

Invoking a Web service with a Java Message Service (JMS) client

Build a solution with WebSphere Integration Developer and WebSphere Enterprise Service Bus

Skill Level: Intermediate

[Philip Norton \(nortonp@uk.ibm.com\)](mailto:nortonp@uk.ibm.com)

Software Engineer, WebSphere ESB Development Team
WSO2 Inc

25 Apr 2006

Learn how to invoke a Web service with a Java™ Message Service (JMS) client, using WebSphere® Enterprise Service Bus (ESB) and WebSphere Integration Developer. In this tutorial you will create a simple Web service, define the necessary server resources, build a mediation module to expose a Web service as a JMS service, and configure a JMS client to invoke the Web service.

Section 1. Before you start

About this tutorial

The WebSphere Enterprise Service Bus (ESB) enables the connection of applications and services. In many cases these applications and services will have been developed over time, using different programming languages, interfaces and standards. This tutorial describes how to use WebSphere ESB to expose a Web service so it can be invoked by a JMS application.

Objectives

In this tutorial we build an end-to-end solution using the visual tooling supplied by WebSphere Integration Developer.

Prerequisites

You'll need WebSphere Integration Developer installed with the WebSphere ESB Integrated Test Environment and the server's default host configured to port 9080.

System requirements

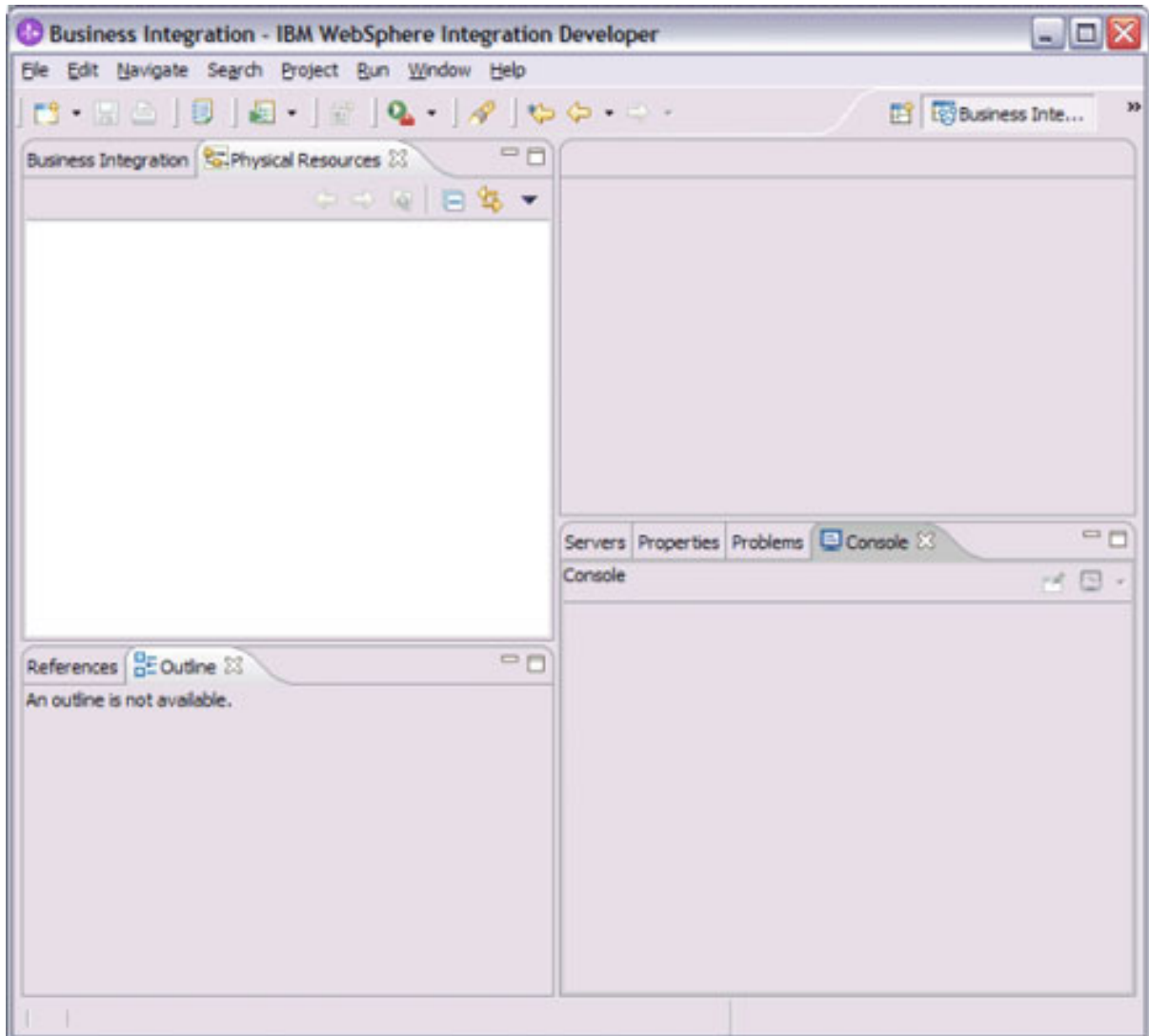
To run the examples in this tutorial, you will need a machine with at least 500MB of memory.

Section 2. Creating a JMS client

This scenario assumes that the Java™ Message Service (JMS) client has been configured to send an order and receive a confirmation. The orders are defined in XML and are sent using TextMessages. To test the JMS client we will use JavaServer Pages technology (JSP) and run it on our server. This will allow us to provide a simple interface to the client which can be displayed using a browser.

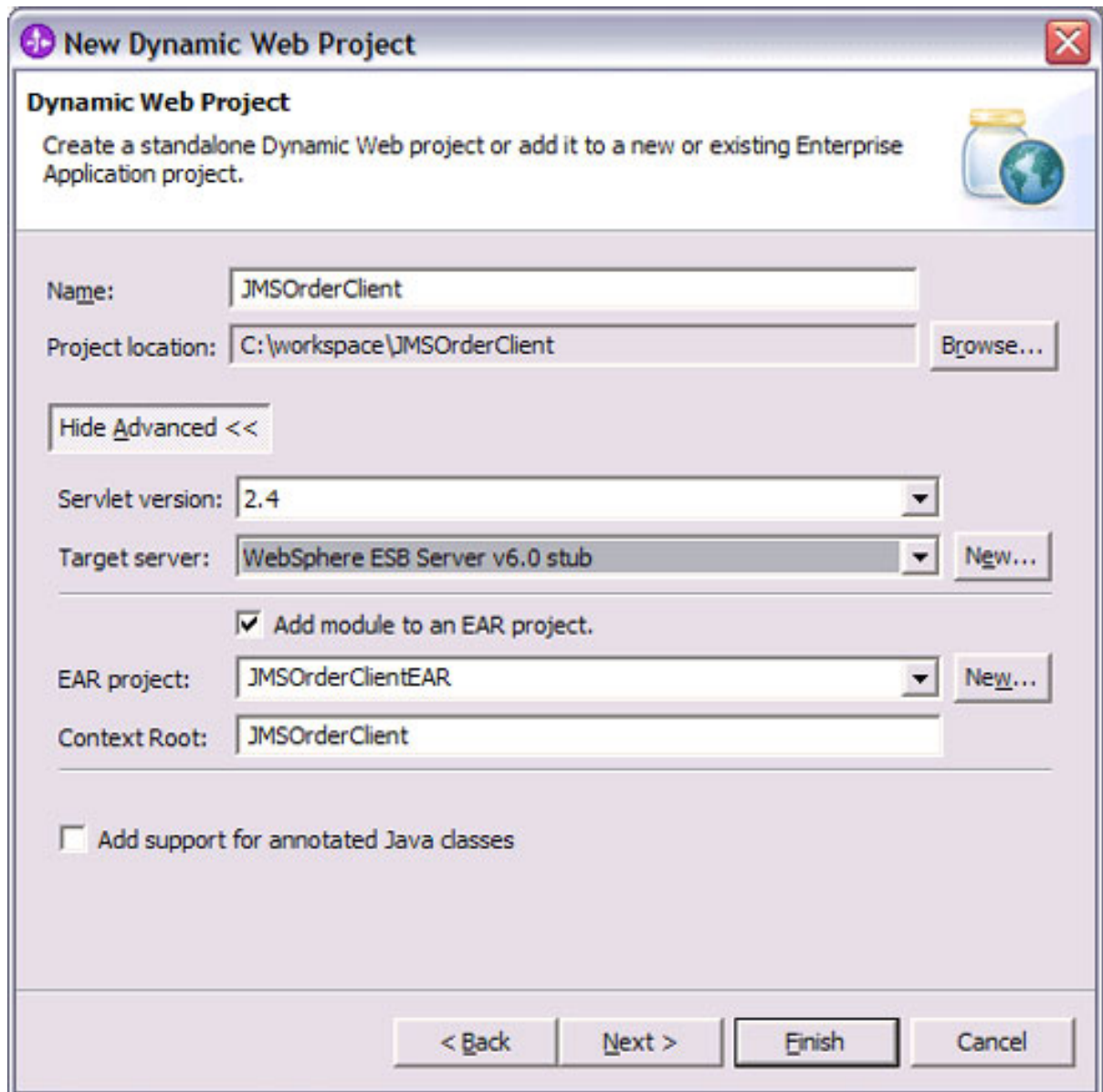
First, open WebSphere® Integration Developer and close the welcome pane to enter the Business Integration perspective. If you happen to close this perspective, you can re-open it by selecting **Window > Open Perspective > Other > Business Integration**. For the first task we need to use the Physical Resources view, as shown in [Figure 1](#). Select **Window > Show View > Physical Resources**:

