediation flow editor.
2. Add a Stop primitive.
3. Add a Fail primitive.
4. Wire the response flow as shown in Figure 4-53.

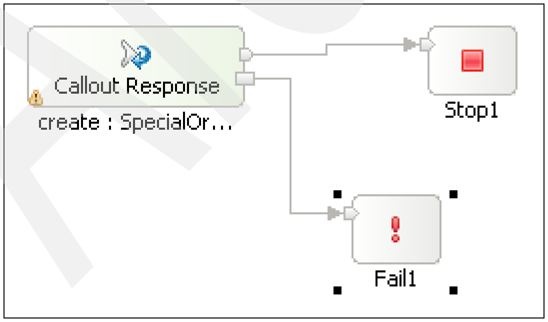


Figure 4-53 Response flow for OrderPreProcessingMediation

5. Put an error message in the Fail primitive (for example `Failure in the response`).
6. Save the mediation flow and assembly diagram.
7. Ensure that no errors are raised.

You can now test the mediation.

4.4 Testing the mediation

To test the mediation:

1. Delete all files in C:\ITSOSpecialOrders.
2. Start the integrated test environment server.
3. Deploy the OrderPreProcessingMediation and OrderManagementProcessEm projects to the test environment using the **Add and Remove projects** option, as shown in Figure 4-54.

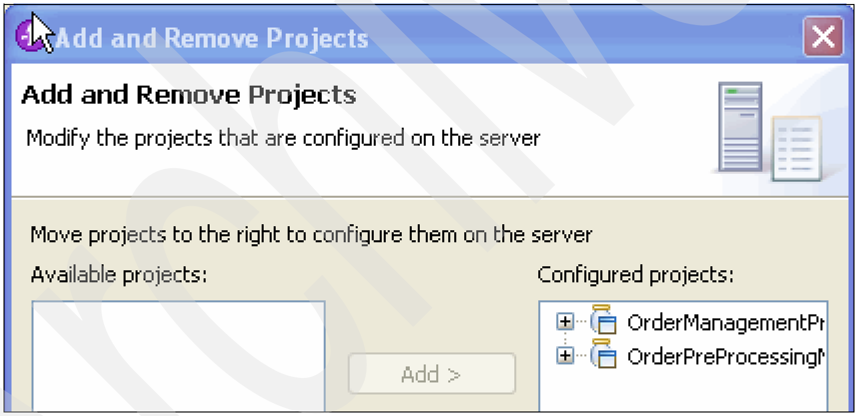


Figure 4-54 Deploying all of the modules for testing

   Wait for the publish operation to complete. The server status is Started and its state is Synchronized.

4. Right-click in the assembly diagram and select **Test Module**.

Testing regular order flow

To test the flow for regular orders:

1. Configure the test properties as shown in Figure 4-55. Make sure `isSpecial` (in the initial request parameters) is set to `false`.

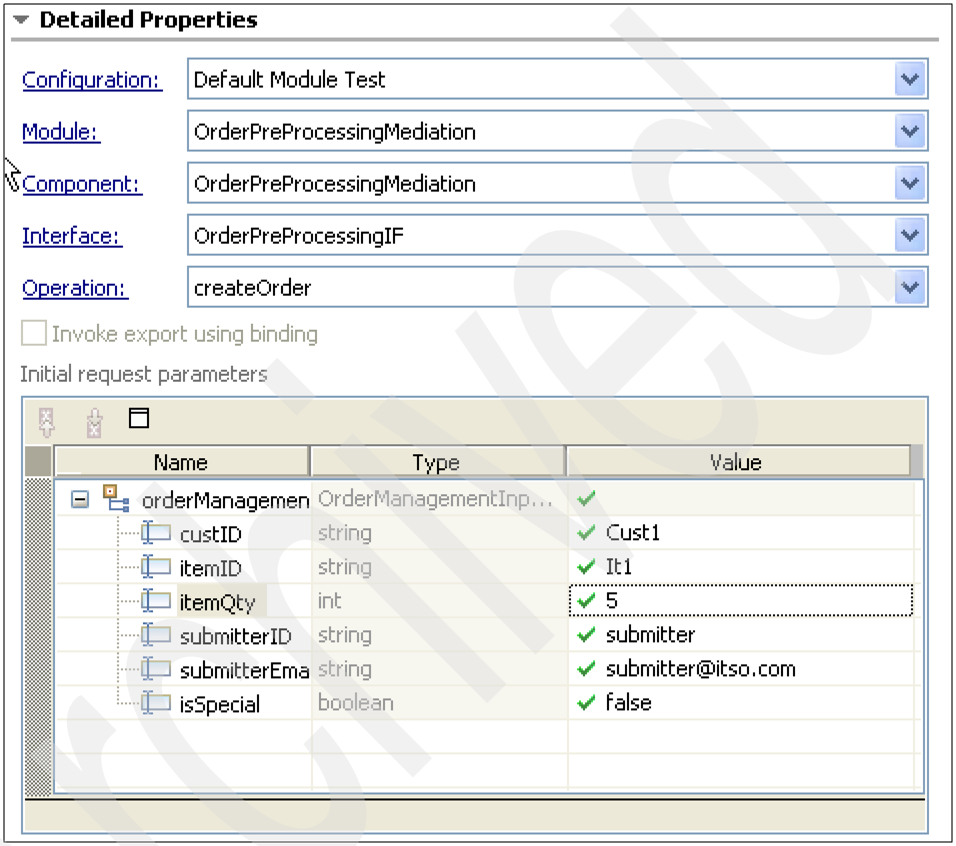


Figure 4-55 Final test parameters

2. Run the test, selecting the WebSphere ESB Server 6.1 test environment.

   You can see in the Events console, shown in Figure 4-56, that the request was sent to the OrderManagementProcessEm module through the OrderManagementProcessImport.

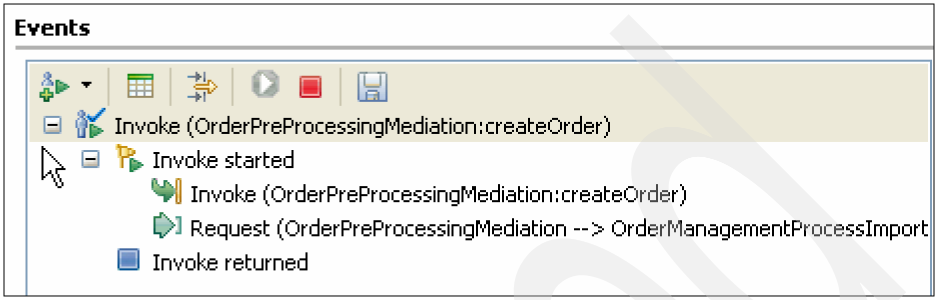


Figure 4-56 Testing the regular order management

3. Ensure that no files are found in C:\ITSOSpecialOrders as shown in Figure 4-57.

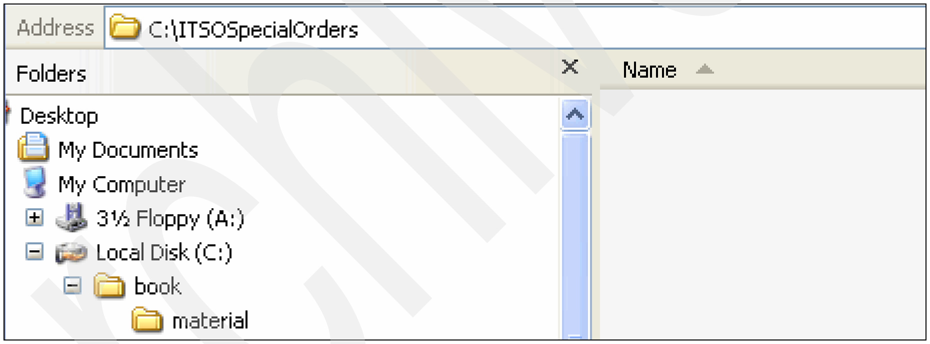


Figure 4-57 No files created for regular orders

Testing the special order flow

To test the flow for special orders:

1. Repeat the steps from the previous test (“Testing regular order flow” on page 292), and change the `isSpecial` value to `true`.
2. This time the request is forwarded to the SpecialOrderImport as shown in Figure 4-58.

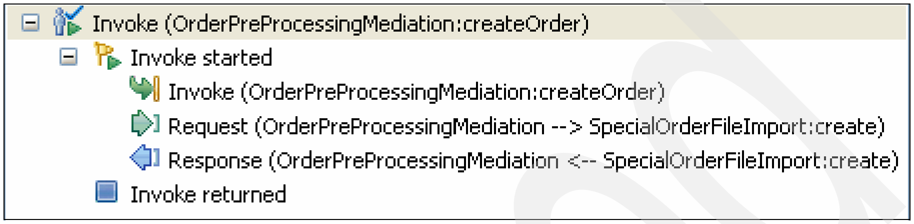


Figure 4-58 Request flow for special orders

Figure 4-59 shows a SpecialOrderRequest.1 file in C:\ITSOSpecialOrders, with your input content plus the `For further implementation` special note text.

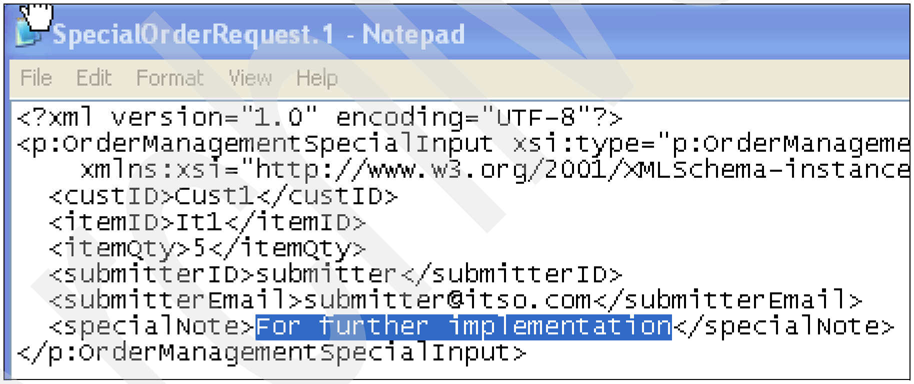


Figure 4-59 Special order information outbound to the file system

Test the Web service export for the component

The previous tests started with the mediation flow component. To test the component starting with the Web service export, repeat the test as described in “Testing regular order flow” on page 292 and select the OrderPreProcessingIFExport1 component as shown in Figure 4-60.

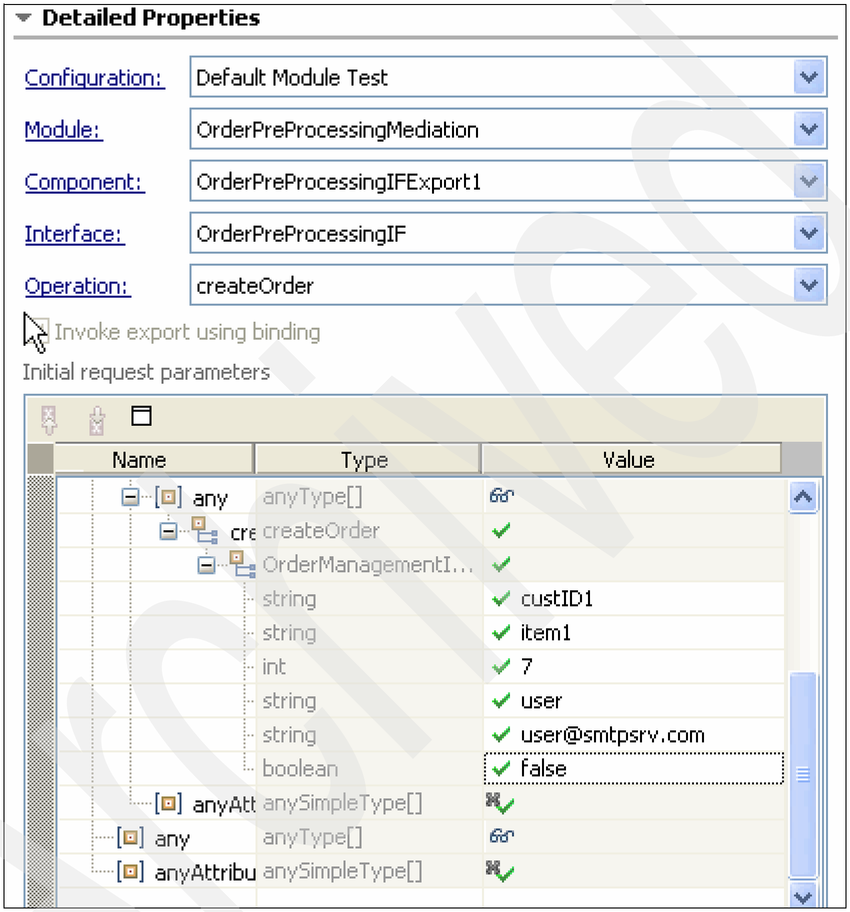


Figure 4-60 Testing the OrderPreProcessingMediation accessing the Web service export

Team development: This is a good time to check in your project to CVS.

5

DBMSServiceMediation module

The DBMS Mediation is used in the Order Management System to access database records in the ORDERDB database. The DBMSServiceMediation module maps data and operations that are defined in the interfaces that are called by the OrderManagement module into the JDBC adapter operations and appropriate ORDERDB table records. This mediation module uses the IBM WebSphere Adapter for JDBC Version 6.1.

This chapter includes the following topics:

- IBM WebSphere Adapter for JDBC
- Creating the mediation module
- Defining the database runtime resources
- Testing the JDBC outbound interface
- Building the mediation flow
- Testing the mediation flow component
- Creating the mediation export component

5.1 IBM WebSphere Adapter for JDBC

The IBM WebSphere Adapter for JDBC is a technology adapter that implements the Java database connectivity (JDBC) API for accessing databases. This adapter is included with WebSphere Integration Developer.

5.1.1 JDBC outbound application

This example uses the JDBC *outbound* services that are provided by the IBM WebSphere Adapter for JDBC. Thus, the application initiates interaction with the database and executes create, update, and retrieve operations on the database. To have an application or application services invoked by certain types of database events, then you use the JDBC *inbound* services instead.

> **Additional material:** This section assumes that you have built the Derby ORDERDB database and populated it with data and that it is located in C:\itso\sampleDB\ORDERDB.
>
> The ORDERDB database is located in the Derby sample files\ORDERDB directory in the additional material that is included with this book. Copy the ORDERDB directory to C:\itso\sampleDB.
>
> You can find information about the additional material in Appendix B, “Additional material” on page 461.
>
> You can find the instructions to build this database in Appendix A, “Creating the ORDERDB Derby database” on page 453.

> **Note:** If you are using Derby Embedded, make sure that there are no active connections to the database, including those from your workspace. The external services wizard must connect to the database to complete.

5.2 Creating the mediation module

To create the new mediation module based on the WebSphere Adapter for JDBC, perform the following steps, using the Business Integration perspective:

1. Select **File** → **New** → **External Service** from the top menu bar.
2. Select **Adapters** and click **Next**.

3. Select **IBM WebSphere Adapter for JDBC** and click **Next**.
4. Click **Add** to the right of the JDBC driver JAR files list (as shown in Figure 5-1). Browse to WID_ROOT\runtimes\bi_v6\derby\lib and select the derby.jar file. This JAR file contains the JDBC driver for accessing a Derby database. Click **Next**.

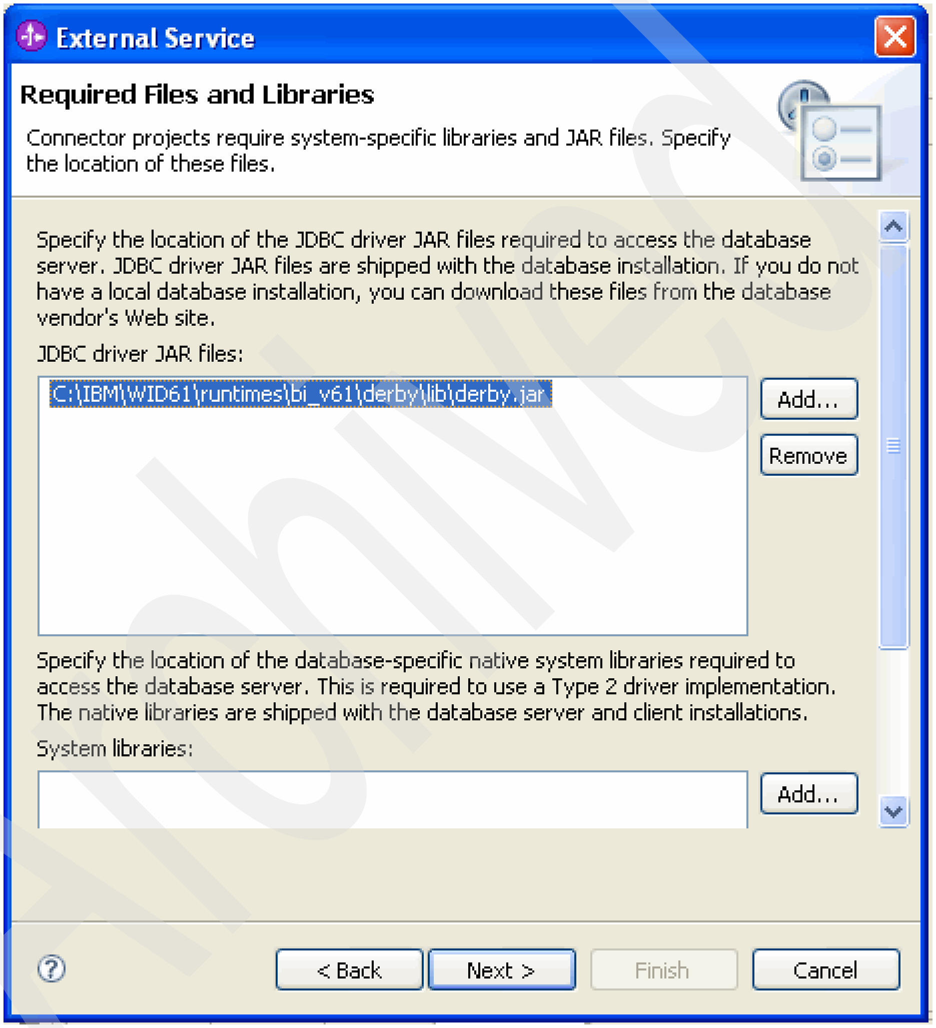


Figure 5-1 Select the Derby JDBC driver

5. Select **Outbound** for the processing direction on the next dialog box and click **Next**, as shown in Figure 5-2.

Figure 5-2 Select the processing direction

6. On the Discovery Configuration panel:

   a. In the left panel expand **Generic JDBC** and select **1.0**.

   b. In the Discovery Configuration panel (Figure 5-3 on page 301), under properties select **Other** for the JDBC driver type. Then, set the JDBC driver classname to `org.apache.derby.jdbc.EmbeddedDriver`. Finally, set the Database URL to:

   ```
   jdbc:derby:C:\itso\sampleDB\ORDERDB
   ```

   This URL must match the database URL of the connect statement that is used during ORDERDB database creation.

c. Enter the user name and password that is used to create ORDERDB. For this example, it is `dbadmin/dbadmin`.

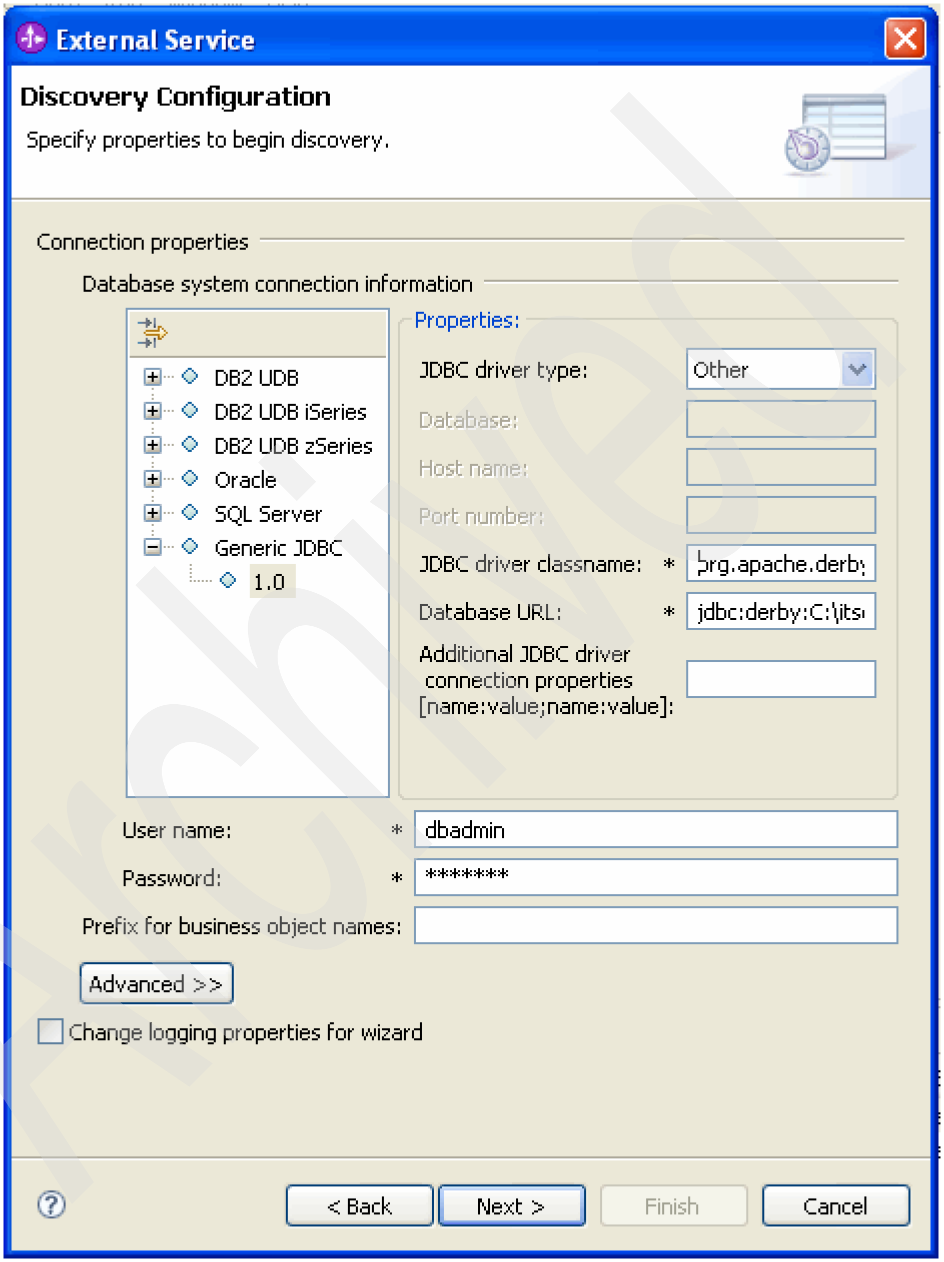


Figure 5-3 Database connection properties

7. Click **Next**.

   If you do not receive an error, go to the next step. However, if you get an error similar to the one shown in Figure 5-4:

   a. Click **OK** on the error panel. Then, click **Cancel** from the Discovery Configuration panel. The data that you entered will not be lost.

Figure 5-4 Error in the database discovery process

   A typical reason for this type of error with Derby is that you might already have established a connection with the ORDERDB database from another application, for example, the Database Explorer or the ij tool. This connection would prevent a connection to the database. Make sure that no other application is connected to ORDERDB. Note that you might have to switch to the Data perspective to check the ORDERDB connectivity in Database Explorer.

   b. To try re-establish connection with the database again, select **File** → **New** → **External Service** and select **Adapter** on the next dialog box as you did earlier in steps 1 and 2.

c. This time, however, you find a connectivity icon under CWYBC_JDBC adapter as shown in Figure 5-5. This icon implies that you have already configured a JDBC adapter for this workspace.

   Select that icon, and then select **Outbound** on the Processing Direction panel. Now you should be back to the panel before the error occurred. Select **Generic JDBC 1.0** again. The property values are the same as you left them. Re-type the password (it is not saved), and click **Next**.

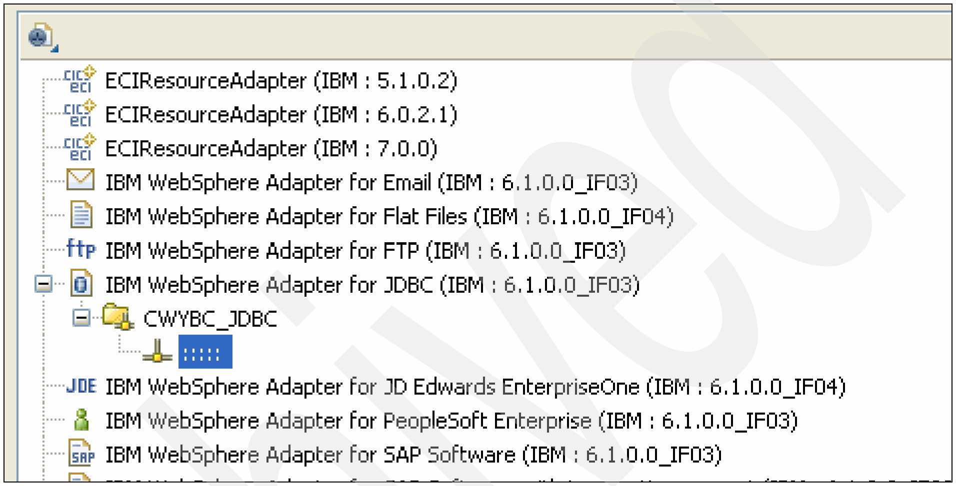


Figure 5-5 Connectivity icon says the adapter has been configured for this workspace

8. If all the data is entered correctly and no other application is connected to the database, you should move successfully to the **Object Discovery and Selection** dialog box, shown in Figure 5-6. Click **Edit Query**.

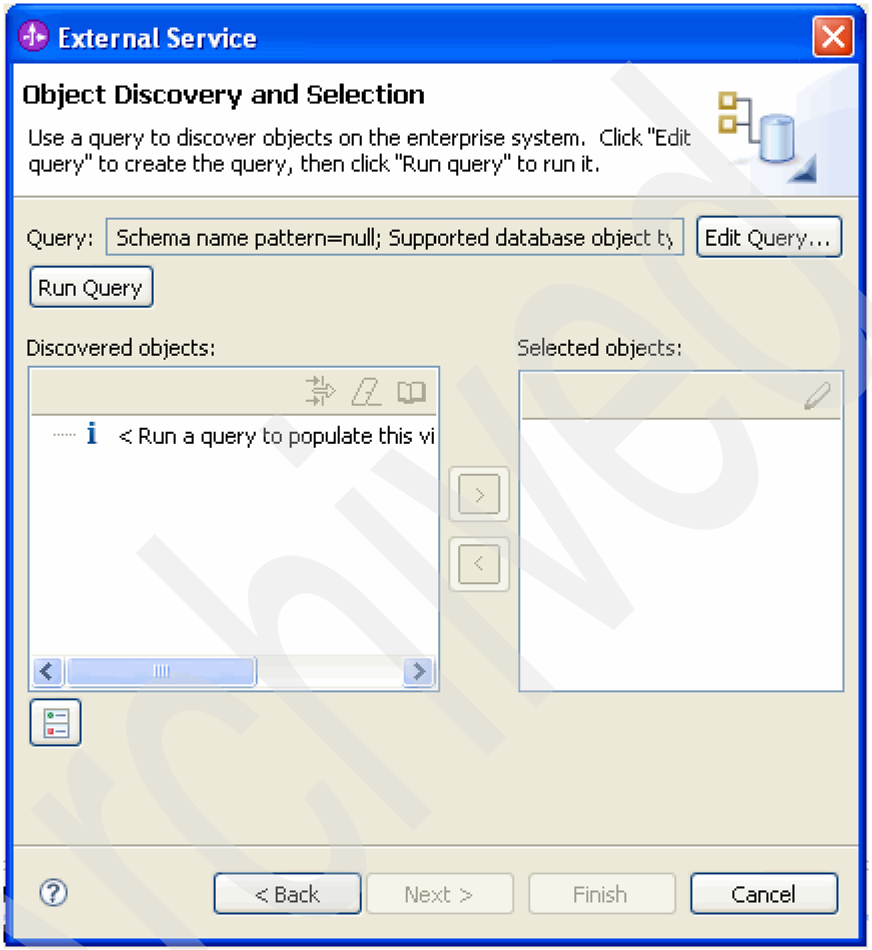


Figure 5-6 Object discover and selection

9. Select **Prompt for additional configuration settings when adding business object** on the Query Properties panel, as shown in Figure 5-7. Then, click **OK**.

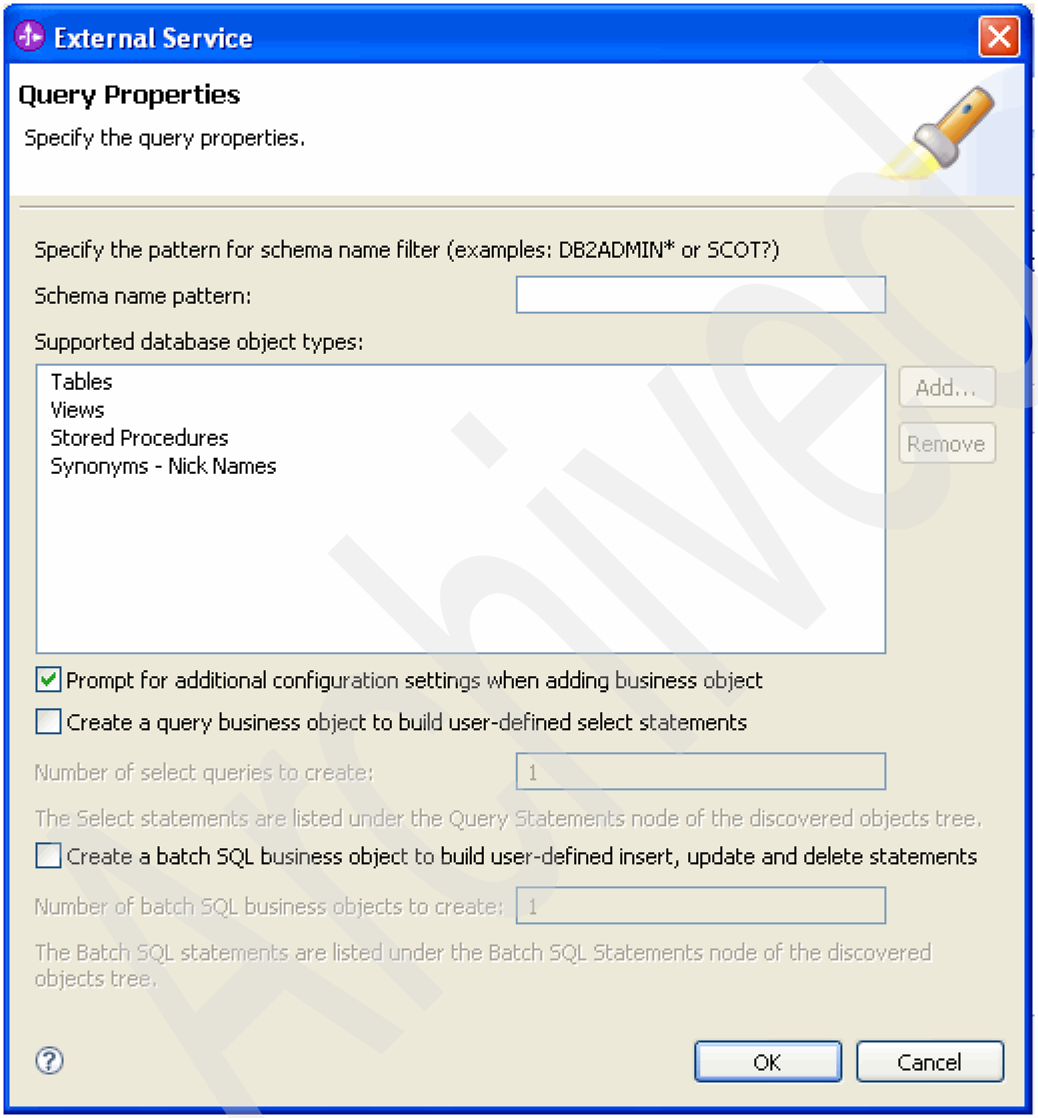


Figure 5-7 Query properties

10. Click **Run Query**.

11. Expand DBADMIN and then Tables. Select **CUSTOMER**, **ORDERHEADER**, and **ITEM** (use the Ctrl key to select multiple items) as shown in Figure 5-8. Then click the **>** icon to move these objects into the Selected objects area.

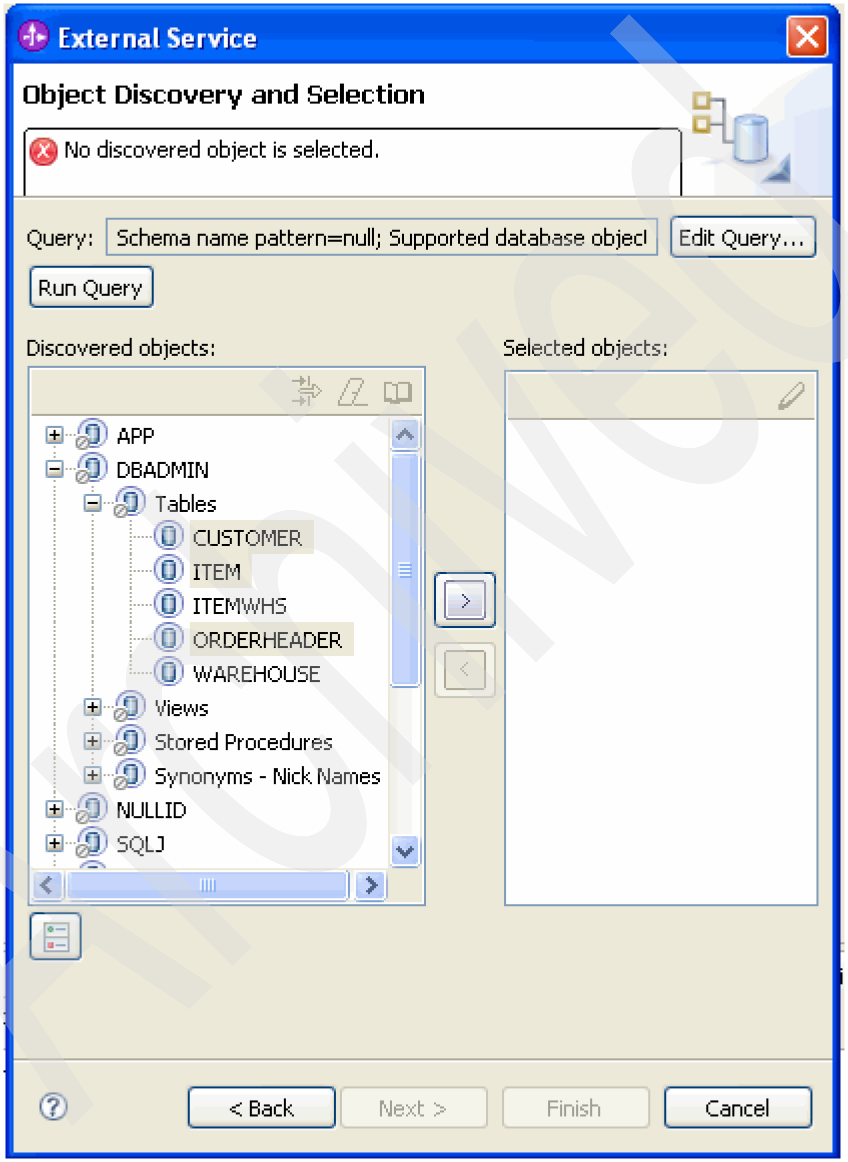


Figure 5-8 Object discovery

12. Click **OK** on Configuration Properties panel.

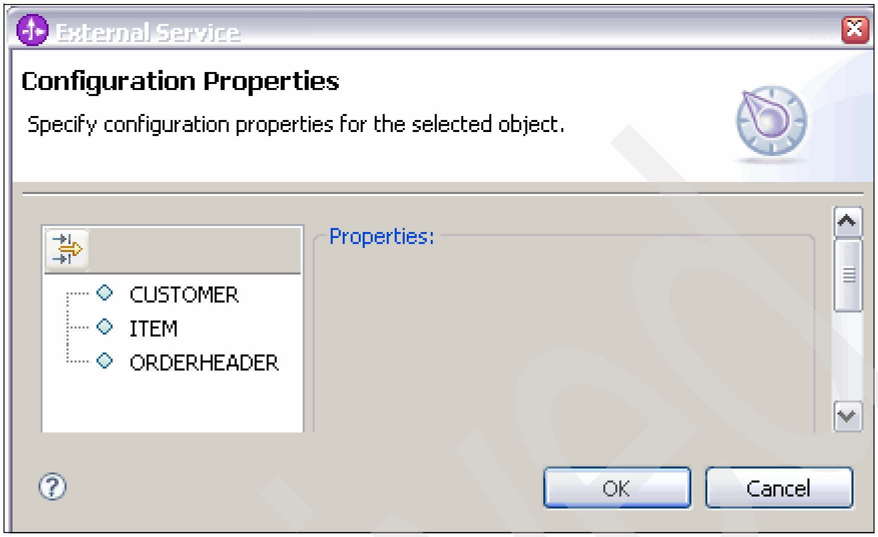


Figure 5-9 Configuration properties for the selected object

13. Now, select only **ITEMWHS** and click the > icon again.

14. On the Configuration Properties for ITEMWHS, shown in Figure 5-10 on page 308, select the following options:

   – **ITEM (DBADMIN)** as a parent table
   – **ITEMID** to build a foreign key relationship
   – Parent object owns child object
   – Preserve ITEMWHS when parent is updated
   – ITEMWHS required for operations on parent

   Click **OK** and then **Next**.

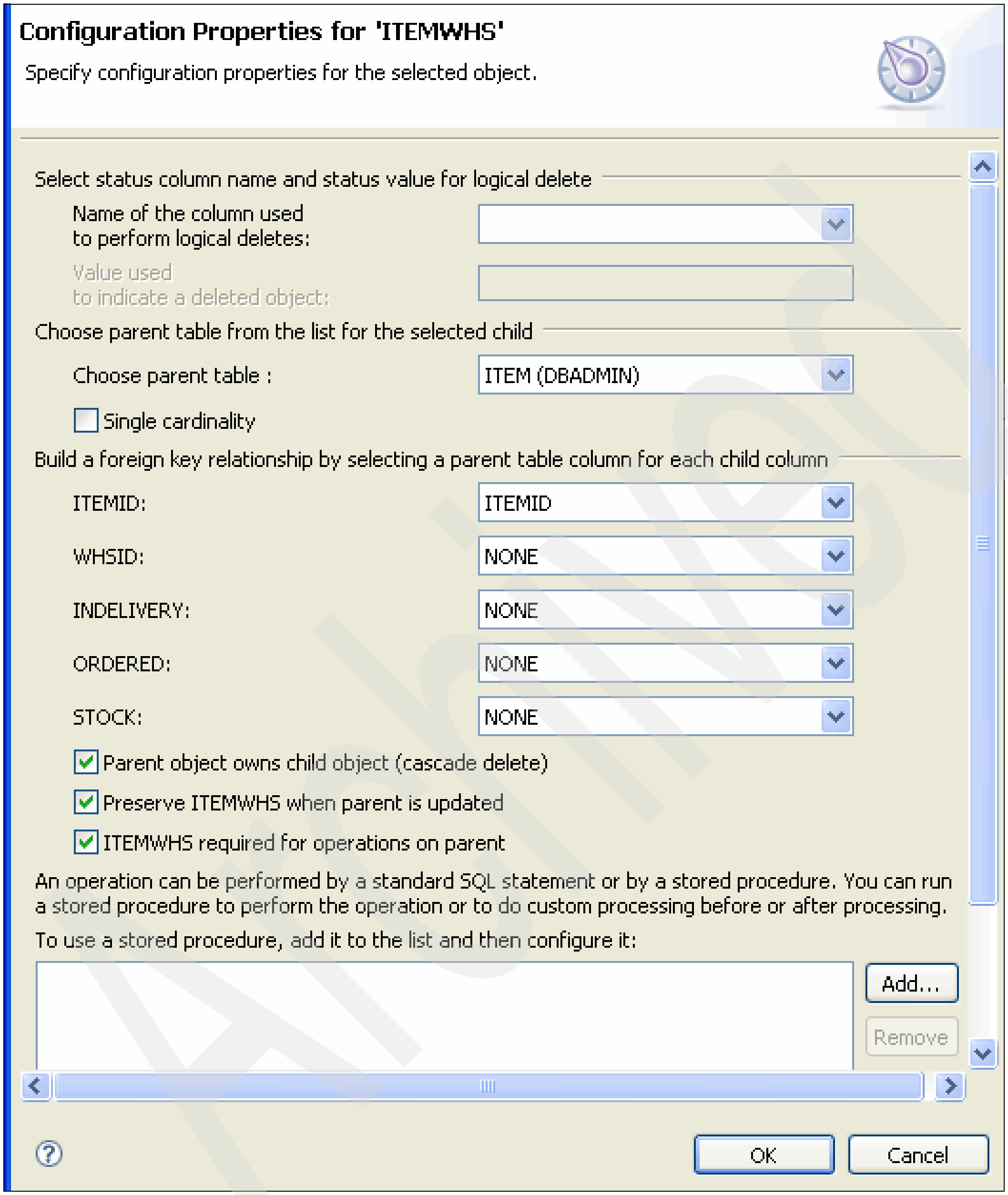


Figure 5-10 Table configuration properties

15. In the Configure Composite Properties panel (shown in Figure 5-11), note that maximum records for the RetrieveAll operation is set to 100 by default and that business graph generation is optional. Click **Next**.

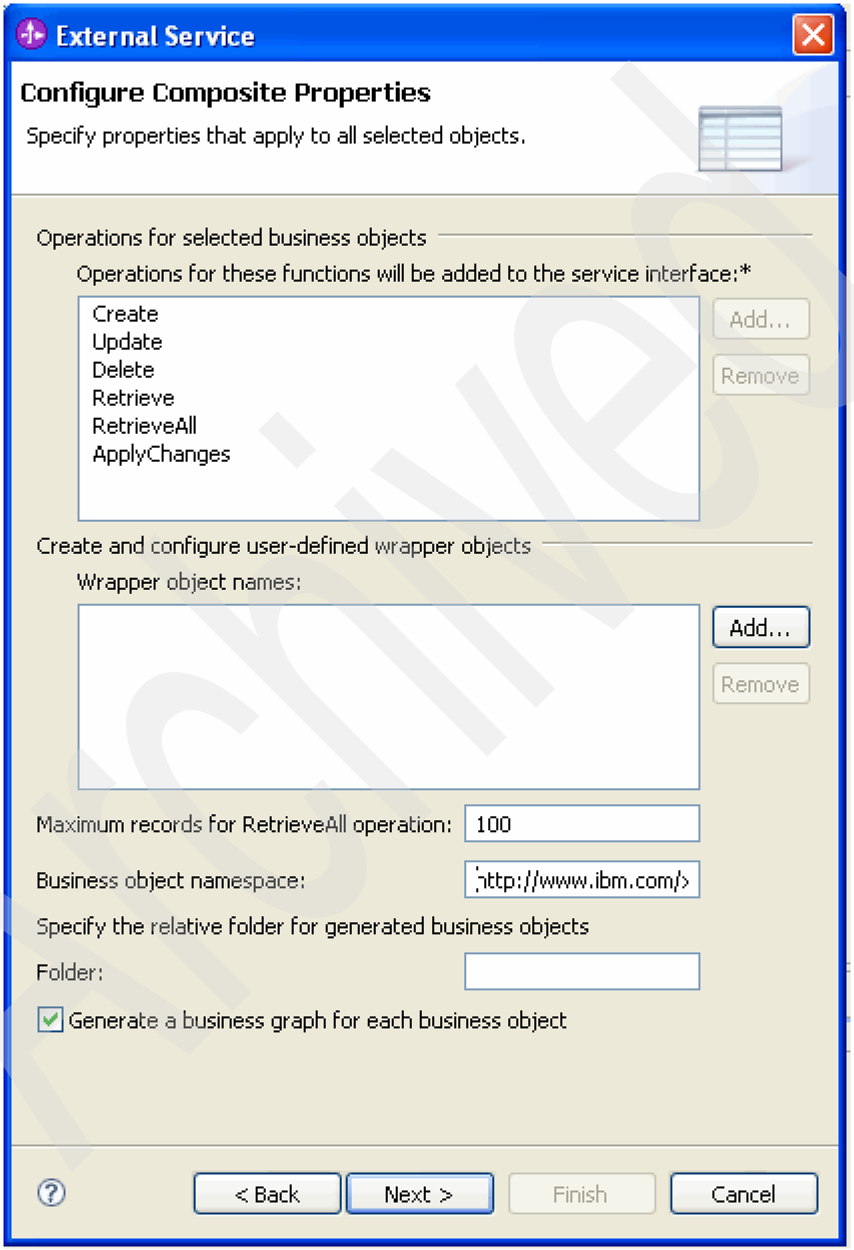


Figure 5-11 Composite properties

16.On the next panel, shown in Figure 5-12:

a. Deselect **Specify a Java Authentication and Authorization Services (JAAS) alias security credential**.

b. Select **With module for use by single application** in the Deploy connector project field.

c. Verify that the connection properties are correct.

d. Click **Next**.

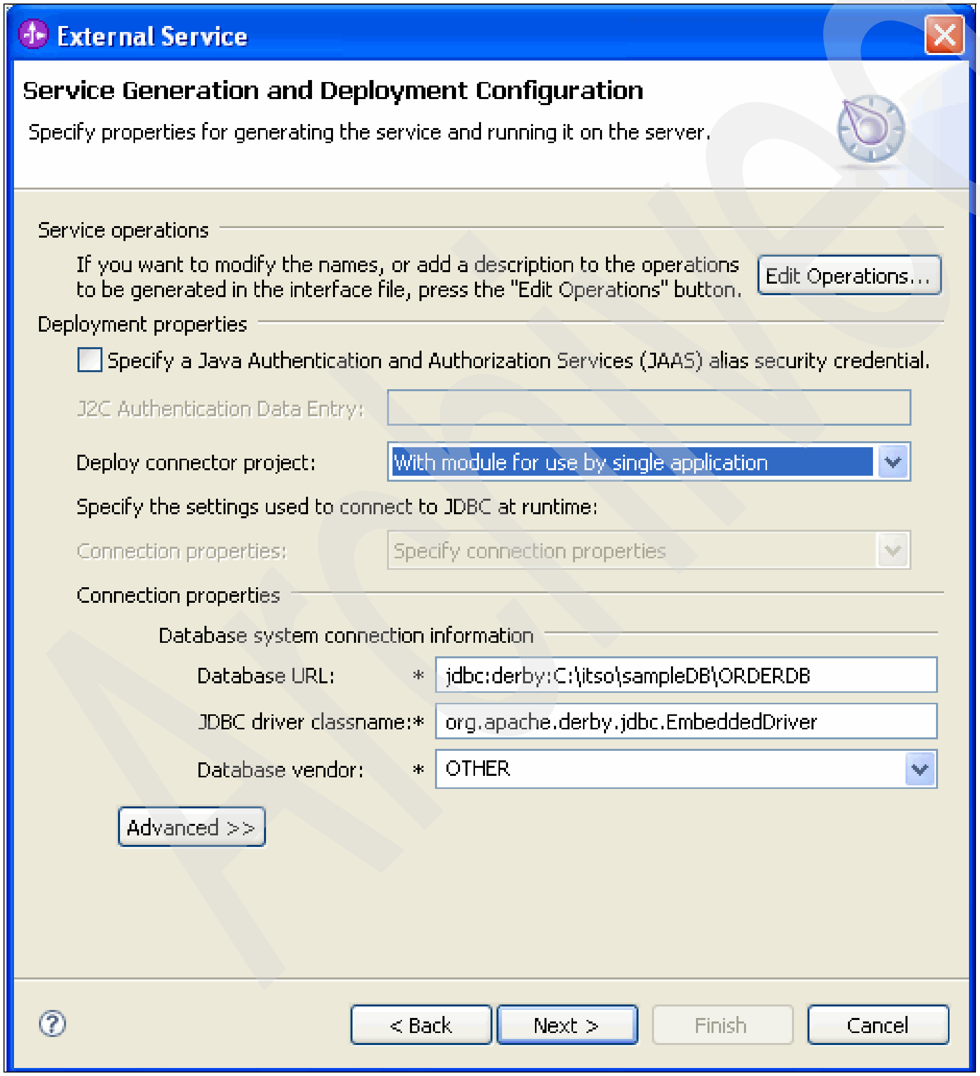


Figure 5-12 Service generation and deployment configuration

17.If it is the first time that you have run the external services wizard, you are prompted to enter a module name to which to deploy this connector project. Because you will create a DBMS service mediation module, you can create a new module now.