Figure 1. The Physical Resources view



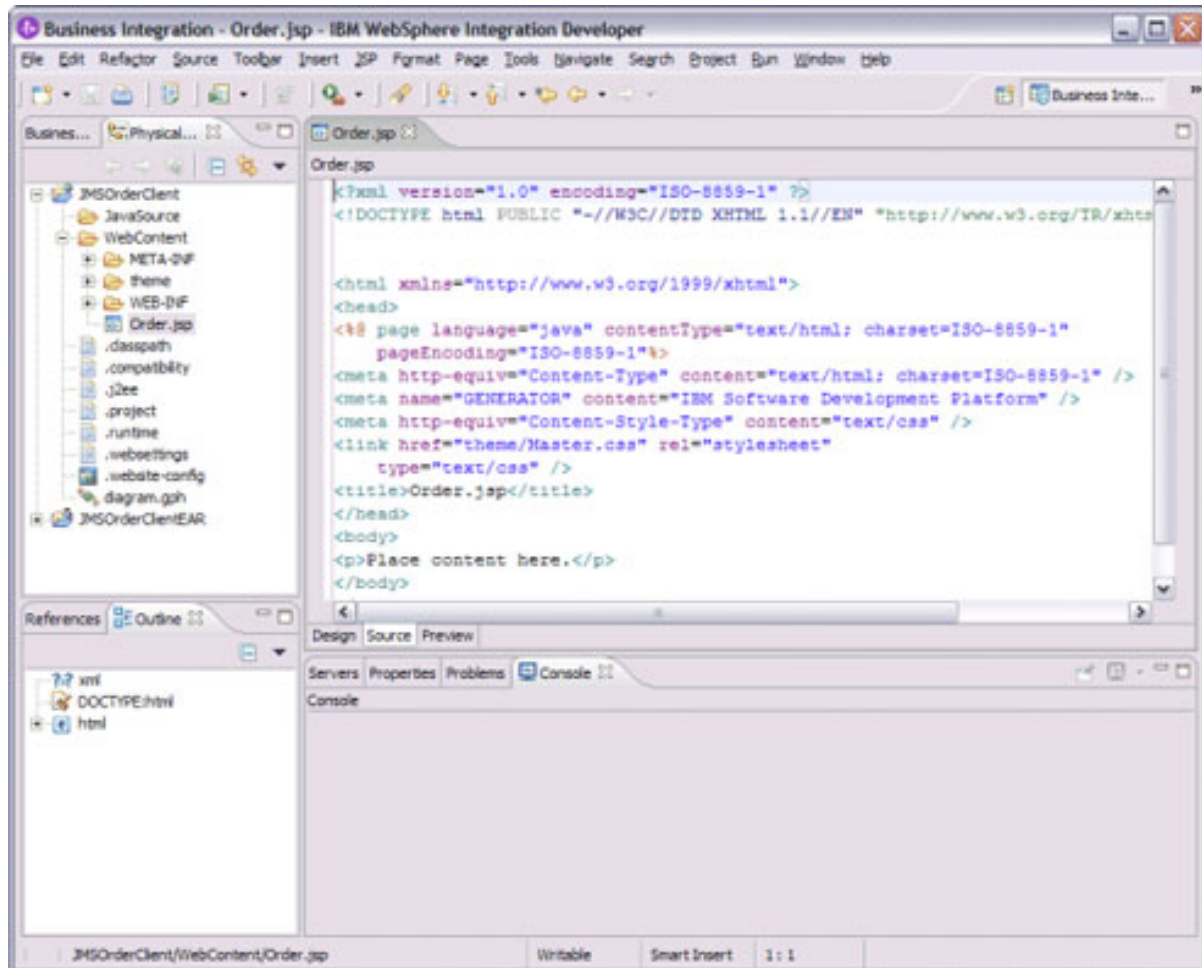
1. Right click in the Physical Resources view and select **New > Other...**
2. Choose a project type of **Web > Dynamic Web Project**. This will open the **New Dynamic Web Project** dialog.
3. Enter `JMSOrderClient` as the name of the project.
4. Click the **Show Advanced>>** button and ensure that the target server is set to **WebSphere ESB Server v6.0 stub**, as shown in [Figure 2](#).
5. Click **Finish**. When asked to switch to the Web Perspective click **No**.

Figure 2. New dynamic Web project



6. Expand the **JMSOrderClient** Web project. Right Click and select **New > Other...**
7. Choose **Web > JSP File** and click **Next**.
8. Give it a name of `Order.jsp` and click **Finish**. The JSP will be displayed as shown in [Figure 3](#).

Figure 3. Order.jsp



9. Replace the contents of Order.jsp with that shown in [Listing 1](#).

Listing 1. Code for JMS client Order.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <meta name="GENERATOR" content="IBM Software Development Platform">
    <title>Order.jsp</title>
  </head>
  <body>
    <div style="text-align: center">
      <h1>Place an order</h1>
      <form method="get" action="Order.jsp">
        <table align="center">
          <tr>
            <td>Product ID:</td>
            <td><input type="text" name="productid"/></td>
          </tr>
          <tr>
            <td>Quantity:</td>
```

```

        <td>
            <select name="quantity">
                <option>1</option>
                <option>2</option>
                <option>3</option>
                <option>4</option>
                <option>5</option>
            </select>
        </td>
    </tr>
</table>
<br/>
<input name="order" type="submit" value="Order"/>
</form>
<%
if (request.getParameter("productid") != null) {
//The Initial Context Factory
String icf = "com.ibm.websphere.naming.WsnInitialContextFactory";
//the Provider URL (This port number may be different depending on your install)
String url = "iiop://localhost:2809/";
//The Queue Connection Factory used to connect to the bus
String sampleQCF = "jms/orderQCF";
//The Queue used to send requests to the mediation module
String sampleSendQueue = "jms/orderInputQueue";
//The Queue used to receive responses from the mediation module
String sampleReceiveQueue = "jms/orderOutputQueue";
//The XML representation of an Order Business Object required by the
//placeOrder operation on the OrderService Interface
String message = "<?xml version=\\"1.0\\" encoding=\\"UTF-8\\"?>";
message += "<xsl:Order xmlns:xsl=\\"http://OrderServiceModule\\">";
message += "<productid>"+request.getParameter("productid")+"</productid>";
message += "<quantity>"+request.getParameter("quantity")+"</quantity>";
message += "</xsl:Order>";

try {
//Create the Initial Context
java.util.Hashtable env = new java.util.Hashtable();
env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, icf);
env.put(javax.naming.Context.PROVIDER_URL, url);
javax.naming.Context ctx = new javax.naming.directory.InitialDirContext(env);

//Lookup the ConnectionFactory
javax.jms.ConnectionFactory factory =
    (javax.jms.ConnectionFactory)ctx.lookup(sampleQCF);
//Create a Connection
javax.jms.Connection connection = factory.createConnection();
//Start the Connection
connection.start();
//Create a Session
javax.jms.Session jmsSession =
    connection.createSession(false, javax.jms.Session.AUTO_ACKNOWLEDGE);
//Lookup the send Destination
javax.jms.Destination sendQueue =
    (javax.jms.Destination) ctx.lookup(sampleSendQueue);
//Create a MessageProducer
javax.jms.MessageProducer producer = jmsSession.createProducer(sendQueue);
//Create the TextMessage that will hold our Order as text
javax.jms.TextMessage sendMessage = jmsSession.createTextMessage();
//Set the content of the message to be the XML defined Order
sendMessage.setText(message);
//Set the operation to call on the OrderService interface to be placeOrder
sendMessage.setStringProperty("TargetFunctionName", "placeOrder");
//Send the message
producer.send(sendMessage);

//Lookup the receive Destination
javax.jms.Destination receiveQueue =
    (javax.jms.Destination) ctx.lookup(sampleReceiveQueue);
//Create a MessageConsumer

```

```
    javax.jms.MessageConsumer consumer = jmsSession.createConsumer(receiveQueue);
    //Wait 15 seconds to receive the response
    javax.jms.TextMessage receiveMessage =
        (javax.jms.TextMessage) consumer.receive(15000);
    //If we receive a response print the contents of the message to the screen
    String confirmation = "Order failed.";
    if (receiveMessage != null) {
        //Print the contents of the message.
        confirmation = "Order Successful.<br/>Order: "+receiveMessage.getText();
    }
    out.println("<p>"+confirmation+"</p>");