Click **New**.

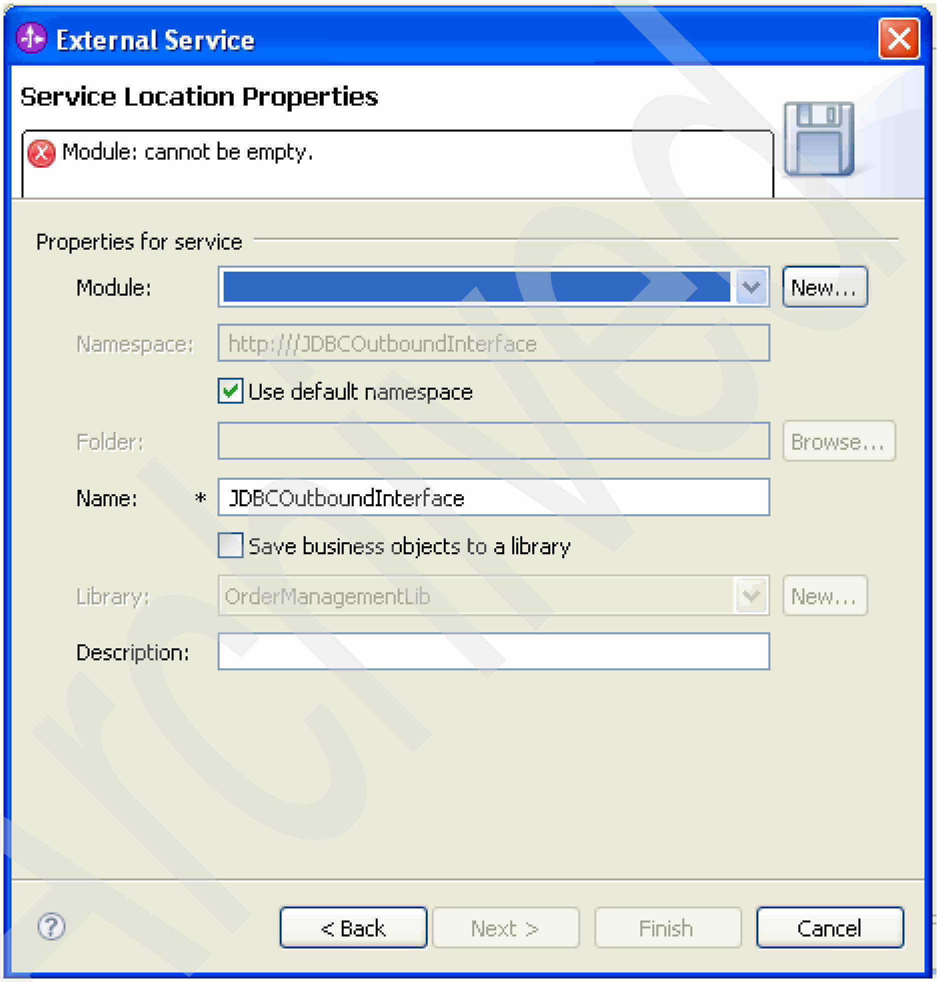


Figure 5-13 Service location properties

18. In the New Integration Project dialog box, shown in Figure 5-14, choose **Create a mediation module project** and click **Next**.

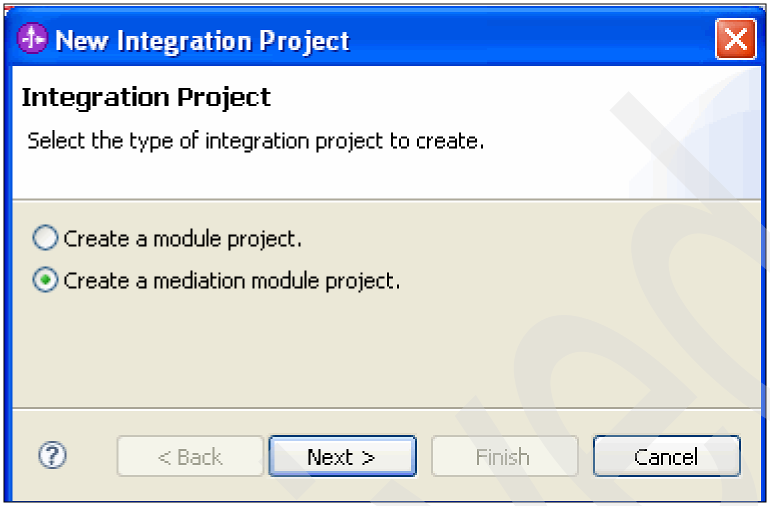


Figure 5-14 Create a new mediation module project

19. On the New Mediation Module panel (shown in Figure 5-15 on page 313):
   - Give the module a name: `DBMSServiceMediation`.
   - Leave **Use default location** selected.
   - Set the Target runtime to **WebSphere ESB Server v6.1**.
   - Leave **Create mediation component** checked.
   - Give mediation component a name: `DBMSServiceMediation`.

   Click **Next**.

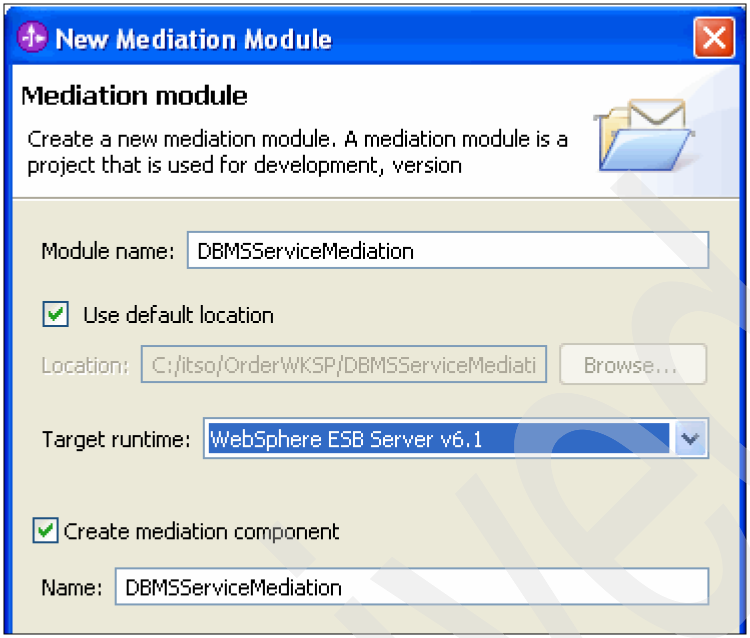


Figure 5-15 New mediation module properties

20. The next panel allows you to select the required libraries for the new module. Select **OrderManagementLib**, and click **Finish**.
21. Click **Finish** on the Service Location panel.
22. If you are prompted to load the updated files on the file system, answer **Yes**. Otherwise, you might lose changes that were produced by the system during previous steps.

    The new mediation module is built and the assembly diagram opens.

    The new CWYBC_JDBC project that displays in the Business Integration view is a JDBC database connectivity module that can be packaged and deployed with any application that requires database services.

    > **Note:** The DBMSServiceMediation component has no business logic at the moment. You will deal with this issue a bit later.

23. Save the changes on the assembly diagram by clicking Ctrl+S.

Figure 5-16 shows the completed assembly diagram.

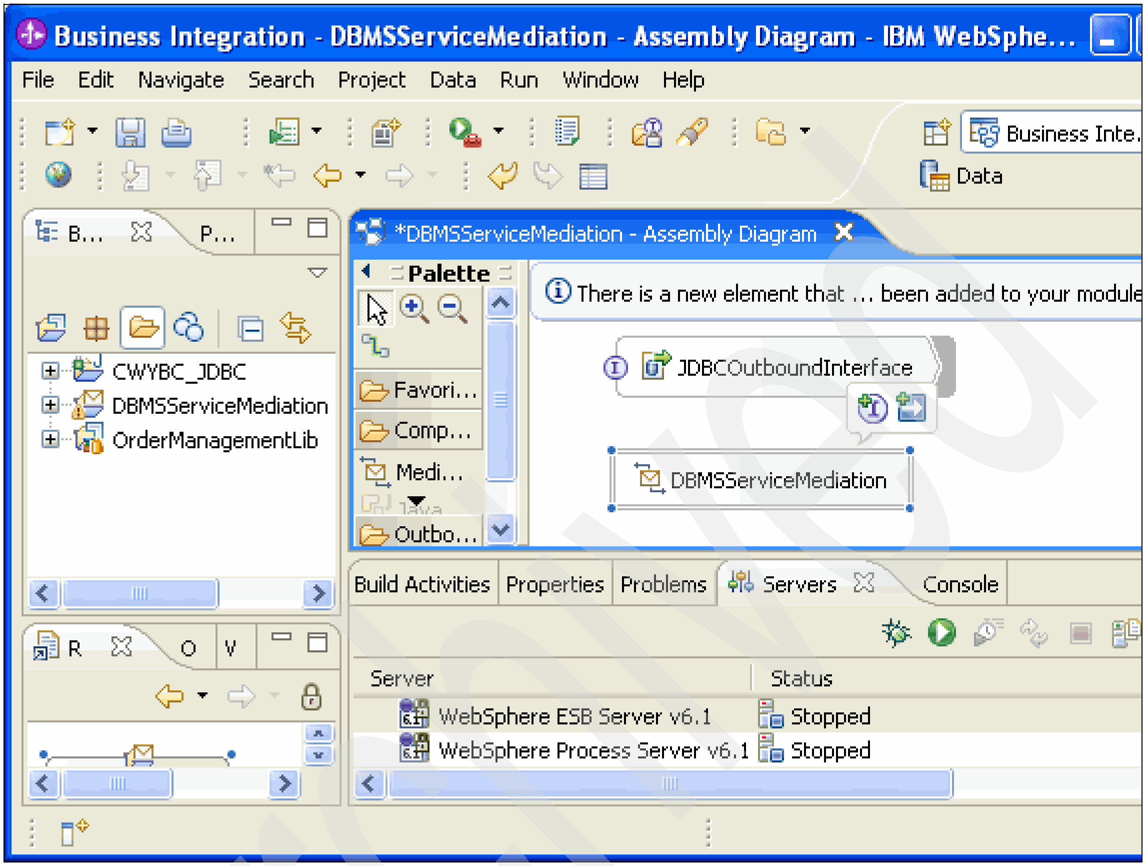


Figure 5-16 DBMSServiceMediation assembly diagram

Update the data types

To update the data types:

1. Expand the DBMSServiceMediation project and expand the Data Types folder to see data types that are generated by the JDBC connector project, as shown in Figure 5-17.

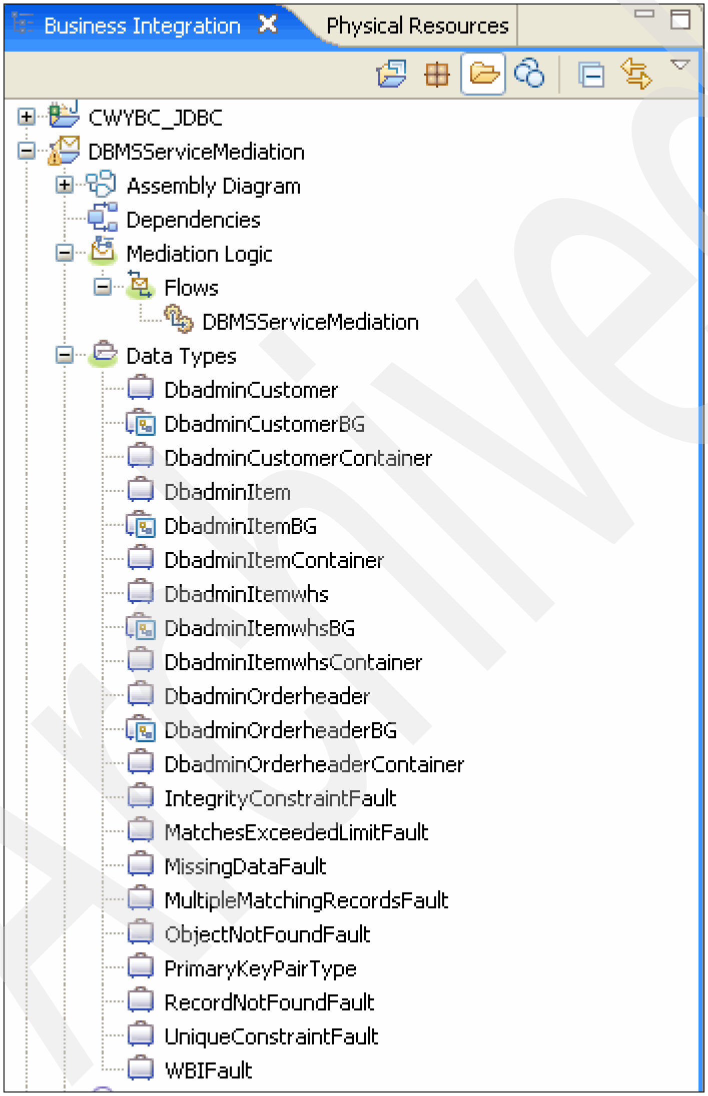


Figure 5-17 New data types created for the adapter

2. Double-click the `DbadminItem` data type.

   In addition to the fields that are defined in the ITEM table, it has a field called `itemwhsdobj` that is an array of DbadminItemwhs objects. Because this object represents warehouses where a particular item is stocked, you rename it to `warehouses`.

   Select **itemwhsobj**, and type `warehouses` over it. Then, press Alt+Shift+R to refactor, as shown in Figure 5-18.

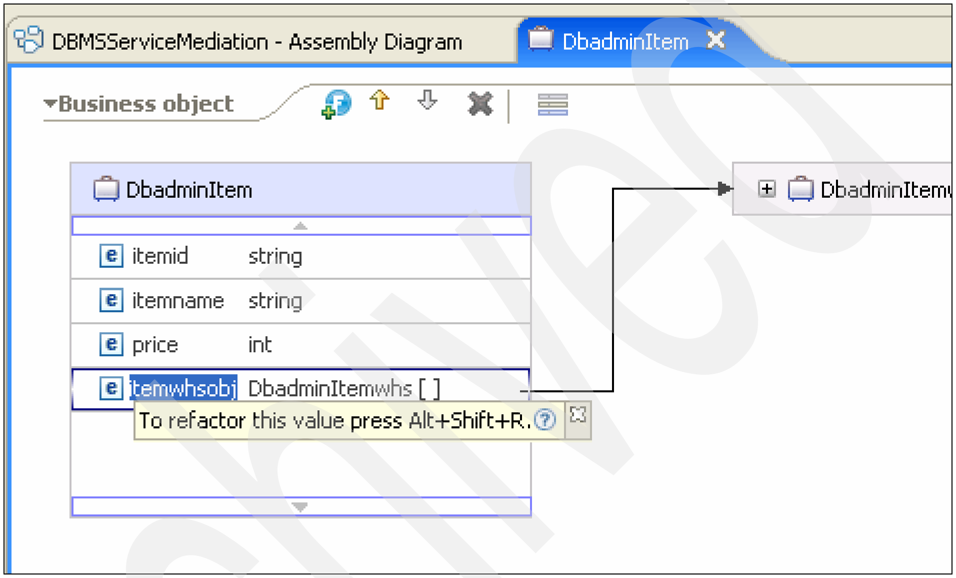


Figure 5-18 Change the field name and refactor

3. Click **OK** to save all modified resources.
4. Then, click **OK** again on the Rename window (shown in Figure 5-19).

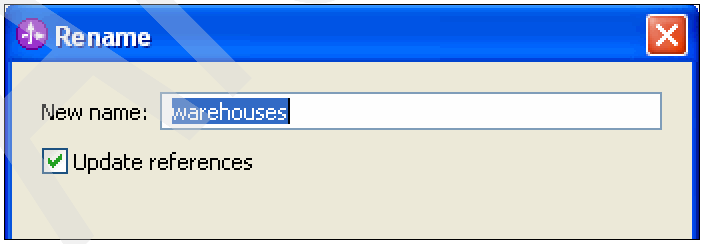


Figure 5-19 Rename the business object field

   When the project finishes rebuilding, make sure that there are no errors. Warnings are usually fine.

The DbadminItem business object looks similar to that shown in Figure 5-20.

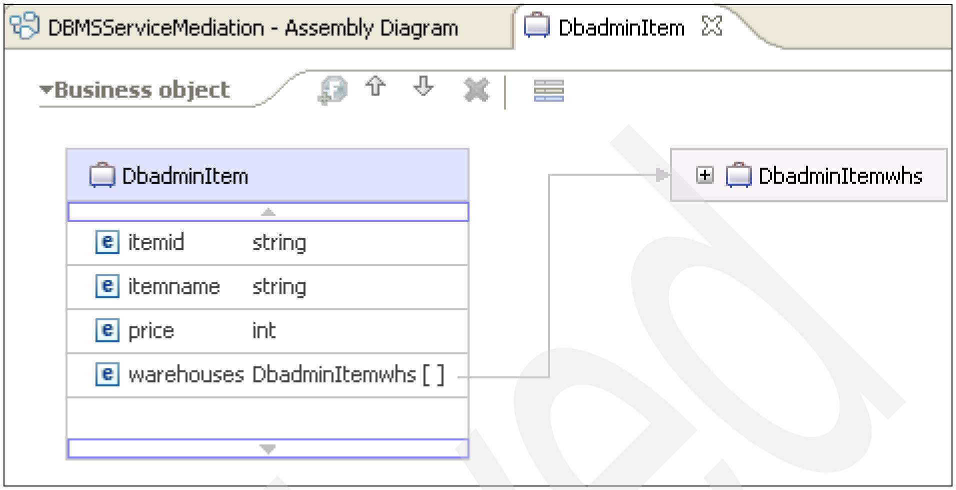


Figure 5-20 DbadminItem business object

5. Now, select the **warehouses** field of DbadminItem, and open the Description tab in the element's Properties. Make sure that the **Array** property is selected, as shown in Figure 5-21.

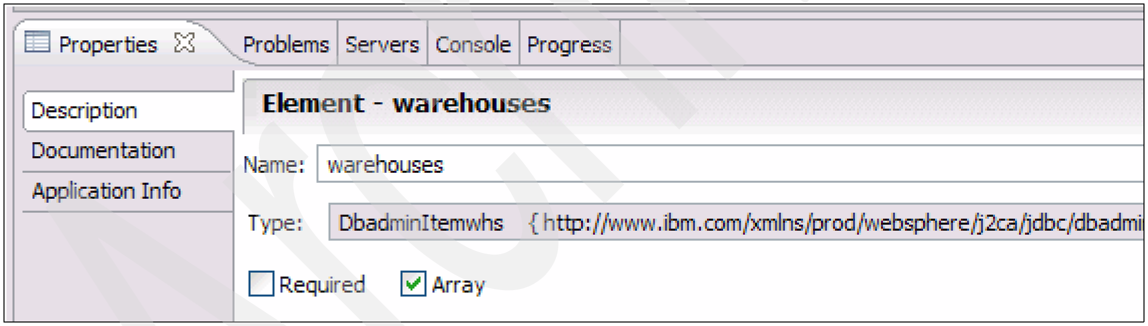


Figure 5-21 Make the warehouses field an array

6. Next, click the Application Info tab, expand JDBC ASI schema, and expand JDBCAttributeTypeMedatdata. Then:

   a. Set the Ownership attribute to `true`.

   b. Delete the `keepRelationship` attribute

   c. Add the `ChildBOType` attribute.

   To add the ChildBOType, select **JDBCAttributeTypeMetadata**, and right-click in the white space. Select **New** → **jdbcasi:ChildBOType** as shown in Figure 5-22.

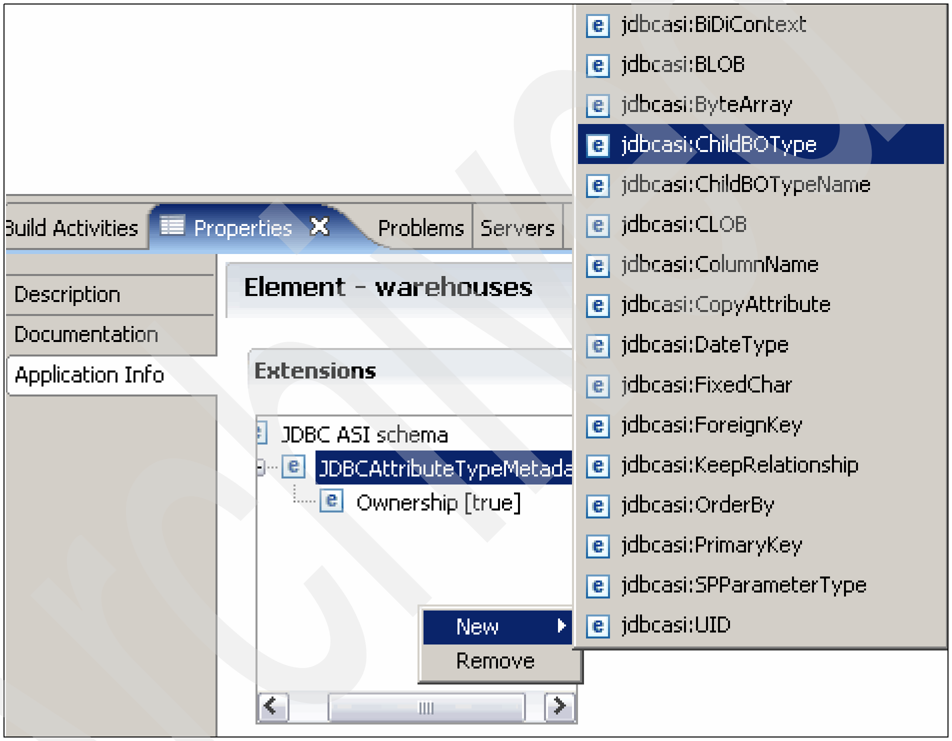


Figure 5-22 Business object field extensions

d. Select **Array** as the text value as shown in Figure 5-23.

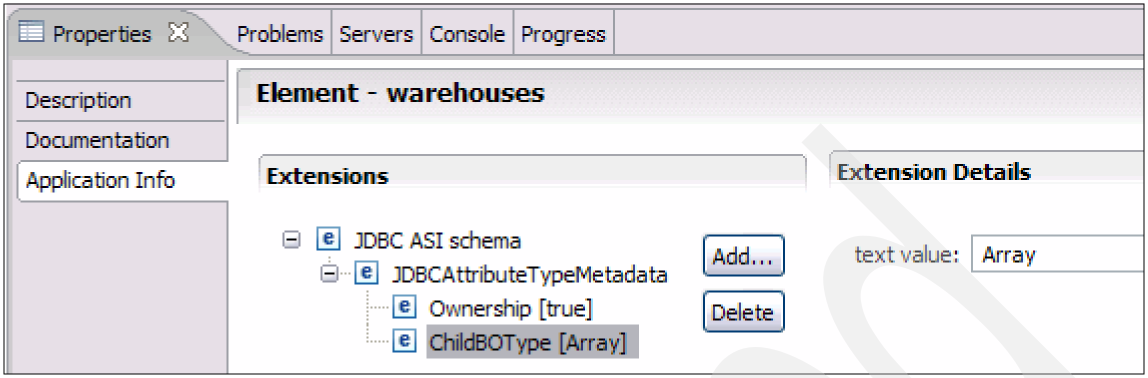


Figure 5-23 The warehouses field application information attributes

7. Review the remaining generated data type elements to make sure that their attributes correctly describe the matching attributes in the database schema, for example, primary key, value required, and so forth.
8. Save all the changes.

Now, you can test the adapter and make sure that the JDBC outbound services are working properly. To deploy the application to the server, you need to create a J2C authentication alias and configure data sources.

5.3 Defining the database runtime resources

To run the mediation on a server, you need to create a J2C authentication alias and data source for the database.

Create the authentication alias

The authentication alias contains the credentials that are required for the mediation to access the ORDERDB database. To create the authentication alias:

1. Start WebSphere ESB v6.1 server from Servers tab of WebSphere Integration Developer. Wait until the server status is started.
2. Open the administrative console by right-clicking the server and selecting **Run administrative console**.
3. In the console, select **Security** → **Security administration, application, and infrastructure**.

4. In the Authentication section on the far right of the panel, expand **Java Authentication and Authorization Service** in the Authentication section. Click **J2C authentication data**.
5. Click **New** and enter the following values
   - Alias: `DerbyDS`
   - User ID: `dbadmin`
   - Password: `dbadmin`

   Click **OK** to save the changes.
6. Click **Save** in the Messages box that displays.

5.3.1 Creating the JDBC provider and data source

The JDBC provider and data source provide the information that is required to access the database. The JDBC provider is specific to the database provider implementation. If you already have a JDBC provider at the proper scope, you do not need to create one again.

To create the JDBC provider:

1. In the navigation area of the console, select **Resources** → **JDBC** → **JDBC Providers**.
2. Set scope at the node level.
3. Click **New** and enter the following values (as shown in Figure 5-24 on page 321):
   - Database type: Derby
   - Provider type: Derby JDBC Provider
   - Implementation type: XA data source
   - Name: `Derby JDBC Provider (XA) for ORDERDB`
4. Click **Next** to verify data.

   Note that the ${DERBY_JDBC_DRIVER_PATH} variable is used by default. If you have trouble using this provider later, ensure that this environment variable points to the correct path on your system.

   Click **Finish** and then **Save**.

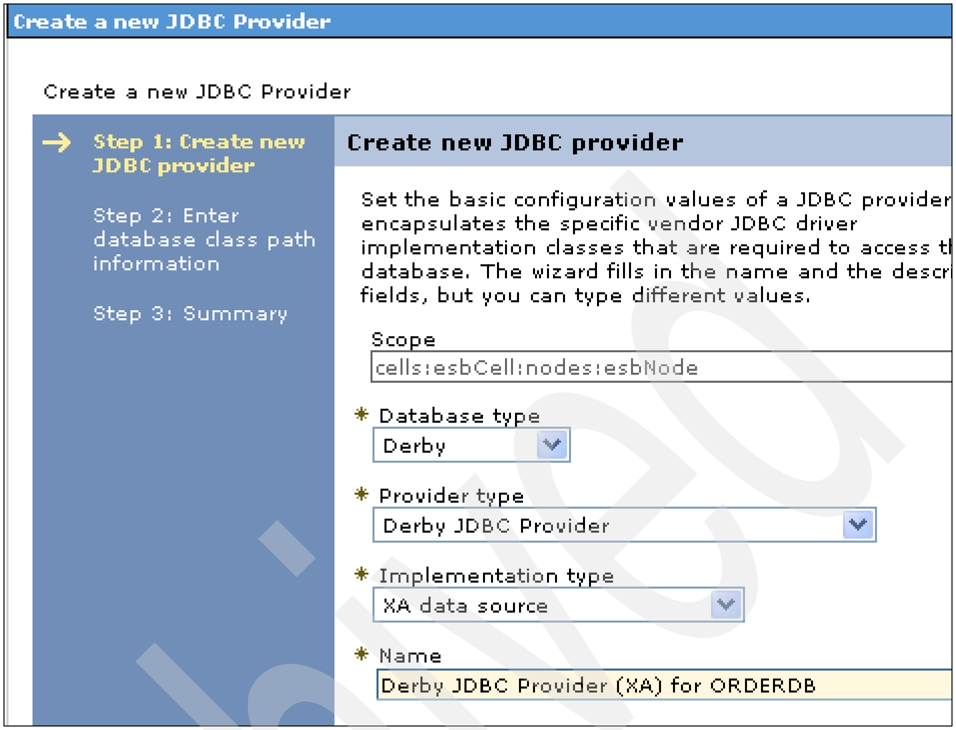


Figure 5-24 Create a new JDBC provider

5. Now the JDBC providers panel looks similar to that shown in Figure 5-25. Click **Derby JDBC Provider (XA) for ORDERDB**.

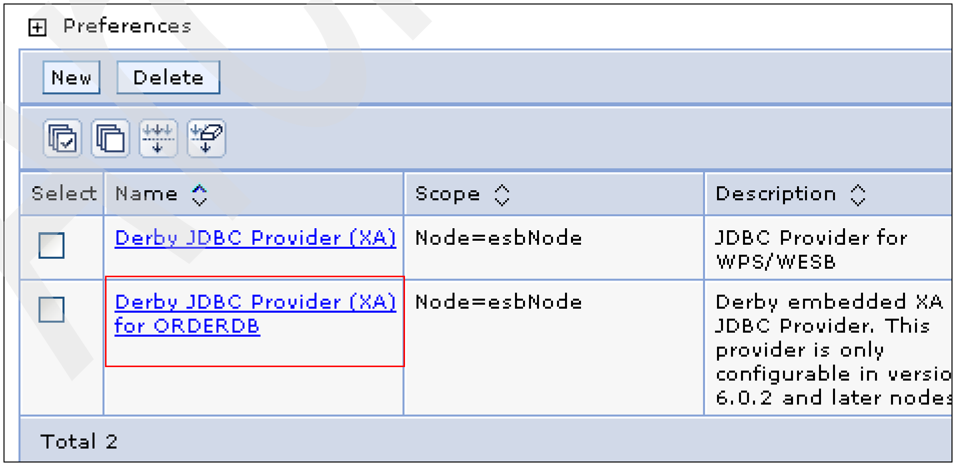


Figure 5-25 List of JDBC providers

6. Click **Data source** under Additional Properties.
7. Click **New**, and enter the following values, as shown in Figure 5-26:
   - Data source name: `Derby JDBC Driver XA DataSource for ORDERDB`
   - JNDI name: `jdbc/DerbyORDERDB`
   - From the drop down list of “Component-managed authentication alias and XA recovery authentication alias, select the authentication alias created for the database.

   Click **Next**.

Figure 5-26 Basic data source information

8. On the next panel, enter the path to the database, C:\itso\sampleDB\ORDERDB, as shown in Figure 5-27. Click **Next**.

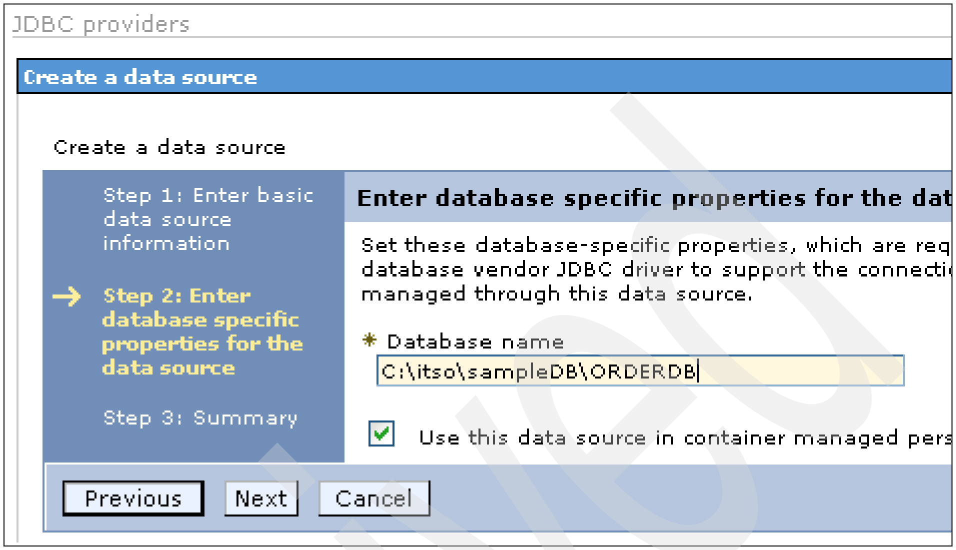


Figure 5-27 Enter the database name

9. Click **Finish** on the next panel. Then, click **Save**. A new Derby JDBC Driver XA DataSource for ORDERDB displays as shown in Figure 5-28.

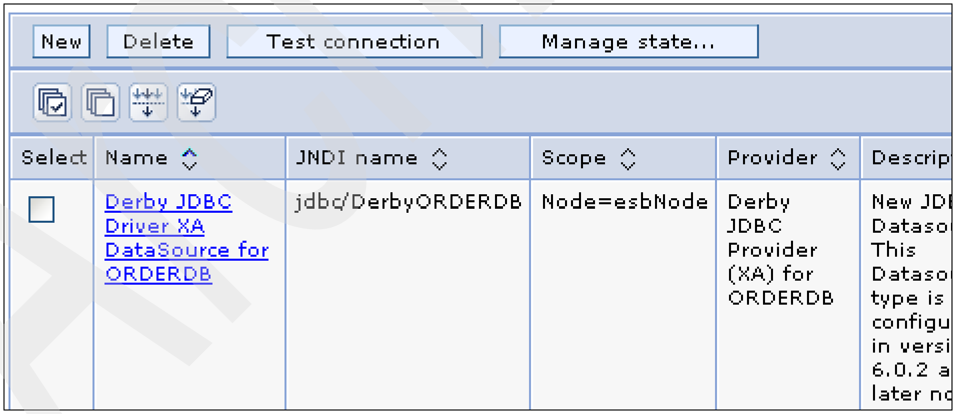


Figure 5-28 New data source

10. Select the box to the left of the data source and click **Test connection**. It should show success. However, you might see an error message similar to that shown in Figure 5-29.

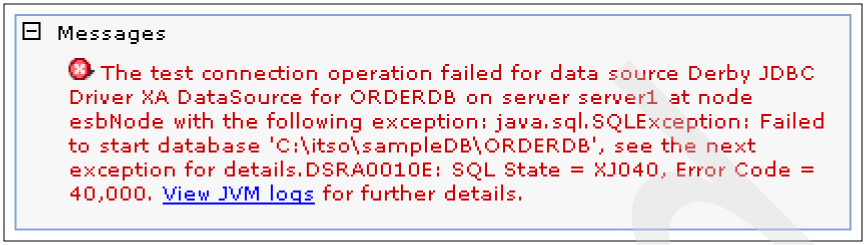


Figure 5-29 Test connection fails

This error can be caused by an unclosed connection to the ORDERDB database that occurred in previous steps. Because you are using Derby Embedded, only one connection can be active at a time. To correct this issue:

a. Make sure that you do not have any running applications that might be accessing the database.

b. Save all unsaved projects in WebSphere Integration Developer and exit from the workspace altogether. You can shut-down WebSphere Integration Developer without stopping the server. In this case, of course, you need to be connected to the Administrative console by a browser outside of WebSphere Integration Developer.

c. Click **Test connection** again. This time it should be successful.

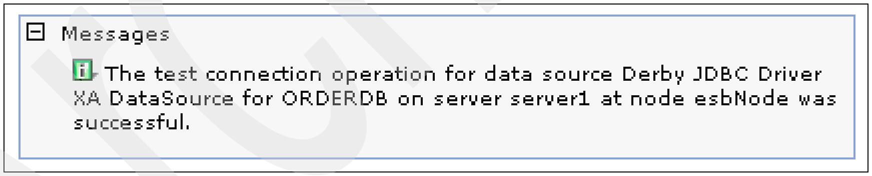


Figure 5-30 Test connection success

Launch WebSphere Integration Developer again, and open the workspace that you exited from on previous step. You should see WebSphere Integration Developer reconnected to the running WebSphere ESB instance.

5.4 Testing the JDBC outbound interface

Next, you can test the JDBC Outbound Interface as follows:

1. Make sure the server is up and running.
2. Open the assembly diagram for the DBMSServiceMediation project, shown in Figure 5-31.

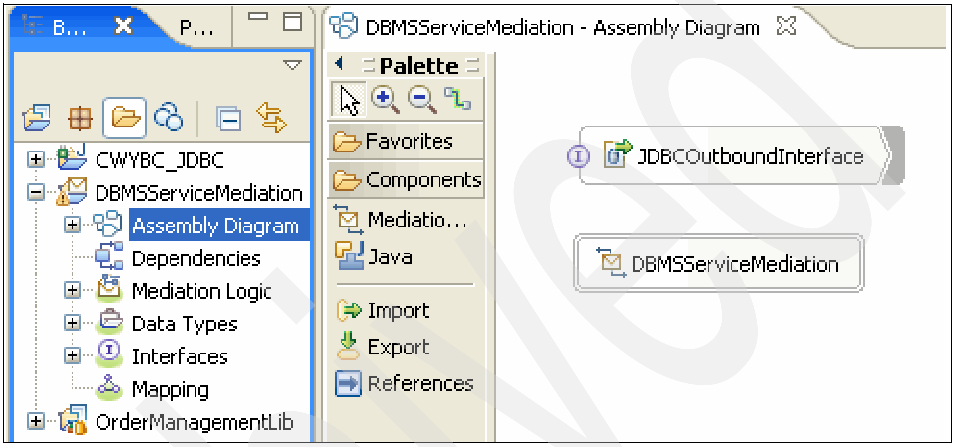


Figure 5-31 DBMSServiceMediation assembly diagram

3. Right-click the JDBCOutboundInterface component and select **Test Component** from the pop-up menu.
4. Click the Configurations tab and make sure that no emulators are defined.
5. In the Events tab, select the **retrieveallDbadminCustomerBG** operation, as shown in Figure 5-32.

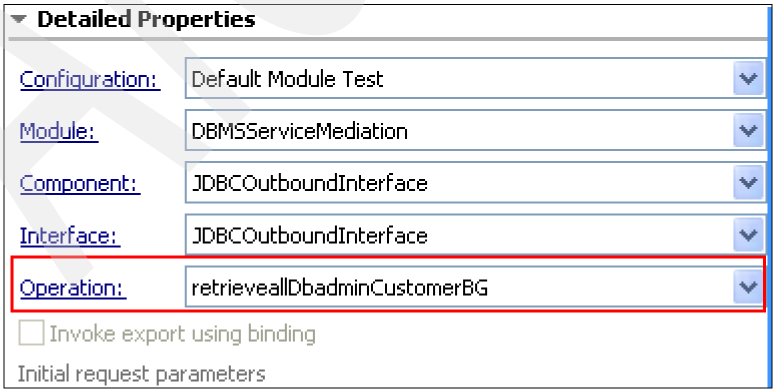


Figure 5-32 Test properties

6. You want to retrieve all the records from the Customer table. Unset the verb and all of the fields.

   In the Initial request parameters panel, select the **verb** and each field under **DbadminCustomer** object by pressing and holding the Ctrl key while selecting each item.

   Right-click in the selected area, and select **Set To** → **Unset** from the pop-up menu as shown in Figure 5-33.

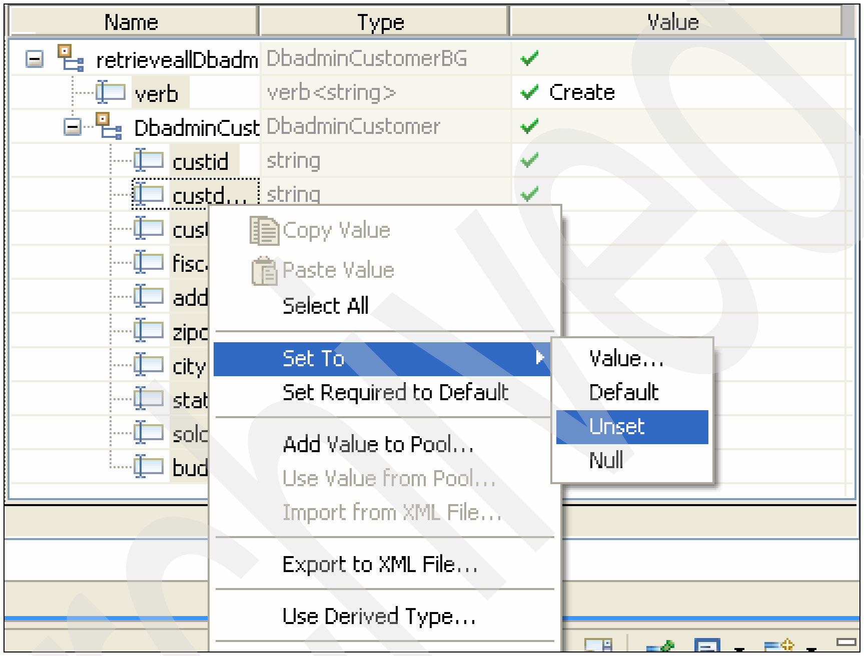


Figure 5-33 Unset the verb

7. The view changes, and a small x sign displays in the Value column as an indicator that no value is set for this parameter. See Figure 5-34.

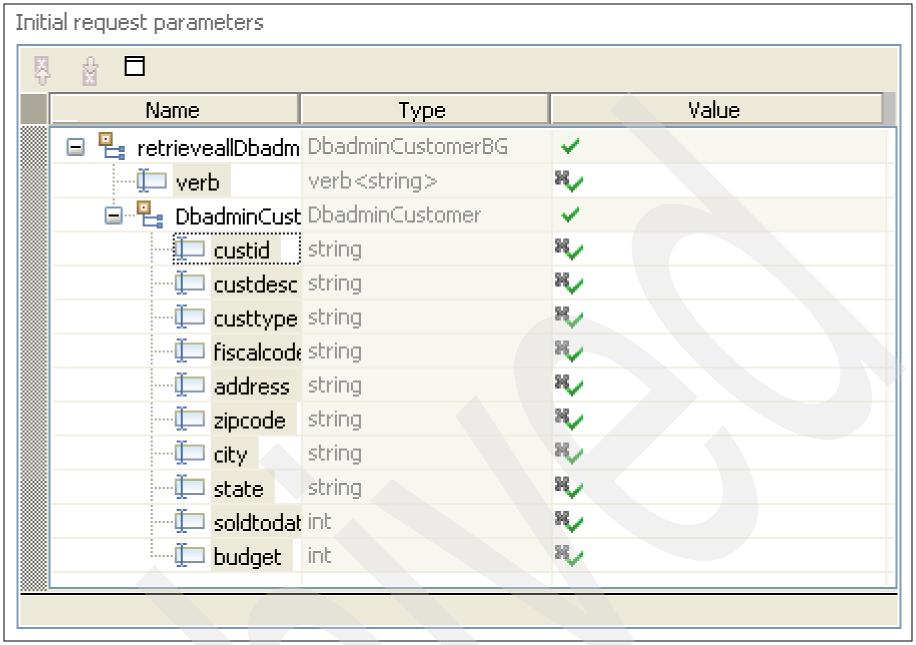


Figure 5-34 New parameter settings

Note, that if you select DbadminCustomer object rather than its fields and unset the values, the view collapses as shown in Figure 5-35.

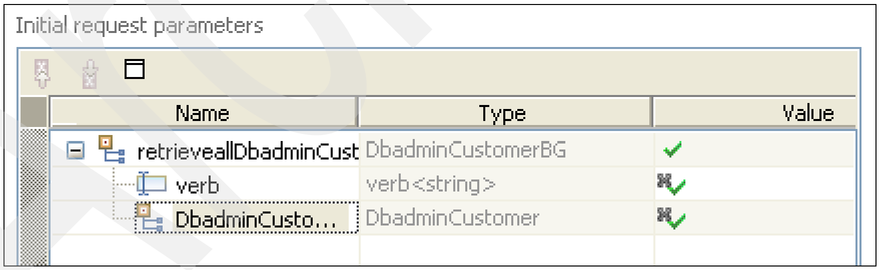


Figure 5-35 Selecting unset for DbadminCustomer

Alternatively, the view might even collapse to Figure 5-36.

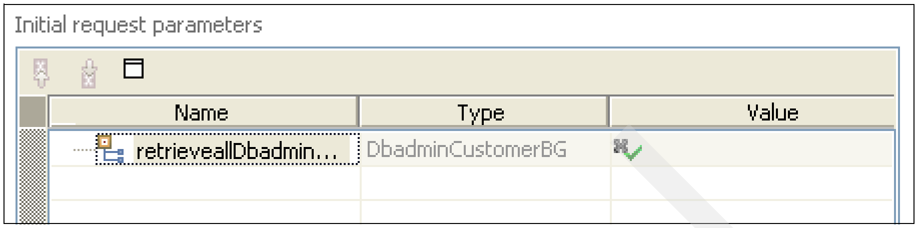


Figure 5-36 Selecting unset for DbadminCustomer

This is wrong. You will get a Null pointer run-time exception. To correct the situation, right-click anywhere on each disabled object and choose **Set To** → **Default**. Alternatively, you can choose **Select All** first and **Set To** → **Default** next.

8. After you set the verb and all fields under DbadminCustomer correctly, click the **Continue** icon on the Events menu bar.

9. If you are doing this for the first time and the application has not been deployed, a Deployment Location window opens. Choose **WebSphere ESB Server v6.1**, and click **Finish**.

   Because the server is secured by default, you get a User Login prompt with default user ID and password (which are `admin` and `admin` by default). Click **OK**, and allow some time for the test client to start and load the test application.

   The Invoke operation should start and return successfully, as shown in Figure 5-37.

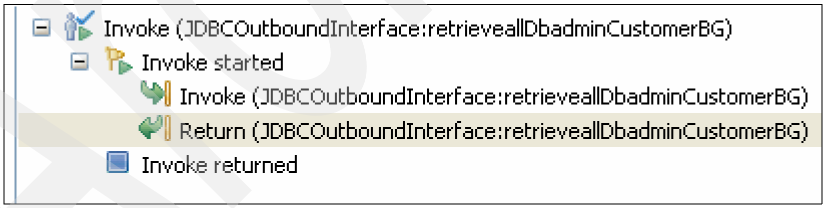


Figure 5-37 Test results

The Detailed Properties panel shows an array of DbadminCustomer objects returned, as shown in Figure 5-38. In fact, all customer records from CUSTOMER table are retrieved.