    //Close the Connection
    connection.close();
}
catch (Exception e) {
    out.println(e);
}
}
%>
</div>
</body>
</html>
```

11. Save Order.jsp.

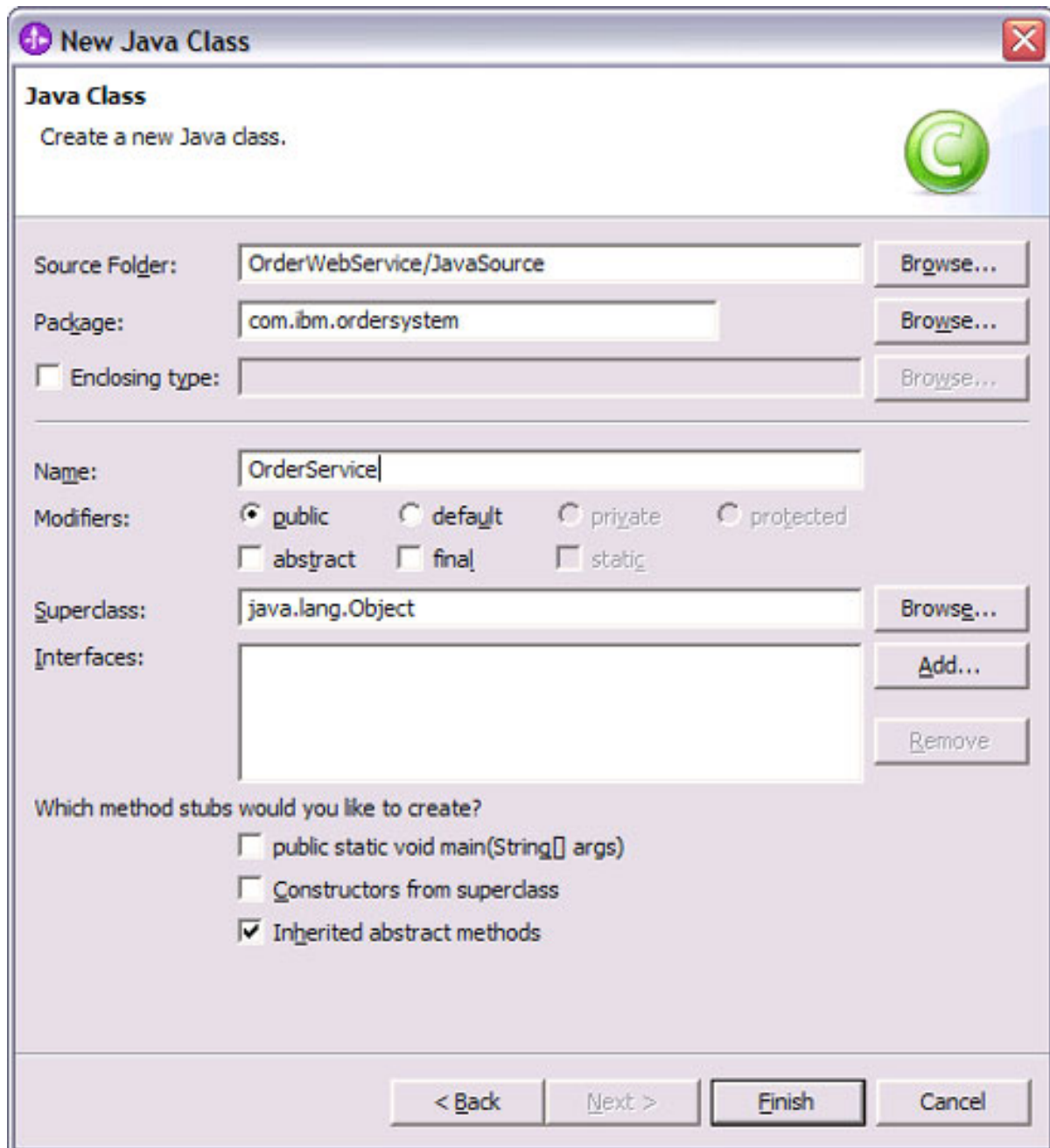
The creation of the JMS client is complete.

Section 3. Creating a Web service

Now it's time to create a Web service that is implemented using a simple Java class. To keep things interesting, we assume that the Web service uses a different interface to the JMS client:

1. In the Physical Resources view, create another dynamic Web project, as described in steps 1-5 of The JMS Client, naming the project `OrderWebService`.
2. Right click on the JavaSource directory and select **New > Other**.
3. Choose **Java > Class** and click **Next**.
4. Set the package to `com.ibm.ordersystem` and the name to `OrderService`, as shown in [Figure 4](#):

Figure 4. New Java class



5. Click **Finish**
6. The Java editor will open with the empty `OrderService` class. Replace the implementation with that shown in [Listing 2](#):

Listing 2. Code for `OrderService.java`

```
package com.ibm.ordersystem;
```

```
import java.util.Random;

public class OrderService {
    //The order method takes an Order object
    public Confirmation order(Order order) {
        Confirmation confirmation = new Confirmation();
        //Check whether the values are valid
        if (!order.getId().equals(null) && order.getQuantity() != 0) {
            //Return a new Confirmation with an id between 0-1000
            confirmation.setId(Integer.toString(new Random().nextInt(1000)));
        }
        //If the Order is invalid return a null Confirmation
        return confirmation;
    }
}
```

7. In the same package create a Java class called `Order`. Replace the implementation with that shown in [Listing 3](#).

Listing 3. Code for `Order.java`

```
package com.ibm.ordersystem;

//Order has an id and a quantity
class Order {
    String id;
    int quantity;

    public Order() {}

    public String getId() {
        return id;
    }

    public int getQuantity() {
        return quantity;
    }
}
```

8. In the same package create a Java class called `Confirmation`. Replace the implementation with that shown in [Listing 4](#):

Listing 4. Code for `Confirmation.java`

```
package com.ibm.ordersystem;

//Confirmation has an id
class Confirmation {

    String id;