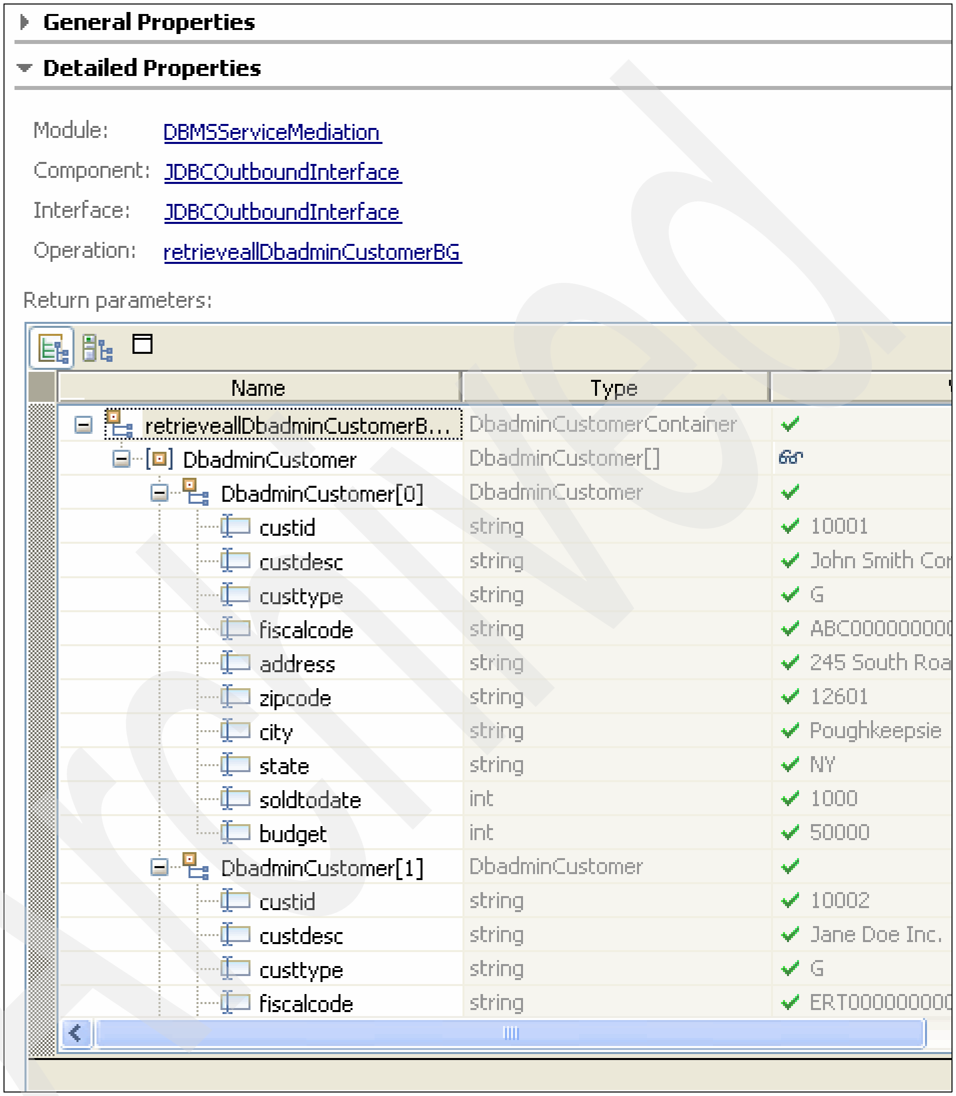


Figure 5-38 Test results

10. As a separate test you might want to set one field to a concrete value and leave the rest unset. For example if you want to see all records for *John Smith Corp.* your Initial request parameters look as shown in Figure 5-39.

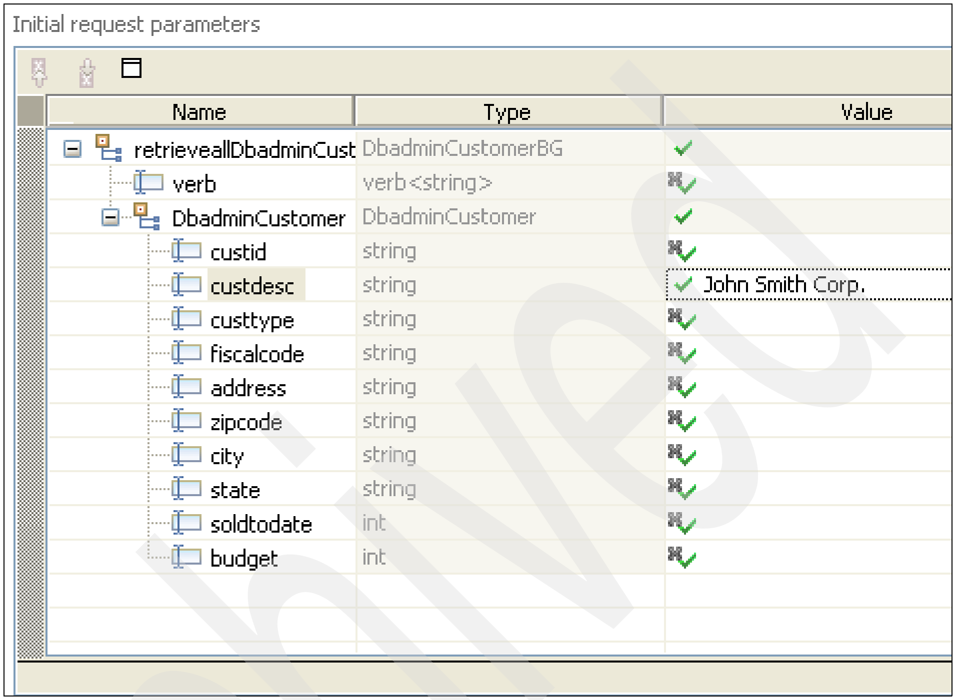


Figure 5-39 Test input for one customer

Click **Continue** and you get all `John Smith Corp.` records, as shown in Figure 5-40.

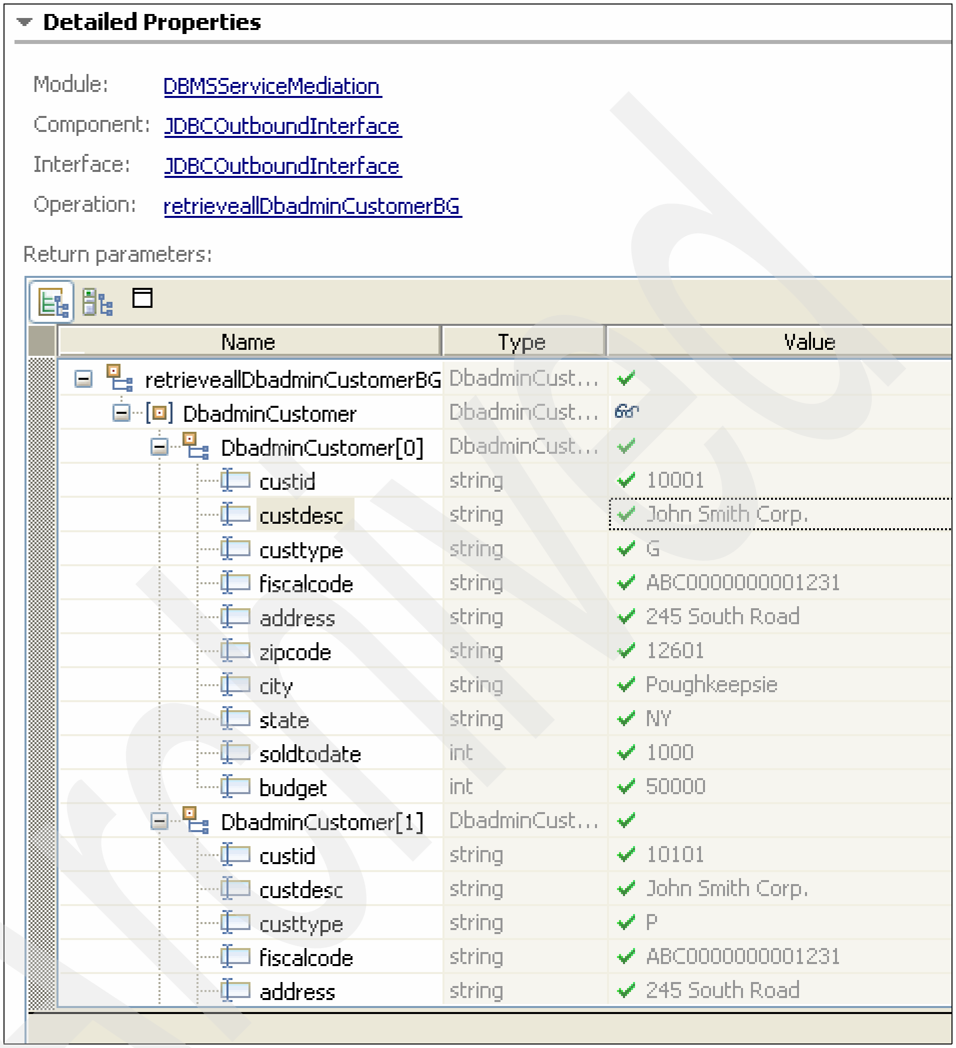


Figure 5-40 Test results

If your input parameters are set to values not found in the CUSTOMER table, you get a RecordNotFound exception as shown in Figure 5-41.

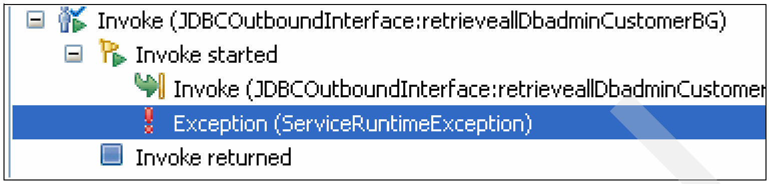


Figure 5-41 Exception for record not found

Instead of the fields returned from the query, a stack trace appears in the detailed properties area, as shown in Figure 5-42.

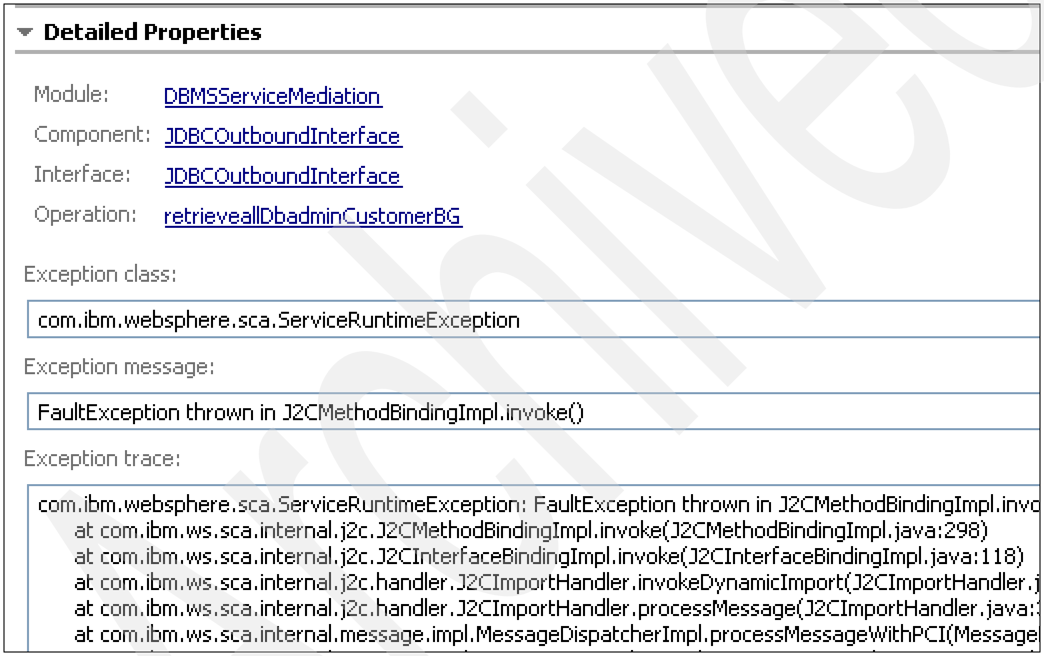


Figure 5-42 Stack trace for the exception

The first line of the Exception trace is quite long. At the very end of it, you can see a possible reason for exception, as follows:

```
com.ibm.websphere.sca.ServiceRuntimeException: FaultException thrown
in J2CMethodBindingImpl.invoke()
com.ibm.j2ca.base.exceptions.RecordNotFoundException: No matching
records found., error code:
```

Now, you are ready to build a mediation module to connect each of the following interfaces with JDBCOutbound interface to tap into database services that are provided by the WebSphere Adapter for JDBC:

- CustomerServiceIF
- ItemServiceIF
- OrderManagementIF
- OrderServiceIF

5.5 Building the mediation flow

So far, you have created a mediation module with no business logic that acts as a placeholder. The mediation component is visible in the assembly diagram shown in Figure 5-43. Now, you can build the mediation flow.

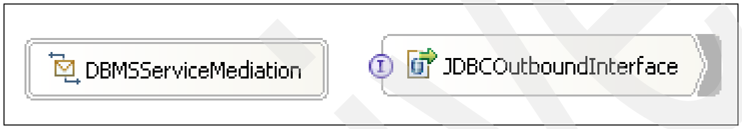


Figure 5-43 DBMSServiceMediation assembly diagram

5.5.1 Adding the interfaces to the mediation flow component

The DBMSServiceMediation module has three interfaces. You need to add these interfaces to the module in the assembly diagram as follows:

1. Add an interface to the DBMSServiceMediation component by right-clicking the component and selecting **Add** → **Interface**.

> **Tip:** You should see a list of interfaces that are available to the module. In this scenario, the interfaces include interfaces from the OrderManagementLib library.
>
> If you do not see the interfaces that you expect to see, make sure that you have defined any libraries that you need as a dependency. For example, right-click **DBMSSeviceModule** in Business Integration view, and choose **Open Dependencies**. You should see the library listed under Libraries.
>
> If the library is not there, click **Add**, select the library in **Library Selection** window, and click **OK**.

2. Choose **CustomerServiceIF** from the list of matching interfaces and click **OK**.

   A small icon displays on the left edge of the DBMSServiceMediation component. Hover the mouse over this component, and it displays `1 Interface found: CustomerServiceIF (WSDL Interface)` as shown in Figure 5-44.

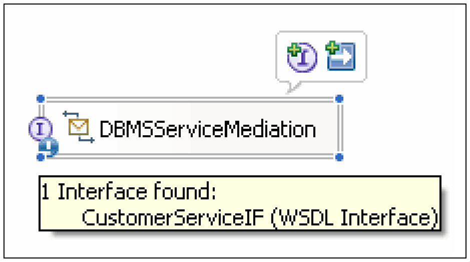


Figure 5-44 Interface for the mediation flow component

3. Similarly add two other interfaces to the mediation flow component:
   - ItemServiceIF
   - OrderServiceIF
4. Wire the DBMSServiceMediation component to the JDBCOutboundInterface as shown in Figure 5-45.

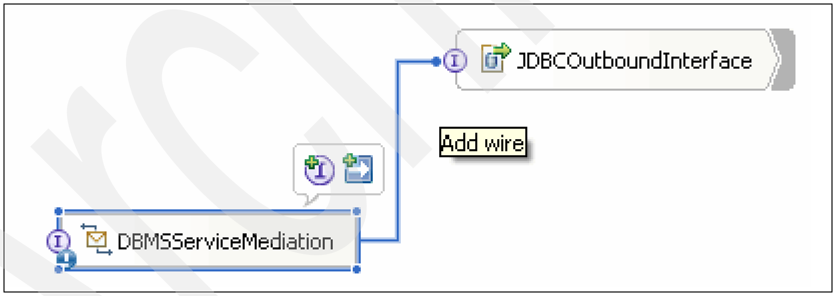


Figure 5-45 Wire the components

   If prompted to allow a matching reference to be created, click **OK**.

5. Save the assembly diagram and make sure that no errors are found.

> **Note:** If you created the reference before you added an interface, the system throws an error. The error does not occur after you add the interface.

5.5.2 Adding mediation logic

If you point to the exclamation mark in the lower left corner of DBMSServiceMediation, you see a warning that the component has no implementation. To fix this issue:

1. Double-click the **DBMSServiceMediation** component, and click **Yes** when prompted whether you want to implement the component now. Then, click **OK** to select the DBMSServiceMediation on the folder.

   The DBMSServiceMediation opens in the mediation flow editor.

   > **Note:** It is worth mentioning that even though you are allowed to add interfaces from within the mediation flow, these interfaces are not public. To add public interface, you have to use assembly diagram.

2. Next connect all OrderManagement operations with the appropriate JDBCOutput adapter operations in the Operations connections section as shown in Figure 5-46.

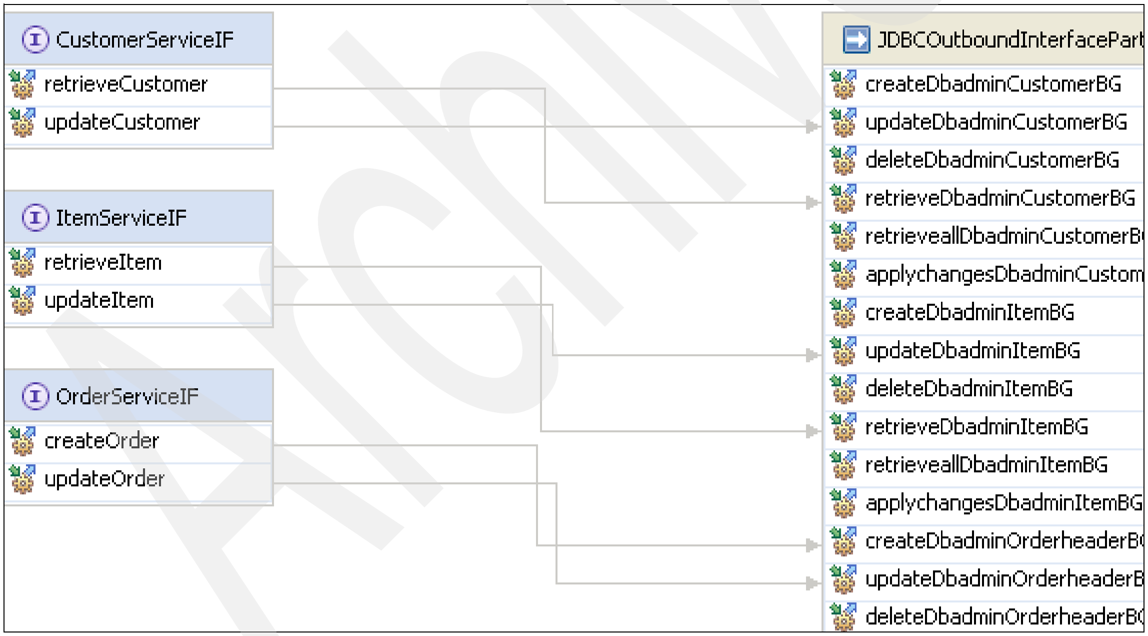


Figure 5-46 Operation connections

Each of these connections represents a mediation flow. Each of the operations that are selected are request-response operations. When you make the connection, a request and response flow is created for each and populated with the appropriate endpoint nodes (that is Input, Callout, and Input Response on the request flow, and Callout Response and Input Response on the response flow).

3. Select each connection in turn, and add the appropriate mediation primitives to the flow to transform the business object that comes in from the interface operation to the business object that is required by the reference partner operation.

XSL transformation for the retrieveCustomer operation

The first flow to update is for the retrieveCustomer operation.

Request flow

To build the request flow for the retrieveCustomer operation:

1. Select the wire between retrieveCustomer and retrieveDbadminCustomerBG in the Operations connection area.
2. Add an XSL Transformation primitive to transform the message. In the Transformation folder of the Palette, click **XSL Transformation primitive** and drop it on the canvas in the flow section.
3. Connect the following as shown in Figure 5-47:
   - The output terminal of the Input node (represents the retrieveCustomer operation of CustomerServiceIF) to the input terminal of XSLTransformation1 node.
   - The output terminal of XSLTransformation1 node to the input terminal of the Callout node (which represents retrieveDbadminCustomerBG operation of JDBCOutboundInterface).

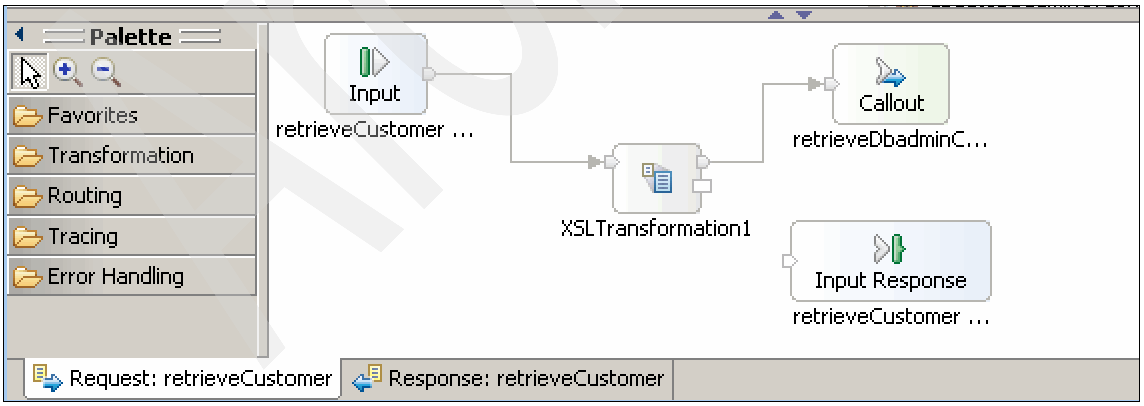


Figure 5-47 retrieveCustomer request flow

4. Double-click the XSLTransformation1 node to open the New XML Mapping wizard (Figure 5-48). Click **Finish** to open the XML mapping editor.

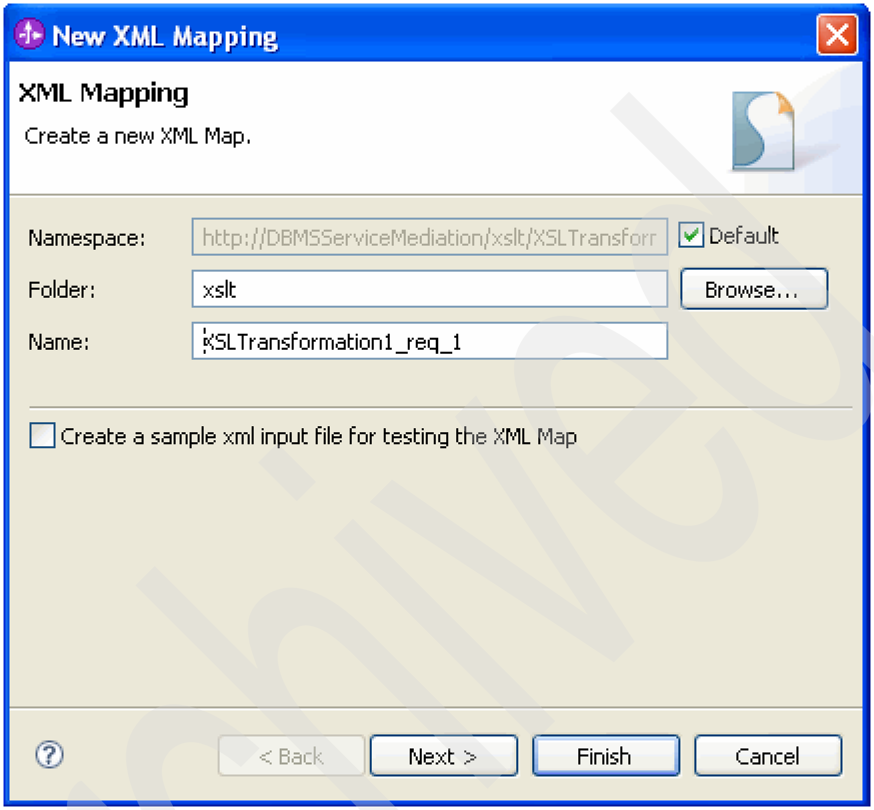


Figure 5-48 New XML mapping

5. In the XML mapping editor, the retrieveCustomer object is on the left and retrieveDbadminCustomerBG object on the right (Figure 5-49).

   Because a customer is uniquely identifiable by the customer ID, start by mapping the customer ID. Select **Wire retrieveCustomer → customer → custID to retrieveDbadminCustomerBG → retrieveDbadminCustomerBGInput → DbadminCustomer → custid**.

   The default mapping operation between the two fields is `Move`. This operation assigns a value from the field on the left to the field on the right.

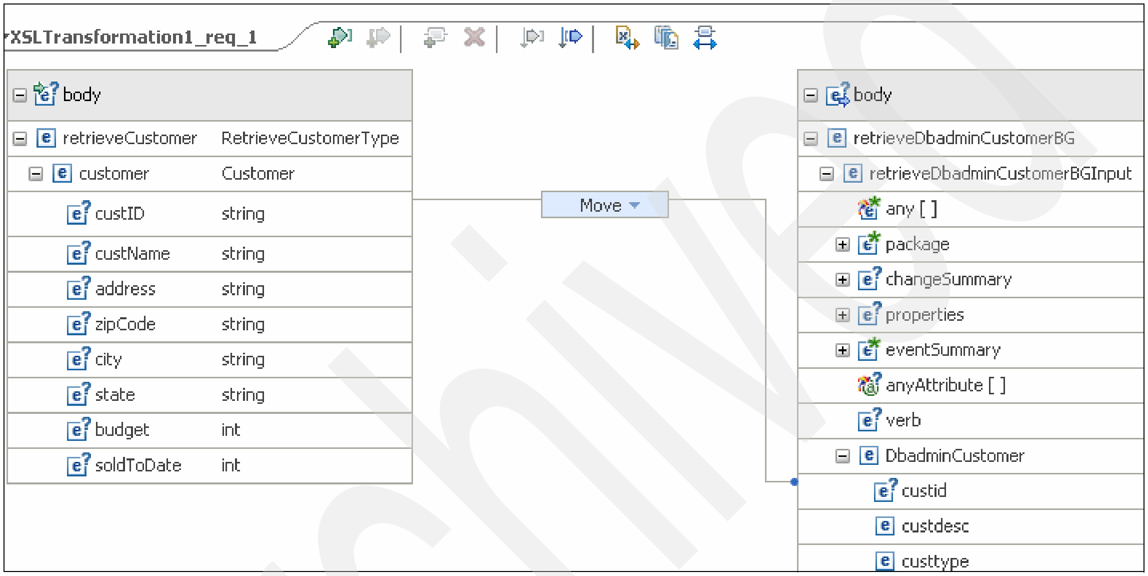


Figure 5-49 XML mapping editor

In the remaining sections, we use a table similar to Table 5-1 to show the mappings that you need to make.

Table 5-1 retrieveCustomer request flow XSLT

retrieveCustomer (Input) field	Action	retrieveDbadminCustomerBG (Output) field
custID	move	custid

6. Close the map.

Response flow

To build the response flow for the retrieveCustomer operation:

1. Select the wire between retrieveCustomer and retrieveDbadminCustomerBG in the Operations connection area.
2. Click the Response tab in the flow section.
3. Add an XSL Transformation primitive and wire it between the Callout Response node and the Input Response node as shown in Figure 5-50.

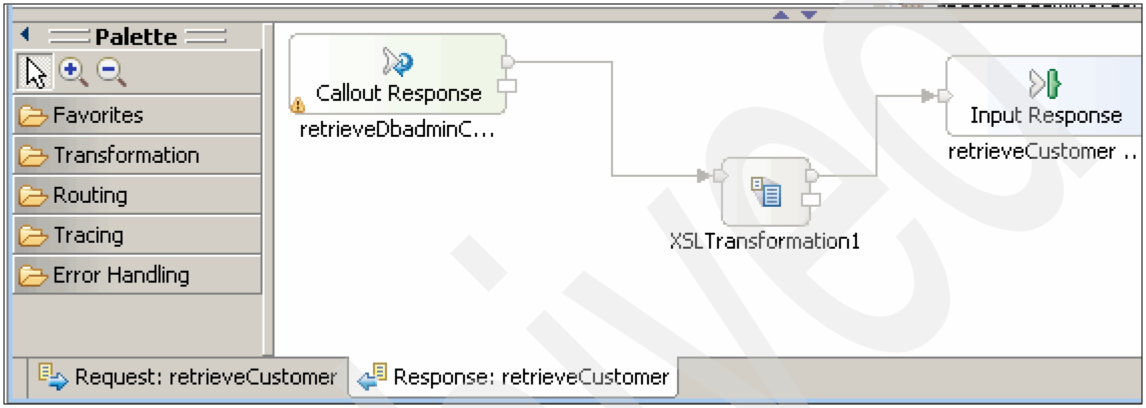


Figure 5-50 retrieveCustomer response flow

4. Open the XML mapping editor for the XSL Transformation primitive, and map the fields as shown in Table 5-2.

Table 5-2 retrieveCustomer response flow XSLT

Input field	Action	Output field
custid	move	custID
custdesc	move	custName
address	move	address
zipcode	move	zipCode
city	move	city
state	move	state
soldtodate	move	soldToDate
budget	move	budget
n/a	assign ‘3’	RC

Tip: To assign a value to a field, right-click the field in the XSL map, and select **Create Transform** to put an Assign operation to the field. Enter a value for the field in the General tab of the Properties view for the Assign transform.

Next, repeat this process to add an XSL Transformation primitive in the rest of the flow (request and response flows of two operations on each of the three interfaces). Some of the mediation operations are quite trivial, others are more complex. For example, to mediate the updateItem operation, you have to use an inline map action to map the array of Warehouses.

XSL Transformation primitive for updateCustomer

The next flow to update is for the updateCustomer operation.

Request flow

To build the request flow for the updateCustomer operation:

1. Select the wire between updateCustomer and updateDbadminCustomerBG in the Operations connection area.
2. Add an XSL Transformation primitive and wire it between the Input node and the Callout node.
3. Open the XML mapping editor, and map the fields as shown in Table 5-3.

Table 5-3 updateCustomer request flow XSLT

Input field	Action	Output field
custID	move	custid
custName	move	custdesc
address	move	address
zipCode	move	zipcode
city	move	city
state	move	state
budget	move	budget
soldToDate	move	soldtodate

Response flow

To build the response flow for the updateCustomer operation:

1. Select the wire between updateCustomer and updateDbadminCustomerBG in the Operations connection area.
2. Add an XSL Transformation primitive and wire it between the Callout Response node and the Input Response node.
3. Open the XML mapping editor for the XSL Transformation primitive and map the fields as shown in Table 5-4.

Table 5-4 updateCustomer response flow XSLT

Input field	Action	Output field
n/a	assign '3'	RC

XSL transformation for the updateItem operation

The next flow to update is for the updateItem operation.

Request flow

To build the request flow for the updateItem operation:

1. Select the wire between updateItem and updateDbadminItemBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Input node and the Callout node.
3. Open the XML mapping editor and map the fields as shown in Table 5-5.

Table 5-5 updateItem Request flow XSLT

Input field	Action	Output field
itemID	move	itemid
itemName	move	itemname
price	move	price
warehouses	Inline map—see Table 5-6	warehouses

Map the warehouses inline map as shown in Table 5-6. Note that some input fields are mapped to multiple output fields.

Table 5-6 updateItem request flow Inline Map for warehousesXSLT

Input field	Action	Output field
	custom: /body/updateItem/item/itemID	itemid
whsID	move	whsid
stock	custom: $stock + (-$itemQtyPartial)	stock
indelivery	custom: $indelivery + $itemQtyPartial	indelivery
itemQtyPartial	custom: $stock + (-$itemQtyPartial)	stock
itemQtyPartial	custom: $indelivery + $itemQtyPartial	indelivery
	assign: 1	ordered

Response flow

To build the response flow for the updateItem operation:

1. Select the wire between updateItem and updateDbadminItemBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Callout Response node and the Input Response node.
3. Open the XML mapping editor and map the field as shown in Table 5-7.

Table 5-7 updateItem response flow XSLT

Input field	Action	Output field
n/a	assign '3'	RC

XSL Transformation for the retrieveItem operation

The next flow to update is for the retrieveItem operation.

Request flow

To build the request flow for the retrieveItem operation:

1. Select the wire between retrieveItem and retrieveDbadminItemBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Input node and the Callout node.

3. Open the XML mapping editor and map the field as shown in Table 5-8.

Table 5-8 retrieveItem request flow XSLT

Input field	Action	Output field
itemID	move	itemid

Response flow

To build the response flow for the retrieveItem operation:

1. Select the wire between retrieveItem and retrieveDbadminItemBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Callout Response node and the Input Response node.
3. Open the XML mapping editor and map the fields as shown in Table 5-9.

Table 5-9 retrieveItem response flow XSLT

Input field	Action	Output field
itemid	move	itemID
itemname	move	itemName
price	move	price
n/a	assign '3'	RC
warehouses	Inline map—see Table 5-10	warehouses

Map the warehouses inline map as shown in Table 5-10.

Table 5-10 retrieveItem response flow Inline Map for warehousesXSLT

Input field	Action	Output field
whsid	move	whsID
indelivery	move	indelivery
stock	move	stock

XSL transformation for the createOrder operation

The next flow to update is for the createOrder operation.

Request flow

To build the request flow for the createOrder operation:

1. Select the wire between createOrder and createDbadminOrderheaderBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Input node and the Callout node.
3. Open the XML mapping editor and map the fields as shown in Table 5-11.

Table 5-11 createOrder request flow XSLT

Input field	Action	Output field
ordID	move	ordid
amount	move	amount
submitterID	move	submitterid
state	move	state
creationDate	custom: concat(substring-before($creationDate, 'T'), ' ', substring-before(substring-after($creationDate, 'T'), '.'))	creationdate
completionDate	custom: concat(substring-before($completionDate, 'T'), ' ', substring-before(substring-after($completionDate, 'T'), '.'))	completiondate
itemQty	move	itemqty
itemID	move	itemid
custID	move	custid

Response flow

To build the response flow for the createOrder operation:

1. Select the wire between createOrder and createDbadminOrderheaderBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Callout Response node and the Input Response node.

3. Open the XML mapping editor and map the fields as shown in Table 5-12.

Table 5-12 createOrder response flow XSLT

Input field	Action	Output field
ordid	move	ordID
custid	n/a	
amount	move	amount
submitterid	move	submitterID
state	move	state
creationdate	custom: concat(substring-before($creationdate, ' '), 'T', substring-after($creationdate, ' '), '.546-05:00')	creationDate
completiondate	custom: concat(substring-before($completiondate, ' '), 'T', substring-after($completiondate, ' '), '.546-05:00')	completionDate
itemqty	move	itemQty
	assign ‘3’	RC

Working with the dateTime data type

The creationDate and completionDate field use the `dataTime` data type. The format that the database system supports is most likely going to be different from the format that WebSphere ESB uses. To resolve the difference, use an XPath expression to transform `dateTime` on request and response flows.

On the request flow, you have to convert `dateTime` to a string that Derby understands. The XPath expression to achieve it is as follows:

```
concat(substring-before($completionDate, 'T'), '  ',
substring-before(substring-after($completionDate, 'T'), '.') )
```

This XPath expression converts from the format shown in Figure 5-51 that used by WebSphere ESB to the format shown in Figure 5-52, used by Derby.

Figure 5-51 WebSphere ESB date / time format

Figure 5-52 Derby date / time format

On the response flow, convert a string that represents dateTime to a `dateTime` datetype as follows:

```
concat(substring-before($completiondate, '  '), 'T',
substring-after($completiondate, '  '),  '.546-05:00')
```

XSL Transformation for the updateOrder operation

The next flow to update is for the updateOrder operation.

Request flow

To build the request flow for the updateOrder operation:

1. Select the wire between updateOrder and updateDbadminOrderheaderBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Input node and the Callout node.
3. Open the XML mapping editor and map the fields as shown in Table 5-13.

Table 5-13 updateOrder request flow XSLT

Input field	Action	Output field
ordID	move	ordid
amount	move	amount
submitterID	move	submitterid
state	move	state
creationDate	custom: concat(substring-before($creationDate, 'T'), ' ', substring-before(substring-after($creationDate, 'T'), '.'))	creationdate
completionDate	custom: concat(substring-before($completionDate, 'T'), ' ', substring-before(substring-after($completionDate, 'T'), '.'))	completiondate
itemQty	move	itemqty
itemID	move	itemid
custID	move	custid

Response flow

To build the response flow for the updateOrder operation:

1. Select the wire between updateOrder and updateDbadminOrderheaderBG in the Operations connection area.
2. Add an XSL transformation primitive and wire it between the Callout Response node and the Input Response node.
3. Open the XML mapping editor and map the field as shown in Table 5-14.

Table 5-14 updateOrder Response flow XSLT

Input field	Action	Output field
	assign ‘3’	RC

Complete the mediation flow

The mediation component can now deal with a database record. Note that you have not added error handling. You will do this before the mediation flow goes into production.

Save each map, mediation flow, and the assembly diagram. Ensure that there are no errors. (Warnings are generally OK.)

Tip: Press Ctrl+Shift+S to save all.

The next step is to run a quick component test to make that things are working properly.

5.6 Testing the mediation flow component

In a manner similar to how we tested JDBCOutboundInterface, select and right-click the **DBMSServiceMediation** component and choose **Test Component** from the pop-up menu. You test the `retrieveItem` operation of ItemServiceIF

Tip: If you see any errors in your mediation logic that you want to correct, a good practice is to remove the project from the server and then add it again to ensure that you have the latest code.

Tip: If you are using a Derby database and have run the external service discovery wizard to connect to it, you might have to restart WebSphere Integration Developer to break the connection. However, you do not have to stop the server.

To test the mediation flow component:

1. Set the following options as shown in Figure 5-53:
   - Component name: `DBMSServiceMediation`
   - Interface: `ItemServiceIF`
   - Operation: `retrieveItem`
   - itemID: `IT_001`

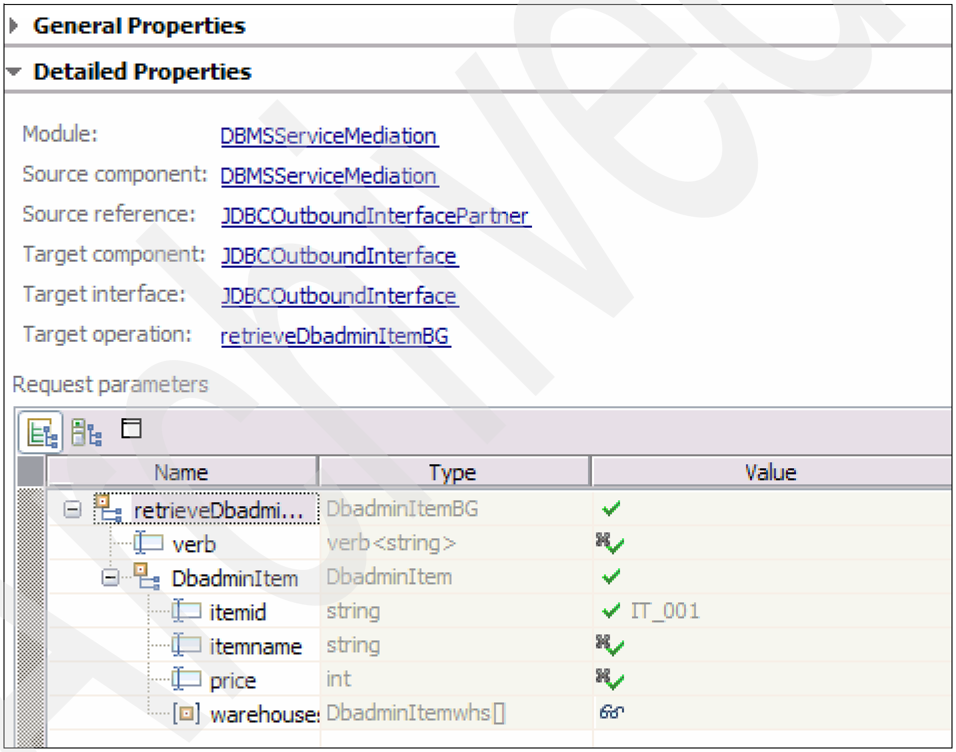


Figure 5-53 Test parameters

2. Run the test.

   If no errors occur, the events diagram looks similar to Figure 5-54.

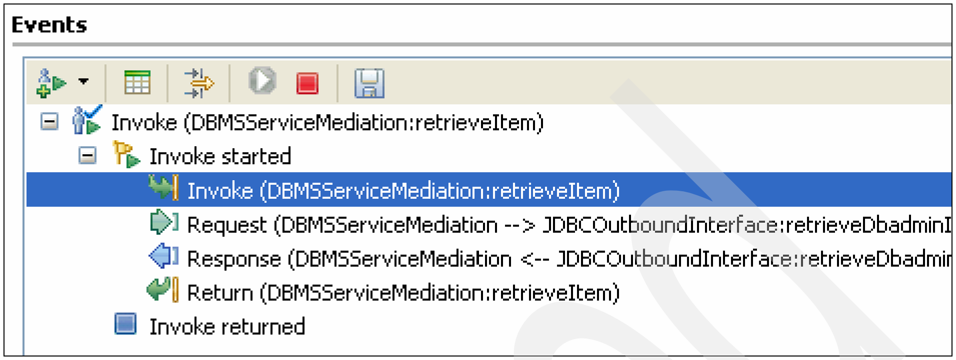


Figure 5-54 Test events result

The return parameters show the item that is returned with the warehouse information, as shown in Figure 5-55.

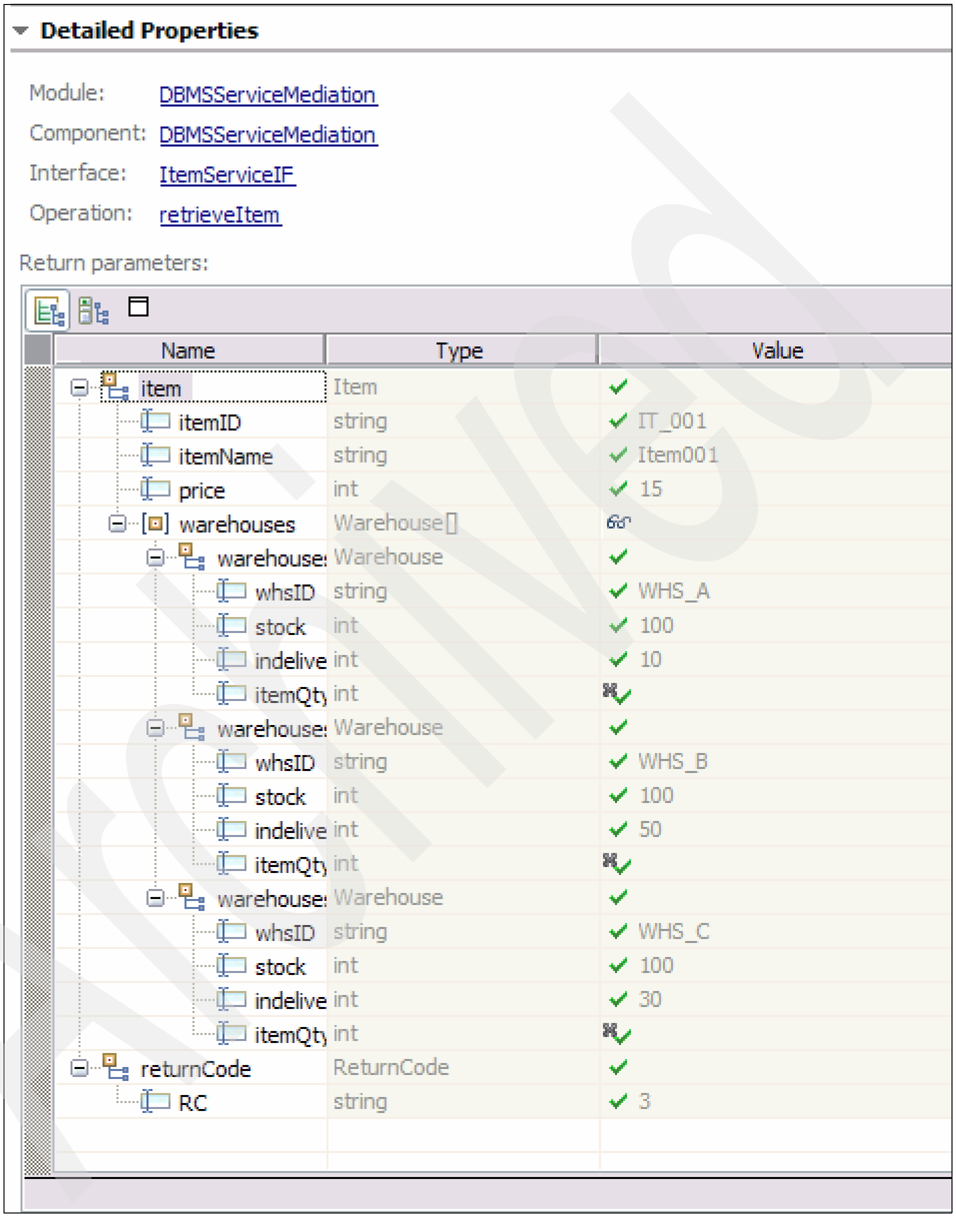


Figure 5-55 Test parameter results

> **Note:** If you enter an item number that dose not exist in the database, you will receive an error.

5.7 Creating the mediation export component

Finally, you generate an export component to make the mediation services offered by DBMSServiceMediation available for use. Follow these steps:

1. Right-click the mediation flow component, and select **Generate Export**. Then, select **Web Service Binding**. See Figure 5-56.

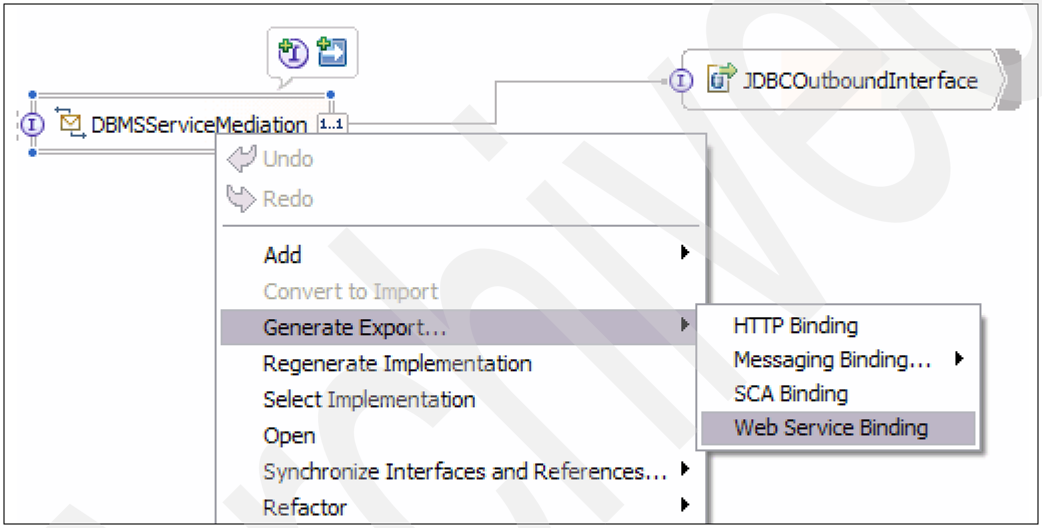


Figure 5-56 Generate an export component with a Web service binding

2. Select all available interfaces as shown in Figure 5-57 and click **OK**.

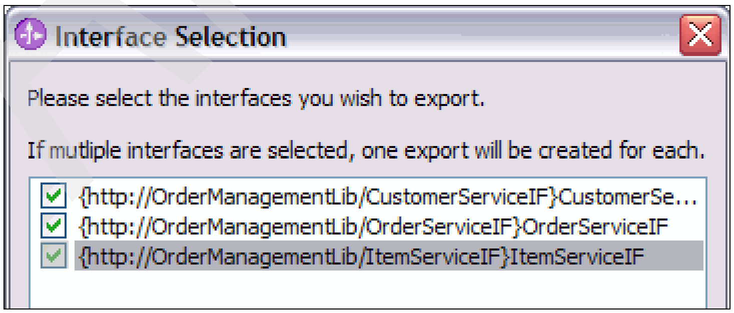


Figure 5-57 Web service binding interface selection

The final assembly diagram looks similar to that shown in Figure 5-58.

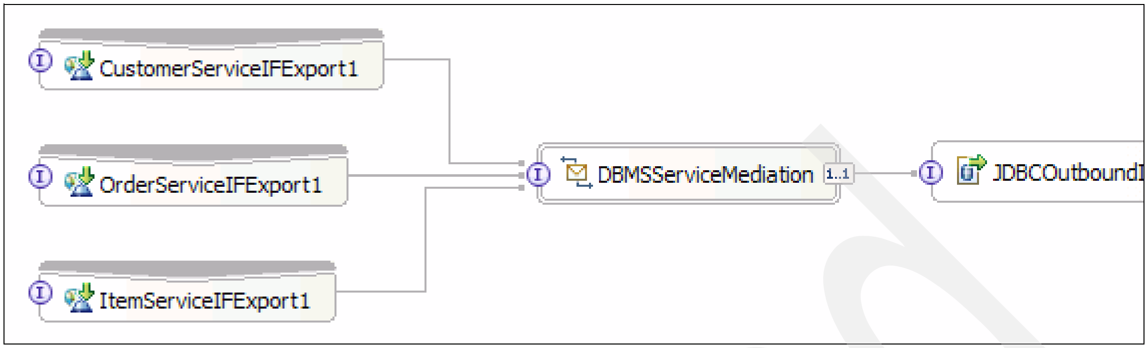


Figure 5-58 Final assembly diagram

Team development: This is a good time to check in your project to CVS.

6

WarehouseAvailabilityMediation module

This chapter explains the implementation of the Warehouse Availability Mediation.

This example mediation illustrates the use of the following primitives and techniques:

- Custom mediation primitive
- Usage of correlation, transient and shared contexts
- Advanced usage of the XSL Transformation primitive
- Fan Out and Fan In primitives
- Inline Service Invoke primitive

The topics in this chapter include:

- Mediation design and setup
- Preparing the Warehouse Web services
- Creating the mediation module and artifacts
- Building the mediation flow
- Testing the mediation

6.1 Mediation design and setup

According to the high-level design, described in 1.2, “Scenario high-level design” on page 9, the Warehouse Availability Mediation determines which warehouses are available to fulfill an order and how to split the order among those warehouses.

Figure 6-1 shows the interface that the mediation implements.

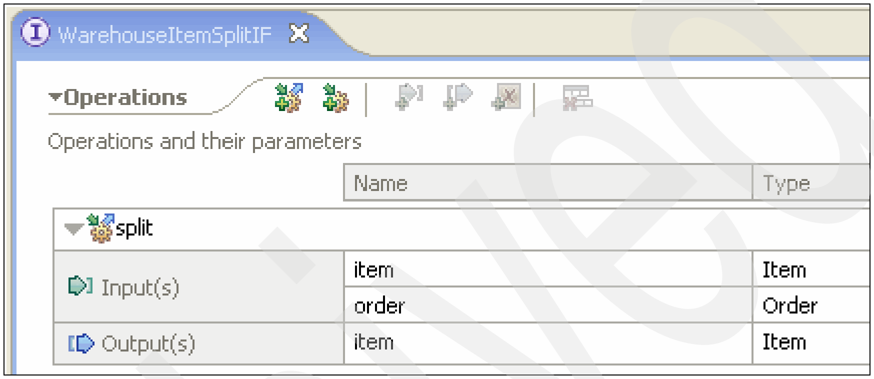


Figure 6-1 Warehouse split interface

The input to the mediation is the item and the order business objects. The quantity ordered is contained within the order business object. The item business object contains an array of possible warehouses that can supply the item.

The output is the item business object with an array of warehouses to supply the item and the quantity they will supply.

See 1.4.2, “Interfaces and business objects definition” on page 17 for details about these business objects.

There are three possible warehouses:

- Warehouse A
- Warehouse B
- Warehouse C

The mediation checks with each warehouse to see if it is available to fulfill the order. The request to the warehouse for its availability is done using a Web service provided by each warehouse. When the responses are returned from the warehouses, the order for the item quantity is split among the available warehouses.

As an example, the mediation receives the following objects:

- An Item business object where `itemID=1` and three warehouses are in the array.
- An Order business object that carries an item quantity of 11.

The mediation checks all the three warehouses for their availability to supply the item. We assume two of three are available (Warehouse A is unavailable). The mediation then splits the item quantity of 11 between the two warehouses, 6 items for the Warehouse B and 5 items for the Warehouse C.

Given this expected behavior, the mediation is implemented as follows:

1. A Fan Out primitive is used to fire the availability request to three Service Invoke primitives, which receive an availability status from the warehouses (`true` or `false`).
2. A Fan In primitive waits for all of the three services to provide a response and fires out the aggregated result.
3. A Custom Mediation primitive splits the quantity over the available warehouses using the following simple algorithm:
   a. Order quantity is divided by the number of the available warehouses.
   b. The net result of the division is assigned to every warehouse.
   c. The eventual residual quantity is added to the first available warehouse

Table 6-1 shows two examples of this simple algorithm. In Example 1, all the warehouses are found available. In Example 2, Warehouse A is found unavailable, and the quantity is split over the remaining two.

Table 6-1 Split algorithm behavior given some possible inputs

Example	Quantity	Warehouse A	Warehouse B	Warehouse C
Example 1	10	4	3	3
Example2	19	Unavailable	10	9

Figure 6-2 shows the design of this logic.

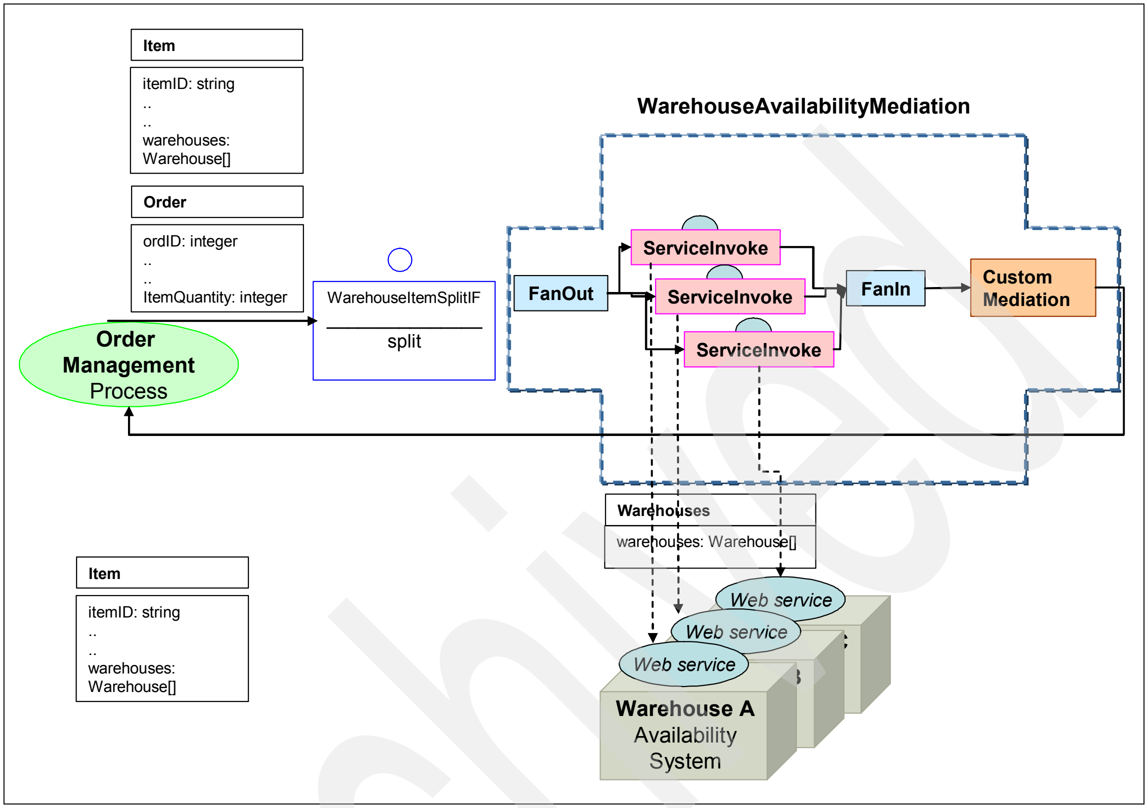


Figure 6-2 WarehouseAvailabilityMediation design

The Warehouse Availability Services expects a new *Warehouses* business object as input. This object wraps the Warehouse array coming with the item as mediation input. The services return a boolean value giving the warehouse availability.

The OrderManagementProcess business application is the mediation client in the scenario, but any client can be used as long as it binds to the SCA WarehouseItemSplitIF interface on the export component used. For example, to test the mediation, we used the WebSphere Integration Developer Test Client. The mediation client receives an item business object from the mediation in response.

The following message transformations play a role in the mediation:

1. Service Invoke primitives invoke the Warehouse Availability Services using the interface that is required by these services. These interfaces require a

Warehouses business object as input. This object wraps the warehouses array. A transformation primitive (business object map or XSLT) converts the order and item business objects to the Warehouses business object. We refer this transformation as *InputToServiceInvoke*.

2. After each Service Invoke, another transformation primitive (either business object map or XSLT) is needed to aggregate results from the Web services invocation. We refer them respectively as *AggregateWarehouseA*, *AggregateWarehouseB*, and *AggregateWarehouseC*.
3. The Custom Mediation primitive expects both the Service Invoke results and order item quantity to perform its algorithm. A transformation primitive is used between the Fan In primitive and the Custom Mediation primitive to prepare the object. We refer this transformation as *FanInToSplitAlgorithm*.

Figure 6-3 shows the transformation primitives that are inserted in the mediation flow.

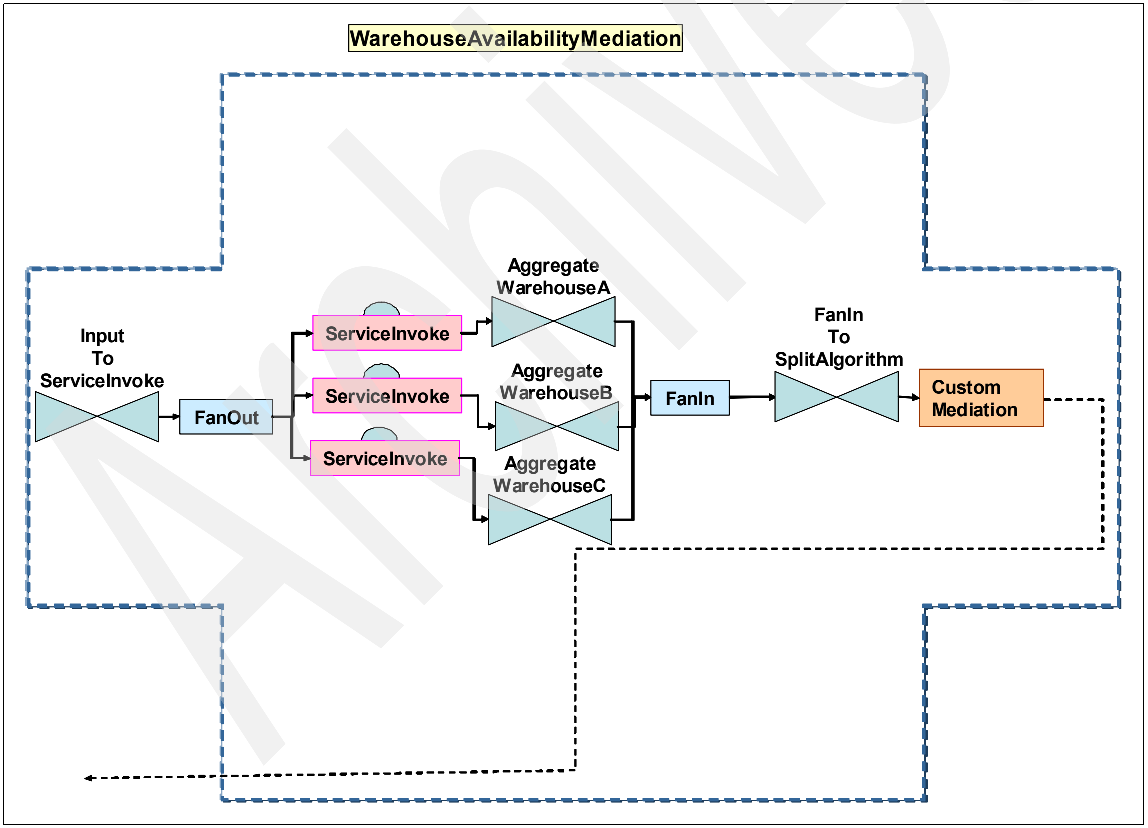


Figure 6-3 Transformation primitives that are required by the WarehouseAvailabilityMediation

The transformation converts input objects to the format that is needed through the mediation.

The problem is that you can lose detail in a transformation that you need later in the mediation. For example, the InputToServiceInvoke transformation eliminates the order object and all fields in the item with the exception of the warehouse array. However, after the responses indicating warehouse availability are returned, the mediation needs the item order quantity from the order object to perform the split algorithm in the Custom Mediation primitive. It also needs information from the item object (item ID and name) because this information must be sent back to the client.

So, the mediation must store this information so it can be retrieved later. This information is stored in the mediation contexts:

- Information that must be preserved unchanged from the start of the mediation to the end must be put in the *correlation* context, which is the case for the item details with the exception of the warehouse list.
- Information that must be stored and used later in the mediation must be placed in the *transient* context, which is the case for the original warehouse list and for the order item quantity.
- Information that must be aggregated in a Fan In/Fan Out usage must be placed in the *shared* context, which is the case of the warehouse availability indicators that are returned by the Web services.

The transformation primitives put this information in the contexts. Because we implement advanced use of the contexts in our example, this process is best done using an XSL Transformation primitive. We move the Service Message Object body information in the context and vice-versa.

6.2 Preparing the Warehouse Web services

The mediation invokes the Warehouse Availability Systems that are provided by three Web services. For testing, the Warehouse Availability Systems are implemented as mediation flows also and are deployed to the WebSphere ESB server.

Note: This section describes the Warehouse services that are invoked by the WarehouseAvailabilityMediation. The Warehouse services are simply a resource for the mediation. This section is of interest only to those who want to set up a simple test for mediations being developed.

If you are following along by developing the mediations in this book, you can simply load these services from the additional material that is supplied online for this book. You can find information about downloading and using this material in Appendix B, "Additional material" on page 461. The Web services and a library that contains the common artifacts are supplied as a project interchange file called WarehouseAvailability.zip.

This chapter also assumes that you have the OrderManagementLib shipped as part of the additional material.

Figure 6-4 shows how the Business Integration view looks after these projects are imported into the workspace.

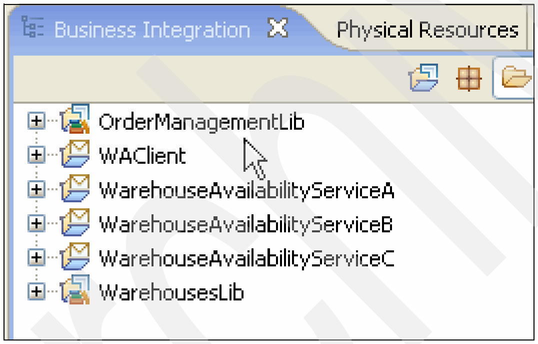


Figure 6-4 Business Integration view after importing Warehouse Availability System Web services

In addition to the artifacts contained in OrderManagementLib, these projects contain:

- Three mediation modules named WarehouseAvailabilityServiceA, WarehouseAvailabilityServiceB, and WarehouseAvailabilityServiceC.

  These mediations contain the implementation of the three Web services. They do not contain any mediation flow but just a Java component. The exports that make the mediations available use a Web service binding.

  Figure 6-5 shows the Web services mediation module assembly diagram.

Figure 6-5 Web services mediation module assembly diagram

Figure 6-6 shows the Java component for WarehouseAvailabilityServiceA. The component uses simple logic to emulate the warehouse services. All three mediations have a similar implementation.

```
public Boolean isAvailable(DataObject warehouses) {

    DataObject WarA = (DataObject) warehouses.getList("wList").get(0);
    //the warehouse A will refuse availability while its quatity
    //partial is 8
    if(WarA.getInt("itemQtyPartial") == 8)
        return false;

    return true;
}
```

Figure 6-6 Warehouse A Availability Web service logic

Warehouse A refuses availability if it receives a quantity of 8. Of course, this is just an example of logic (it is not necessarily realistic). This example provides the mediation tester a way to generate a “not available” response. In a production environment, the logic behind the Web services is not of interest to the mediation. The only thing that matters is the interface and binding properties.

Figure 6-7 shows the implementation for Warehouse B and Warehouse C. They always return an available status.

```
public Boolean isAvailable(DataObject warehouses) {
    //always available
    return true;
}
```

Figure 6-7 Warehouse A and Warehouse B Web service always return available status

- A WarehousesLib library, which is used by the three Web services, contains the interface, the business object, and the three port definitions for the Web services that are used to import the Web services to the mediation. See Figure 6-8.

Figure 6-8 The Warehouse Services common library

> **Attention:** This library makes indirect use of the OrderManagementLib artifacts through the definition of the Warehouses business object. You will have errors on it if you do not have the OrderManagementLib in the workspace.

- A WAClient mediation module is also provided. You might find this module useful for testing the Web services before developing the mediation that makes use of them.

  This mediation has just the three SCA Web services imports, as shown in Figure 6-9. You can use the Integrated Test client to verify that you can get these imports and use them in the WarehouseAvailabilityMediation module.

Figure 6-9 The Web Services client assembly diagram

You can deploy the Web services in the test environment as follows:

1. Start the WebSphere ESB Server test environment.
2. Deploy WarehouseAvailabilityServiceA, WarehouseAvailabilityServiceB, and WarehouseAvailabilityServiceC to the server, shown in Figure 6-10.

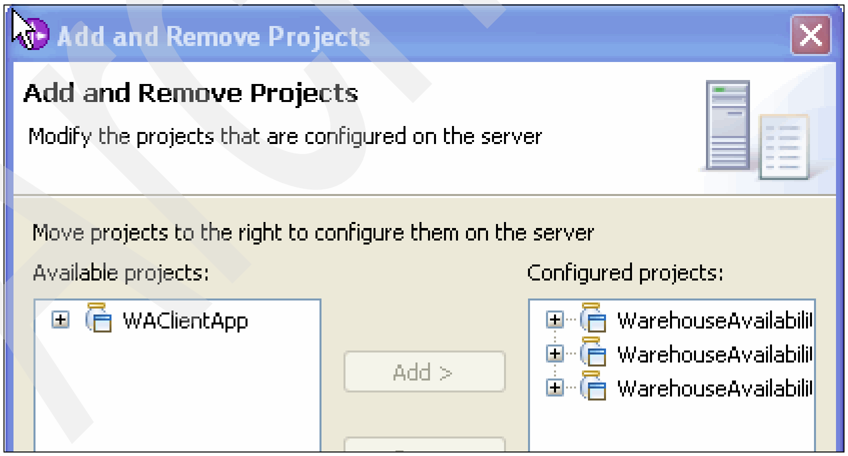


Figure 6-10 Warehouse Availability Systems Web services deployment

3. Locate and open one of the three Web service port declarations in WarehousesLib, as shown in Figure 6-11.

Figure 6-11 Locating Web Services port declarations

4. Locate the `soap:address` element in the port and copy the *entire* content of the location attribute (the URL), as shown in Figure 6-12.

> **Attention:** Figure 6-12 just shows part of this element!

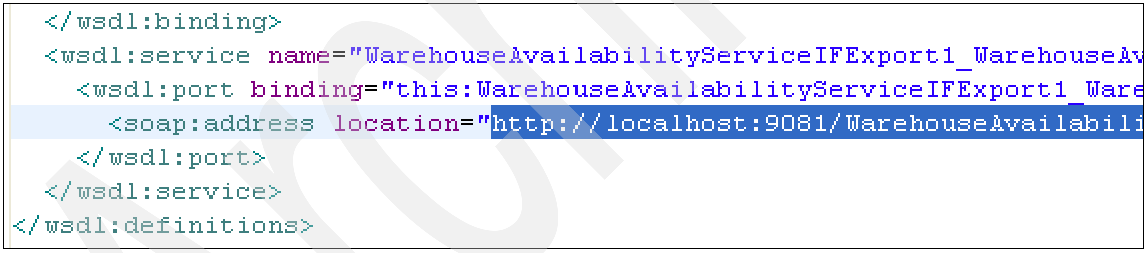


Figure 6-12 Locating the soap:address element in a Web service port definition

5. Open a browser and paste the copied URL in the address field. Then, submit it. The response shown in Figure 6-13 displays.

Figure 6-13 Successful response from a Warehouse Availability System Web service

If you see an error instead, the WebSphere ESB Web container is probably using a port other than the 9081 used in this example. In this case, you need to determine the actual Web service port.

A quick method to do this might be just changing the port in the address you typed on the browser (try 9080. 9081, 9082, and so forth).

The better method is look at the Virtual Hosting setting in the WebSphere ESB administrative console as follows:

1. Open the WebSphere ESB administrative console (right-click the test environment server instance in WebSphere Integration Developer, and click **Run administrative console**).
2. Select **Applications** → **Enterprise Applications** and locate the WarehouseAvailability applications, as shown in Figure 6-14 on page 365.

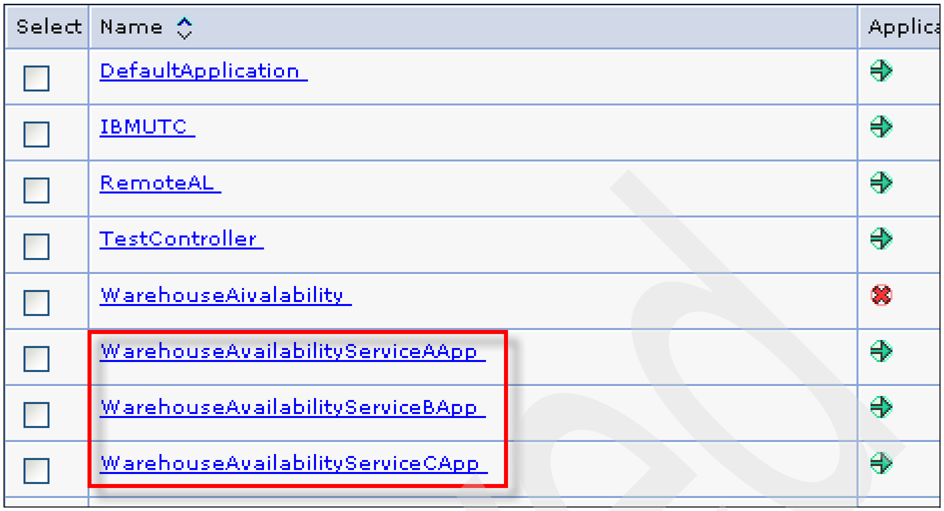


Figure 6-14 WarehouseAvailabilityService applications in the WebSphere ESB console

3. Open one of the applications and select **Web Module Properties** → **Virtual host**. Here you get the name of the virtual host over which the Web services is running (`default_host` in our case as shown in Figure 6-15).

Figure 6-15 Warehouse Availability Web module virtual host

4. Now, navigate to **Environment** → **Virtual Hosts** in the console left menu. Locate the virtual host that is used by the Warehouse Availability Service application (`default_host` in our example) and select it.

5. Select **Additional Properties** → **Host aliases** to see the ports to which the virtual host is listening. Look for the port `9xxx`. The port in our case is `9081`, as shown in Figure 6-16.

Select	Host Name	Port
☐	*	9081
☐	*	80
☐	*	9444
☐	*	5063
☐	*	5062
☐	*	443

Figure 6-16 Virtual host specific ports

6. Ensure that you have the correct port by attempting to reach the service again using a Web browser. After you know the port number, you *must update it* in *all three* Web service port definitions in the WarehousesLib.

```
  <wsdl:service name="WarehouseAvailabilityServiceIFExport1_Wa
    <wsdl:port binding="this:WarehouseAvailabilityServiceIFExp
      <soap:address location="http://localhost:9081/WarehouseA
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 6-17 Updating soap address location port

When you know that you have the correct port number, perform one final test using the WAClient application:

1. If you changed the Web services ports, you must complete the following steps:
   a. Open the WAClient assembly diagram.
   b. Delete all three imports that you find there for the Web services.
   c. Create all three imports again and implement the WarehouseAvailabilityServiceIF interface.
   d. Generate an export component with a Web service binding for each service, pointing to their respective updated Web service port.

2. Deploy the WAClient application to the WebSphere ESB test environment.
3. Use the Integrated Test Client (that is, right-click the WAClient and select **Test** → **Module**) to test the three Web services. Figure 6-18 shows the test client properties that are used for the test.

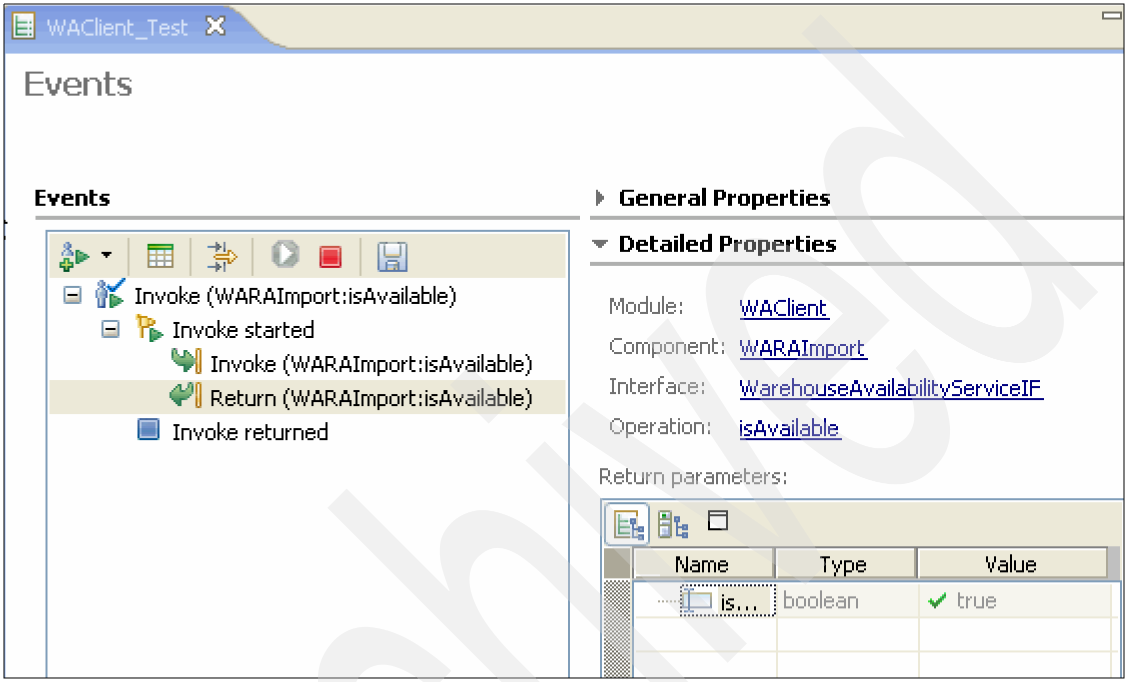


Figure 6-18 Testing Web services using the provided client application

In the WarehouseAvailabilityMediation module, you use these same import types. Thus, this test is a reliable test for the services.

6.3 Creating the mediation module and artifacts

Now, you can create the mediation module. Follow these steps:

1. Create the module, and name it `WarehouseAvailabilityMediation`.
2. Select **WebSphere ESB Server v6.1** as the target runtime on the first mediation module creation wizard panel.
3. Select both the **OrderManagementLib** and **WarehousesLib** as required libraries.

6.3.1 Creating the business objects

If you look at the business objects that are needed according to the high-level design and low-level design, the business objects that are needed are already present in the existing libraries:

- The OrderManagementLib contains the interface that the mediation is expected to implement (WarehouseItemSplitIF) and contains the business objects that it uses (Order and Item plus the Warehouse item, which is needed by the Item business object itself).
- The WarehousesLib contains the WarehouseAvailability Systems Web services interface (WarehouseAvailabilityServiceIF) and contains the Warehouses business object that is expected as input from the Web services.

The correlation, transient, and shared contexts are used for temporary storage along the mediation. These contexts must have a structure that is feasible for hosting the temporary values. These structures must be established creating appropriate business objects or using existing business objects while they match the required structure.

Now, we examine this required structures versus existing business objects.

Correlation context: Item business object

The correlation context has to keep all of the Item information, with the exception of the warehouses array. This information must be returned, unchanged, at the end of the mediation. The information to be stored in the correlation context is clearly a subset of the item business object, so you use the item business object for the correlation context.

Transient context: New business object

The transient context has to keep the item order quantity from the order business object and the warehouses list from the item object. No existing business object matches these requirements, so you must create a specific business object.

Use the New Business Object wizard to create a business object in WarehouseAvailabilityMediation:

1. Name the object `WarehouseMediationFlowTransientContext`.
2. Use the Derived Business Object panel in the New Business Object wizard to get properties from the existing business objects for this new object, as shown in Figure 6-19.

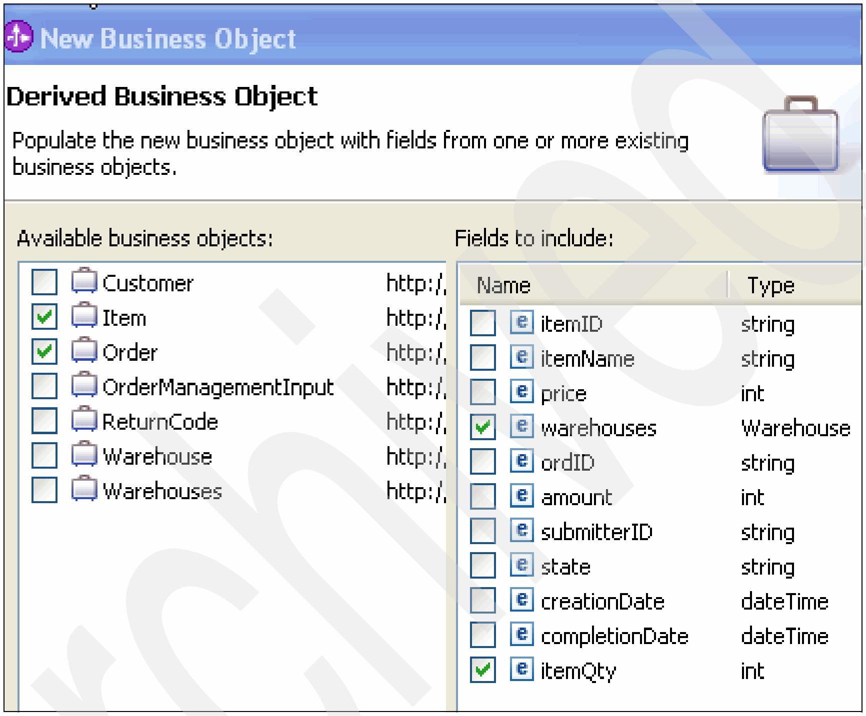


Figure 6-19 Populating the WarehouseMediationFlowTransientContext with existing business object fields

The results look similar to that shown in Figure 6-20.

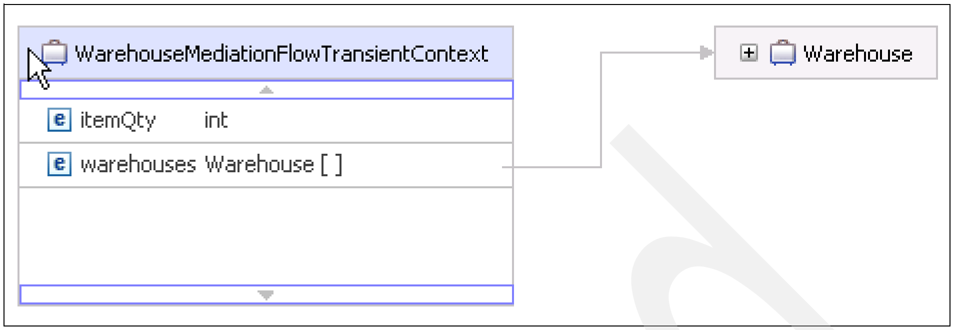


Figure 6-20 WarehouseMediationFlowTransientContext business object

Shared context: New business object

The shared context has to aggregate results coming from the three Warehouse Availability Systems Web services. Each service returns a boolean value. You do not have an existing object that fits this need, so you must create a new object.

Create a business object similar to the one shown in Figure 6-21, and name it `WarehouseMediationFlowSharedContext`.

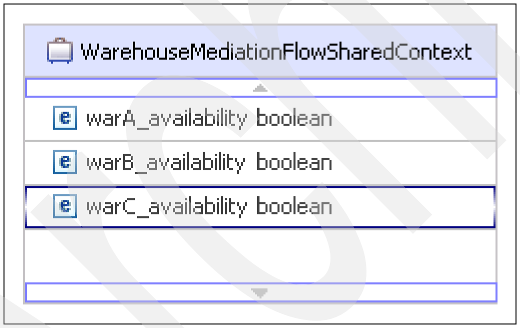


Figure 6-21 WarehouseMediationFlowSharedContext business object structure

That is it. No more business objects are needed inside the mediation module to make it work.

6.3.2 WarehouseAvailabilityMediation interfaces

The WarehouseAvailabilityMediation uses the following interfaces:

- The mediation must implement the WarehouseItemSplitIF interface at the client boundary (with OrderManagementProcess), so it is defined in the OrderManagementLib library.
- The mediation must use the Warehouse Availability System Web services through their common WarehouseAvailabilityServiceIF interface. This interface is included in the WarehousesLib because it is used while creating the services themselves.

Because both of these libraries are included as a dependency in the module, you do not need to create new interfaces.

6.3.3 Importing the Warehouse Availability System Web services

> **Tip:** Define all of the components in the assembly diagram and wire them before implementing the actual mediation flow. This method ensures that the interfaces that are implemented and the services that are used are included when you generate the implementation for the mediation flow.

Now, you need to bind the flow to the Warehouse Availability System Web services and define the SCA export to make the mediation consumable by an external client. Follow these steps:

1. Open the WarehouseAvailabilityMediation assembly diagram.
2. Get three Import components from the palette and place them to the right of the WarehouseAvailabilityMediation mediation flow.

3. Rename the three services (using refactoring) to `GetWarehouseA_Availability`, `GetWarehouseB_Availability`, and `GetWarehouseC_Availability`, as shown in Figure 6-22.

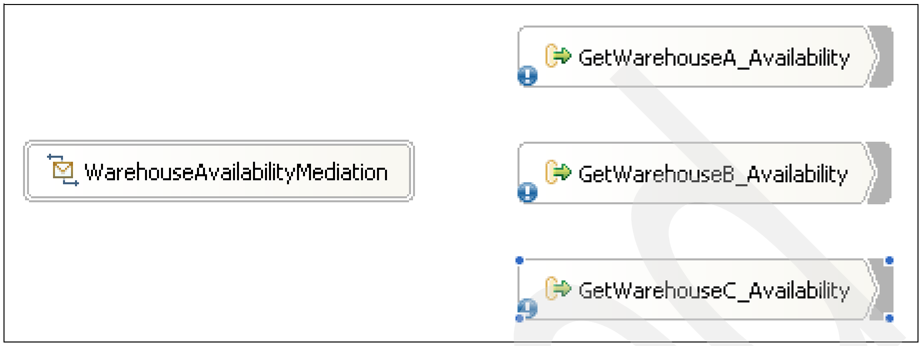


Figure 6-22 WarehouseAvailabilityMediation assembly diagram after dragging imports and refactoring the names

4. Add the WarehouseAvailabilityServiceIF interface to each of the Import components.
5. Generate a Web service binding for each interface:
   a. Select **Use an existing web service port**, as shown in Figure 6-23.

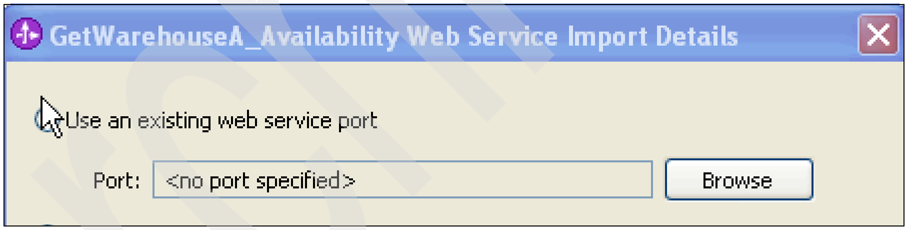


Figure 6-23 Binding the Import component to an existing Web service endpoint

b. Click **Browse** to select the appropriate port for each Web service.

As shown in Figure 6-24, use:

- `WarehouseAvailabilityServiceIFExportI1` for `GetWarehouseA_Availability`
- `WarehouseAvailabilityServiceIFExportI2` for `GetWarehouseB_Availability`
- `WarehouseAvailabilityServiceIFExportI3` for `GetWarehouseC_Availability`

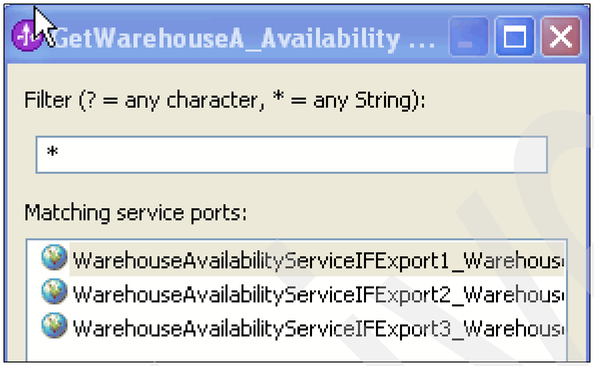


Figure 6-24 Binding the Import components with the Web services

6. Wire the mediation flow component to each of the Import components and allow the matching references to be created automatically.
7. Add the WarehouseItemSplitIF Interface to the WarehouseAvailabilityMediation component and generate the export component using a Web service binding with soap/http for the WarehouseAvailabilityMediation component.
8. Save the assembly diagram.

   Figure 6-25 shows the mediation after wiring.

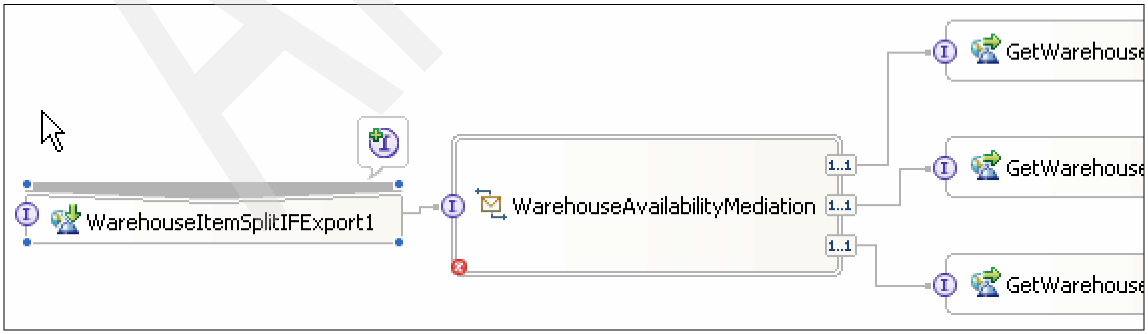


Figure 6-25 WarehouseAvailabilityMediation after wiring

9. You see errors for the mediation flow implementation, which has become incompatible with the interfaces that are implemented and references that are used. To clean up the errors:

   a. Right-click the WarehouseAvailabilityMediation mediation flow.

   b. Select **Regenerate implementation** and confirm overwriting of the previous implementation.

   The mediation flow editor opens, as shown in Figure 6-26. It shows the interfaces and references that you defined.

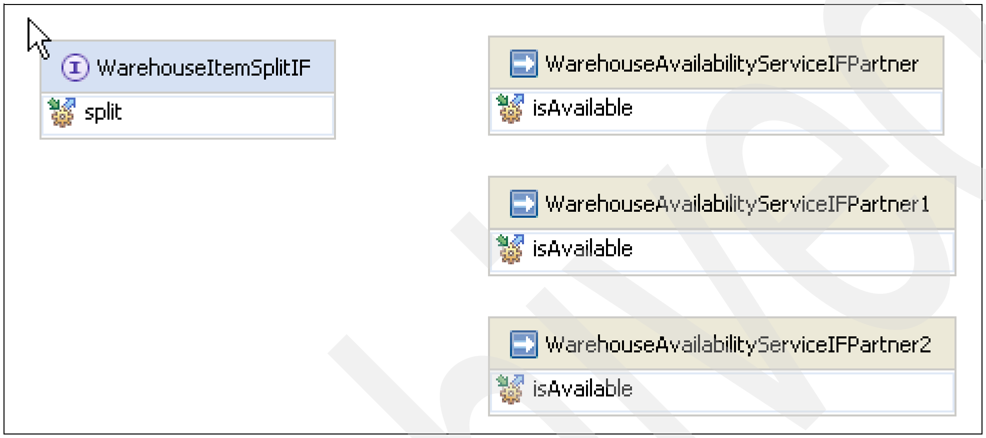


Figure 6-26 Mediation flow editor just after regenerating implementation

10. Save the assembly diagram and proceed to the mediation flow development.

> **Important:** The WarehouseItemSplitIF split operation and the three WarehouseAvailabilityServiceIF partners are never wired together in the mediation flow editor. The three partners are used *inline* to get information from each (warehouse availability). Each response contributes to the final response but is not the actual response. As a consequence, we only develop the request flow in this example. No response flow is required.

6.4 Building the mediation flow

Next, you build the mediation flow. The flow needs to perform the following tasks:

1. Handle with contexts to store values which are not involved in the mediation logic.
2. Invoke the Web services for the three Warehouses using the Fan In and Fan Out primitives.
3. Calculate the item quantity split among the warehouses based on the availability responses.

6.4.1 Warehouse A unavailability test case

In addition to the required logic, you add some initial logic to the flow that is needed to generate a case of unavailability for Warehouse A. To simulate this test case, you need to have an invocation of the Warehouse A Web service with a partial quantity of 8.

The quantity split is done after the service invocation, so you add logic at the beginning to generate an "8 quantity partial" test case while the item order quantity is 19. This primitive has nothing to do with the mediation flow and is simply for testing. It must be removed before the mediation can be considered production ready.

To generate an unavailability test case, follow these steps:

1. Insert a Custom Mediation primitive, and change its display name in `Unavailability Generator`.
2. Go to the Details pane of the Properties view, and select Java as the implementation, as shown in Figure 6-27 on page 376.

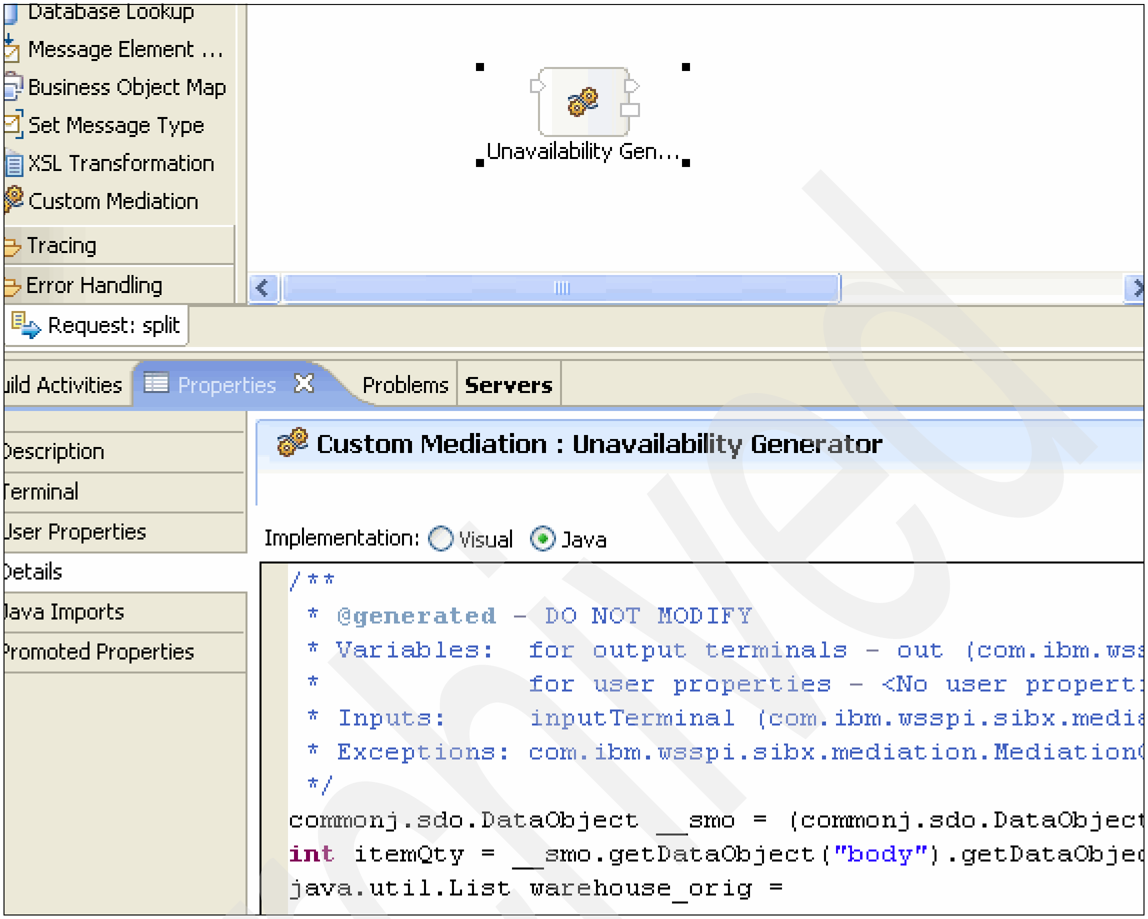


Figure 6-27 Unavailability generator Custom Mediation primitive code

3. Add the code shown in Example 6-1.

> **Additional material:** This code is also provided in the additional material that is available online for this book. The file name is UnavailabilityGenerator.txt. For more information about how to download the addition material, see Appendix B, "Additional material" on page 461.

Example 6-1 Code for the Unavailability Generator primitive

```
commonj.sdo.DataObject __smo = (commonj.sdo.DataObject)smo;
int itemQty =
__smo.getDataObject("body").getDataObject("split").getDataObject("order").getInt("itemQty");
```