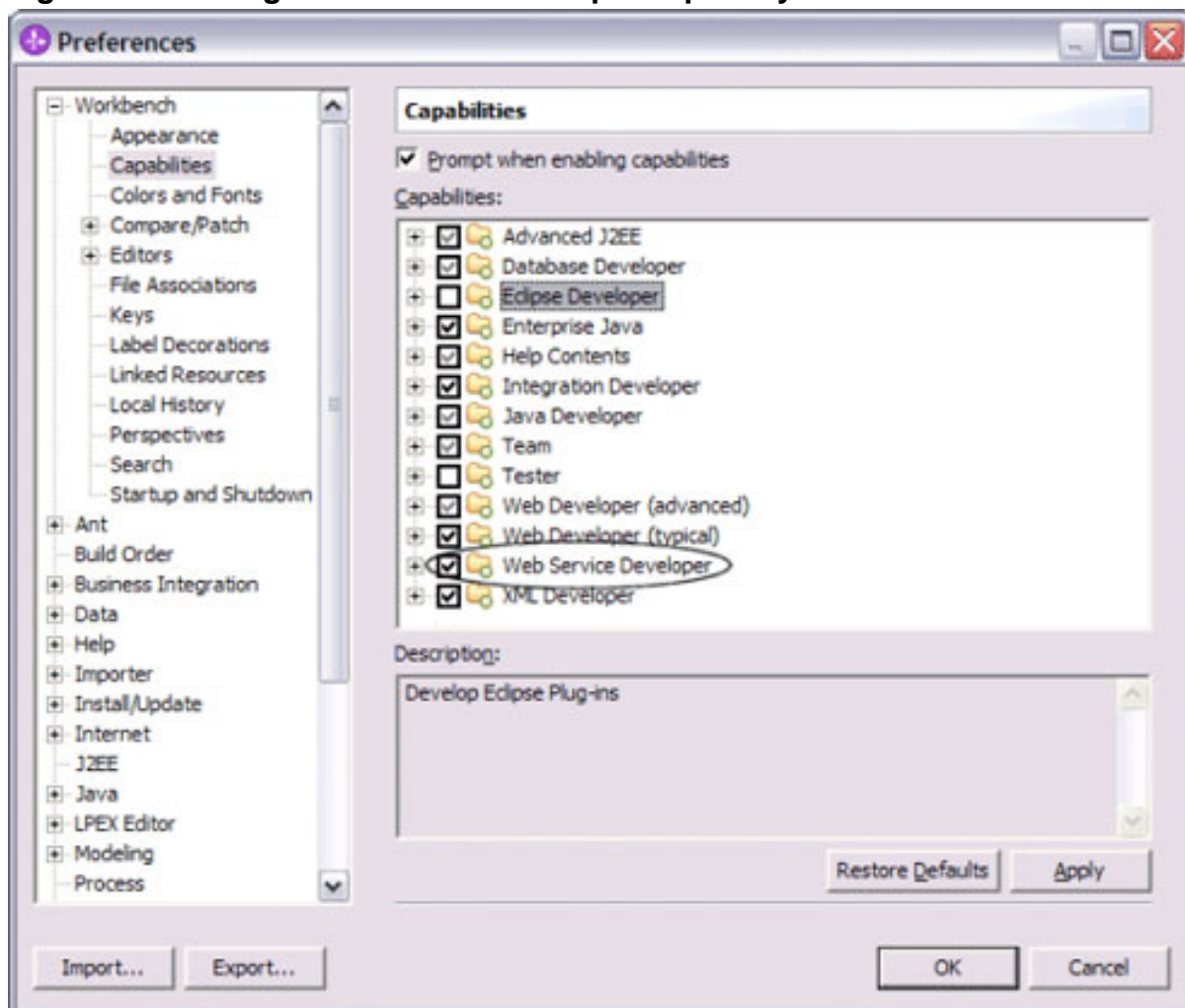
    public Confirmation() {}

    public String getId() {
```

```
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }
}
```

- To create a Web service we need to enable the Web Service Developer Capability. Select **Window > Preferences**.
- Select **Workbench > Capabilities** and ensure the **Web Service Developer** is ticked, as shown in [Figure 5](#):

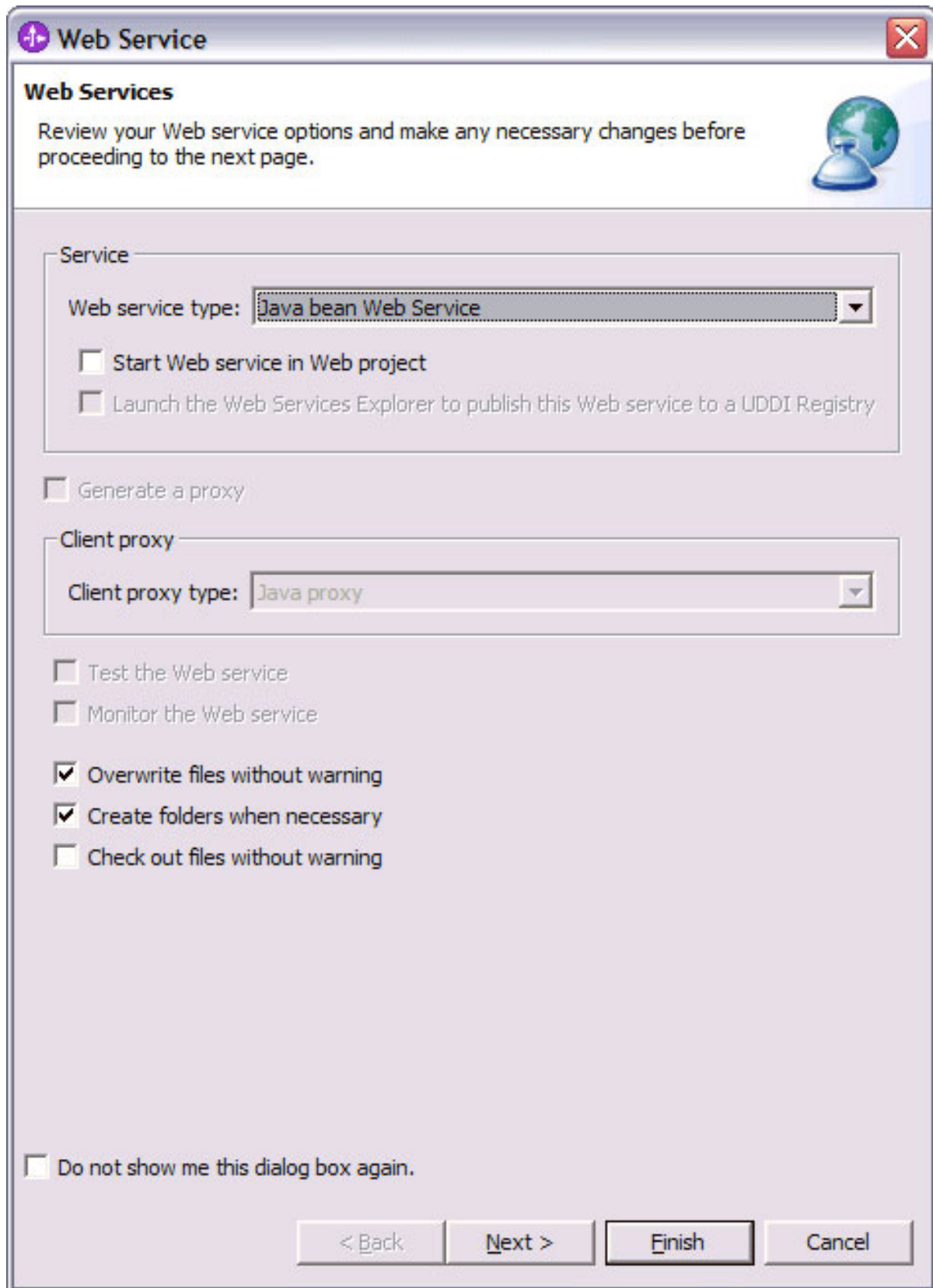
Figure 5. Enabling Web Service Developer capability



- Click **OK**.

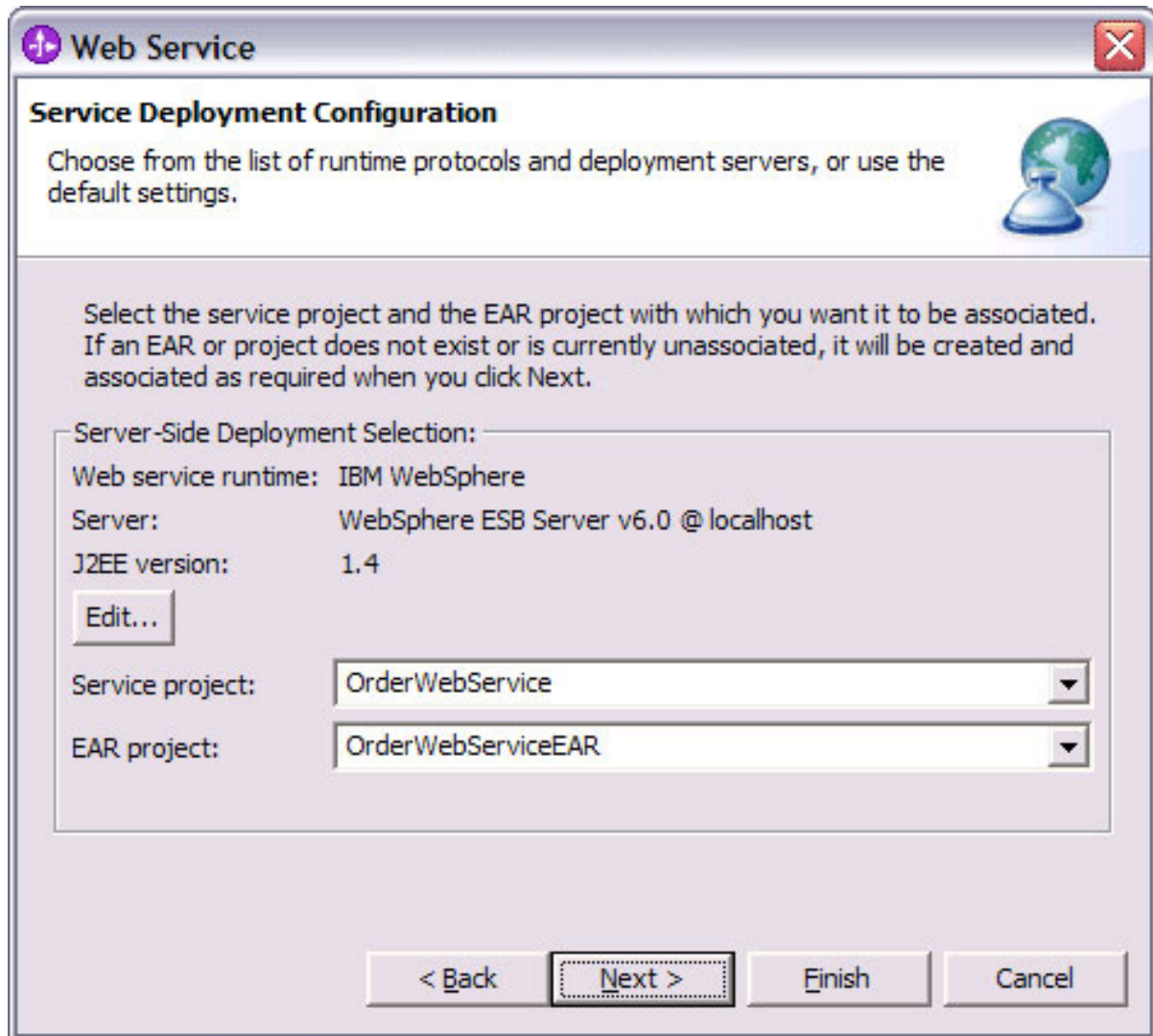
12. Right click on **OrderService.java** and select **Web Services > Create Web Service** . This will open the Web Service wizard as shown in [Figure 6](#):

Figure 6. Web Service wizard



13. Untick the **Start Web service in Web project** check box.
14. Click **Next**. This will take you to the Object selection panel. In the Bean field **com.ibm.ordersystem.OrderService** will be displayed.
15. Click **Next**. This will display the Service Deployment Configuration panel. Ensure the settings match those in [Figure 7](#):

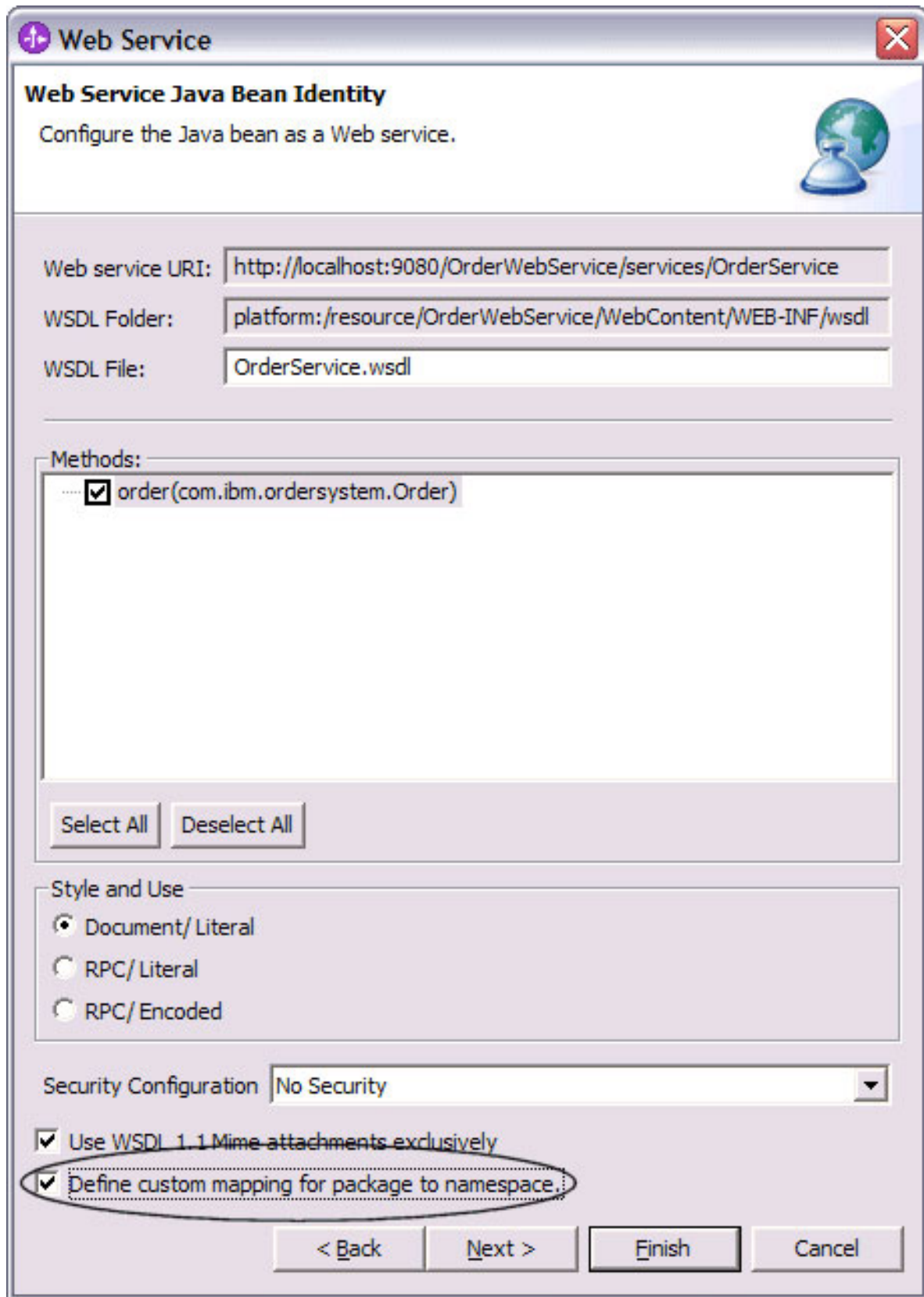
Figure 7. Service Deployment Configuration



16. Click **Next** to display the **Service Endpoint Interface Selection** panel.
17. Click **Next** to display the **Web Service Java Bean Identity** panel.
18. Check the **Define custom mapping for package to namespace** check

box, as shown in [Figure 8](#):

Figure 8. Define custom mapping



19. Click **Next**.
20. On the **Web service mapping for package to namespace** panel, press the **Add** button
21. Set package to `com.ibm.ordersystem` .
22. Set the namespace to `http://ordersystem.ibm.com` .
23. Click **Finish**. This will generate several Java files and a WSDL file describing the Web service.

The generation of the Web service is now complete.

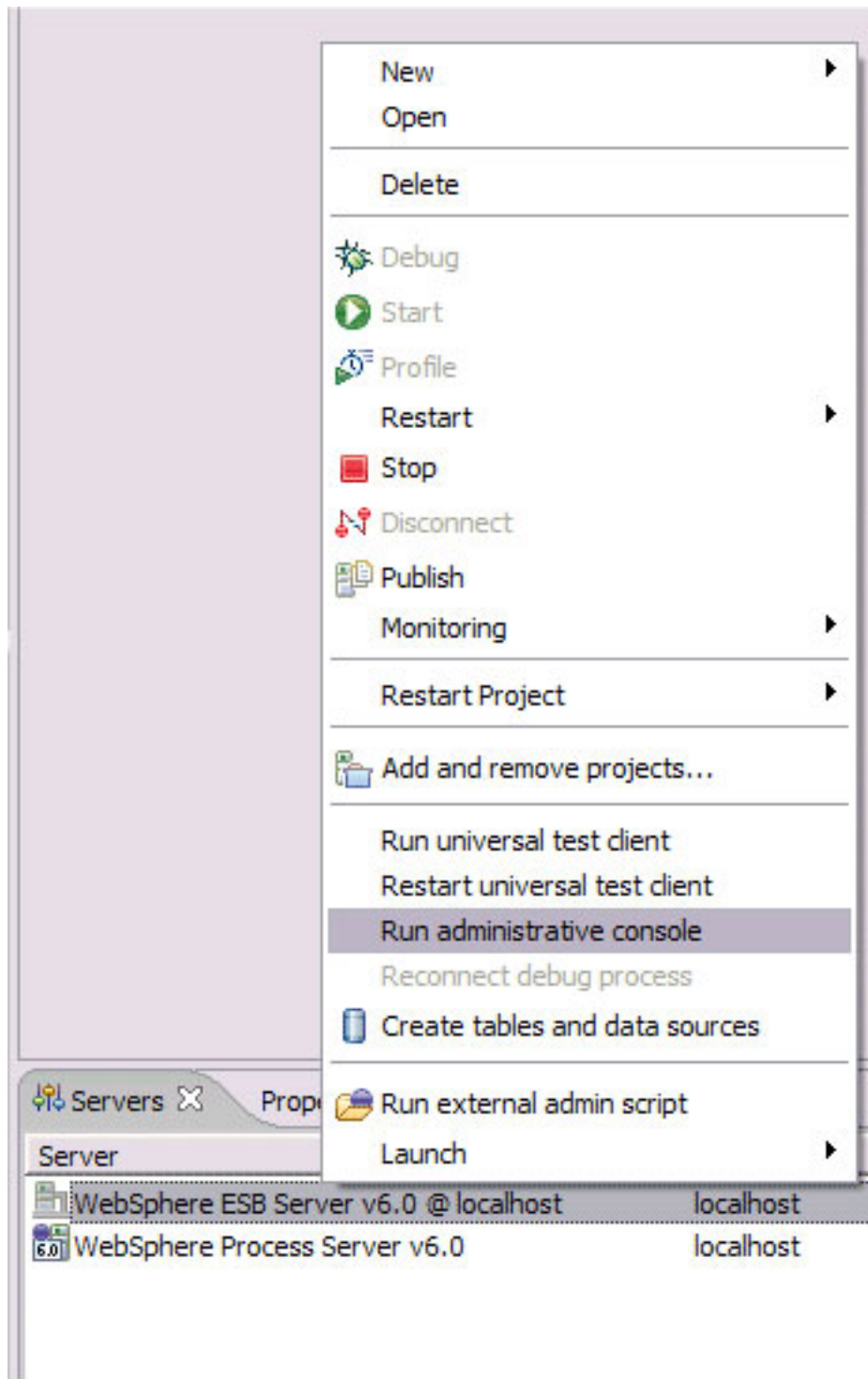
Section 4. Defining server resources

Before creating the mediation module, we need to create the resources required for the JMS client and the mediation module to communicate. These include:

- **Input Queue:** Stores request messages
- **Output Queue:** Stores response messages
- **JMS Input Queue:** Used by the JMS client and mediation module to access the input queue
- **JMS Output Queue:** Used by the JMS client and mediation module to access the output queue