```
java.util.List warehouse_orig =
__smo.getDataObject("body").getDataObject("split").getDataObject("item").getList("warehouses");
java.util.List warehouse_new = new java.util.ArrayList();
int warehouse_orig_size = warehouse_orig.size();

   for(int i=0; i< warehouse_orig_size; i++){
      commonj.sdo.DataObject warehouse = (commonj.sdo.DataObject) warehouse_orig.get(i);

      if((i==0) && (itemQty == 19)){ //if quantity is 19 I want the first partial to be 8
         warehouse.setInt("itemQtyPartial", 8); //this will trigger warehouse A to be
unavailable
         //NOTE THIS HAS NOTHING WITH THE ACTUAL ITEM QTY SPLIT WHICH WILL BE MADE IN CUST MED
2
      }

      warehouse_new.add(warehouse);

   }

__smo.getDataObject("body").getDataObject("split").getDataObject("item").setList("warehouses",
warehouse_new);
out.fire(__smo);
```

4. Save the mediation flow.

   This code simply gets the order quantity and forces the first warehouse quantity partial to be 8 while the total order quantity is 19. When we want to force an unavailability response from Warehouse A, we simply use 19 as the order quantity.

5. Finally, wire the Input node to the Unavailability Generator in terminal and save the mediation flow, as shown in Figure 6-28.

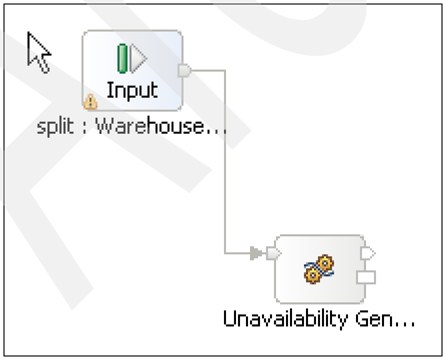


Figure 6-28 Unavailability Generator in the mediation flow

6.4.2 Storing input message data in the contexts at the Input node

In the mediation flow editor, you must set context types as follows:

1. Select the Input node and access the Details tab of the Properties view, as shown in Figure 6-29.

   Initially, the correlation, transient, and shared context fields are unspecified.

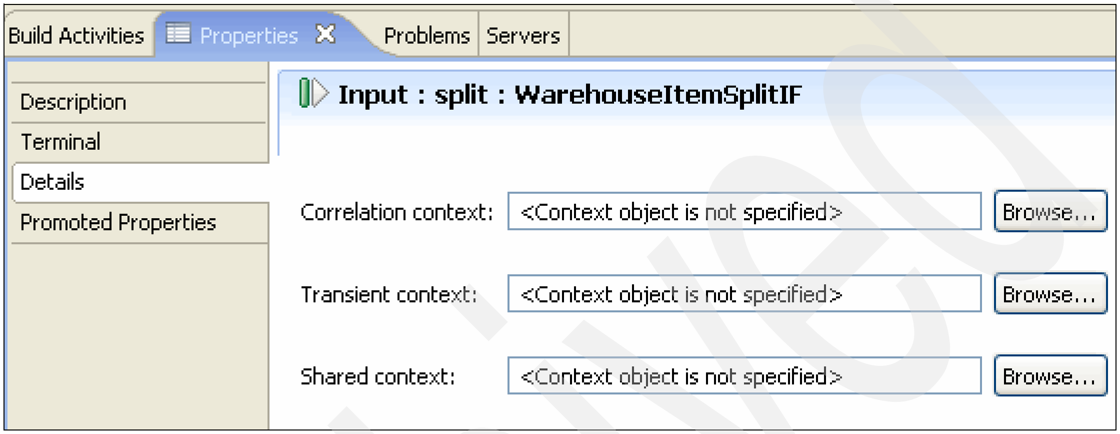


Figure 6-29 Mediation input node with contexts unspecified

2. For each field, use the Browse button to set the context type as follows:
   a. The correlation context must be of type **Item**.
   b. The transient context must be of type **WarehouseMediationFlowTransientContext**.
   c. The shared context must be of type **WarehouseMediationFlowSharedContext**.

   Figure 6-30 shows the results.

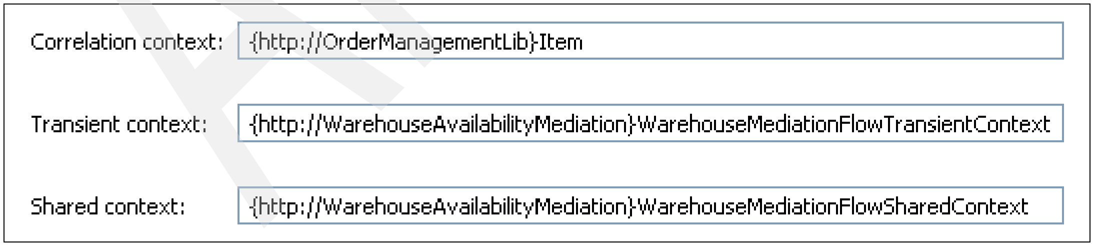


Figure 6-30 Setting contexts type

3. Save the mediation flow.

Now, you must move the Input content to the appropriate context, which requires an XSL Transformation primitive. This primitive is also used to prepare the message that is used for the Web services invocation. Follow these steps:

1. Drag a new XSL Transformation primitive from the Palette and change its display name to `Input To Service Invoke`.
2. Wire the Unavailability Generator primitive out terminal to the Input To Service Invoke in terminal and save the mediation flow editor.
3. Double-click the Input To Service Invoke transformation primitive to generate its implementation and to start the New XML Mapping wizard. In the wizard:
   a. In the first panel, name the primitive `InputToServiceInvoke` and click **Next**.
   b. In the second panel, change the message root to "/", as shown in Figure 6-31.

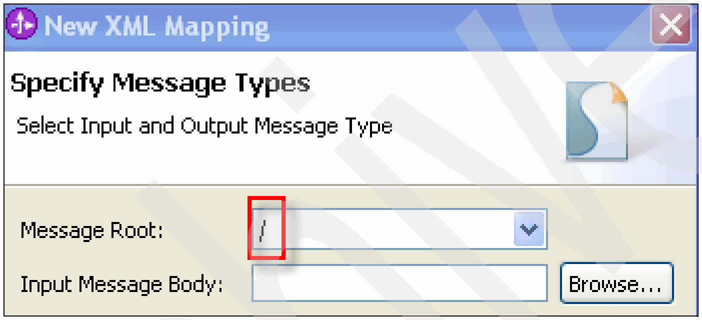


Figure 6-31 Selecting the message root for the XSLT Transformation

c. Click **Browse** next to the Input Message Body to go to the Change Message Type panel (shown in Figure 6-32). Complete the panel options as follows:

- Browse to the WarehouseItemSplitIF interface.
- Select the **split** operation.
- Select **Input** as the message category.
- Select **splitRequestMsg** as final message type.

Click **OK**.

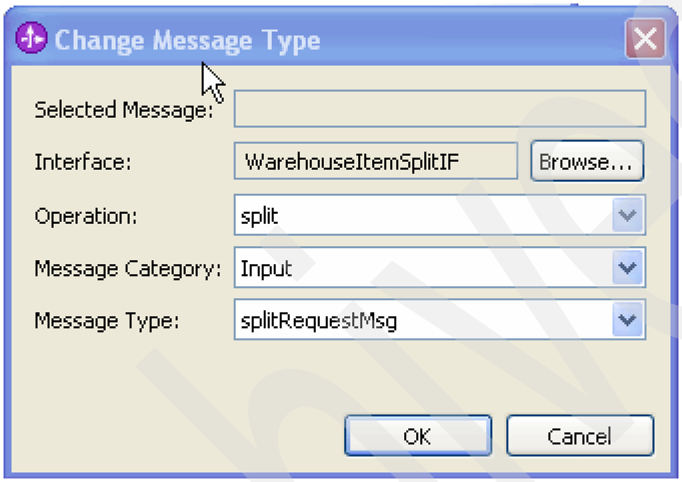


Figure 6-32 InputToServiceInvoke input message selection

Note: As you select the transformation types (input and output), you have a useful Browse tool that helps you to navigate through interfaces and their related objects. Consider where the input message comes from or where the output message is going to when performing this selection.

In this case, for example, our mediation implements WarehouseItemSplitIF, and this transformation is the first in the mediation flow. So, it gets the same input message as the one declared in WarehouseItemSplitIF.

d. Click **Browse** next to the Output Message Body to open the Change Message Type panel (shown in Figure 6-33). Then, complete the panel options as follows:

- Browse and select the WarehouseAvailabilityServiceIF Interface.
- Select the **isAvailable** operation.
- Select **Input** as the message category.
- Select **isAvailableRequestMsg** as final message type.

Click **OK** to return to the Specify Message Types panel (shown in Figure 6-34 on page 382).

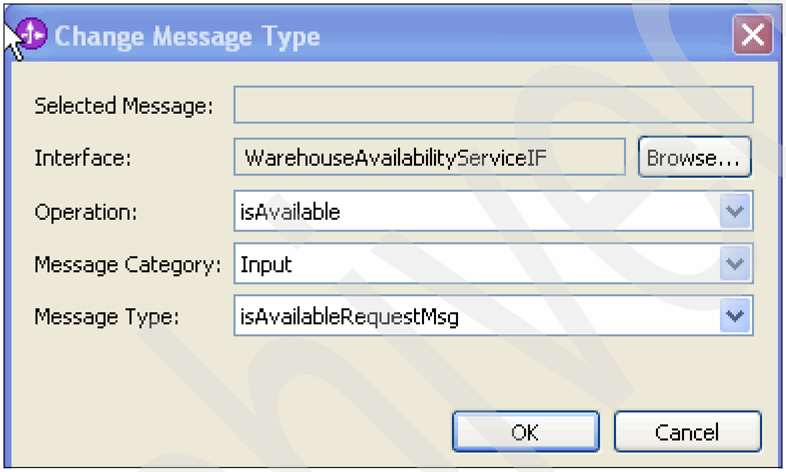


Figure 6-33 InvokeToService output message type

Note: Again, the output message type depends on the transformation target. In this case, the output message is used to invoke the Warehouse Availability System Web services, so it must be of the type those services require as input.

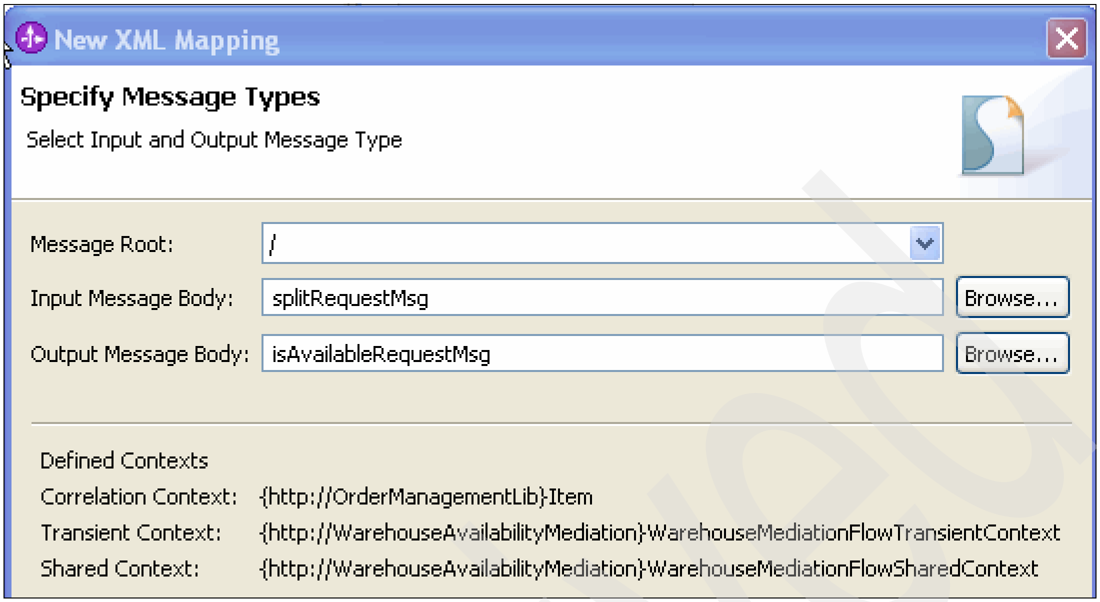


Figure 6-34 InputToServiceInvoke Input and Output types

e. Note how the correlation, transient and shared context correspond to what is specified in the Input node. Click **Finish**.

4. Now, define the transformation behavior. Click the **Map source to target based on names and types** icon at the top of the XML mapping editor, shown in Figure 6-35. This utility defines wiring between the source and target for all of the objects that it finds compatible in name and type. This method is a safe way to ensure that the contexts are always matched and carried unchanged through the mediation.

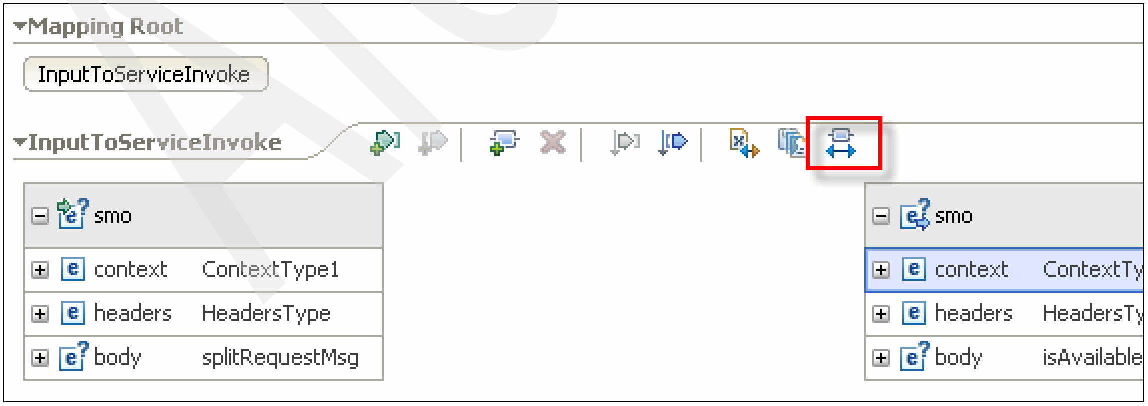


Figure 6-35 Mapping source and target based on names and types

Tip: When you are working extensively with contexts, use the following process to reduce the possibility of losing data as it traverses the mediation flow:

1. Use the **Map source to target based on names and types** utility to match all the contexts.
2. Then, for those elements that need to be moved somewhere else in the output message, remove the wirings that are created automatically and create the specific wirings that are needed.

In our case, the utility created maps between the contexts and headers as shown in Figure 6-36.

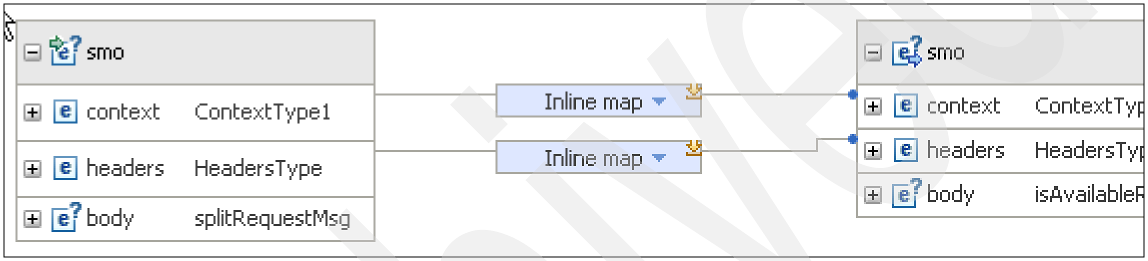


Figure 6-36 Automatic mappings created by the Map By Name and Type tool

3. Next, remove and update the wiring for specific transformation elements. This transformation has to:
   - Put the order quantity in the transient context.
   - Put the item warehouses list both in the transient context (for the split algorithm which will use it later) and the body (for the Warehouse Availability Systems Web services which will be invoked just after the transformation).
   - Put all of the item attributes except the warehouses list in the correlation context.

   The elements that you must take out from the automatic mapping and map as needed are:

   - Order quantity
   - Item warehouses list
   - Other Item attributes

4. Remove the wiring for the XSL elements that must be moved elsewhere as follows:

   a. Double-click the Inline map that connects the two contexts. The mapping details opens, and it shows five inline maps.

   b. Click the Inline map between the correlation fields and delete it. (The correlation context is filled with the message body in this transformation.)

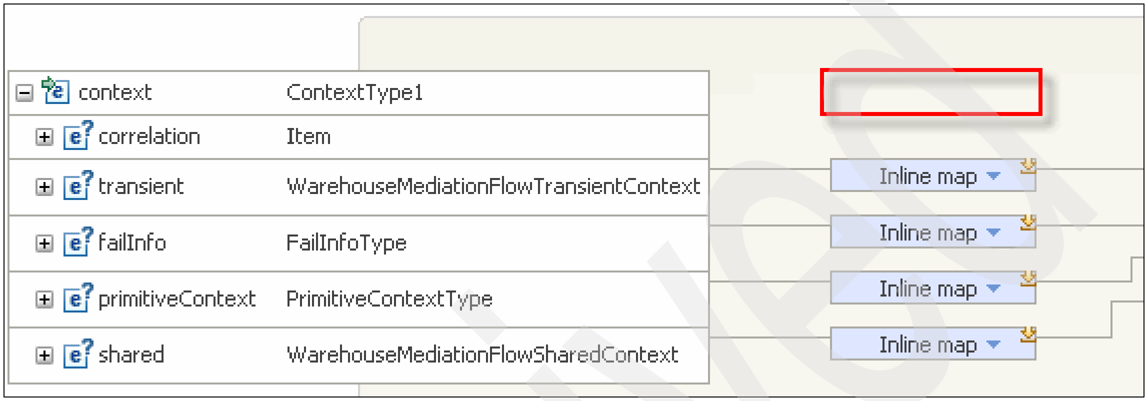


Figure 6-37 Getting into details for the contexts mapping

   c. Similarly, delete the Inline map between the transient fields. (The transient content are filled with the Input body).

   d. Save the primitive.

   e. Return to the previous map by clicking the arrow in the upper right corner of the contexts Inline map rectangle (shown in Figure 6-38).

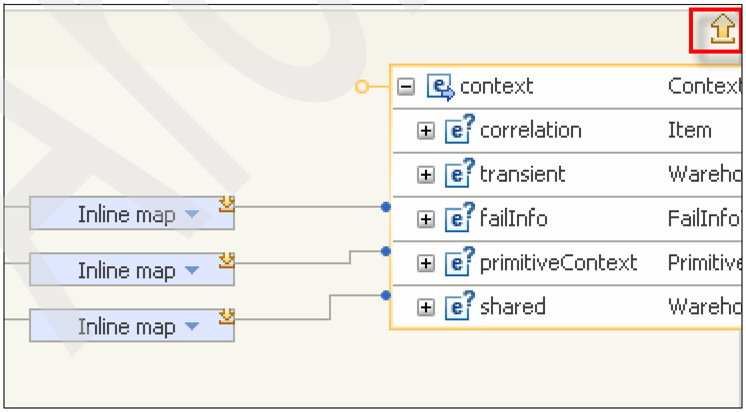


Figure 6-38 Moving back navigating the Inline Maps definitions

5. Create specific wiring for the body elements as follows:

   a. Expand the **body** → **split** → **Item** elements in the input message.

   b. Expand the **context** → **correlation** for the output message.

   c. Wire all of the **body** → **split** → **Item** elements to the **context** → **correlation** → **Item** elements but the warehouses array as shown in Figure 6-39.

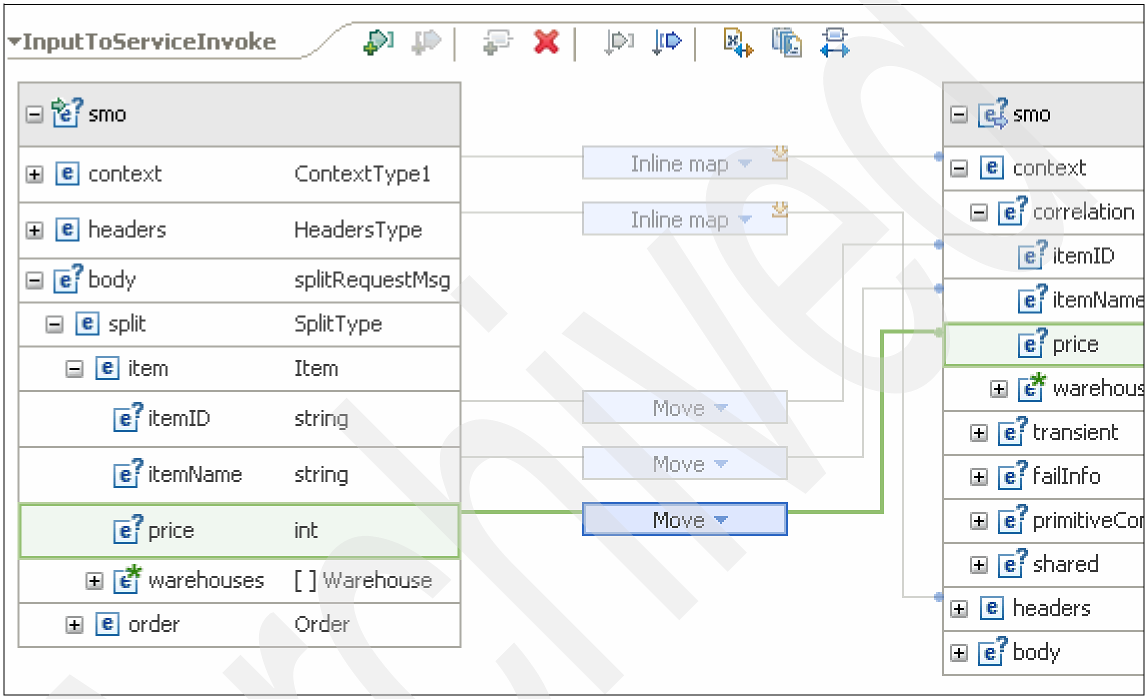


Figure 6-39 Moving Input Item attributes in the correlation context

d. Now, expand **context** → **transient** in the output message.

e. Wire the warehouses array in the input message **body** → **split** → **Item** → **warehouses** to **transient** → **warehouses** in the output message as shown in Figure 6-40.

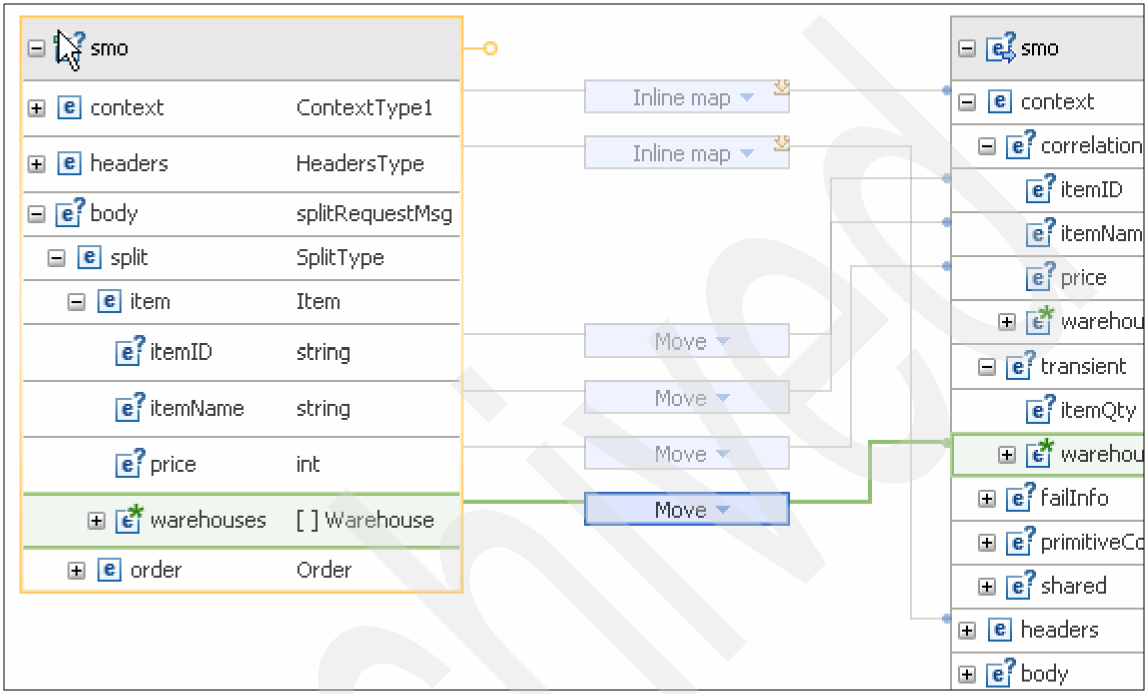


Figure 6-40 Input warehouses list mapped to the transient context

Tip: Always map critical data using an Inline map and then map each of the elements in the array manually to keep total control over the mapping. We show this method in the next step.

f. For the warehouses array, change the **Move** operation to an **Inline map** operation using the blue down arrow next to **Move**. Double-click this Inline Map to open the implementation details. Then, wire each array element manually as shown in Figure 6-41.

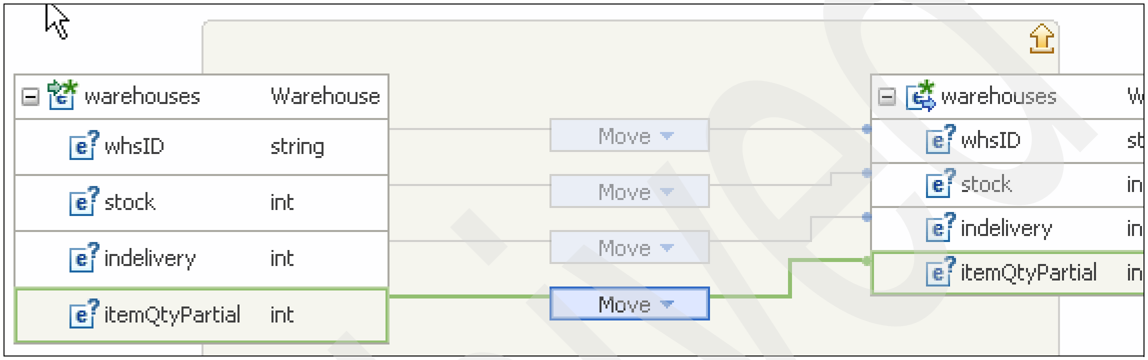


Figure 6-41 Explicit mapping for warehouse array elements

g. Return to the main mapping editor using the up arrow upper right.

h. Now expand **body** → **isAvailable** → **warehouses** in the output message and wire the input message **body** → **split** → **item** → **warehouses** to **body** → **isAvailable** → **warehouses** → **wList**, as shown in Figure 6-42.

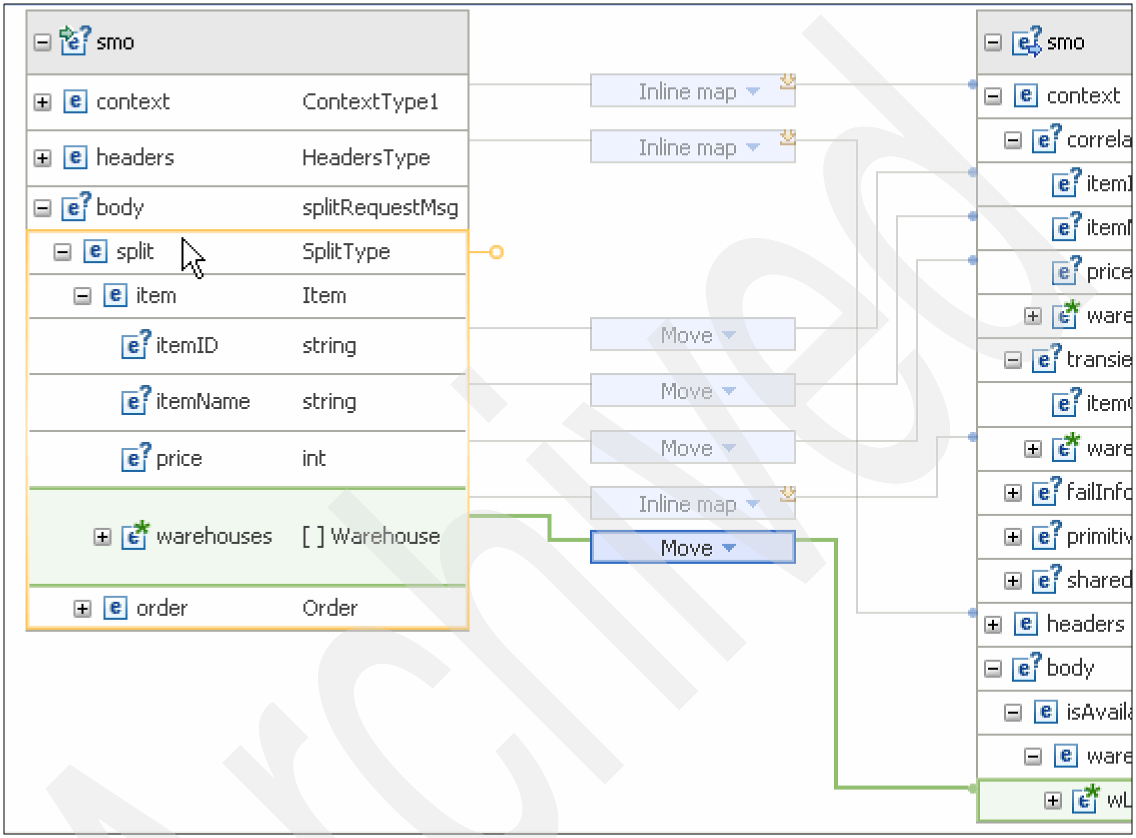


Figure 6-42 Mapping Input warehouses to output body

i. As before, convert this mapping to an **Inline map** and explicitly map every array field in the Inline map implementation.

j. Now expand **body** → **split** → **Order** in the input message.

k. Wire the input message **body** → **split** → **Order** → **itemQty** to the output message **context** → **transient** → **itemQty**

6. Save and close the XML mapping editor.

   Now the correlation and transient context contain the elements that you need to pass through the mediation (Figure 6-43), and the body is ready to be fired to the Warehouse Availability Systems Web services.

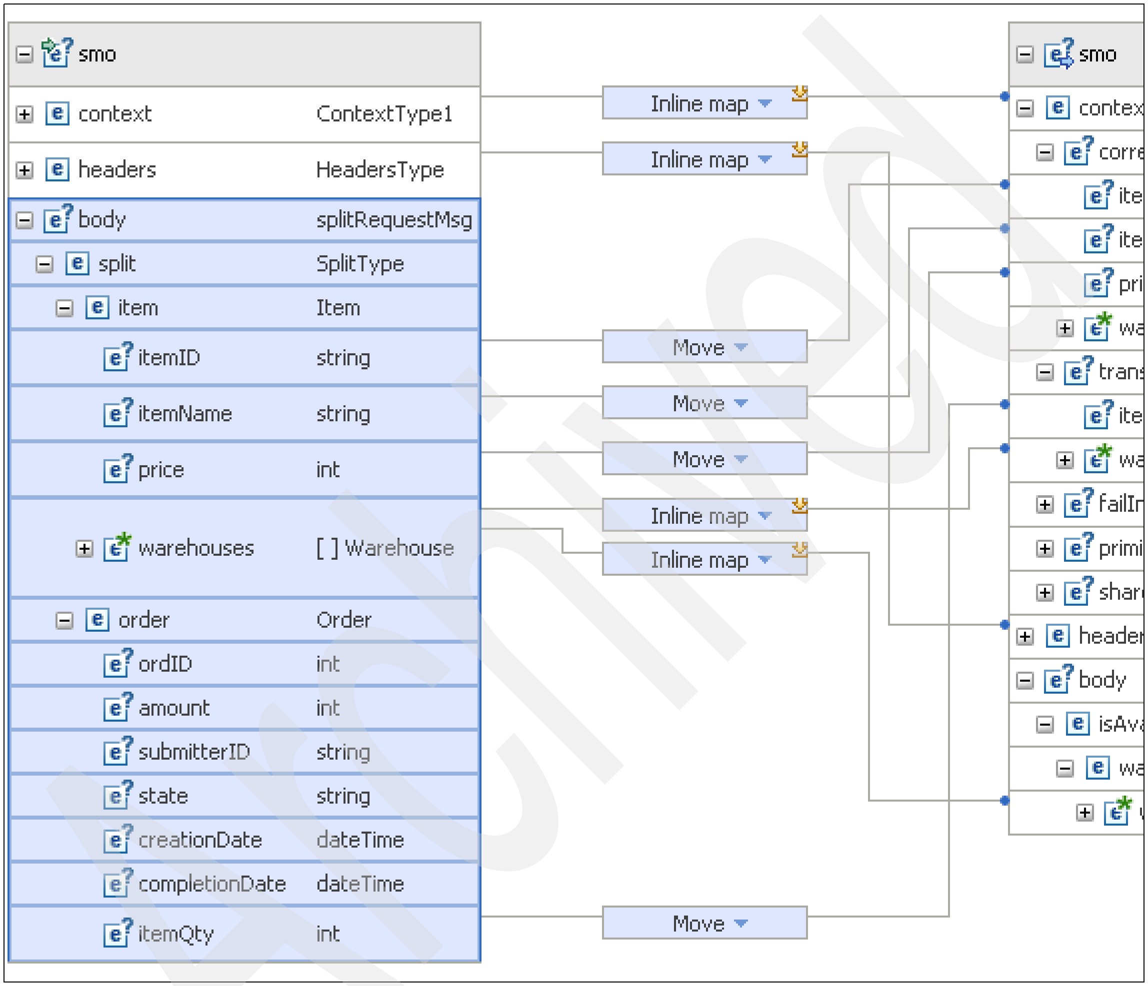


Figure 6-43 InputToServiceInvoke complete XSLT map

7. Save the mediation flow and check that no errors are raised for the InputToServiceInvoke map.

6.4.3 Getting warehouse availability

This portion of the mediation completes the following actions:

- Split the mediation flow using the Fan Out primitive.
- Query the Warehouse Availability System Web services.
- Aggregate the results coming from the services in the shared context.
- Join the separate flow paths back into one using Fan in primitive

Add and wire the nodes

First, you need to add the primitives to the flow and wire them together as follows:

1. Drag a Fan Out primitive to the mediation flow editor from the Palette (Routing section) and change the display name to `Fan Out Warehouses Web Services`.
2. Wire the out terminal of the InputToServiceInvoke transformation to the in terminal of the Fan Out primitive, as shown in Figure 6-44, to determine automatically the type of the Fan out in terminal.

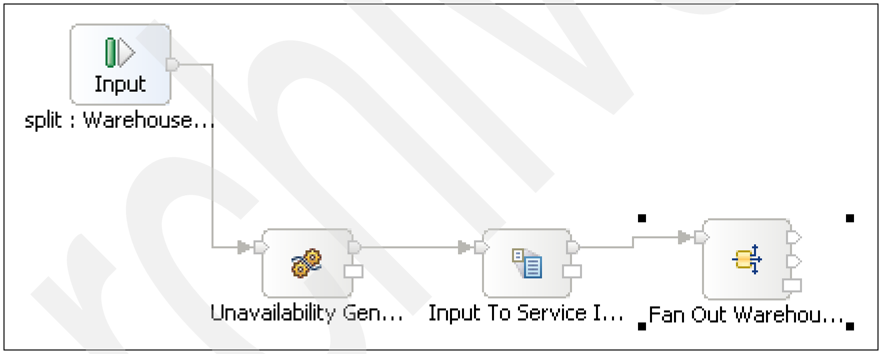


Figure 6-44 Fan Out Warehouses Web Services in the flow editor

3. Select the Fan Out Warehouse Web Services configuration and open the Details pane of the Properties view.

   You want to invoke each Web service exactly one time using the message that you built using the InputToServiceInvoke transformation. Take the default to fire the message exactly once for every output connection, as shown in Figure 6-45 on page 391.

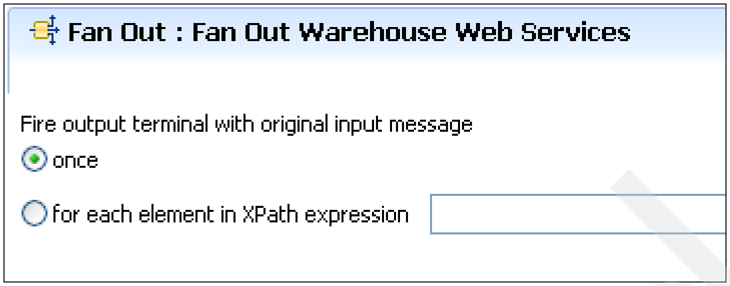


Figure 6-45 Fan out configured to fire the message once

4. Drag a Service Invoke primitive from the Palette (Routing section), and place it to the upper right of the Fan Out Warehouse Web Services. As you release the mouse, you are prompted for the Reference to be bound to this service invoke. Select **WarehouseAvailabilityServiceIFPartner** as shown in Figure 6-46.

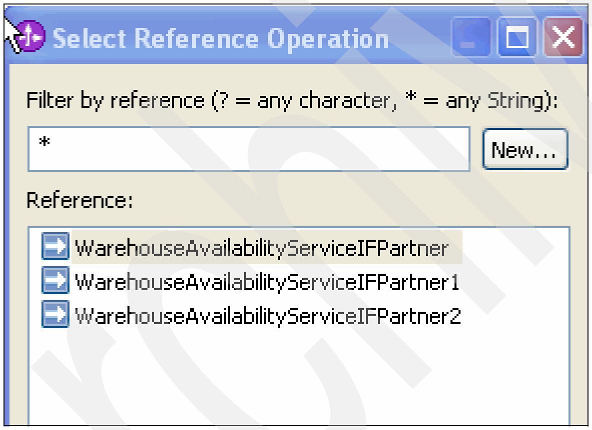


Figure 6-46 First service invocation against first partner, Warehouse A Web service

Change the display name of the service invoke to `Warehouse A Availability`.

5. Drag two more Service Invoke primitives to the right of the Fan Out Warehouse Web Services (center and bottom position). Bind them respectively to WarehouseAvailabilityIFPartner1 and WarehouseAvailabilityIFPartner2. Name them, respectively, `Warehouse B Availability` and `Warehouse C Availability` as shown in Figure 6-47.

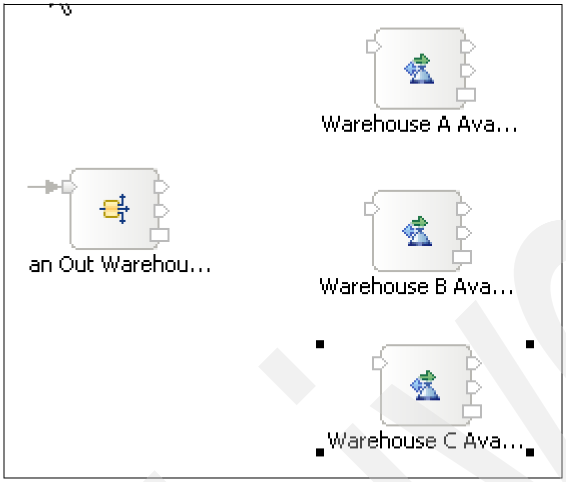


Figure 6-47 Warehouse Availability Systems invocation

6. Drag three XSL Transformation primitives next to the three Service Invoke primitives and change their display name, respectively, to `Agg War A`, `Agg War B`, and `Agg War C`, as shown in Figure 6-48.

   At this time, do not worry about errors. They will be resolved as you implement the transformations later.

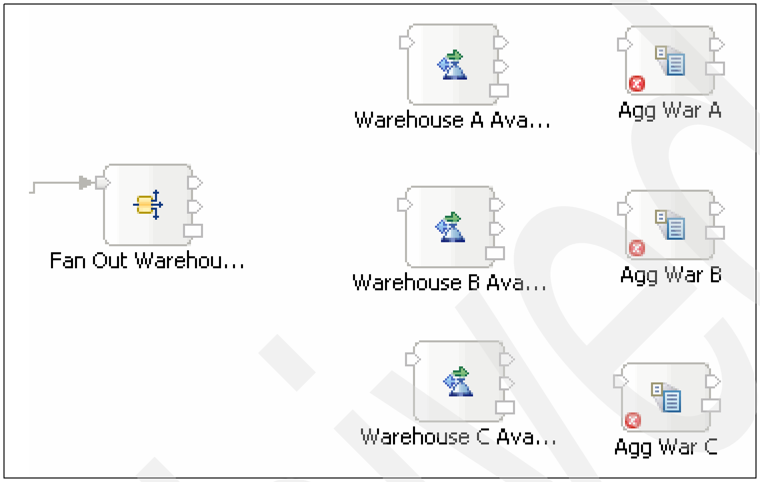


Figure 6-48 Aggregation transformation added to the mediation editor

7. Drag a Fan In primitive (from the Palette, Routing section) next to the three aggregation XSL Transformation primitives. As you drag the Fan In primitive, you need to specify the Fan Out primitive to which it is related, in this case, `Fan Out Warehouse Web Services`, as shown in Figure 6-49.

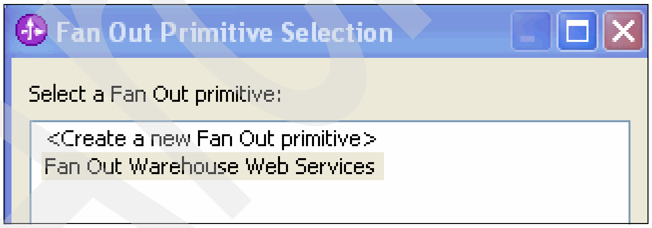


Figure 6-49 Fan in and Fan Out correspondent

8. Change the Fan In display name in `Fan In Warehouses Web Services`, as shown in Figure 6-50.

   All of the primitives for Warehouse Service Availability check are on the mediation flow editor now. You can ignore any errors for now.

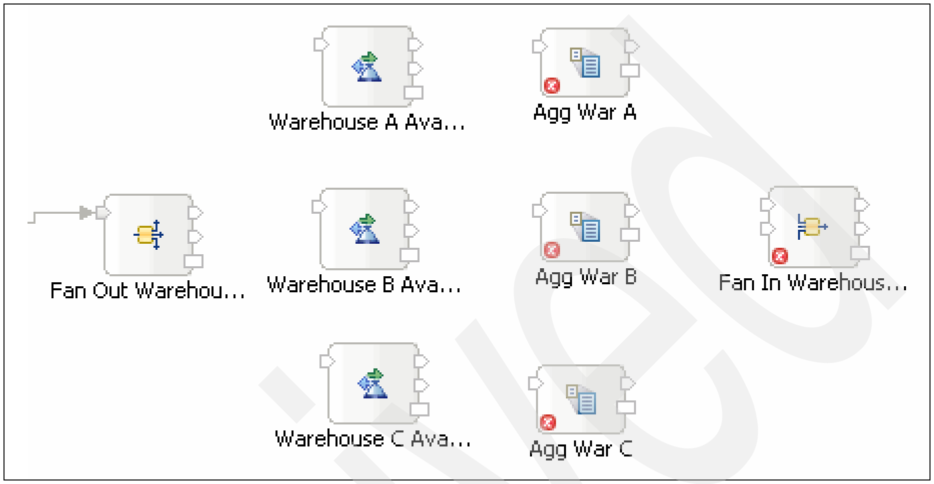


Figure 6-50 Warehouse Availability mediation flow primitives

9. Wire all of the primitives as shown in Table 6-2 (in the order listed in the table).

Table 6-2 Warehouse Availability mediation flow wiring

Order number	Starting point	Ending point
1st	Fan Out Warehouse Web Services **out** terminal	Warehouse A Availability **in** terminal
2nd	Fan Out Warehouse Web Services **out** terminal	Warehouse B Availability **in** terminal
3rd	Fan Out Warehouse Web Services **out** terminal	Warehouse C Availability **in** terminal
4th	Warehouse A Availability **out** terminal	Agg War A **in** terminal
5th	Warehouse B Availability **out** terminal	Agg War B **in** terminal
6th	Warehouse C Availability **out** terminal	Agg War C **in** terminal

Order number	Starting point	Ending point
7th	Agg War A **out** terminal	Fan In Warehouses Web Services **in** terminal
8th	Agg War B **out** terminal	Fan In Warehouses Web Services **in** terminal
9th	Agg War C **out** terminal	Fan In Warehouses Web Services **in** terminal

Attention: Be sure to select the terminal indicated in Table 6-2. In particular, the Service Invoke primitive can be confusing because it has a three output terminals (timeout, fail, and out).

10.Save the editor. Figure 6-51 shows the expected mediation editor.

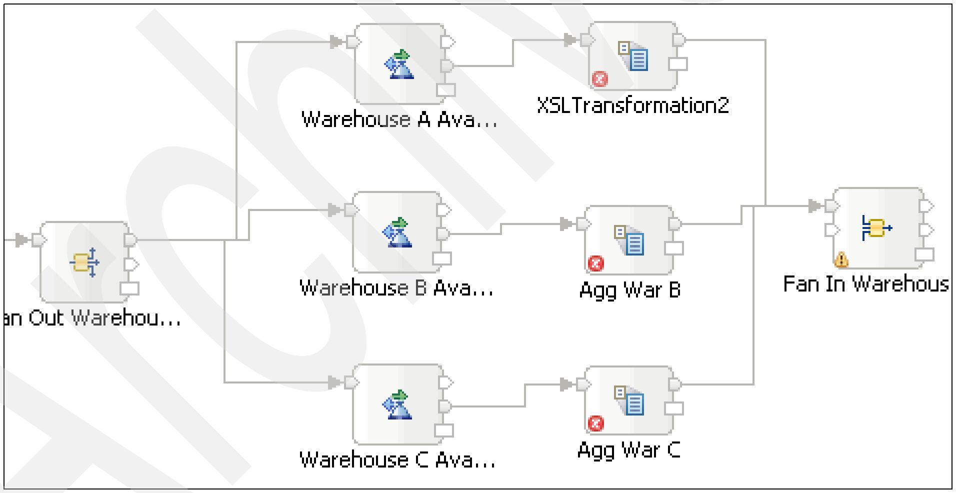


Figure 6-51 Warehouse Availability mediation flow wired

The next steps take you through the implementation of each primitive, including the three XSLT maps that aggregate results coming from the Web services in the shared context. The Implementation is quite similar to what you performed for InputToServiceInvoke, so we provide a less detailed description.

Agg War A XSL Transformation primitive

Implement the Agg War A primitive as follows:

1. Double-click the primitive to create the map:
   a. Name it `AggregateWarehouseA`.
   b. Change the Message Root to "/".
   c. Leave the Input message type as you find it (`isAvailableResponseMsg`).
   d. For the output message, select:
      - Interface: `WarehouseItemSplitIF`
      - Operation: `split`
      - Message category: `Output`
      - Message type: `splitResponseMsg`

   The configuration looks similar to that shown in Figure 6-52.

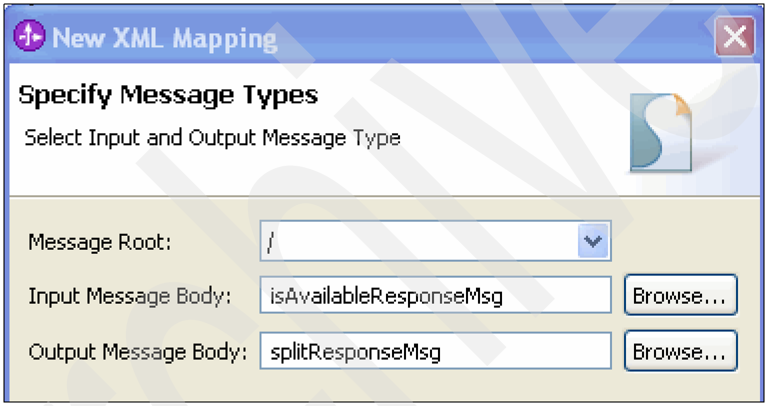


Figure 6-52 AggregateWarehouseA Input Output configuration

2. Use the **Map source to target based on names and types** tool to create an automatic copy of all of the headers and contexts.
3. Open the context inline map. Then, open the shared context Inline map, and remove the `warA_availability` mapping (Figure 6-53 on page 397).

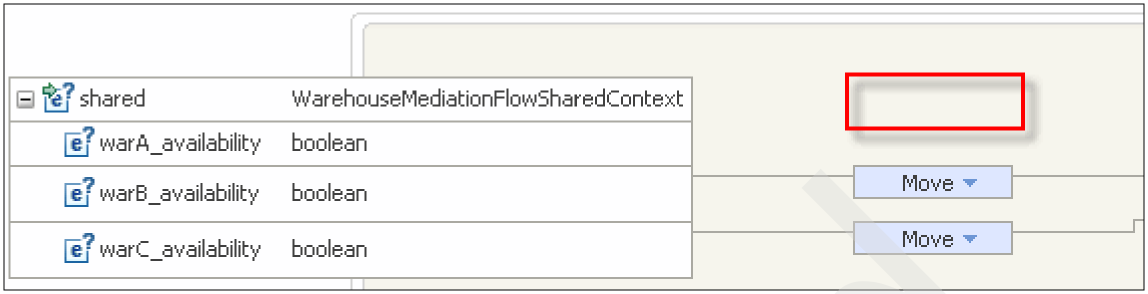


Figure 6-53 Removing Warehouse A availability copy from shared map

> **Important:** You are not removing the `warB_availability` and `warC_availability` mapping, which is intentional. You are running three Web services and aggregating through maps. You cannot know in which order the maps will be performed. So, you must always carry what you find in the shared context, because it might already contain results from the other Web services

4. Go back to the original map (clicking the up arrow in the right upper corner of the Inline map editor twice). Wire the input message `body` → `isAvailableResponse` → `isAvailable to the output message context` → `shared` → `WarA_availability` as shown in Figure 6-54.

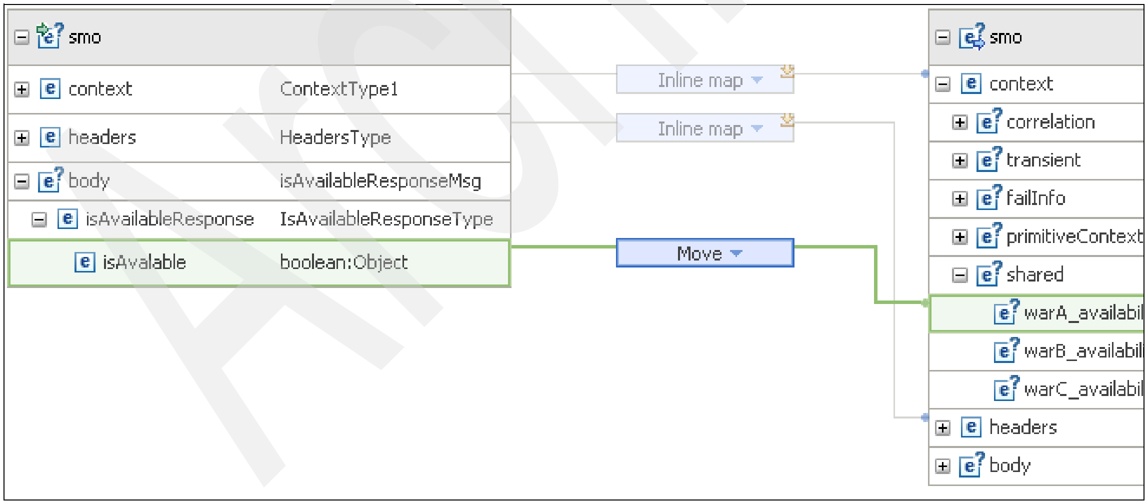


Figure 6-54 Aggregating Warehouse A availability to the shared context

5. Save and close the map.

Agg War B XSL Transformation primitive

Implement the Agg War B XSLT map as you implemented the Agg War A XSLT except for the following:

- Name the map `AggregateWarehouseB`.
- In the shared context inline map, remove the `warB_availability` mapping, as shown in Figure 6-55.

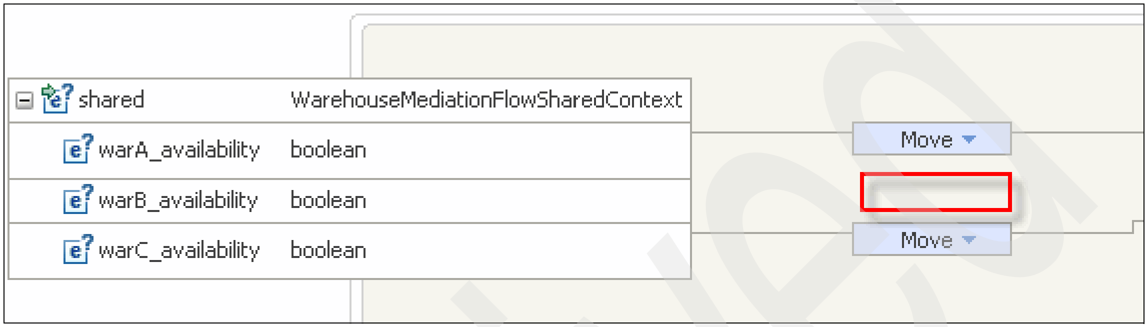


Figure 6-55 warB_availability removed from the shared context Inline map

- Map the input message `body` → `isAvailableResponse` → `isAvailable` to the output message `context` → `shared` → `warB_availability`, as shown in Figure 6-56.

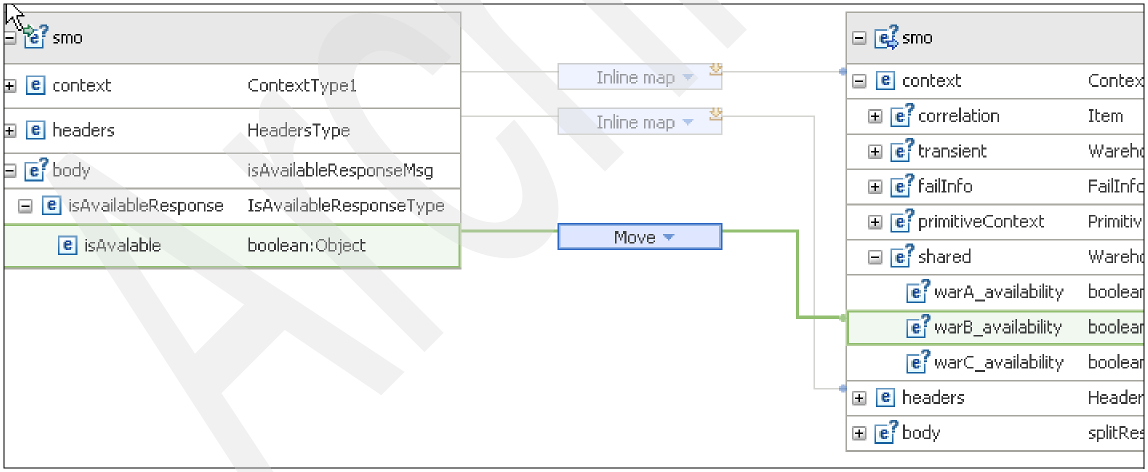


Figure 6-56 Aggregating warehouse B availability to shared context

Agg War C XSL Transformation primitive

Implement the map for Agg War C as you did for Agg War A except for the following:

- Name the map `AggregateWarehouseC`.
- In the shared context inline map, remove the `warC_availability` mapping, as shown in Figure 6-57.

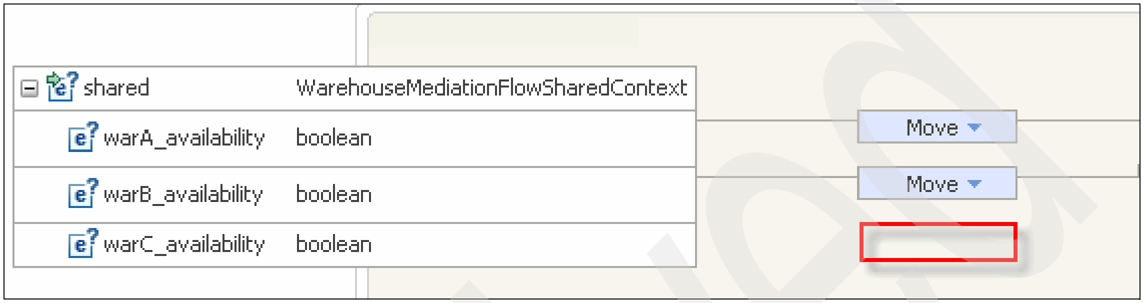


Figure 6-57 warC_availability removed from the shared context Inline map

- Map the input message `body` → `isAvailableResponse` → `isAvailable` to the output message `context` → `shared` → `warC_availability` as shown in Figure 6-58.

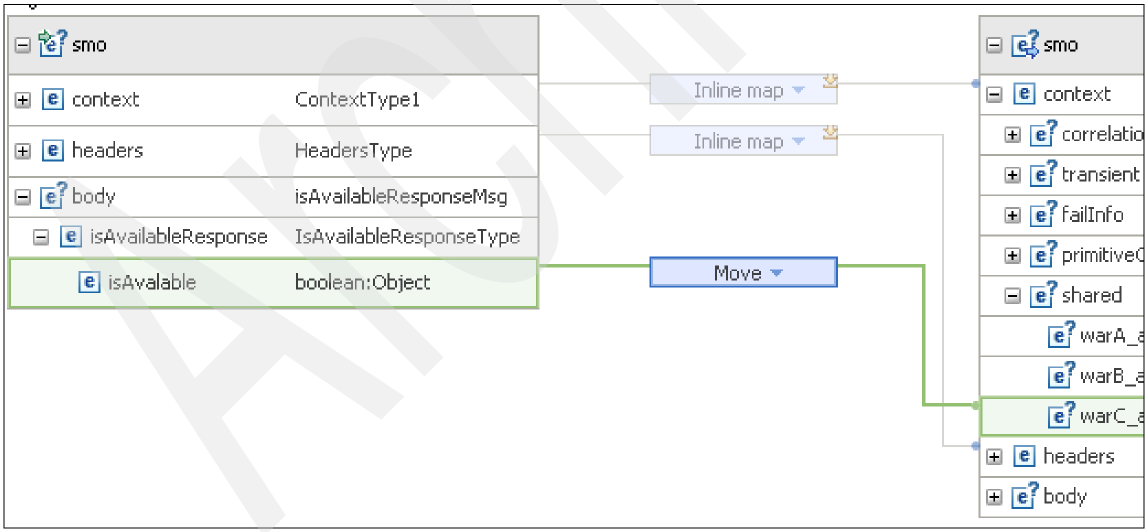


Figure 6-58 Aggregating warehouse C availability to shared context

The aggregation maps are now complete.

Fan In primitive

You now configure the Fan In primitive to ensure that it waits for all of the service responses and aggregation to complete before firing the aggregated message. Follow these steps:

1. Select **Fan In Warehouses Web Services**. In the Details tab of the Properties view, enter `3` in the **Fire output terminal when** field, as shown in Figure 6-59.

Figure 6-59 Fan in must wait for the three messages to fire the aggregated output

2. Save the mediation flow editor.

The mediation flow now contains no errors.

6.4.4 Performing quantity split over available warehouses

With the previous implementation steps complete, the WarehouseAvailabilityMediation flow now works with messages that have the following characteristics:

- Warehouse availability status in the shared contexts, aggregated by AggregateWarehouseA, AggregateWarehouseB, and AggregateWarehouseC XSL maps with the results coming from the warehouse Availability Systems Web services.
- Original warehouses list in the transient context moved there by the InputToServiceInvoke XSL map and carried on by the AggregateWarehouseA, AggregateWarehouseB, and AggregateWarehouseC XSL maps.
- Order item quantity in the transient context moved there by the InputToServiceInvoke XSL map and carried on by the

AggregateWarehouseA, AggregateWarehouseB, and AggregateWarehouseC XSL maps.

- Item attributes (all but the warehouses list) in the correlation context for what we implemented in the InputToServiceInvoke transformation.

You now create a Fan In to Split Algorithm transformation that prepares the input object for the split algorithm. This transformation basically moves the Item correlation context in the message body. The algorithm builds the warehouses list in the Item object based on the warehouse availability in the shared contexts.

Add the primitives and wire them

To add and wire the primitives, follow these steps:

1. Drag an XSL Transformation primitive to the mediation editor and change the display name to `Fan In to Split Algorithm`.
2. Drag a Custom Mediation primitive to the mediation editor and change the display name in `Warehouse Quantity Split`.
3. Wire all of the primitives according to Table 6-3 (in the order listed in the table).

Table 6-3 Warehouse Availability mediation flow wiring

Order number	Starting point	Ending point
1st	Fan In Warehouse Web Services **out** terminal	Fan In to Split Algorithm **in** terminal
2nd	Fan In to Split Algorithm **out** terminal	Warehouse Quantity Split **in** terminal
3rd	Warehouse Quantity Split **out** terminal	Input Response **in** terminal

4. Save the mediation flow.

Figure 6-60 shows the wired mediation flow.

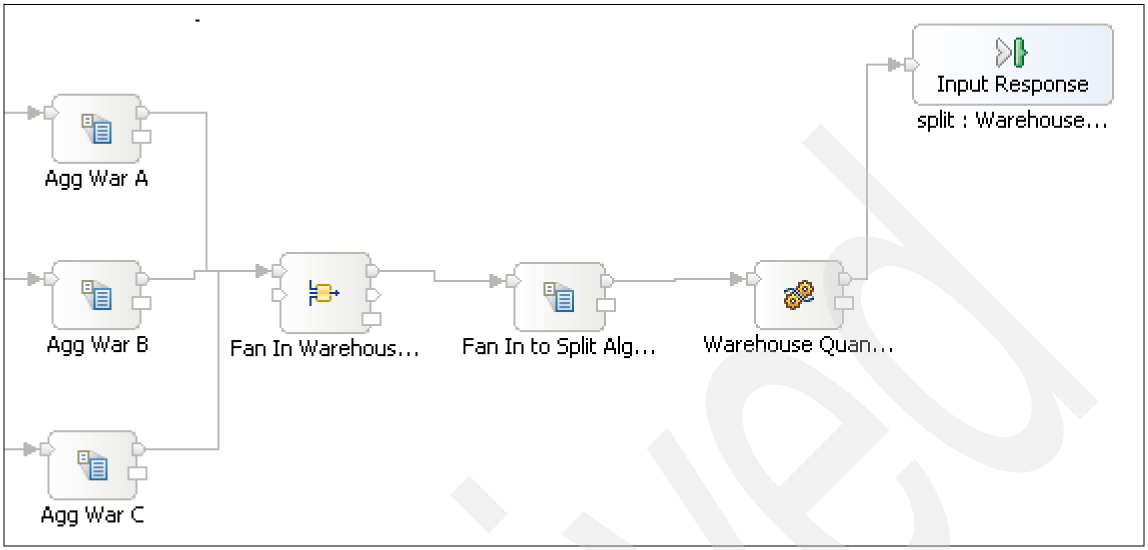


Figure 6-60 Split quantity mediation flow completely wired

Fan In Split Algorithm

Now, you must implement the Fan In Split Algorithm transformation. It is very similar to what you performed for the other transformations. Follow these steps:

1. Double-click the **Fan In Split Algorithm** to implement it:
   - Name it `FanInSplitAlgorithm`.
   - Change the message root to "/".
   - Click **Browse** to the right to the Output Message Body to select the splitResponseMsg (Output of the split operation, WarehouseItemSplitIF interface), as shown in Figure 6-61 on page 403.

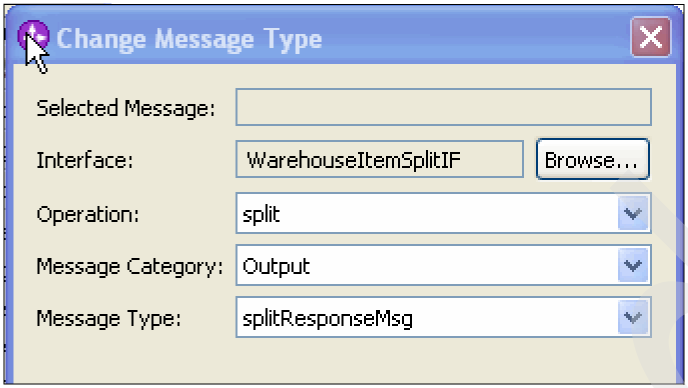


Figure 6-61 Selecting splitResponseMsg as XSL transformation output

The input and output message bodies are the same type, as shown in Figure 6-62. Click **Finish**.

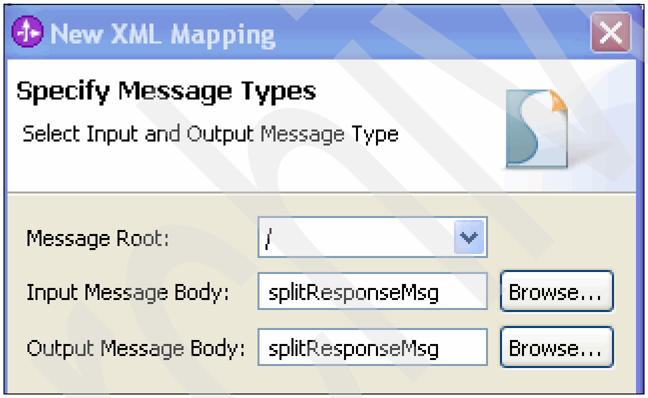


Figure 6-62 Fan In to Split Algorithm Input and Output message types

2. In the XML mapping editor (Figure 6-63 on page 404):
   a. Use the **Map source to target based on names and types** tool to create an automatic copy of all of the headers and contexts.
   b. Open the inline map for the correlation context.
   c. Remove all but the warehouses array mapping (because you must move the itemID, name, and price to the output body).

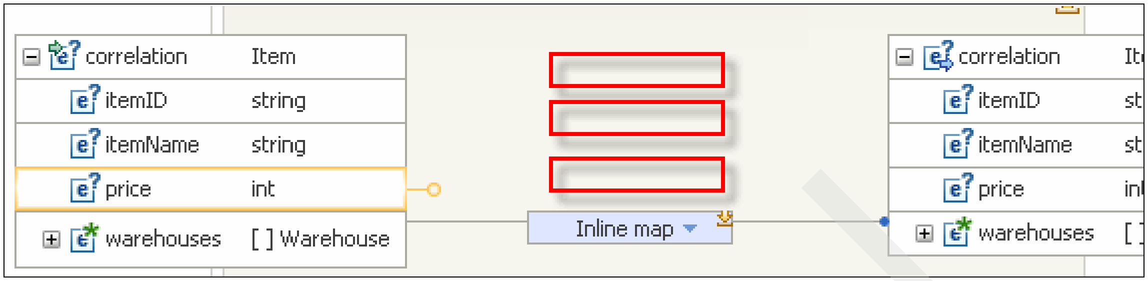


Figure 6-63   Deleting automatic mappings from the correlation context inline map

d. Return to the original map, and remove the body inline map because you have to move the correlation context content into the body instead. See Figure 6-64.

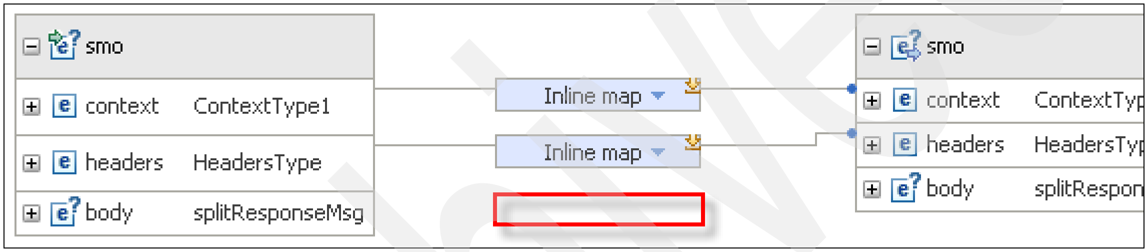


Figure 6-64   Body Inline map removal

e. Expand the input message `context` → `correlation` and the output `body` → `splitResponseMsg` → `Item`.

f. Map itemID, itemName, and price from the input message `context` → `correlation to the` output message `body` → `splitResponse` → `Item` as shown in Figure 6-65.

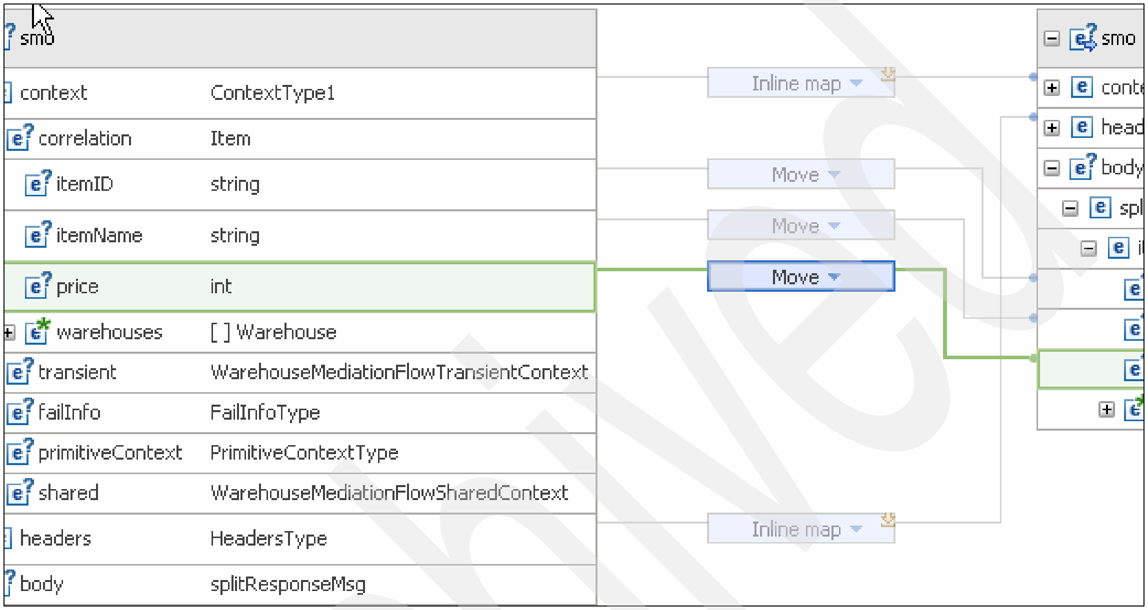


Figure 6-65 Mapping correlation context to output body

3. Save and close the XSLT map

Warehouse Quantity Split primitive

The transformed output message is now the input to the final split algorithm. The algorithm splits the total quantity over the available warehouses. This process is performed in the Warehouse Quantity Split Custom Mediation primitive, as shown in Figure 6-66.

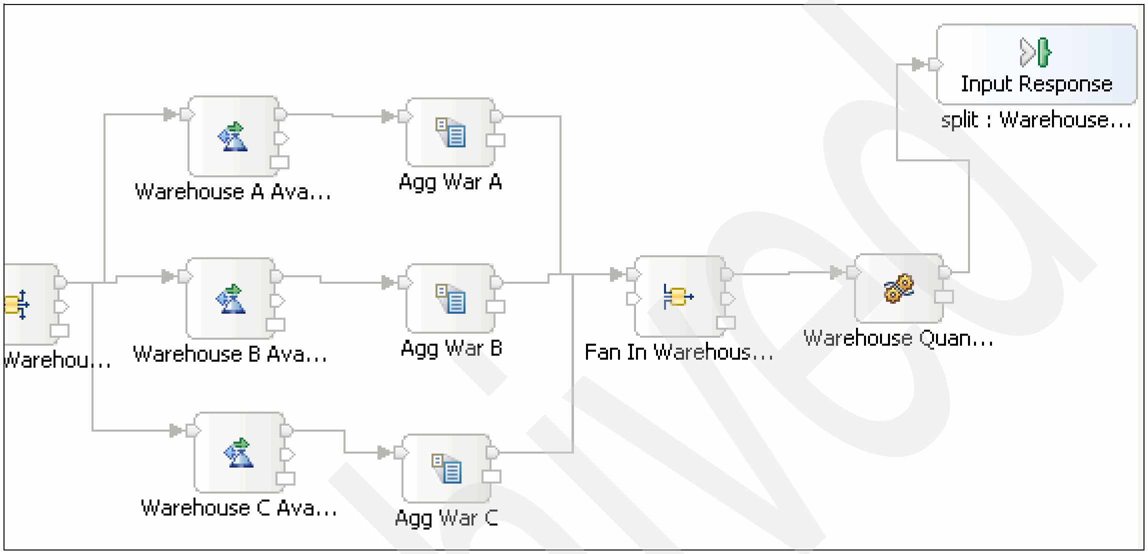


Figure 6-66 Warehouse Quantity Split Custom Mediation primitive in the flow editor

Addition material: The code that is used for the Warehouse Quantity Split primitive is included in the additional materials for this book in the file WarehouseQuantitySplit.txt. See Appendix B, "Additional material" on page 461.

To complete the Custom Mediation primitive, perform the following steps:

1. Select the Warehouse Quantity Split primitive.
2. In the Details tab of the Properties view, select **Java** as the implementation, and then add the code shown in Example 6-2.

Example 6-2 Java implementation for Warehouse Quantity Split