- **JMS Queue Connection Factory:** Used by the JMS client and mediation module to create a connection to the messaging engine hosting the Input and Output Queues
- **JMS Activation Specification:** Used by the mediation module to register a queue with the module's message listener

In the Servers view, right click on the WebSphere ESB server and select **Run administrative console**, as shown in [Figure 9](#), and log in to the Administrative console.

Figure 9. Run administrative console



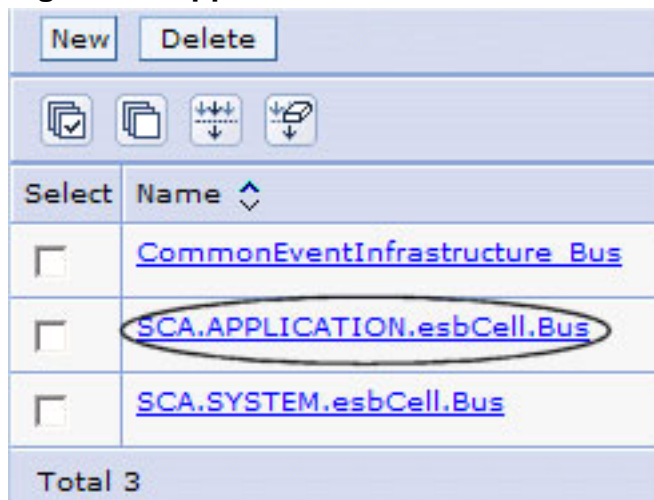
Creating input and output queues

The physical queues are created on the default Service Integration Bus, for use by

WebSphere ESB applications. Let's create a queue to store requests and a queue to store responses.

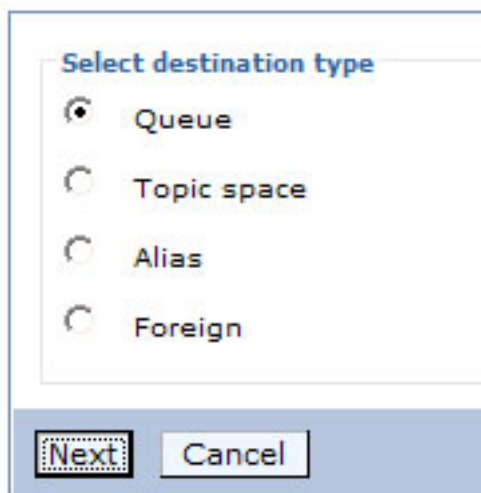
1. In the navigation, click **Service Integration > Buses**.
2. A list of buses is displayed, as shown in [Figure 10](#). Click **SCA.APPLICATION.esbCell.Bus**.

Figure 10. Application Bus



3. Click **Destinations**.
4. Press the **New** button.
5. Destination type **Queue** is selected by default, as shown in [Figure 11](#):

Figure 11. Destination type



6. Click **Next**.
7. Set the Identifier field to `OrderInputQueue`, as shown in [Figure 12](#):

Figure 12. Queue identifier

→ **Step 1: Set queue attributes**

Step 2: Assign the queue to a bus member

Step 3: Confirm queue creation

Set queue attributes

Configure the attributes of your new queue

* Identifier
OrderInputQueue

Description

Next Cancel

8. Click **Next**. This will take you to the **Assigning a Queue to a Bus Member** panel.
9. Click **Next** again.
10. Click **Finish**.
11. Repeat steps 5-11 to create a Queue called `OrderOutputQueue`.
12. You will see both queues listed as shown in [Figure 13](#):

Figure 13. Input and output queues

Select	Identifier	Type
<input type="checkbox"/>	Default.Topic.Space	Topic space
<input type="checkbox"/>	OrderInputQueue	Queue
<input type="checkbox"/>	OrderOutputQueue	Queue
<input type="checkbox"/>	SYSTEM.Exception.Destination.esbNode.server1-SCA.APPLICATION.esbCell.Bus	Queue
Total 4		

13. Save your changes.

Creating JMS queues

The JMS queues are stored in JNDI and describe how to access the physical queues. These are configured to point at **OrderInputQueue** and **OrderOutputQueue**.

1. In the navigation click **Resources > JMS Providers > Default Messaging**.
2. Click **JMS queue**, as shown in [Figure 14](#):

Figure 14. JMS Queue

Connection Factories