```
commonj.sdo.DataObject __smo = (commonj.sdo.DataObject)smo;
java.util.List warehouse_orig =
__smo.getDataObject("context").getDataObject("transient").getList("warehouses");
int warehouse_orig_size = warehouse_orig.size();
System.out.println("Original warehouses list size "+warehouse_orig_size);
```

```
int warehouse_number = 0;
java.util.List warehouse_new = new java.util.ArrayList();
boolean[] availability = {false, false, false};
if(__smo.getDataObject("context").getDataObject("shared").getBoolean("warA_availabili
ty")){
   warehouse_number++;
   availability[0] = true;
}
if(__smo.getDataObject("context").getDataObject("shared").getBoolean("warB_availabili
ty")){
   warehouse_number++;
   availability[1] = true;
}
if(__smo.getDataObject("context").getDataObject("shared").getBoolean("warC_availabili
ty")){
   warehouse_number++;
   availability[2] = true;
   }
System.out.println("Warehouses AVAILABLE number "+warehouse_number);
int itemQty =
__smo.getDataObject("context").getDataObject("transient").getInt("itemQty");
int baseQty;
int residualQty;
{// divide
   baseQty =  itemQty / warehouse_number;
}
{// rest
   residualQty =  itemQty % warehouse_number;
}
int whoGetsResid = 0; //index of the one which will get residual
for(int i=0; i< warehouse_orig_size; i++){
   commonj.sdo.DataObject warehouse = (commonj.sdo.DataObject) warehouse_orig.get(i);

   if(!availability[i]){
      whoGetsResid++;
   }
   else{
      int itemQtyPartial = baseQty;
      if (i==whoGetsResid)
         itemQtyPartial += residualQty;
      warehouse.setInt("itemQtyPartial", itemQtyPartial);
      warehouse_new.add(warehouse);
   }