- [JMS connection factory](#)
- [JMS queue connection factory](#)
- [JMS topic connection factory](#)

Destinations

- [JMS queue](#)
- [JMS topic](#)

Activation Specifications

- [JMS activation specification](#)

3. Press the **New** button.
4. Set **Name** to `orderInputQueue`.
5. Set **Java Naming and Directory Interface (JNDI) name** to `jms/orderInputQueue`.
6. From **Bus name** drop down list select **SCA.APPLICATION.esbCell.Bus**.
7. From **Queue name** drop down list select **OrderInputQueue**.
8. Your JMS Queue should look like [Figure 15](#):

Figure 15. Creating orderInputQueue

General Properties

Administration

* **Scope**

* **Name**

* **JNDI name**

Description

Connection

Queue name

Bus name

Delivery mode

9. Click **OK**.
10. Press the **New** button.
11. Set **Name** to orderOutputQueue.
12. Set **JNDI name** to jms/orderOutputQueue.
13. From **Bus name** drop down list select **SCA.APPLICATION.esbCell.Bus**.
14. From **Queue name** drop down list select **OrderOutputQueue**.
15. Your JMS Queue should look like [Figure 16](#):

Figure 16. Creating orderOutputQueue

General Properties

Administration

* Scope
cells:esbCell:nodes:esbNode

* Name
OrderOutputQueue

* JNDI name
jms/OrderOutputQueue

Description

Connection

Queue name
OrderOutputQueue

Bus name
SCA.APPLICATION.esbCell.Bus

Delivery mode
Application

16. Click **OK**.
17. Save your changes.

Creating JMS QueueConnectionFactory

The queue connection factory is stored in JNDI and describes how to create a connection to the messaging engine. The messaging engine is available through the service integration bus **SCA.APPLICATION.esbCell.Bus** .

1. In the navigation click **Resources > JMS Providers > Default**

Messaging.

2. Click **JMS queue connection factory**, as shown in [Figure 17](#):

Figure 17. Creating JMS QueueConnectionFactory



3. Press the **New** button.
4. Set **Name** to `orderQCF`.
5. Set **JNDI name** to `jms/orderQCF`.
6. From **Bus name** drop down list select **SCA.APPLICATION.esbCell.Bus**.
7. Your JMS Queue Connection Factory should look like [Figure 18](#):

Figure 18. Creating orderQCF

General Properties

Administration

* Scope

* Name

* JNDI name

Description

Category

Connection

* Bus name

Target

8. Click **OK**.
9. Save your changes.

Creating the JMS activation specification