}
```

```
__smo.getDataObject("body").getDataObject("splitResponse").getDataObject("item").setList("warehouses", warehouse_new);
out.fire(__smo);
```

This code cycles through the original list of warehouses, removing unavailable warehouses from the list and adding their quantity partial to the first available warehouse in the new list.

The development phase is over. The next step is to test the module.

6.5 Testing the mediation

To test the mediation module:

1. Start the test environment server.
2. Deploy the WarehouseAvailabilityMediation and the three WarehouseAvailabilityService modules to the test environment as shown in Figure 6-67.

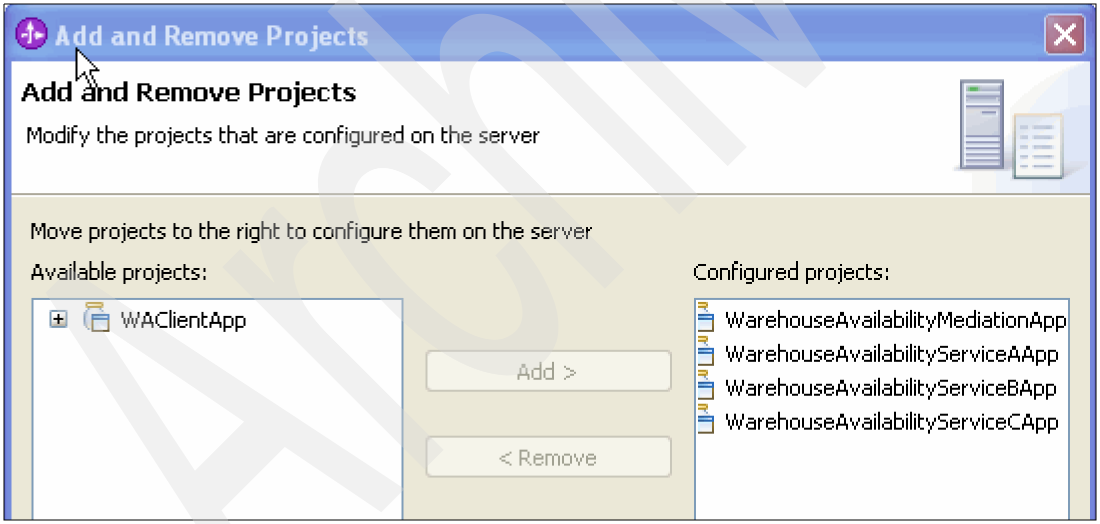


Figure 6-67 Deploy Web services and mediation module for testing

3. Launch a module test instance by right-clicking the **WarehouseAvailabilityMediation** module in the Business Integration view and selecting **Test** → **Module**.
4. Select the WarehouseAvailabilityMediation as the component, and leave the other settings unchanged as shown in Figure 6-68.

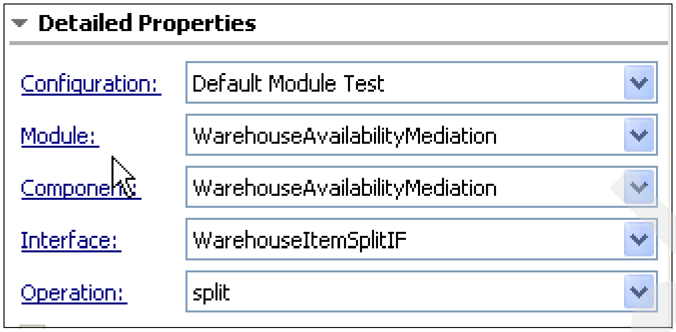


Figure 6-68 Detailed properties for module testing

5. The values set for the initial request parameters must be as shown in Figure 6-69 on page 410.

   Enter values in accordance with the type.

   To add the elements to the array, right-click the warehouses entry, and select **Add Elements**.

   > **Note:** The value used for the Order itemQty value basically determines the result.

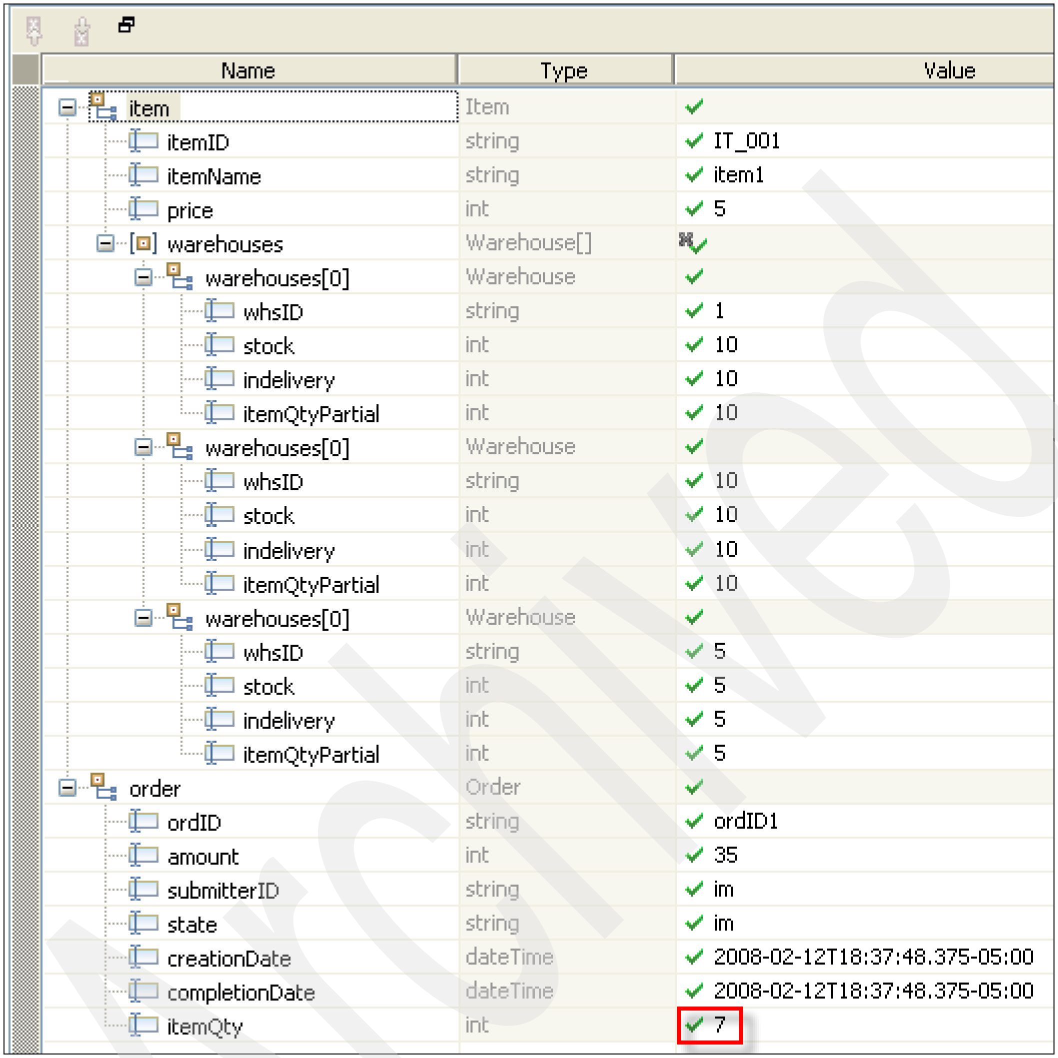


Figure 6-69 Initial request parameters for testing

In this case, you submit a request with a total quantity of 7. This value does not trigger any warehouse to return an unavailability status (only the value of 19 does that).

6. Run the test, ensuring the WebSphere ESB Server v6.1 test environment is selected.

   When the test completes, the request and response flow displays in the Events window as shown in Figure 6-70.

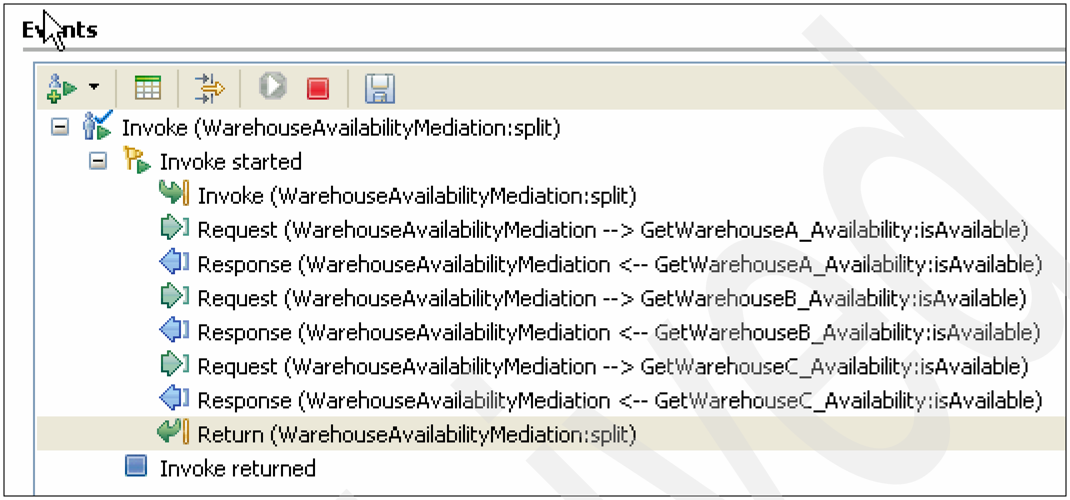


Figure 6-70 Test Request Response flow

> **Tip:** If you experience an unexpected request or response flow or if you get an unexpected exception, try cleaning up the environment as follows
>
> 1. Close the Test client session.
> 2. Remove the WarehouseAvailabilityMediation from the server.
> 3. Clean the projects by using **Project** → **Clean**.
> 4. Confirm cleaning for all of the project in the workspace.
> 5. Deploy the WarehouseAvailabilityMediation to the server.
> 6. Repeat the test.

Looking at the test results, you have all three warehouses returned in the array, and the item quantity is split according to the algorithm. Because the order quantity was 7 and all of the three warehouses are available, their quantity partial (look at the itemQtyPartial attributes in Figure 6-71 on page 412) is respectively 3, 2, and 2.

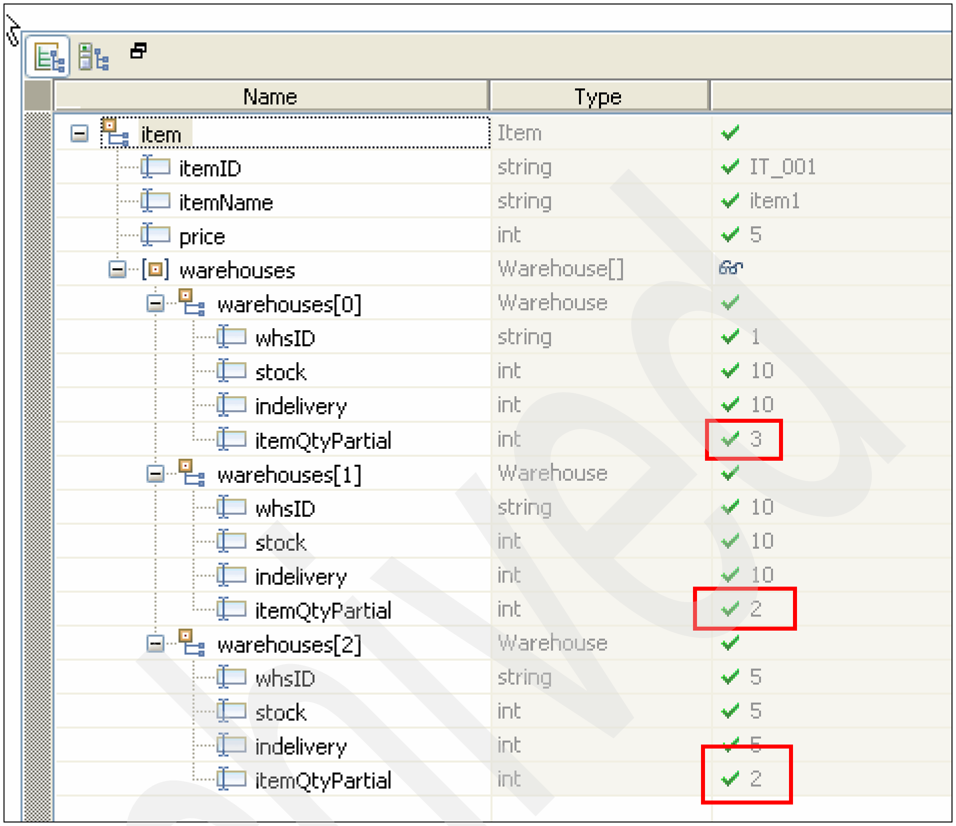


Figure 6-71 Results from the "all warehouse available" test case

7. Repeat the test using the same Input parameters but setting 19 as itemQty as shown in Figure 6-72.

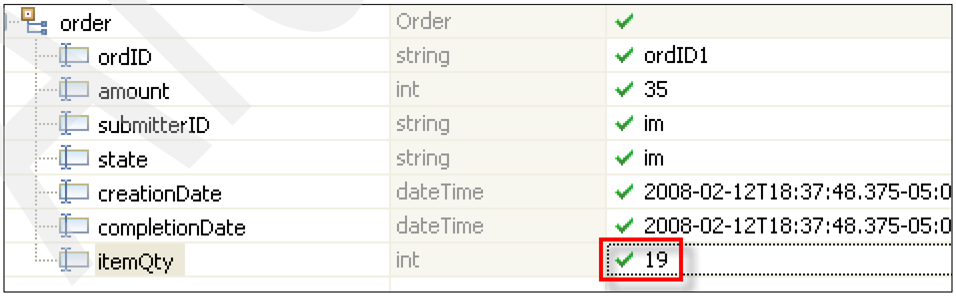


Figure 6-72 Second test with the 19 as itemQty

Now, Warehouse A is found unavailable, and only the other two warehouses are included in the array. As a consequence, order item quantity is split between the two itemQtyPartial (see Figure 6-73).

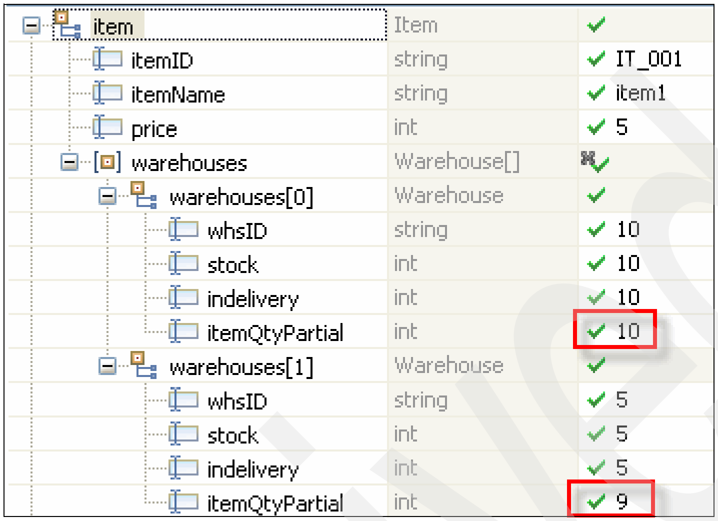


Figure 6-73 Warehouse A found unavailable, quantity split among Warehouse B and Warehouse C

> **Note:** By following instructions in this test section, you test all the mediation components except the export component that provides the Web service access to the mediation.
>
> To test the mediation starting with the Web service export, you have to select the WarehouseItemSplitIFExport1 component and complete the request input fields as shown in Figure 6-74.

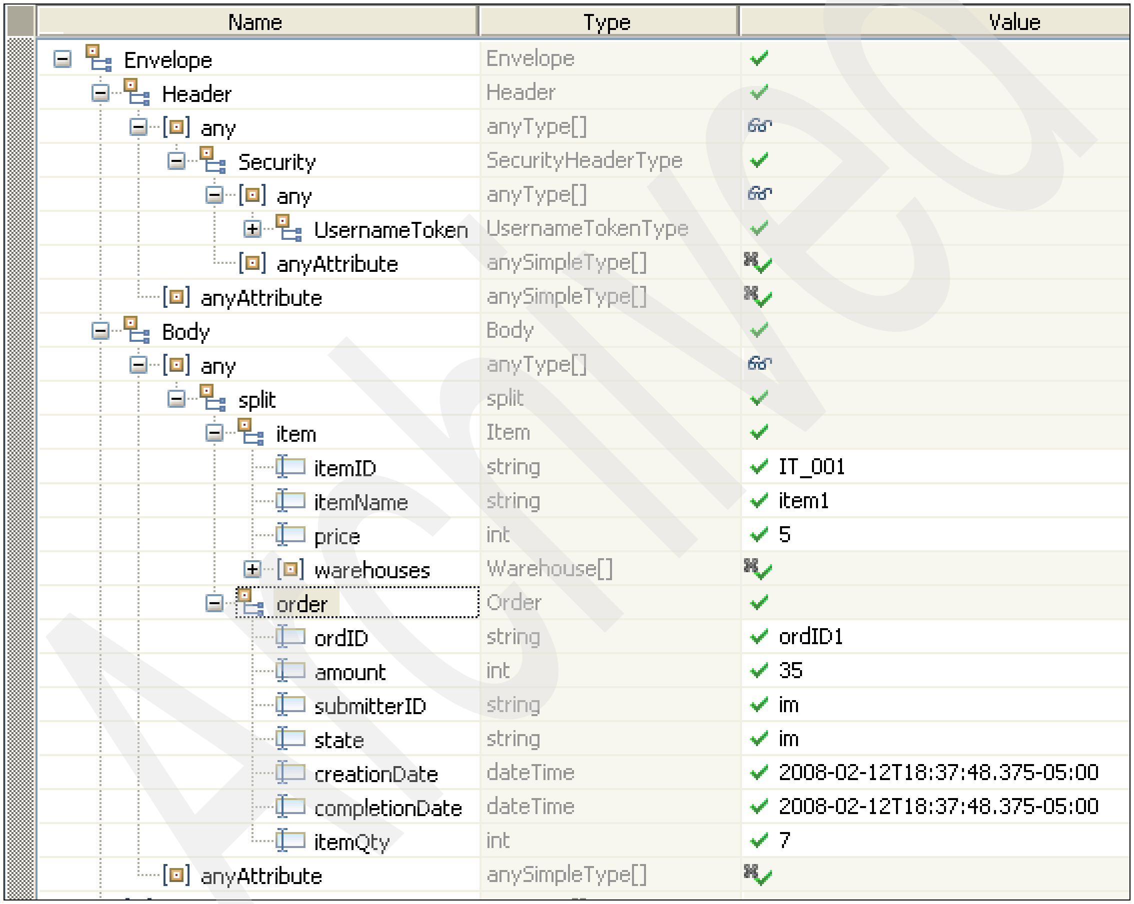


Figure 6-74 Testing WarehouseAvailabilityMediation accessing the Web service export

> **Team development:** Now is a good time to check in your project to CVS.

7

End-to-end test

This chapter shows how to combine the WebSphere Process Server and WebSphere ESB components and how to run the scenario from end-to-end in the WebSphere Integration Developer test environment.

The test cases that we run in this chapter are a special order and four variations of a regular order. The four variations of a regular order are:

- Automatic approval
- Automatic rejection
- Human approval
- A warehouse split

This chapter includes the following topics:

- Preparing to deploy
- Combining the components
- End-to-end testing
- End-to-end testing of an imported project

7.1 Preparing to deploy

In WebSphere Integration Developer, you set up the supporting structure, including the users and groups for human approval and the e-mail client and server as you developed the end-to-end scenario from a business integration perspective. As you combine the WebSphere Process Server and WebSphere ESB components, you also need to add the WebSphere ESB supporting structure, including the ORDERDB database, the warehouse Web services, and the file system for special orders.

This section describes these preparation steps. The sections that follow describe the end-to-end test of the combined WebSphere Process Server and WebSphere ESB components in the WebSphere Integration Developer stand-alone server test environment.

7.1.1 Test data

In 2.9, "Obtain or agree upon the sample data" on page 42, the test data that is used to emulate different interactions between components is defined. For end-to-end testing, we focus mainly on the entry point of the process, which is the OrderPreProcessingMediation module that accepts regular and special orders.

In addition to the four regular order scenarios (APPROVE, REJECT, HUMAN, and ITEM SPLIT), we focused on when developing the business integration application, we now need a case for Special Orders.

This section includes the sample data for each use case.

Additional material: XML files with these test case values are included in the additional material available for download in the Sample Data folder. See Appendix B, "Additional material" on page 461.

Regular Order: Approve

Figure 7-1 shows the pre-processing input for Regular Order: Approve.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 15
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 7-1 Sample pre-processing input for Regular Order: Approve

This sample data is stored in the following files in the additional material:

- To test from OrderManagementBP: Sample Data\orderManagementInput_APPROVE.xml
- To test from OrderPreProcessingMediation: Sample Data\orderManagementInputSb_APPROVE.xml

Regular Order: Warehouse split

Figure 7-2 shows the pre-processing input for Regular Order: Warehouse split.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10001
itemID	string	✔ IT_001
itemQty	int	✔ 19
submitterID	string	✔ user1
submitterEmail	string	✔ user1@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ false

Figure 7-2 Sample pre-processing input for Regular Order: Warehouse split

This sample data is stored in the following files in the additional material:

- To test from OrderManagementBP: Sample Data\orderManagementInput_SPLIT.xml
- To test from OrderPreProcessingMediation: Sample Data\orderManagementInputSb_SPLIT.xml

Regular Order: Reject

Figure 7-3 shows the pre-processing input for Regular Order: Reject.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10002
itemID	string	✔ IT_002
itemQty	int	✔ 10
submitterID	string	✔ user2
submitterEmail	string	✔ user2@kcg16hw.itso.ral.ibm.com

Figure 7-3 Sample pre-processing input for Regular Order: Reject

This sample data is stored in the following files in the additional material:

- To test from OrderManagementBP: Sample Data\orderManagementInput_REJECT.xml
- To test from OrderPreProcessingMediation: Sample Data\orderManagementInputSb_REJECT.xml

Regular Order: Human

Figure 7-4 shows the pre-processing input for Regular Order: Human.

Name	Type	Value
orderManagementInput	OrderManagementInput	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com

Figure 7-4 Sample pre-processing input for Regular Order: Human

This sample data is stored in the following files in the additional material:

- To test from OrderManagementBP: Sample Data\orderManagementInput_HUMAN.xml
- To test from OrderPreProcessingMediation: Sample Data\orderManagementInputSb_HUMAN.xml

Special Order

Figure 7-5 shows the pre-processing input for Special Order.

Name	Type	Value
orderManagementInputSb	OrderManagementInputSb	✔
custID	string	✔ 10003
itemID	string	✔ IT_003
itemQty	int	✔ 20
submitterID	string	✔ user3
submitterEmail	string	✔ user3@kcg16hw.itso.ral.ibm.com
isSpecial	boolean	✔ true

Figure 7-5 Sample pre-processing input for Special Order

This sample data is stored in the Sample Data\orderManagementInputSb_SPECIAL.xml file in the additional material.

7.1.2 Defining the users and groups for human tasks to the server

The users and groups that are required by the human task are defined using the server's administrative console. Select:

- **Users and Groups** → **Manage Users** to add new users.
- **Users and Groups** → **Manage Groups** to add new groups and add members to groups.

See 3.5.4, "Configuring users and groups in the integrated test environment" on page 168 for instructions about setting up the users and groups for the human tasks in the scenario.

7.1.3 Defining the e-mail client and server to the module

The scenario requires that you have an e-mail client and server to read and send e-mails. The e-mail adapter assumes that you are using an e-mail server on `localhost` and listening for SMTP protocol on port `25` with user name and password `admin/admin`. If you are not using these settings, follow these instructions to change the e-mail server's host name, port, and protocol:

1. Open the EmailOutbound module, and open its assembly diagram.
2. Click **EmailImport** and go to the Binding tab in the Properties view.
3. Modify as needed the host name, port number, and protocol fields.

4. If you need to change the user name and password from the current values of `admin/admin`, then click **Advanced >>**. Modify the user name and password fields as needed.
5. Save your changes

7.1.4 Creating the database and defining it to the server

The DBMSServiceMediation provides access to the ORDERDB database. You must create this database and define a data source for it in the server. These actions occur during the test phase for the DBMSServiceMediation.

ORDERDB is a Derby database. It is created at the following location using the instructions in Appendix A, “Creating the ORDERDB Derby database” on page 453:

C:\ITSO\sampleDB\ORDERDB

The data source is created from the server’s administrative console. You can find instructions for this tasks in 5.3, “Defining the database runtime resources” on page 319.

7.1.5 Deploying the warehouse Web service applications to the server

The scenario uses three Web service applications that act as the warehouses that supply the items. For information about how these mediations are created, see 6.2, “Preparing the Warehouse Web services” on page 358.

To make these services available, the following applications are deployed to the server.

- WarehouseAvailabilityServiceAApp
- WarehouseAvailabilityServiceBApp
- WarehouseAvailabilityServiceCApp

These applications are deployed from the Servers view as follows:

1. Select the server, right-click, and select **Add and Remove Projects**.
2. Select each of the three warehouse service applications, and click **Add >** to move them from the Available projects window to the Configured projects window.
3. Then, click **Finish**.

7.1.6 Creating the file system for special orders

The OrderPreProcessing mediation uses a file system to store messages for special orders. The C:\itsoSpecialOrders folder must be created on the server's operating system. If you have not already done so, create a folder C:\itsoSpecialOrders.

7.2 Combining the components

In this section, we describe how to bring in each additional WebSphere ESB component incrementally and test it with the WebSphere Process Server components.

As shown in the component diagram in Figure 7-6 on page 422, you can see immediately the three mediations (WebSphere ESB components) that you need to add to the WebSphere Process Server components. The OrderManagement system and the two rules were implemented in the OrderManagement module, while the Email module, which is another WebSphere Process Server component, is integrated already.

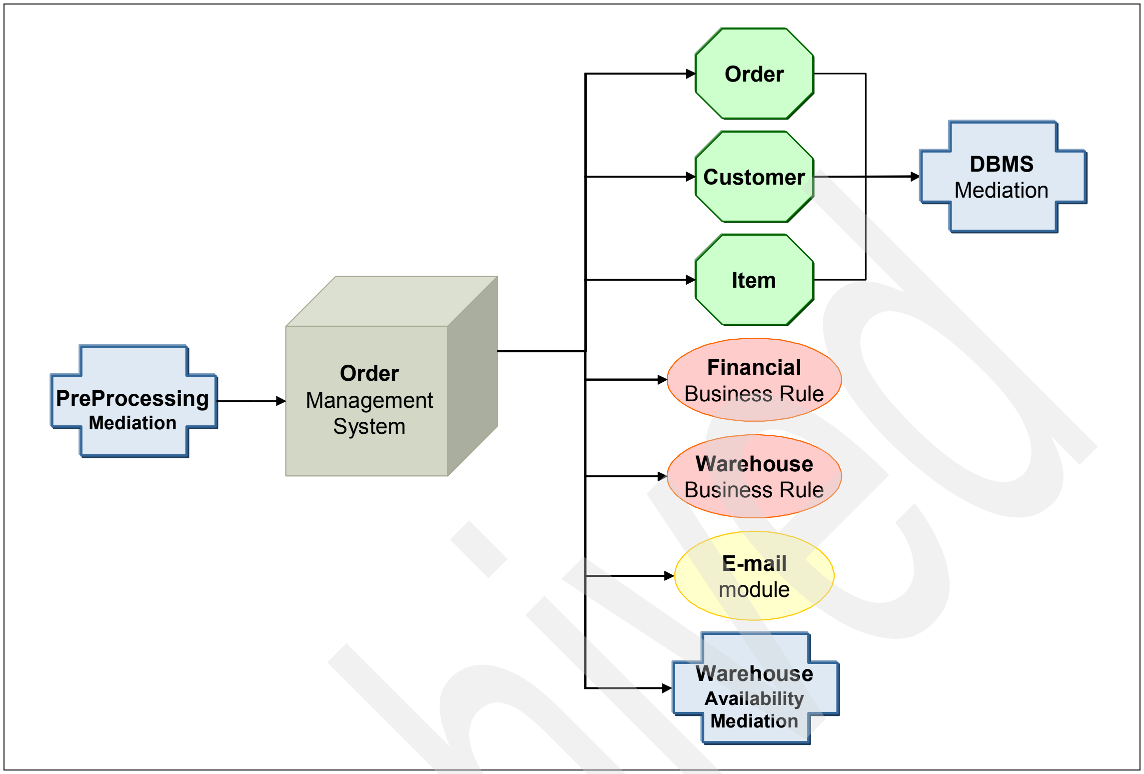


Figure 7-6 Component diagram of the OrderManagement system

Figure 7-6 shows us to how to combine the WebSphere Process Server components incrementally. To the WebSphere Process Server components, you add successively:

- The DBMSServiceMediation
- The WarehouseAvailabilityMediation
- The OrderPreProcessingMediation

Finally, you need to export the EAR files so that they are available for deployment to a WebSphere Process Server environment outside of the WebSphere Integration Developer test environment.

7.2.1 Adding DBMSServiceMediation

In this section, you add the DBMSServiceMediation to the WebSphere Process Server components and then test them together. The order, customer, and item services are implemented in the DBMSServiceMediation. You integrate the mediation with the WebSphere Process Server components by:

1. Setting up a workspace with all the components.
2. Verifying or creating the DBMSServiceMediation export components.
3. Verifying and updating your test environment deployment port.
4. Adding the respective imports to the OrderManagement assembly diagram.

> **Prerequisites:** This mediation uses the ORDERDB database. This database must be available and defined to the server.

Populating the workspace with all components

The workspace must include all the WebSphere Process Server components, plus the DBMSServiceMediation. In a team development environment, each component has been stored in the repository.

To populate the workspace with all components:

1. Import the following modules and libraries that you need for this phase of testing from CVS or from a project interchange file that is supplied by the developer (as shown in Figure 7-7):
   - OrderManagementLib
   - OrderManagement
   - EmailOutbound and its associated adapter project CWYEM_Email
   - DBMSServiceMediation and its associated adapter project CWYBC_JDBC.

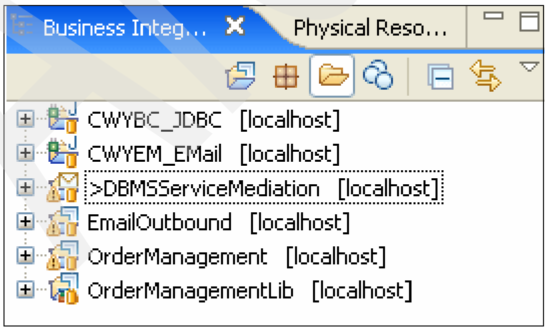


Figure 7-7 Business Integration view with the required components

Verifying the DBMSServiceMediation export components

To verify the DBMSServiceMediation export components, follow these steps:

1. Open the DBMSServiceMediation assembly diagram (Figure 7-8).

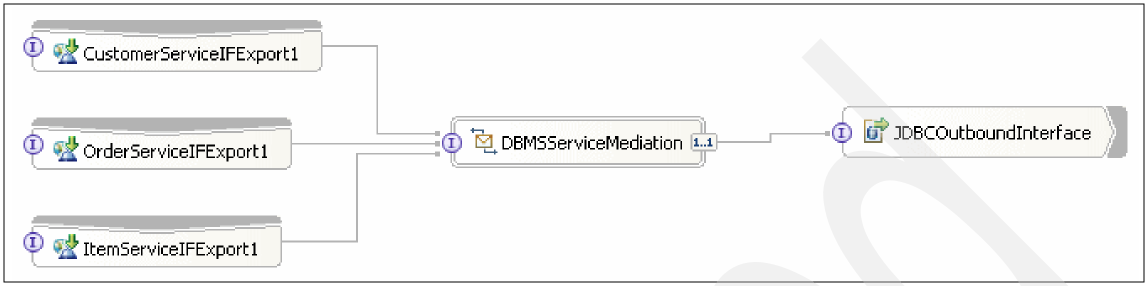


Figure 7-8 DBMSServiceMediation assembly diagram

2. If the three export components do not display in the assembly diagram, you need to generate them as follows:
   a. RIght-click DBMSServiceMediation and select **Generate Export** → **Web Service Binding**.
   b. Click **OK**.

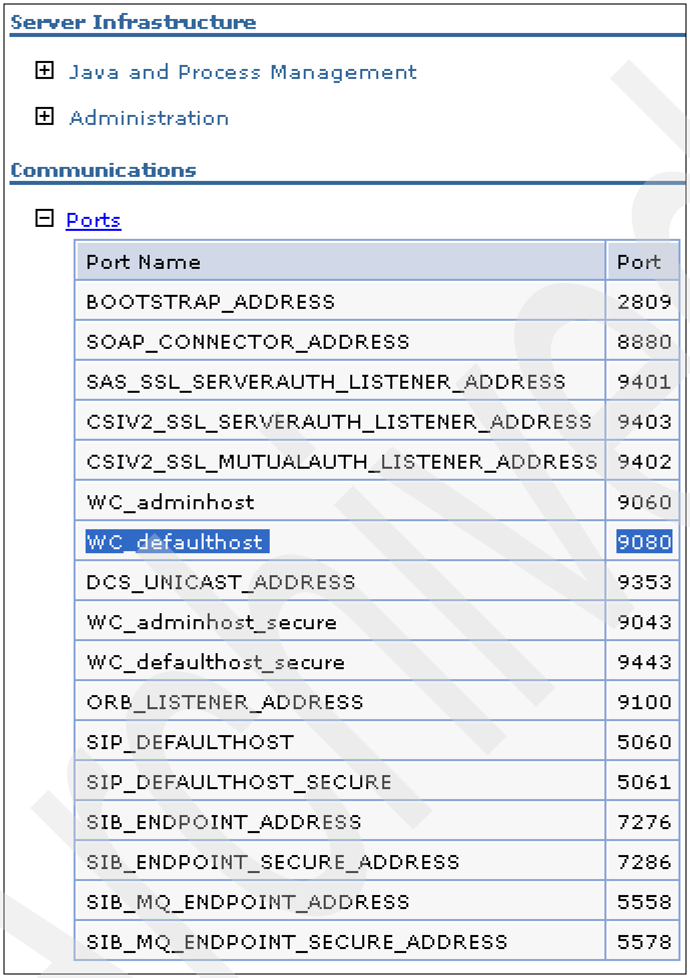
      The three new export components displays as depicted in Figure 7-8.
   c. Save the assembly diagram.
3. If the three export components are created already when you open the assembly diagram, you might need to verify, and perhaps update, their ports prior to creating their import components. We explain how to do this in the next section.

Verifying your test environment deployment port

To verify your test environment deployment port, follow these steps:

1. Select the server in the Servers view, right-click, and select **Run administrative console**. Log in with user ID and password `admin/admin` (the default).
2. Select **Servers** → **Application servers**.

3. Double-click **server1** to open its configuration page. Then, scroll down to the Communications section and expand Ports. Find wc_defaulthost and make a note of its port value. In this case, 9080, as shown in Figure 7-9.



Port Name	Port
BOOTSTRAP_ADDRESS	2809
SOAP_CONNECTOR_ADDRESS	8880
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9401
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9403
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9402
WC_adminhost	9060
WC_defaulthost	9080
DCS_UNICAST_ADDRESS	9353
WC_adminhost_secure	9043
WC_defaulthost_secure	9443
ORB_LISTENER_ADDRESS	9100
SIP_DEFAULTHOST	5060
SIP_DEFAULTHOST_SECURE	5061
SIB_ENDPOINT_ADDRESS	7276
SIB_ENDPOINT_SECURE_ADDRESS	7286
SIB_MQ_ENDPOINT_ADDRESS	5558
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578

Figure 7-9 wc_defaulthost port value

Updating the Web service ports if necessary

To update the Web service ports, follow these steps:

1. In the Business Integration view, expand **OrderManagementLib** → **Web Service Ports**.
2. Right-click the customer service port, CustomerServiceIFExport1_CustomerServiceIFHttp_Servicem, and select **Open With** → **WSDL Editor**.

3. The interface opens in the WSDL editor. If you are not in the Design pane when it opens, switch to it. Look at the Web service URL and if necessary change the port to the port that you identified previously (Figure 7-10).

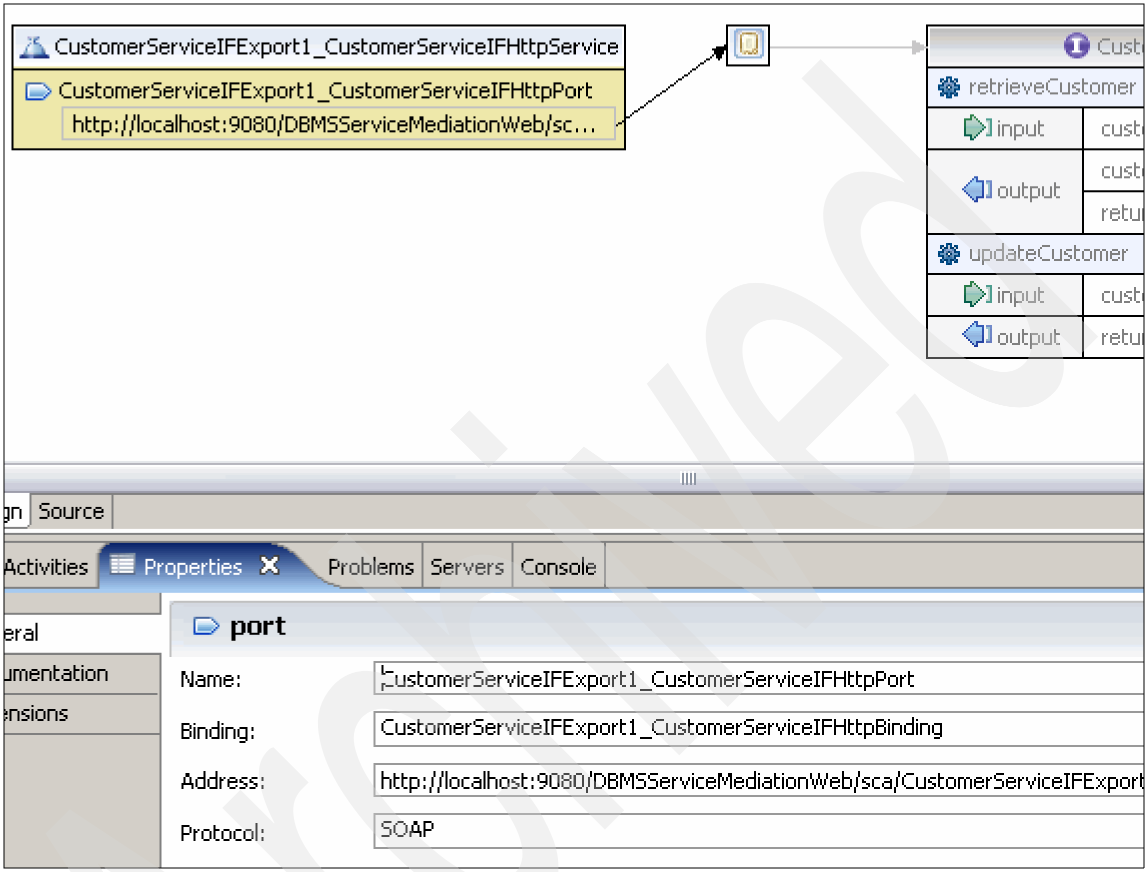


Figure 7-10 Web service port in the WSDL editor

4. Save the changes.
5. Copy the corrected CustomerService Web service URL, http://localhost:9080/DBMSServiceMediationWeb/sca/CustomerServiceIFExport1, to a text document.
6. Repeat these steps for the item service (ItemServiceIFExport1_ItemServiceIFHttp_Service) and the order service (OrderServiceIFExport1_OrderServiceIFHttp_Service).

Testing the supporting Web services ports

To test the Web services port, follow these steps:

1. Deploy the DBMSServiceMediation to the server.
2. Open a Web browser, and test the URL for customer service in the browser.

   If you get an error, you might need to check the URL in the WSDL editor
3. Repeat similar tests to test the item services in a browser:

   `http://localhost:9080/DBMSServiceMediationWeb/sca/ItemServiceIFExport1`

   And order services in a browser:

   `http://localhost:9080/DBMSServiceMediationWeb/sca/OrderServiceIFExport1`
4. Remove the DBMSServiceMediation from the server.

Updating the assembly diagram's Web service ports

To update the assembly diagram's Web service port, follow these steps:

1. Open the DBMSServiceMediation assembly diagram.
2. If necessary, update the customer export component:
   a. On the DBMSServiceMediation assembly diagram, click **CustomerServiceIFExport1** and go to the Bindings tab of the Properties view.
   b. Verify the Address value and update the port if necessary, `http://localhost:9080/DBMSServiceMediationWeb/sca/CustomerServiceIFExport1`.
3. If necessary, update the item export component:
   a. On the DBMSServiceMediation assembly diagram, click **ItemServiceIFExport1** and go to the Bindings tab of the Properties view.
   b. Verify the Address value and update the port if necessary, `http://localhost:9080/DBMSServiceMediationWeb/sca/ItemServiceIFExport1`
4. If necessary, update the order export component:
   a. On the DBMSServiceMediation assembly diagram, click **OrderServiceIFExport1**and go to the Bindings tab of the Properties view.
   b. Verify the Address value and update the port if necessary, `http://localhost:9080/DBMSServiceMediationWeb/sca/OrderServiceIFExport1`

Now that these exports exist in the DBMSServiceMediation and are pointing to the correct ports, you can use them as import components in the

OrderManagement module to allow the OrderManagementBP to use the customer, item, and order services.

Adding the import components to the assembly diagram

To add the import components to the assembly diagram, follow these steps:

1. Open the OrderManagement assembly diagram (Figure 7-11).

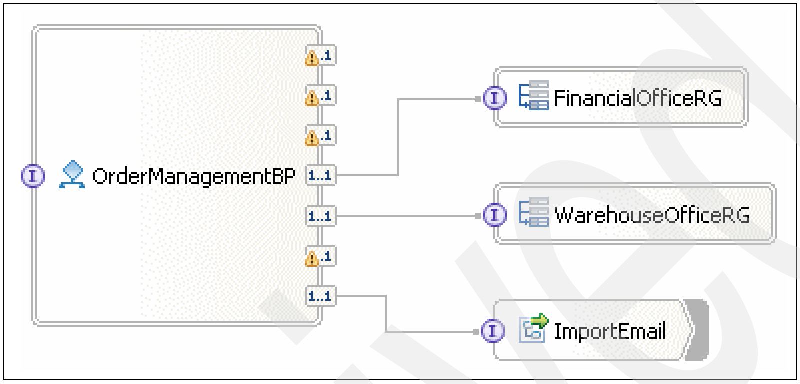


Figure 7-11 OrderManagement assembly diagram

2. In the Business Integration view, expand the DBMSServiceMediation and assembly diagram.
3. Drag CustomerServiceIFExport1 onto the OrderManagement Assembly diagram. If you are prompted for type of export, select **Import with Web Service Binding** and click **OK**.

4. Rename the import component `CustomerServiceImport` as shown in Figure 7-12.

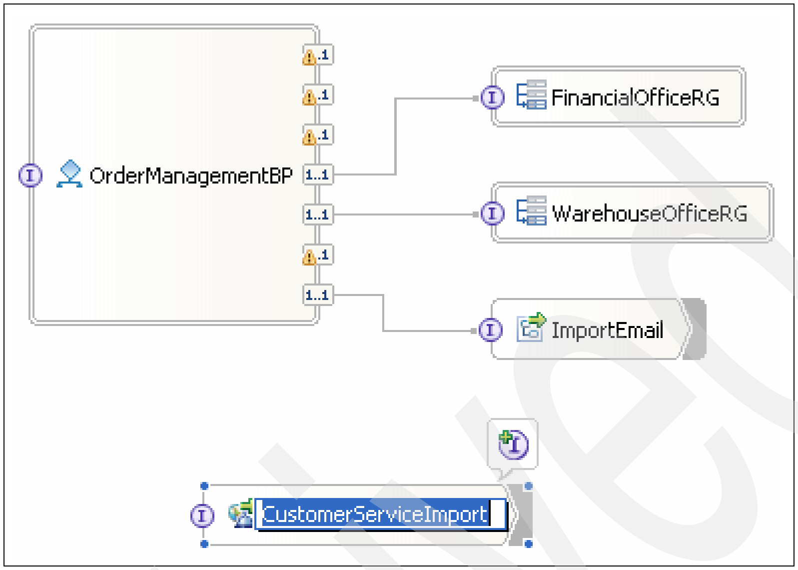


Figure 7-12 OrderManagement assembly diagram with customer import

5. Repeat similar steps for item and order so that the diagram looks as shown in Figure 7-13.

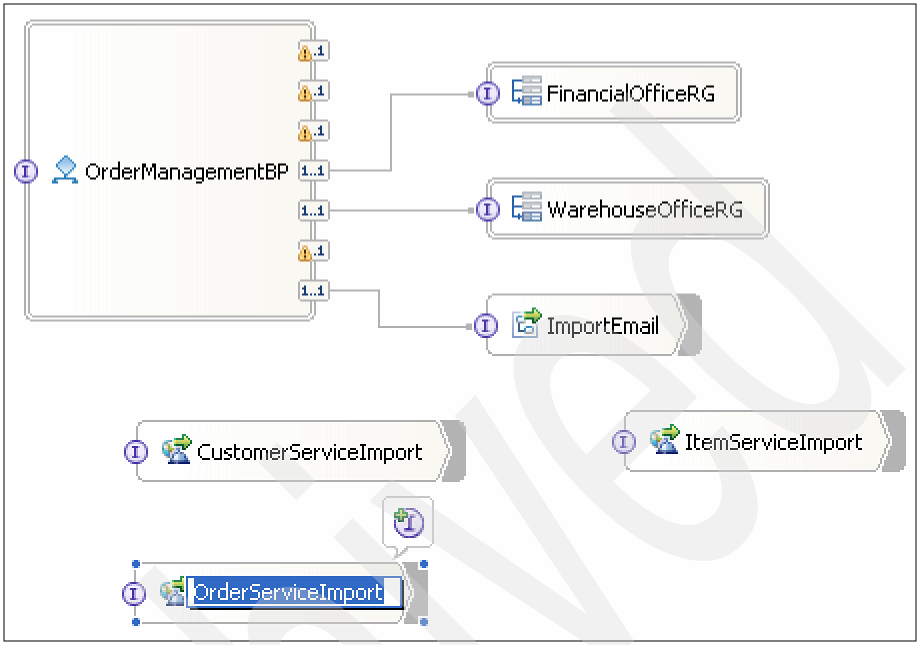


Figure 7-13 OrderManagement assembly diagram with customer, item, and order imports

6. Right-click **OrderManagementBP** and select **Wire to Existing**.
7. Verify that the OrderManagement assembly diagram looks similar to Figure 7-14.

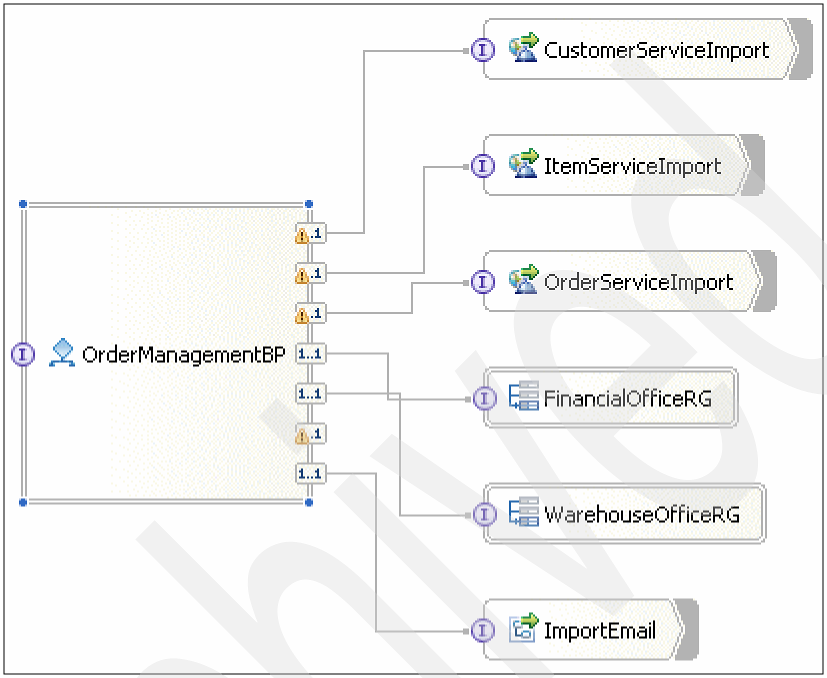


Figure 7-14 OrderManagementBP wired to customer, item, and order imports

8. Save the OrderManagement assembly diagram. There are no errors in the Problems view.

You have completed integrating the DBMSServiceMediaton with the WebSphere Process Server components. In the next step, you test them together.

Integrating DBMSServiceMediation

Before you test the DBMSServiceMediaton with the WebSphere Process Server components, we explain a workaround that you must perform. We then show how to test all components.

Important: Due to an issue in the Component Tester, you need to perform a workaround anytime that you want to test the DBMSServiceMediation with the WebSphere Process Server components. This bug is reported as PMR02321,758,758.

Running the component test work-around for DBMSServiceMediation

To test the DBMSServiceMediation with other components, you must perform the following prerequisite steps. These steps do not apply to testing in a server outside the integrated test environment. Follow these steps:

1. Remove all projects from the test environment.
2. Deploy the DBMSServiceMediation to the test environment.
3. Open the DBMSServiceMediation assembly diagram. Right-click **JDBCOutboundInterface** and select **Test Component**.
4. Select the **retrievelallDbadminCustomerBG** operation.
5. In the initial request parameters section, unset all attributes by selecting all the DbadminCustomer attributes. Right-click, and select **Set To** → **Unset** as shown in Figure 7-15.

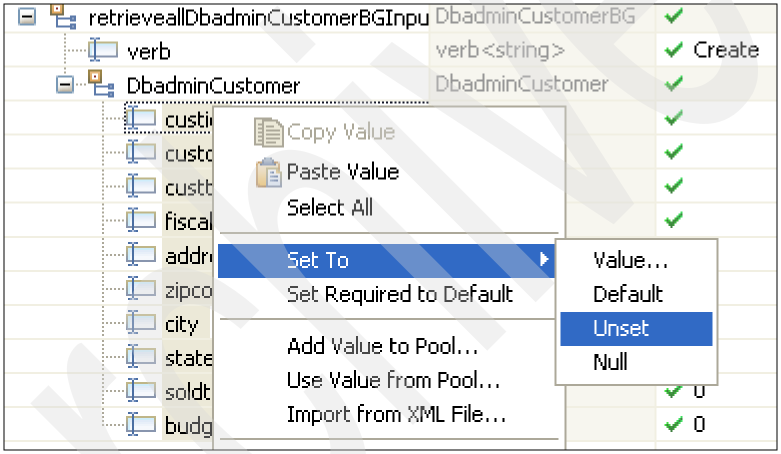


Figure 7-15 Unsetting all DBadminCustomer attributes

6. Verify that your settings look as shown in Figure 7-16.

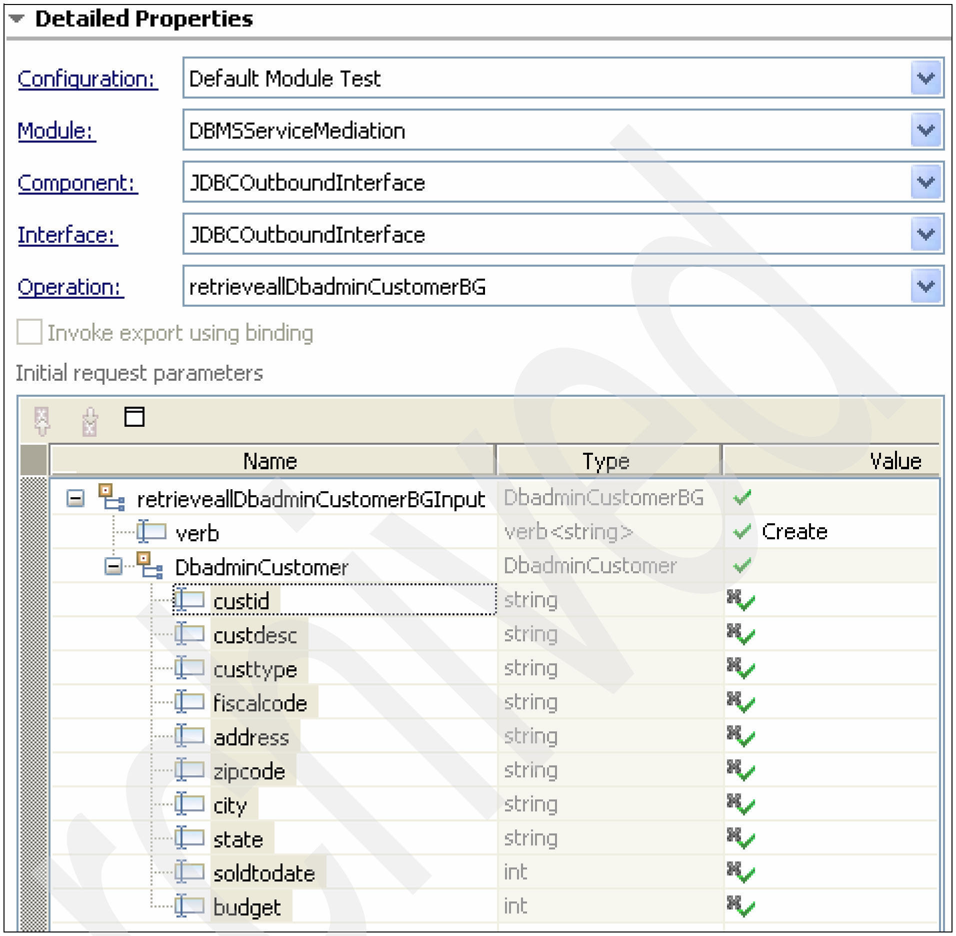


Figure 7-16 retrieveallDbadminCustomerBG settings for testing

7. Click the **Continue** icon to run the test.

This test retrieves a list of all the customers in the database. If errors occur during the test, ensure that you have correctly defined the database to the runtime environment (see 5.3, “Defining the database runtime resources” on page 319).

Testing DBMSServiceMediation with other components

You can now deploy the additional modules and then test as follows:

1. Use **Add and Remove Projects** to deploy the OrderManagement and the EmailOutbound modules.
2. Use the APPROVE test case values to test the components as shown in Figure 7-17.

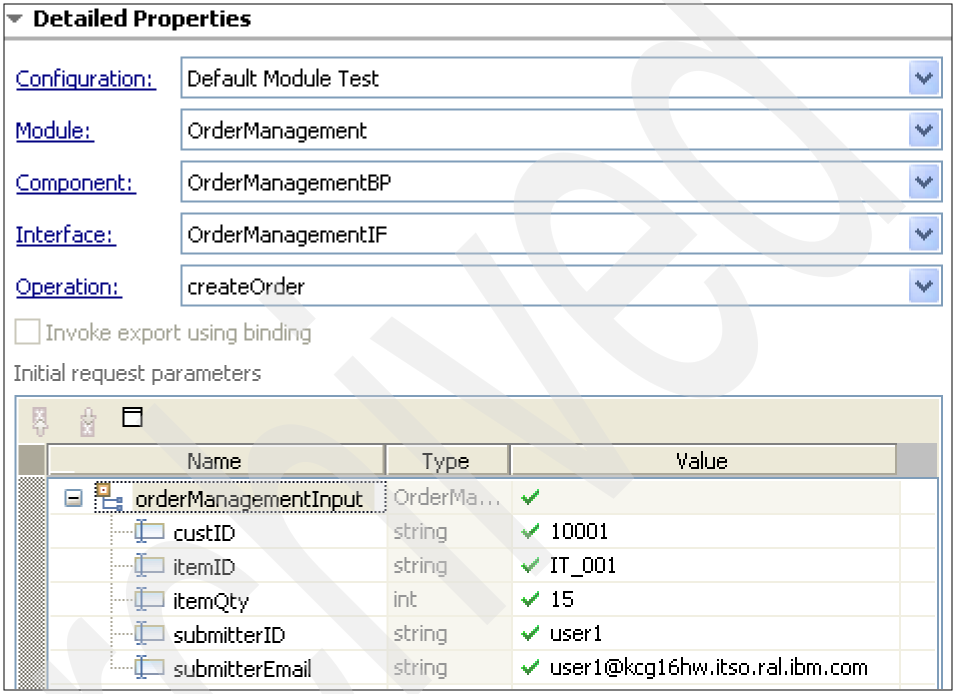


Figure 7-17 Test DBMSServiceMediation and OrderManagement using the approve test case

3. Verify that only the WarehouseItemSplit is emulated as shown in Figure 7-18.

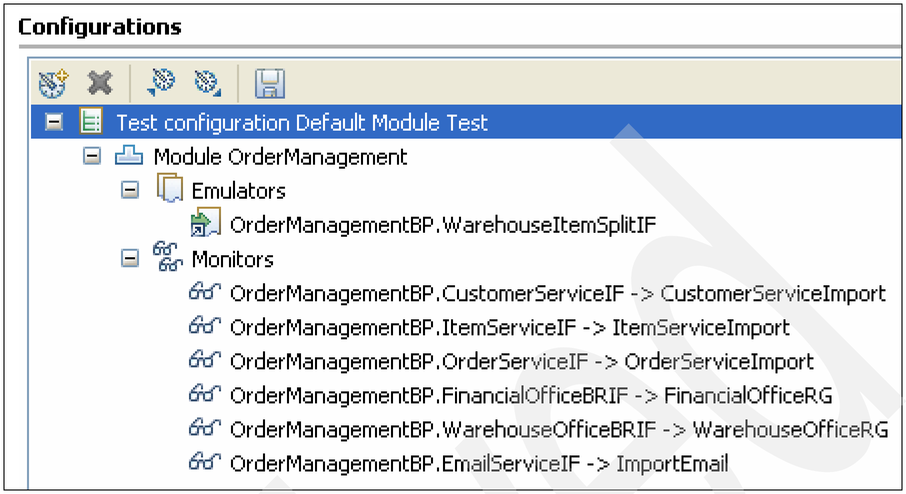


Figure 7-18 Test DBMSServiceMediation and OrderManagement emulation values

4. You receive an e-mail verifying that the order was placed.
5. You could also use the JDBCOutboundInterface in the DBMSServiceMediation to test retrieveallDbadminOrderheaderBG and retrieve at least one order.

This concludes the DBMServiceMediation and OrderManagement integration and testing.

7.2.2 Adding WarehouseAvailabilityMediation

You next integrate the WarehouseAvailabilityMeditaion with the WebSphere Process Server components by:

1. Setting up a workspace with all the components.
2. Updating and testing the supporting Web service ports.
3. Testing the supporting Web service ports.
4. Updating the WarehouseAvailabilityMediation assembly diagram to point to the updated ports.
5. Verifying or updating the WarehouseAvailabilityMediation port.
6. Adding the warehouse availability import components to the OrderManagement assembly diagram.
7. Testing the WebSphere Process Server components with DBMSServiceMediation and WarehouseAvailabilityMediation.

Setting up the workspace the with all components

In the same workspace where you have the WebSphere Process Server components and DBMSServiceMediation, add the WarehouseAvailabilityMediation components by importing them from CVS or a project interchange file that is supplied by the mediation developer. You need the following components in the workspace:

- OrderManagementLib
- OrderManagement
- EmailOutbound and associated adapter project CWYEM_Email
- DBMSServiceMediation and associated adapter project CWYBC_JDBC
- WarehousesLib
- WarehouseAvailabilityServiceA
- WarehouseAvailabilityServiceB
- WarehouseAvailabilityServiceC
- WarehouseAvailabilityMeditaion.

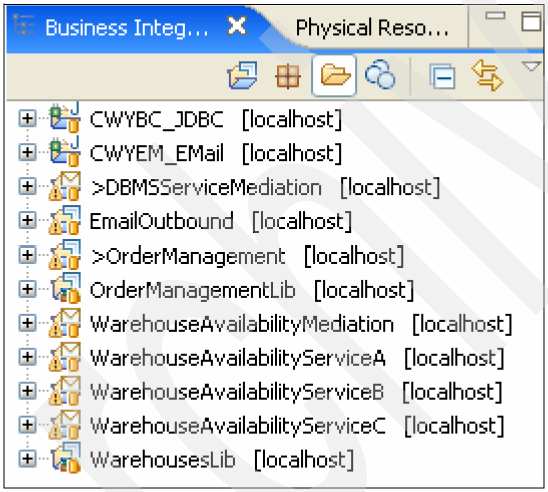


Figure 7-19 Business Integration view with WebSphere Process Server components, DBMSServiceMediation, and WarehouseAvailabilityMediation

Updating and testing the Web services ports

To update and test the Web services ports:

1. In the Business Integration view, expand **WarehousesLib** → **Web Service Ports**.
2. Right-click the first port and select **Open With** → **WSDL Editor** as shown in Figure 7-20 on page 437.

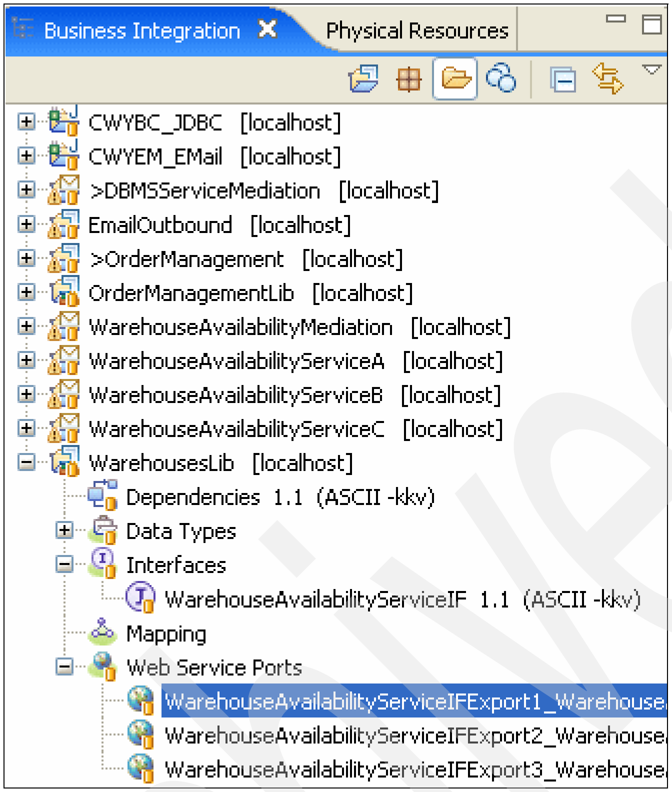


Figure 7-20 WarehousesLib Web service ports

3. The interface opens in the WSDL editor. If you are not in the Design pane, switch to it.
4. Look at the Web service URL, and change the port to the port that you identified previously, `9080`, as shown in Figure 7-21.

Figure 7-21 WarehouseAvailabilityServiceA Web service URL

5. Save the changes.
6. Copy the corrected WarehouseAvailabilityServiceA Web service URL, `http://localhost:9080/WarehouseAvailabilityServiceAWeb/sca/WarehouseAvailabilityServiceIFExport1`, to a text document.

7. Repeat these steps for WarehouseAvailabilityServiceB and WarehouseAvailabilityServiceC Web services.
8. Deploy WarehouseAvailabilityServiceA, WarehouseAvailabilityServiceB, and WarehouseAvailabilityServiceC to the test environment.
9. Open a Web browser and enter the URL for WarehouseAvailabilityServiceA. You see a message similar to that shown in Figure 7-22

Figure 7-22 WarehouseAvailabilityServiceA Web service test from Web browser

10. If you get an error, check the URL in the WSDL editor.
11. Repeat similar tests to test the WarehouseAvailabilityServiceB and WarehouseAvailabilityServiceC in a browser.
12. Undeploy the Web services.

Updating the assembly diagram

Now that you have tested the supporting Web services, you can update the WarehouseAvailabilityMediation bindings to point to the correct Web service ports. The WarehouseAvailabilityMediation invokes each of these Web services.

Update WarehouseAvailabilityMediation assembly diagram to point to updated supporting Web services' ports:

1. Open the WarehouseAvailabilityMediation assembly diagram.
2. Click **GetWarehouseA_Availability**.
3. Go to the Binding tab of the Properties view.
4. Update the Address to point to appropriate port (for example, 9080) as shown in Figure 7-23 on page 439.

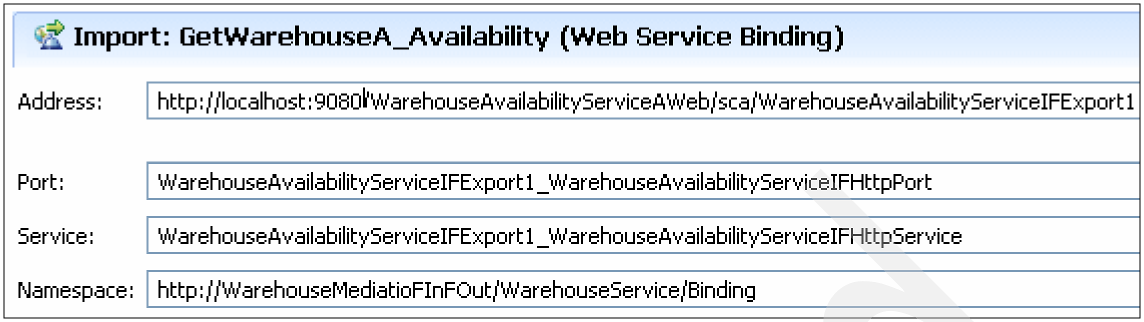


Figure 7-23 GetWarehouseA_Availability Binding

5. Repeat similar steps to update GetWarehouseB_Availability and GetWarehouseC_Availability bindings.
6. Save the assembly diagram.

Updating and testing the WarehouseAvailabilityMediation port

To update and test the port:

1. In the Business Integration view, expand **OrderManagementLib** → **Web Service Ports**.
2. Right-click the WarehouseAvailabilityMediation port and select **Open With** → **WSDL Editor**.
3. The interface opens in the WSDL editor. If you are not in the Design pane, switch to it.
4. Look at the Web service URL and change the port to the port that you identified previously.
5. Save the changes.
6. Copy the corrected WarehouseAvailabilityMediation Web service URL, `http://localhost:9080/WarehouseAvailabilityMediationWeb/sca/WarehouseItemSplitIFExport1`, to a text document.
7. To test the WarehouseAvailabilityMediation Web service URL deploy WarehouseAvailabilityMediation to the test environment.
8. Open up a Web browser.
9. Test the URL for WarehouseAvailabilityMediation in the browser.
10. If you get an error, check the URL in the WSDL editor.
11. Undeploy the WarehouseAvailabilityMediation.

Adding the warehouse availability import components

Follow these steps:

1. Open the OrderManagement assembly diagram.
2. In the Business Integration view, expand the WarehouseAvailabilityMediation assembly diagram.
3. Drag **WarehouseItemSplitIFExport1** onto the OrderManagement assembly diagram. If you are prompted for type of export, select **Import with Web Service Binding** and click **OK**.
4. Right-click **OrderManagementBP** and select **Wire to Existing**.
5. Verify that the OrderManagement assembly diagram looks as shown in Figure 7-24.

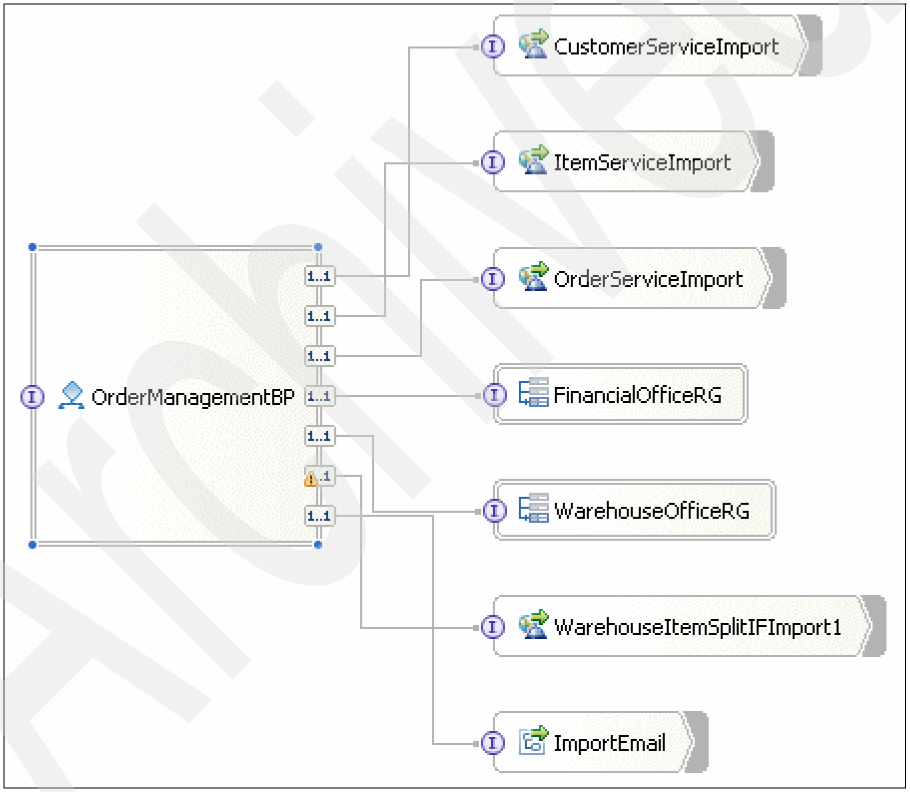


Figure 7-24 OrderManagementBP wired to all import components

6. Save the OrderManagement assembly diagram. There are no errors in the Problems view.

Testing the components together

You have completed integrating the WarehouseAvailabilityMediation with the WebSphere Process Server components. Now, you test the WebSphere Process Server components with DBMSServiceMediation and WarehouseAvailabilityMediation. Follow these steps:

1. Perform the workaround for DBMSServiceMediation.
2. Deploy EmailOutbound, WarehouseAvailabilityServiceA, WarehouseAvailabilityServiceB, WarehouseAvailabilityServiceC, WarehouseAvailabilityMediation, and OrderManagement to the test server.
3. Use the APPROVE test case to test the components as shown in Figure 7-25.

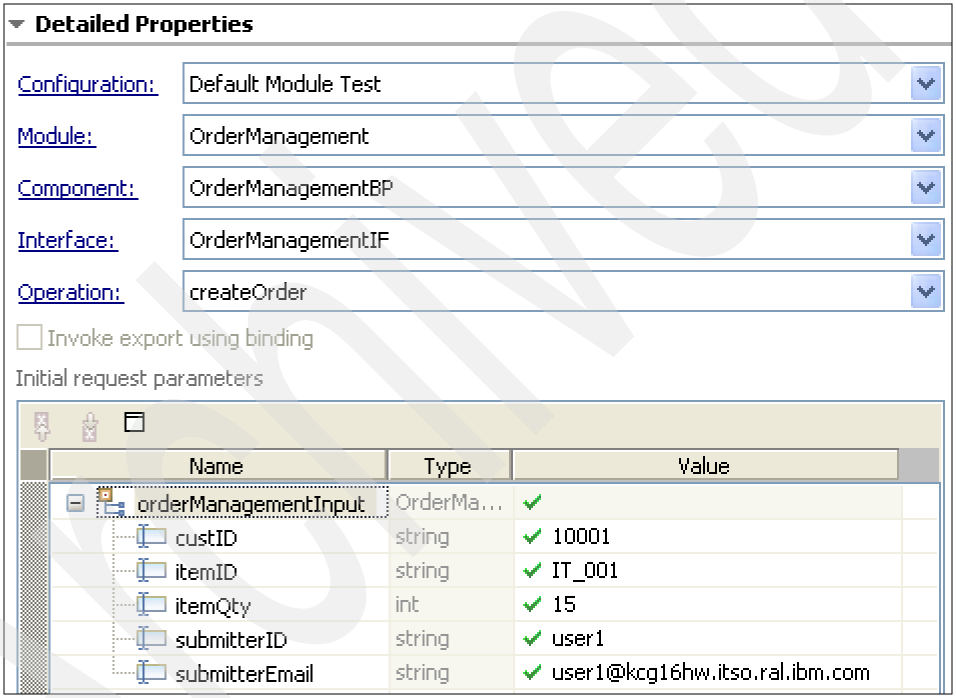


Figure 7-25 Test OrderManagement using the APPROVE test case

4. Verify that there are no emulators set because you specified all emulators. If you have an emulator set, remove it.
5. You receive an e-mail that confirms that the order was placed. Verify that the warehouse split returned three available warehouses.
6. Use similar steps to test the SPLIT test case (`itemQty = 19`). You receive an e-mail confirmation, and you can verify that the warehouse split returned only two available warehouses.

The only mediation that you have not integrated is the OrderPreProcessing mediation. We explain how to integrate it in the next section.

7.2.3 Adding OrderPreProcessing mediation

As shown in the Component diagram of the OrderManagement system (Figure 7-6 on page 422), all components are integrated except the OrderPreProcessing mediation. To integrate it, you generate an export component of the OrderManagementBP and then use it as an import component in the OrderPreProcessing mediation module.

In this section, we explain the following steps:

- Setting up the workspace with all components
- Generating the OrderManagementBP export component
- Updating OrderPreProcessing mediation
- Testing with an emulation for OrderManagement

Setting up the workspace with all components

In the same workspace, add the OrderPreProcessing mediation. You now have the projects shown in Figure 7-26.

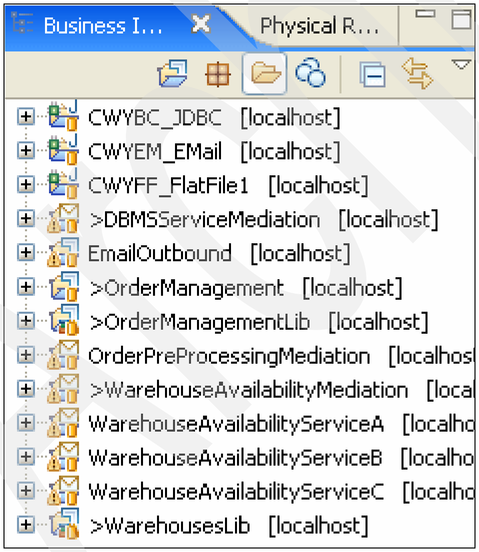


Figure 7-26 Business Integration view with all components

Note: If you have other modules in the workspace, you might want to export this workspace and then delete any additional modules (for example, OrderManagementProcessEm). If you leave additional clients that are used when developing the WebSphere ESB components, you might have errors due to WSDL or import component collisions.

Generating the OrderManagementBP export component

To generate the OrderManagement export component, follow these steps:

1. Open the OrderManagement assembly diagram.
2. Right-click OrderManagementBP and select **Generate Export...** → **Web Service Binding**.
3. Save the assembly diagram.
4. Verify the final OrderManagement assembly diagram looks as shown in Figure 7-27.

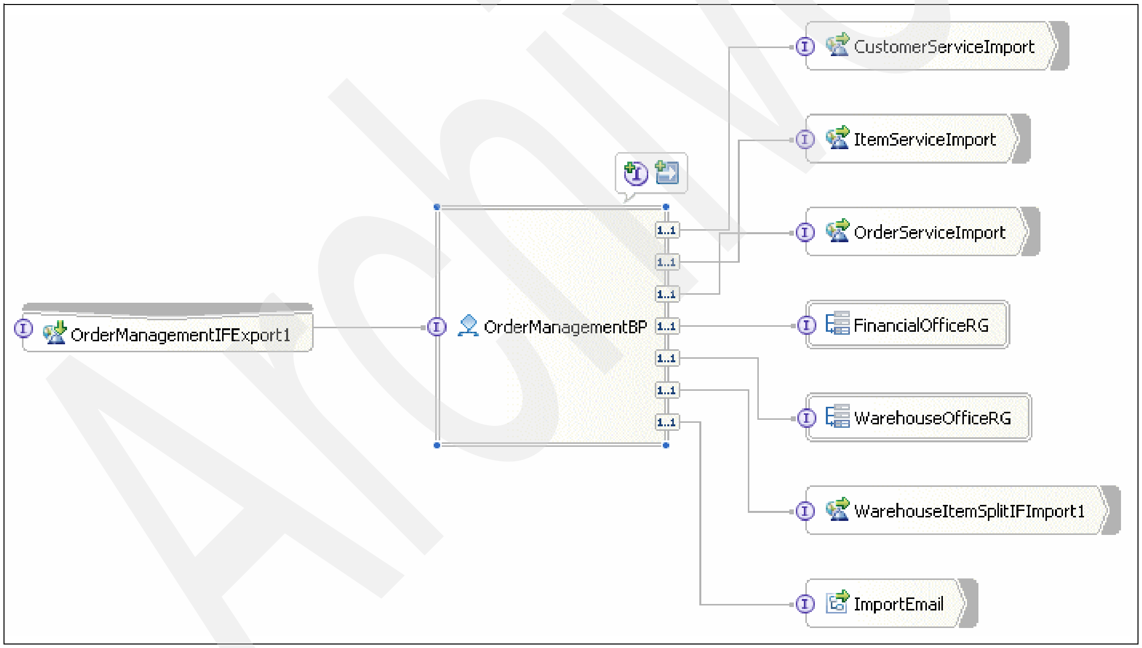


Figure 7-27 OrderManagementBP with all import and export components

Updating OrderPreProcessing mediation

To update the OrderPreProcessing mediation, follow these steps:

1. Open the OrderPreProcessing assembly diagram as shown in Figure 7-28.

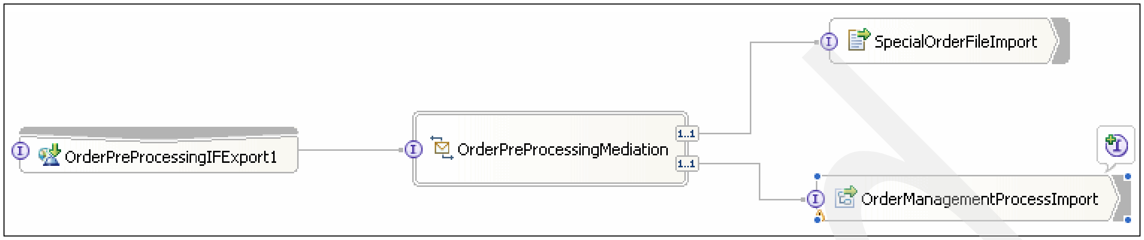


Figure 7-28 OrderPreProcessing assembly diagram

2. Right-click **OrderManagementProcessImport** and select **Delete**.
3. Expand the **OrderManagementLib** → **Web Service Ports** in the Business Integration view.
4. Drag **OrderManagementIFExport1** onto the OrderPreProcessing assembly diagram. If you are prompted for the type of component to create, select **Import with Web Service Binding**, then click **OK.**
5. Rename the import component `OrderManagementImport` as shown in Figure 7-29.

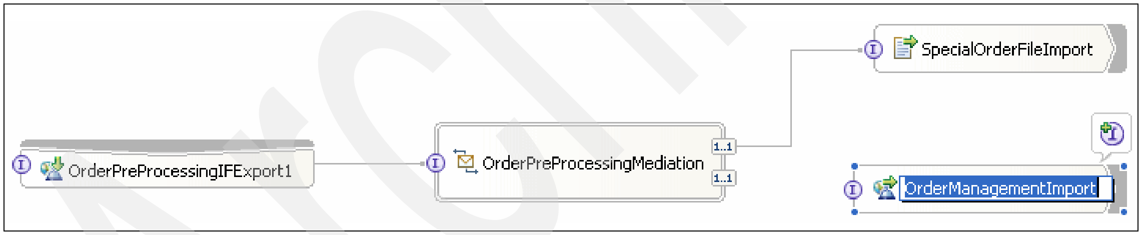


Figure 7-29 OrderPreProcessing assembly diagram

6. Right-click **OrderPreProcessingMediation** and select **Wire to Existing**.

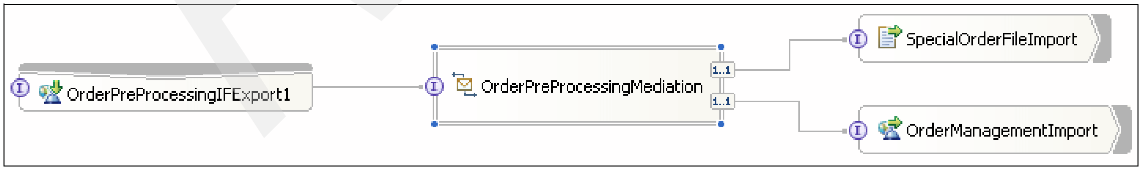


Figure 7-30 OrderPreProcessing assembly diagram

7. Click OrderPreProcessingIFExport1, and go to the Binding tab of the Properties view.
8. Verify the Address value and update the port if necessary, `http://localhost:9080/OrderManagementWeb/sca/OrderManagementIFExport1`.
9. Save the OrderPreProcessing assembly diagram. No errors display in the Problems view.

Testing with an emulation for OrderManagement

Now, that you have integrated the OrderPreProcessing module, you can test from end-to-end. However, you leave that for an upcoming section and instead only look at two test cases (regular order for approval and a special order) and emulate the OrderManagement module. Follow these steps:

1. Make sure that you have created the c:\itsoSpecialOrders folder.
2. Open the OrderPreProcessing assembly diagram, right-click an empty area of the assembly diagram, then select **Test Module**.
3. Set up an emulator for the OrderManagementImport as follows:
   a. Switch to the Configurations tab.
   b. Select **Emulators**, right-click and select **Add** → **Emulator** as shown in Figure 7-31.

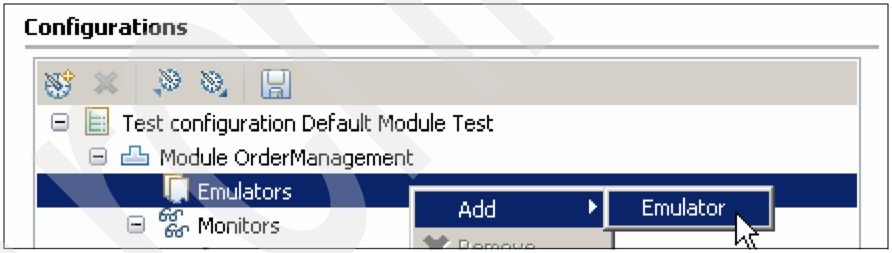


Figure 7-31 Add an emulator

4. Select **Module OrderPreProcessingMediation**. Click **Next**. Select **Components** and select **OrderManagementImport** (as shown in Figure 7-32). Click **OK**.

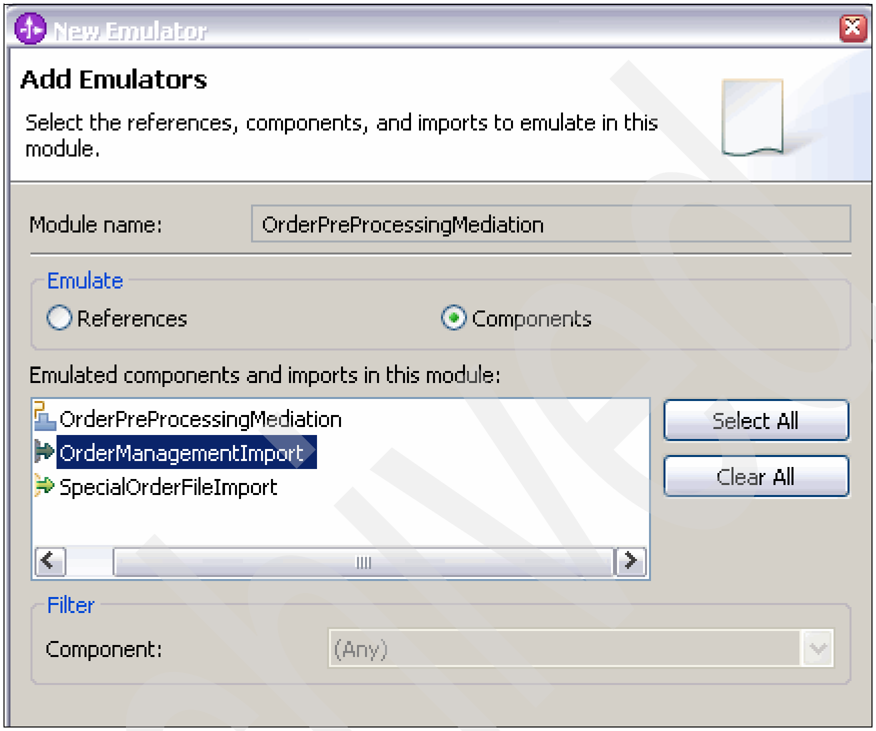


Figure 7-32 Select the component to emulate

5. Go back to the Events tab. Set the initial request parameters using Approve test case (OrderManagementInputSb_APPROVE.xml) as shown in Figure 7-33.

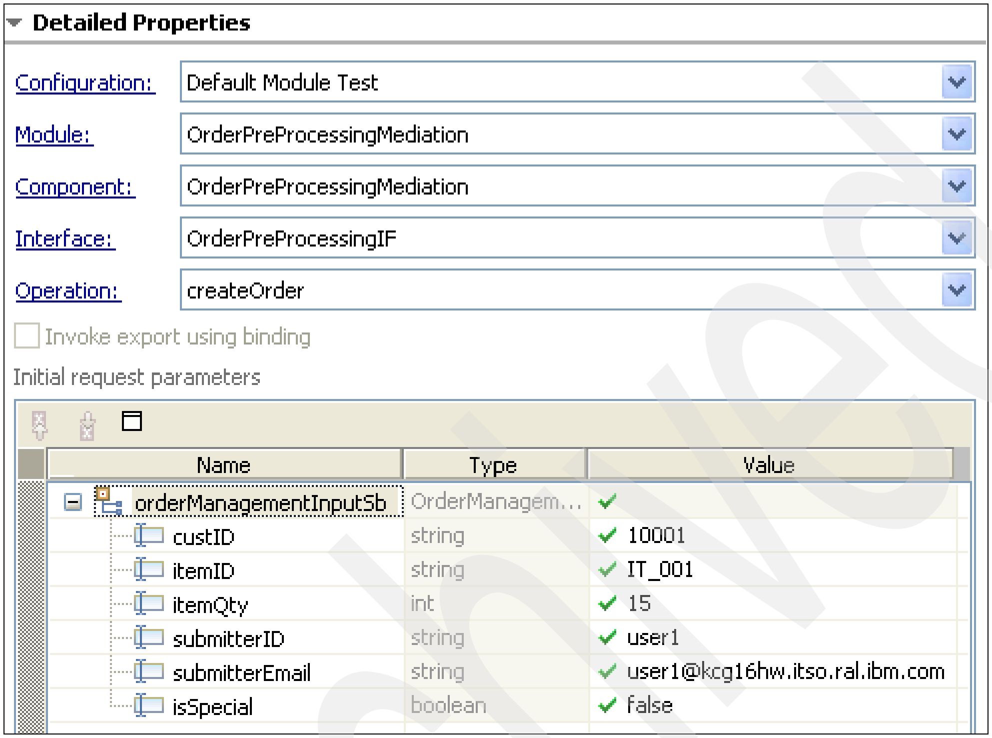


Figure 7-33 OrderPreProcessing input parameters

6. Run the test and it stops automatically at the OrderManagementImport emulation point that you set up earlier. Accept the default values and click **Continue** to finish the test.
7. Run another test for a special order by setting the initial request parameters using the OrderManagementInputSb_SPECIAL.xml.
8. Run the test. You see a new file (SpecialOrderRequest.*) in the c:\itsoSpecialOrders.

7.3 End-to-end testing

This section shows how to perform an end-to-end test if you have followed all previous sections of the chapter. If you are only interested in importing the final project interchange provided with the Additional materials, skip to 7.4, "End-to-end testing of an imported project" on page 450.

Now that you have combined the WebSphere Process Server and WebSphere ESB components, you perform the end-to-end tests from within WebSphere Integration Developer.

As we described earlier, the five test cases that we use to test the process from end-to-end are four regular order scenarios (APPROVE, REJECT, HUMAN, and ITEM SPLIT) and a Special Order. You can find the input file for each scenario in the OrderManagementSystem/SampleData directory of the additional material. The files are named OrderManagementInputSb_*.xml.

To perform end-to-end testing, follow these steps:

1. Open the workspace with all components.
2. Perform the workaround as described in "Running the component test work-around for DBMSServiceMediation" on page 432.
3. Deploy the remaining modules to the test server.
4. Open the OrderPreProcessing assembly diagram, right-click an empty area of the assembly diagram, and then select **Test Module**.

5. Test the special order use case (Figure 7-34):
   a. Make sure there are no emulators set.
   b. Set the input parameters using the OrderManagementInputSb_SPECIAL.xml input data.

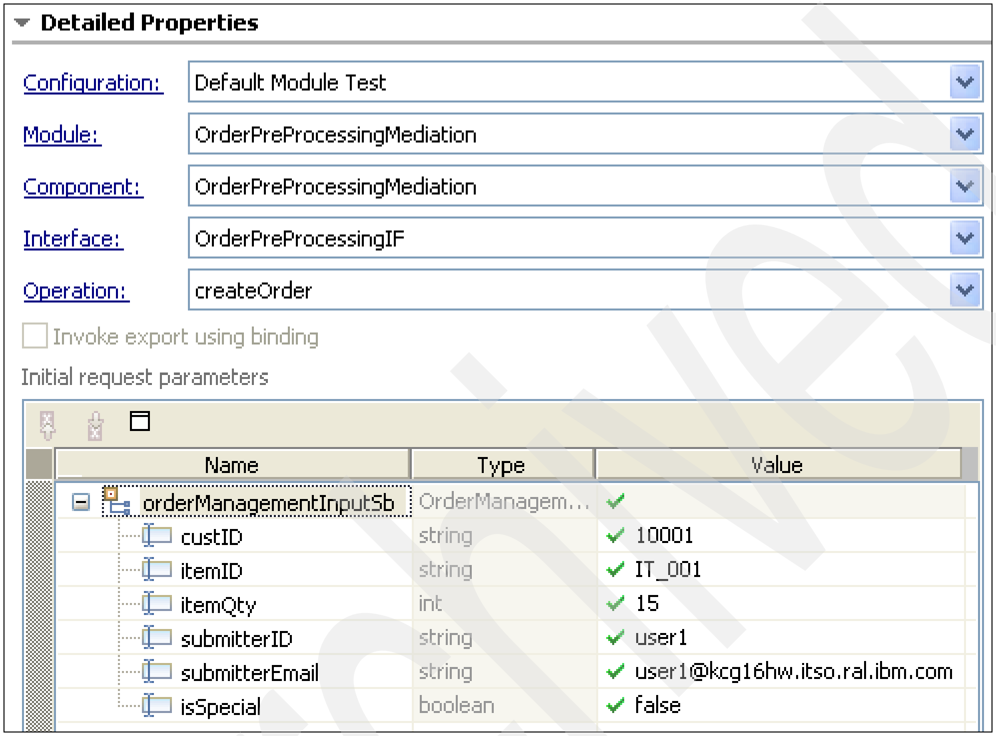


Figure 7-34 OrderPreProcessing input parameters

   c. Run the test. You should see a new file (SpecialORderRequest.*) in the c:\itsoSpecialOrders.

6. Test the regular order APPROVAL use case:
   a. Make sure that there are no emulators set.
   b. Set the input parameters using the OrderManagementInputSb_APPROVAL.xml input data.
   c. Run the test. You should receive an e-mail that confirms that the order was approved.

Note: If you have run many tests with the DBMS mediation or from OrderManagement using the DBMS mediation, you might need to reset the database to the original values. Otherwise, you might get sent to human approval and vice versa, when you do not expect to.

7. Test the regular order REJECT use case:
   a. Make sure there are no emulators set.
   b. Set the input parameters using the OrderManagementInputSb_REJECT.xml input data.
   c. Run the test. You should receive get an e-mail that confirms that the order was rejected.
8. Test the regular order HUMAN use case:
   a. Make sure that there are no emulators set.
   b. Set the input parameters using the OrderManagementInputSb_HUMAN.xml input data.
   c. Run the test. Use the BPC explorer to perform a financial and warehouse approval using the users we setup earlier in the section. Based on your financial and warehouse decisions, you should receive an e-mail confirming the order was accepted or rejected.
9. Test the regular order SPLIT use case:
   a. Make sure that there are no emulators set.
   b. Set the input parameters using the OrderManagementInputSb_SPLIT.xml input data.
   c. Run the test. You should receive an e-mail that confirms that the order was accepted. The only difference here to the approval use case is that the logs shows `Warehouses AVAILABLE number 2`.

This concludes end-to-end testing.

7.4 End-to-end testing of an imported project

This section describes how to test the projects if you have a fresh workspace and import the projects from the additional material.

To run end-to-end testing, you must perform the steps described in 7.1, “Preparing to deploy” on page 416. If you do not want to perform these steps, you can emulate one or all of the WebSphere ESB components, human tasks, and the e-mail service.

If you are not emulating any components and your `wc_defaulthost` port is not 9080, you need to update the Web service ports and bindings in the Web Service Ports folder of the WarehousesLib and OrderManagementLib libraries and on the assembly diagrams where they are imported.

Example 7-1 shows a summary of the Web services that the scenario uses.

Example 7-1 Summary of Web services used in this example