The activation specification is stored in JNDI and registers a queue with the mediation module message listener. We will create an activation specification configured to use the **OrderInputQueue**.

1. In the navigation click **Resources > JMS Providers > Default Messaging**.

2. Click **JMS activation specification**, as shown in [Figure 19](#):

Figure 19. Creating JMS Activation Specification



3. Press the **New** button.
4. Set **Name** to `orderAS`.
5. Set **JNDI name** to `jms/orderAS`.
6. Set **Destination JNDI name** to `jms/orderInputQueue`.
7. From the **Bus name** drop down list select **SCA.APPLICATION.esbCell.Bus**.
8. Your JMS Activation Specification should look like [Figure 20](#):

Figure 20. Creating orderAS

General Properties

Administration

* Scope

* Name

* JNDI name

Destination

* Destination type

* Destination JNDI name

Message selector

Bus name

Acknowledge mode

Target inbound transport chain

9. Click **OK**.
10. Save your changes.
11. Close the Administrative Console.

Section 5. Building a mediation module

The mediation module is required to perform two functions:

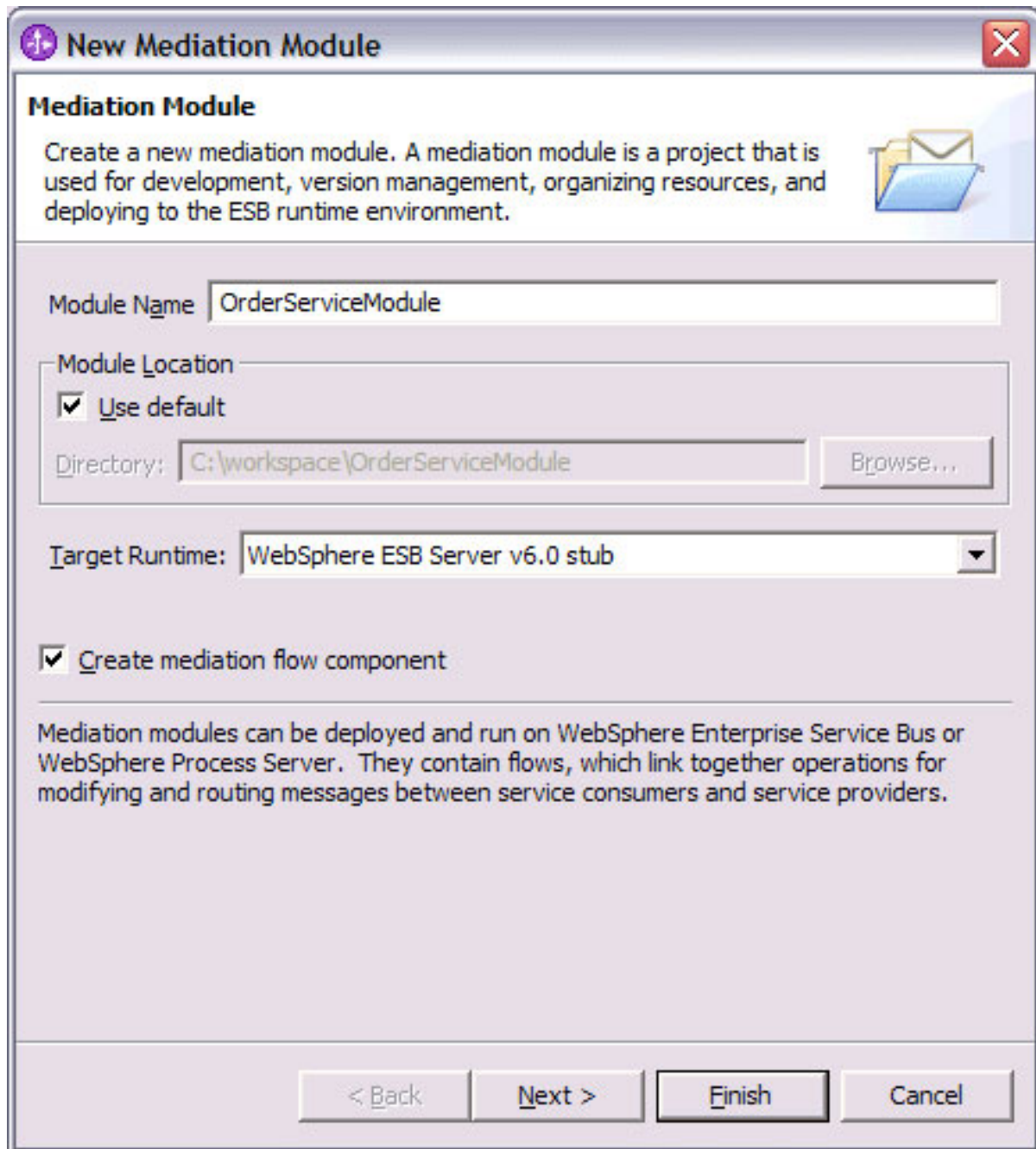
1. Convert a JMS request to a SOAP/HTTP request
2. Map the OrderWebService interface to an interface that can be used by the JMS client

Creating a mediation module

The mediation module contains the logic to connect a service requester to a service provider.

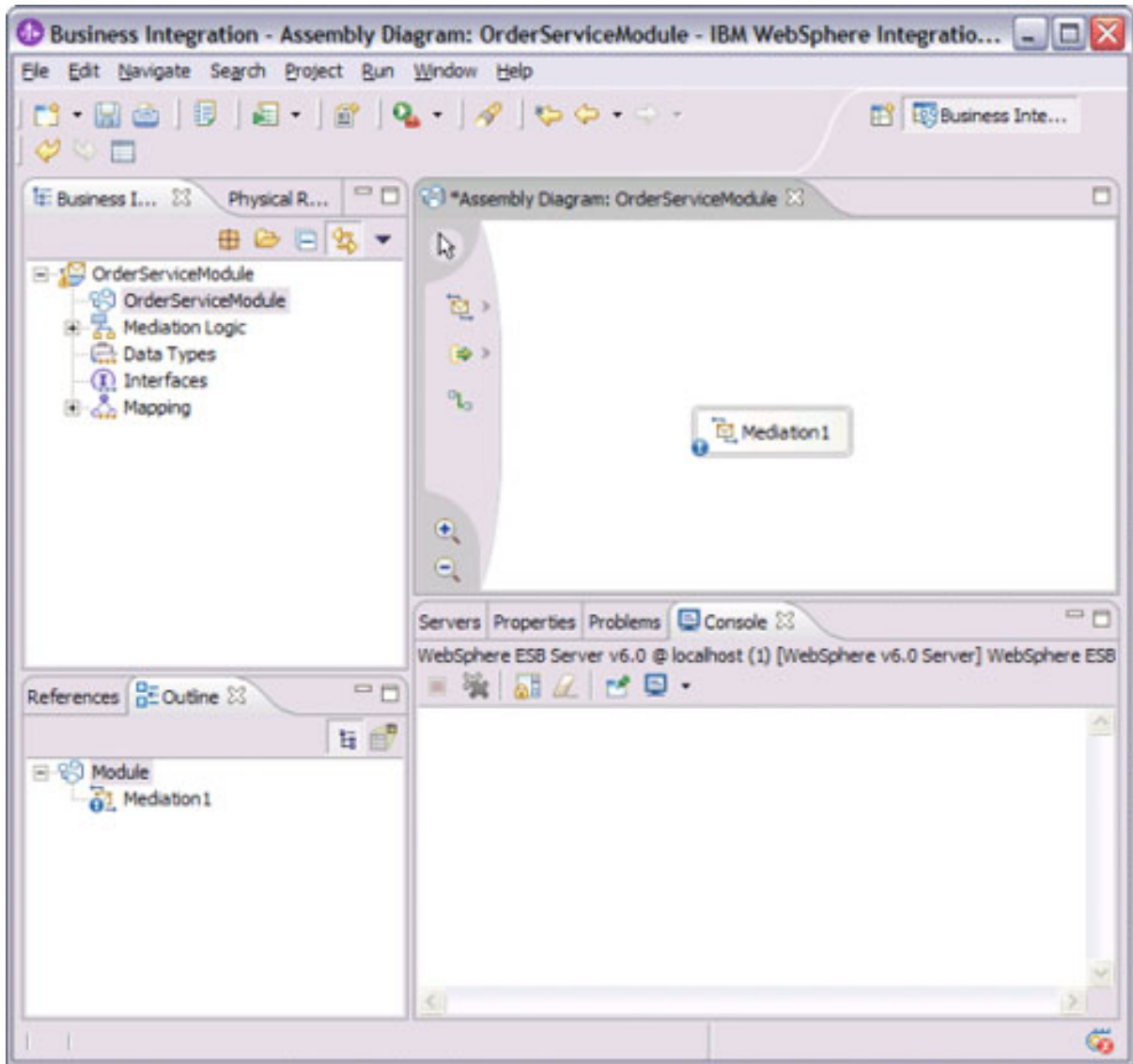
1. In the **Business Integration** view, right click and select **New > Mediation Module**. This opens the New Mediation Module wizard.
2. Set the **Module Name** to `OrderServiceModule`.
3. Ensure the **Target Runtime** is set to **WebSphere ESB Server v6.0 stub**, as shown in [Figure 21](#):

Figure 21. New mediation module



4. Click **Finish**.
5. Expand the **OrderServiceModule** project and double click on **OrderServiceModule** to open the Assembly Editor, as shown in [Figure 22](#):

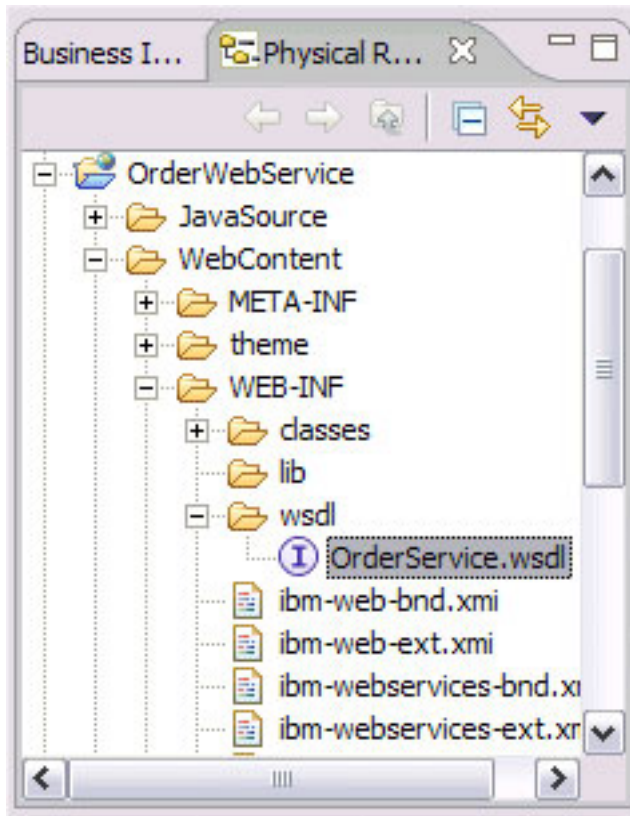
Figure 22. Assembly editor



To make the OrderService available to OrderServiceModule we need to copy OrderService.wsdl into the module:

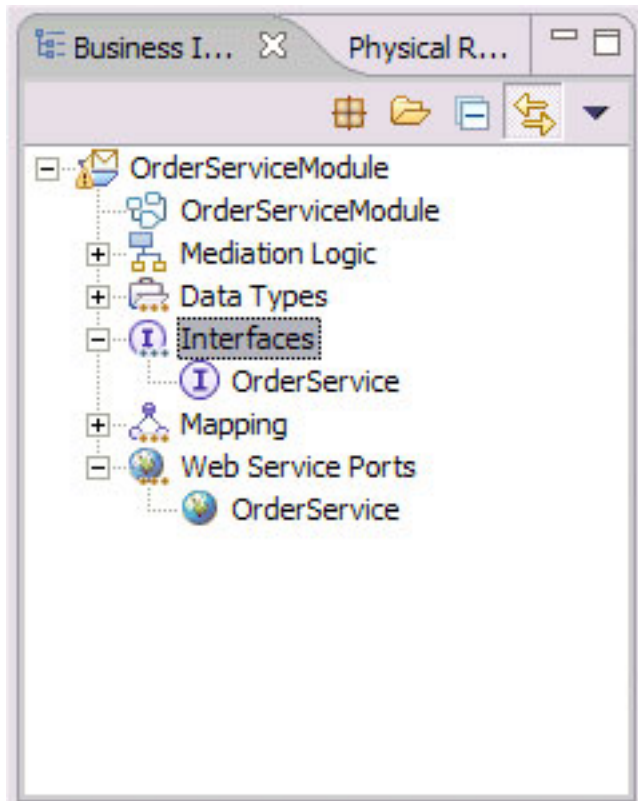
6. Switch to the **Physical Resources** view.
7. Navigate to the **OrderWebService** project and find **WebContent/WEB-INF/wsdl/OrderService.wsdl**, as shown in [Figure 23](#):

Figure 23. OrderService.wsdl



8. Right click and select **Copy**.
9. Switch back the **Business Integration** view.