```
DBMS web services
http://localhost:9080/DBMSServiceMediationWeb/sca/CustomerServiceIFExport1
http://localhost:9080/DBMSServiceMediationWeb/sca/OrderServiceIFExport1
http://localhost:9080/DBMSServiceMediationWeb/sca/ItemServiceIFExport1

Supporting web services (Warehouse A,B,C web services)
http://localhost:9080/WarehouseAvailabilityServiceAWeb/sca/WarehouseAvailabilityServi
ceIFExport1
http://localhost:9080/WarehouseAvailabilityServiceBWeb/sca/WarehouseAvailabilityServi
ceIFExport2
http://localhost:9080/WarehouseAvailabilityServiceCWeb/sca/WarehouseAvailabilityServi
ceIFExport3

Warehouse availability mediaton web service (uses suppporting web services)
http://localhost:9080/WarehouseAvailabilityMediationWeb/sca/WarehouseItemSplitIFExpor
t1

OrderManagement web service
http://localhost:9080/OrderManagementWeb/sca/OrderManagementIFExport1

OrderPreprocessing web service
http://localhost:9080/OrderPreProcessingMediationWeb/sca/OrderPreProcessingIFExport1
```

Run the five tests and check your results.

Team development: Now is a good time to check in your project to CVS.

A

Creating the ORDERDB Derby database

The Order Management System uses a Derby database called ORDERDB. This appendix shows you how this database is created.

Additional material: The examples in this chapter use SQL scripts and data files that are provided as additional material with this book (see Appendix B, “Additional material” on page 461). The are contained in the Derby Sample Files\OrderDbProj folder. This chapter assumes these files have been copied to c:\itso\RedBookData.

Creating the database

To create the ORDERDB database using WebSphere Integration Developer:

1. Open the Data perspective by selecting **Window** → **Data Perspective**. If the Data perspective is not in the list select **Other** and then **Data** in the Open Perspective panel.
2. In the Database Explorer, right-click **Connections**. and select **New Connection** as shown in Figure A-1. (You could also simply click the New Connection icon ().

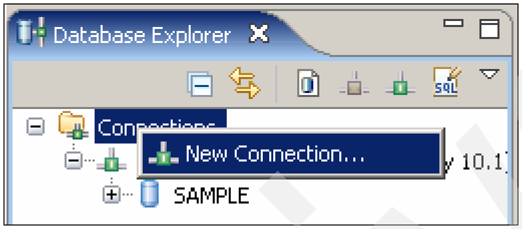


Figure A-1 Add a new connection

3. On the New Connection panel (Figure A-2 on page 455) select the following options:
   a. Clear **Use default naming convention** and enter a name for the connection:

      `ORDERDB Connection`

   b. In the Select database manager field, select **Derby 10.1**.
   c. Select **Derby Embedded JDBC Driver** as the JDBC driver class.
   d. Select the database location that you want to use in the Connection URL details.

      In our example, we want to create a database called ORDERDB and to place it in C:\itso\sampleDB. So we enter:

      `C:\itso\sampleDB\ORDERDB.`

      The directory does not have to exist. If it does not, it is created. However, if this path does exist but ORDERDB is not a Derby database, you will get an error.

   e. Make sure **Create the database if required** is selected to ensure the database creation.

f. Set the class location to:

`WID_root\runtimes\bi_v61\derby\lib\derby.jar`

g. Enter a user ID and password to be used for access to the database.

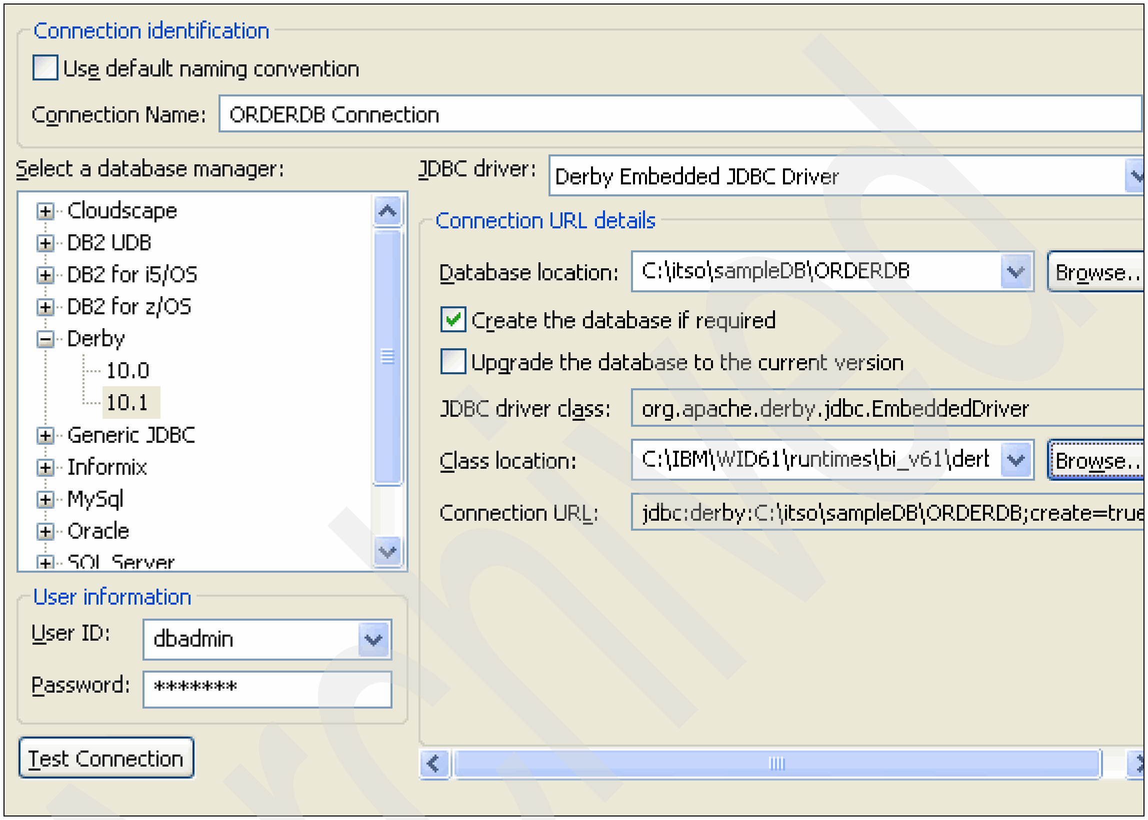


Figure A-2 Create a new database connection

4. Click **Test Connection** to create the database and test the connection.

   If the connection is unsuccessful, you might see a different message.

   If the test fails, one possible reason is that the path exists but that it does not contain a valid Derby database. Correct the path and make sure Test Connection runs successfully.

5. Click **Finish**.

Next, create tables that are used by the application.

Creating the tables

You can create new tables using an SQL script that contains the table definitions. Follow these steps:

1. Click the **Open SQL Editor** icon as shown in Figure A-3 to open a **New_Statement_1** tab.

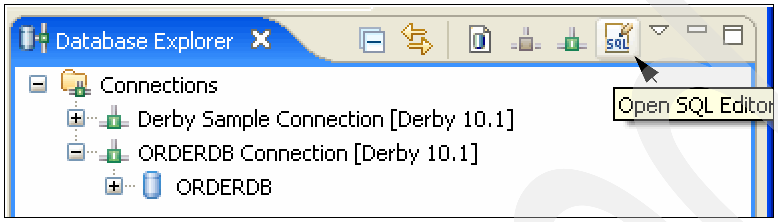


Figure A-3 Open SQL Editor icon

In this tab, type or paste here SQL statements. Enter the appropriate SQL statements to create the tables.

> **Additional material:** For this example, we copied the content from the All_ORDERDB_TABLES.ddl file into this window.

2. After you have entered the statements, right-click anywhere in SQL editor panel and select **Run SQL** from the pop-up menu as shown in Figure A-4.

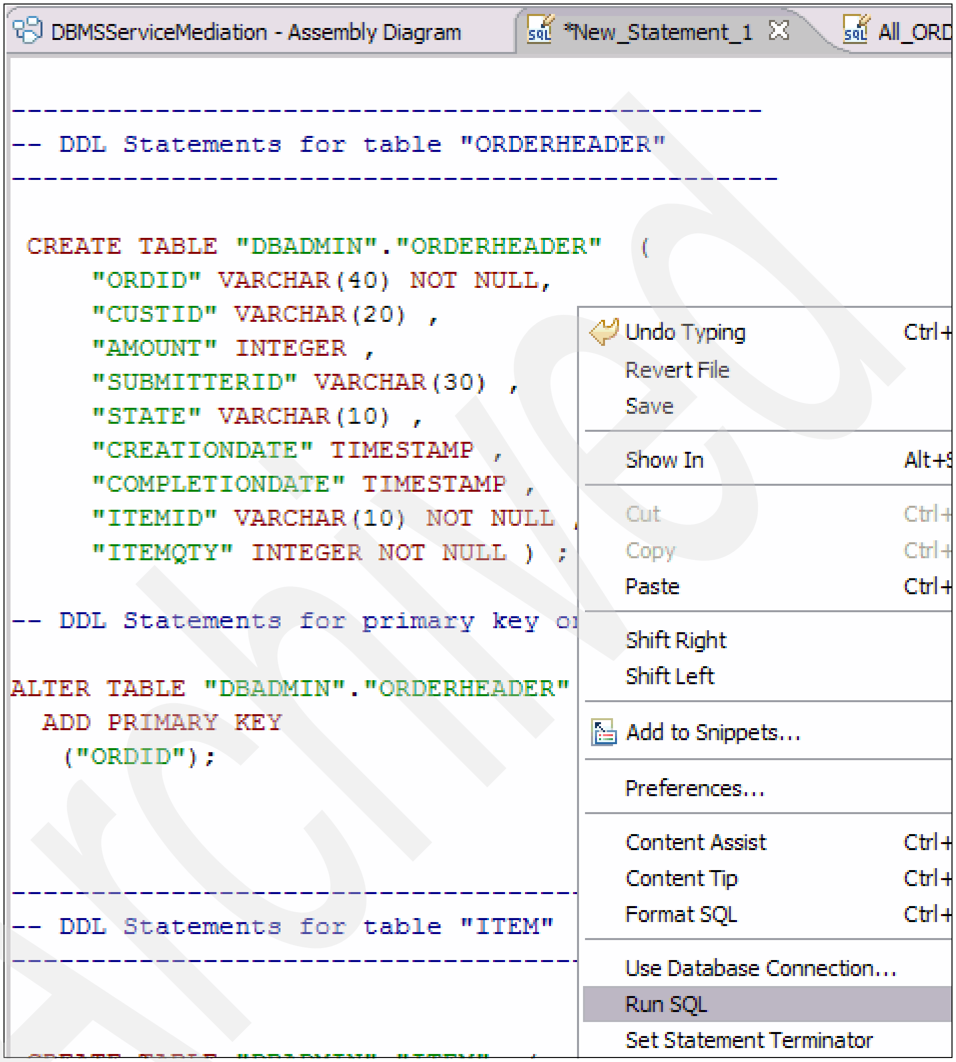


Figure A-4 Enter the SQL statements and select Run SQL

3. In the Connection Selection panel, select **Use an existing connection** and select the **ORDERDB Connection**.
4. Click **Finish**.

The DataOutput report displays as shown in Figure A-5. Each SQL statement is executed, and its result is reported in the Status column.

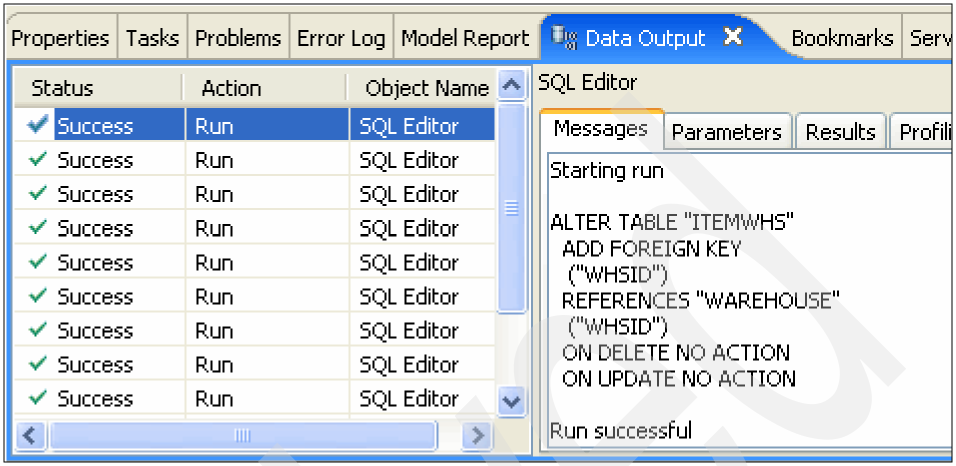


Figure A-5 Output from SQL statements

Now, you can see new tables under dbadmin schema, as shown Figure A-6. If the new tables are not there, you might have to refresh the view.

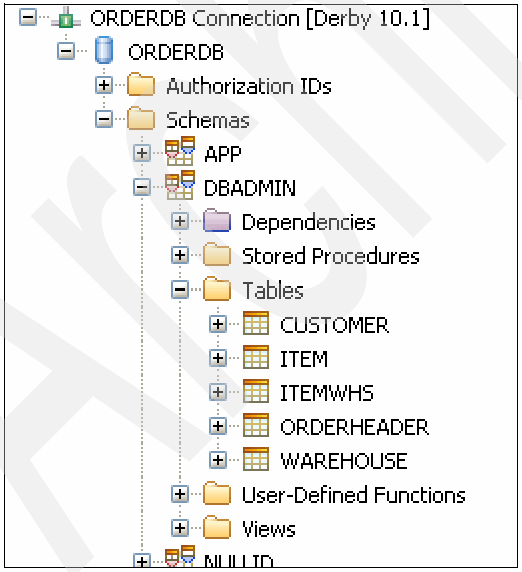


Figure A-6 ORDERDB tables

Loading data into tables

The data that is loaded into the tables is stored in files in the additional material. To load the data into the table:

1. In the Database Explorer view, expand **Connections** → **ORDERDB Connection** → **ORDERDB** → **Schemas** → **DBADMIN**.
2. Select the Customer table, right-click, select **Data** from the pop-up menu, and select **Load** as shown in Figure A-7.

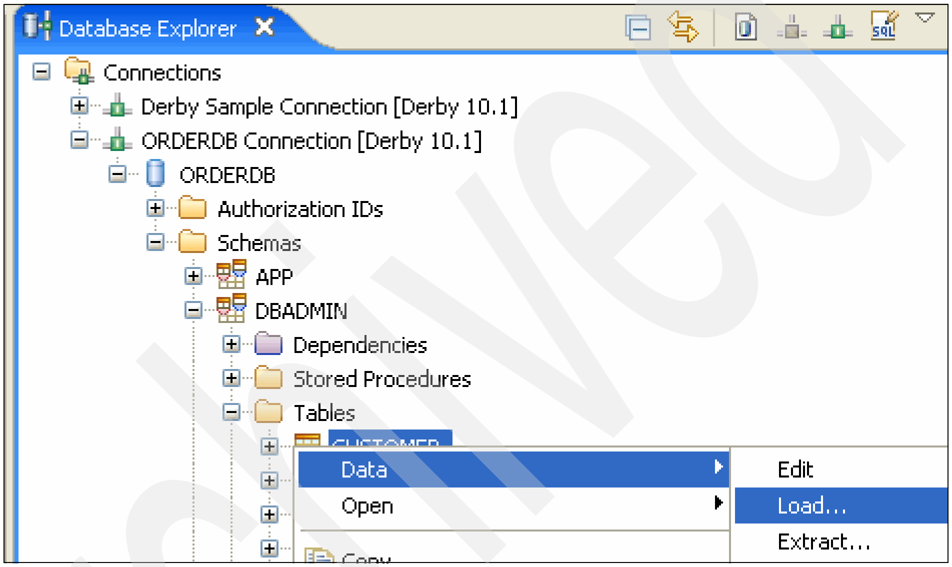


Figure A-7 Load data into the table

3. Select the input data file and review the file format parameters to make sure they correspond to the format used in the data file you plan to load. See Figure A-8.

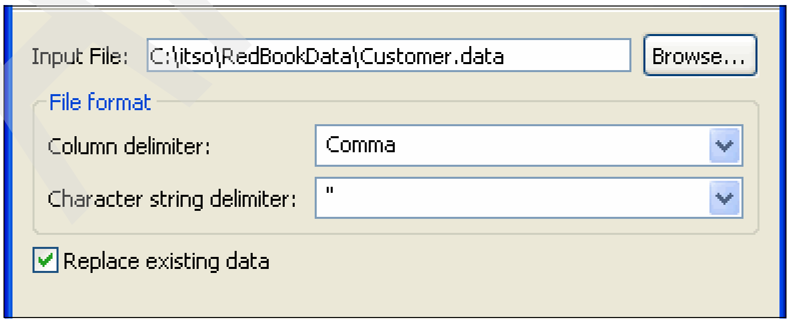


Figure A-8 Select the file from which to load the data

4. Click **Finish**.
5. Repeat the process for each table that you need to populate. In our example, this is the ITEM, WAREHOUSE, and ITEMWHS tables.

   The order in which you load the tables in ORDERDB is important. If you try to load ITEMWHS.data before the other tables, you will get SQL errors due to constraints that are imposed by foreign keys.

You can see the results in the DataOutput window, as shown in Figure A-9.

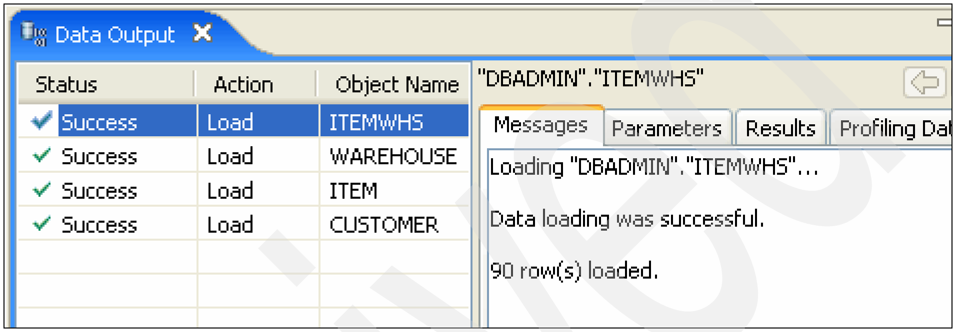


Figure A-9 Results of the data load

Working with the database

Some common actions that you can take against a database in the Database Explorer are:

- Edit data in a table: In the Database Explorer view, select the table, then right-click and select **Data** → **Edit** from the pop-up menu.

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

- Close connections: To close the database connection gracefully, select the connection, right-click, and select **Disconnect** from the pop-up menu.

  To reconnect select the connection, right-click and select **Reconnect** from the pop-up menu.

To remove the Derby database from your system, just remove the folder where the database is located. In our case, the directory is C:\itso\sampleDB\ORDERDB. Removing the database folder deletes the schema and tables from the connection, but the connection remains.

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 associated with this book is available in softcopy on the Internet from the IBM Redbooks publications Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247642`

Alternatively, you can go to the IBM Redbooks publications Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247642.

Using the Web material

The additional Web material that accompanies this book includes the following files:

- **BusinessExampleSnippets**: This directory contains text files used to populate the Java snippets in Chapter 3, “Order Management Process business integration module” on page 67.
- **Derby sample files\ORDERDB:** This directory contains the ORDERDB database used by the scenario. Create a local directory C:\itso\sampleDB and copy this directory to it to have the complete database ready for use in the sample.
- **Derby sample files\OrderDbProj**: This directory contains DDL and data files used to build the ORDERDB database. You do not need these files unless you plan to practice building Derby databases.
- **Sample Data**: This directory contains XML files with the test data that corresponds to the test cases that we use throughout the book.
- **Order Management System**: This directory contains project interchanges files that contain the complete Order Management System scenario. Note that an e-mail server is not included. To test that piece of the scenario, you need to have access to an existing e-mail server.

To import the project interchange files to WebSphere Integration Developer:

1. Select **File** → **Import** → **Other** → **Project Interchange**. Click **Next**.
2. Navigate to the *download_location*\Order Management System\OrderManagementE2E.zip file. Click **Open**.
3. Select all the files and click **Finish**.

How to use the Web material

Create a subdirectory (folder) on your workstation, and decompress the contents of the Web material zipped file into this folder.

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 publications” on page 464. Note that some of the documents referenced here might be available in softcopy only.

- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1: Development*, SG24-7608
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus: Part 3: Run time,* SG24-7643
- *Business Process Management: Modeling through Monitoring Using IBM WebSphere V6.0.2 Products*, SG24-7148
- *Building SOA Solutions Using the Rational SDP*, SG24-7356

Online resources

These 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`

How to get IBM Redbooks publications

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

E

F

G

H

I

J

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 2: Scenario

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 2: Scenario

Learn to create business integration applications

Learn to create mediations

Learn end-to-end test techniques

This IBM Redbooks publication illustrates the concepts and techniques that are associated with building business integration applications and mediations by example. It starts by designing a solution for an order management process. The solution includes a business process and several mediations. It then shows how each component of the solution is created and tested in a development environment.

This book also illustrates the use of three adapters:

- The IBM WebSphere Adapter for JDBC
- The IBM WebSphere Adapter for Flat Files
- The IBM WebSphere Adapter for Email Version

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

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

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

SG24-7642-00 ISBN 0738431044