10. Right click on **Interfaces** and select **Paste**.
11. The interface and port are both defined in the OrderService.wsdl so **OrderService** will be viewable under **Interfaces** and **Web Service Ports**, as shown in [Figure 24](#):

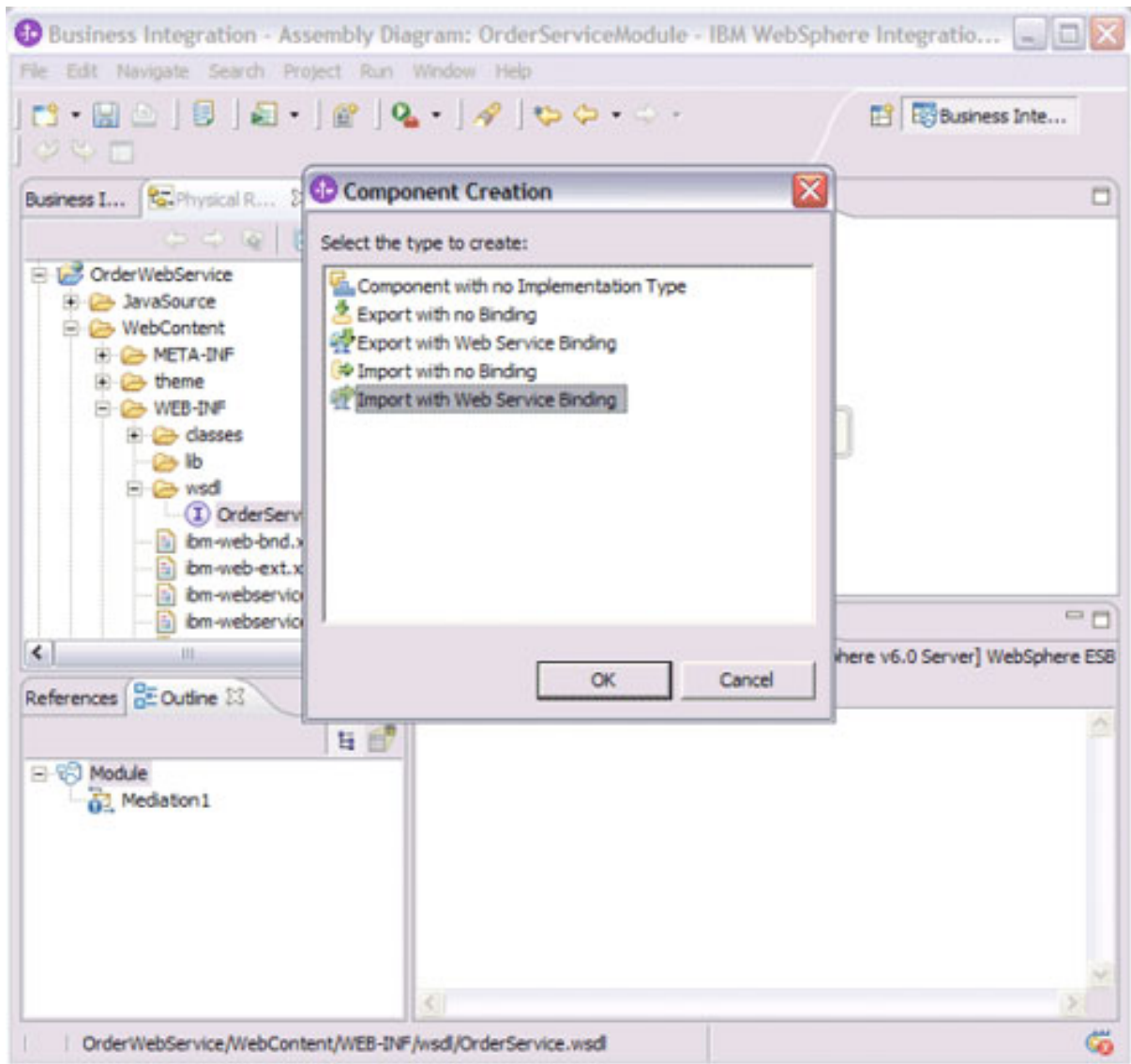
Figure 24. Importing OrderService.wsdl



Creating an import

1. Drag the **OrderService** interface onto the Assembly Editor.
2. In the **Component Creation** dialog, choose **Import with Web Service Binding**, as shown in [Figure 25](#):

Figure 25. Dragging OrderService



3. Click **OK**.

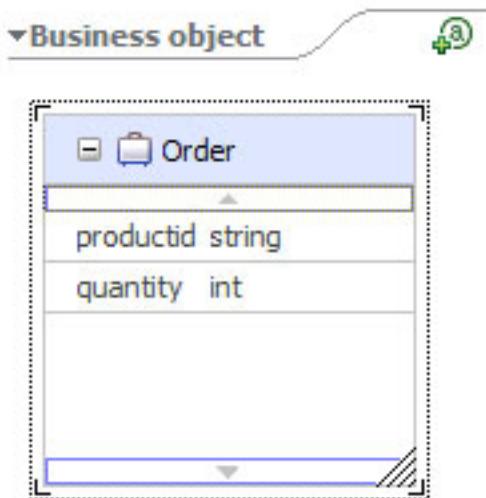
Creating a new interface

Before creating the export we need to define the Interface used by the JMS client.

1. Right click on **Data Types** and select **New > Business Object**.
2. In the **Name** field enter `Order`.
3. Click **Finish**.

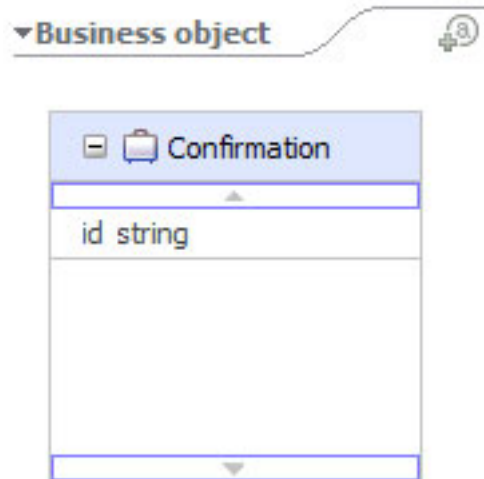
4. Click the **Add Attribute** button to add a String called `productid`.
5. Click the **Add Attribute** button to add an int called `quantity`.
6. The Order business object should look like [Figure 26](#):

Figure 26. Creating order business object



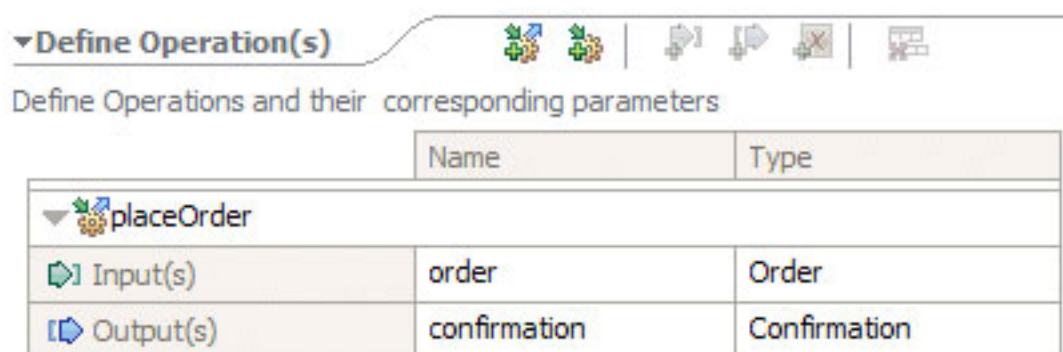
7. Save the order business object.
8. Right click on **Data Types** and select **New > Business Object**.
9. In the **Name** field enter `Confirmation`.
10. Click **Finish**.
11. Click the **Add Attribute** button to add a string called `id`.
12. The confirmation business object should look like [Figure 27](#):

Figure 27. Creating confirmation business object



13. Right click on **Interfaces** and select **New > Interface**.
14. In the **Name** field enter `JMSOrderService`.
15. Click the **Add Request Response Operation** button and name the operation `placeOrder`.
16. Click the **Add Input** button and name it `order` and select type **Order**.
17. Click the **Add Output** button and name it `confirmation` and select type **Confirmation**.
18. The `placeOrder` operation should look like [Figure 28](#):

Figure 28. Creating placeOrder operation



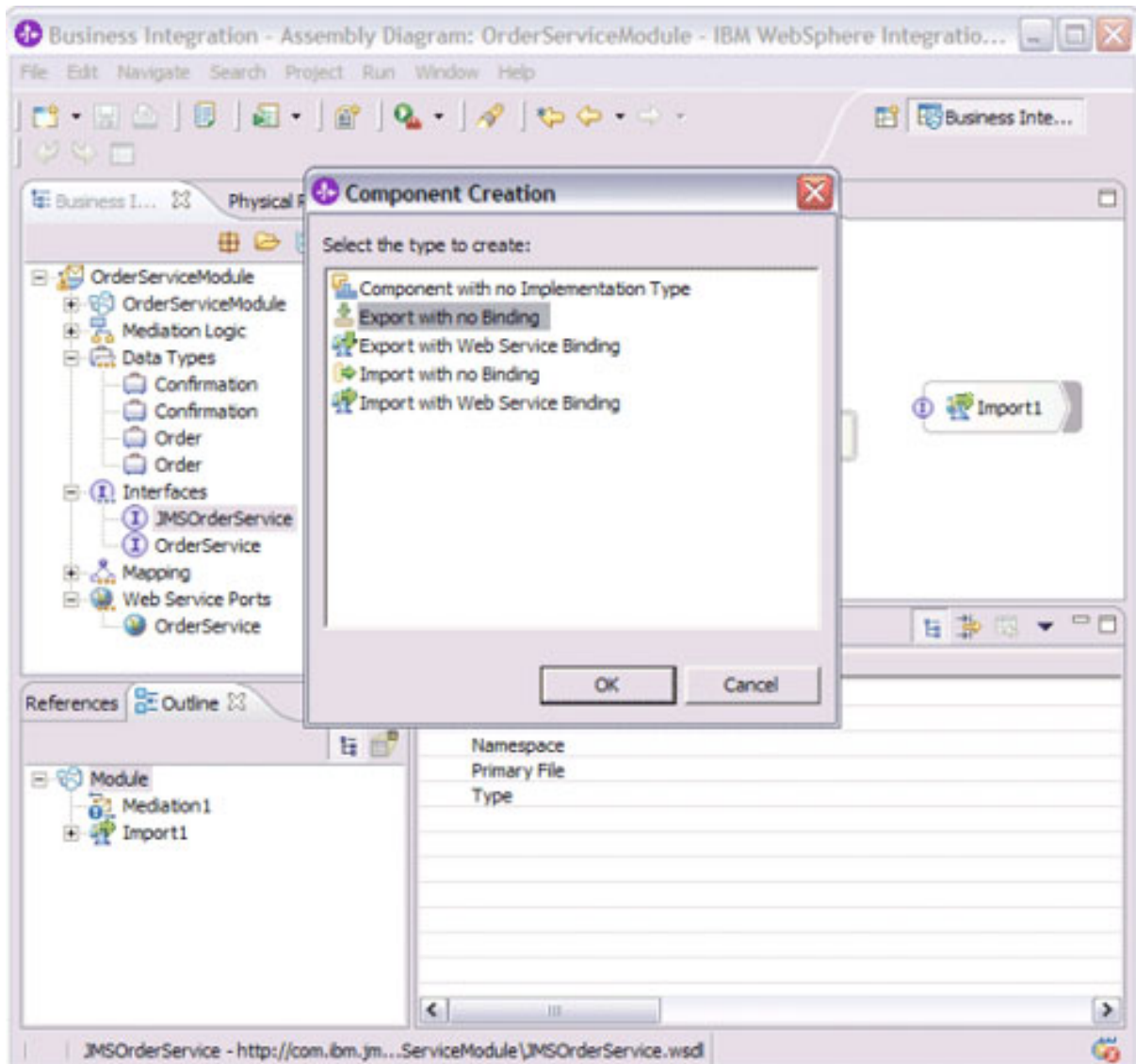
19. Save the **JMSOrderService** interface and close the Interface Editor.

Creating an export

Now we create the export that describes how mediation module will receive JMS request messages.

1. Drag the **JMSOrderService** onto the Assembly Editor.
2. In the Export Creation wizard select **Export with no Binding**, as shown in [Figure 29](#):

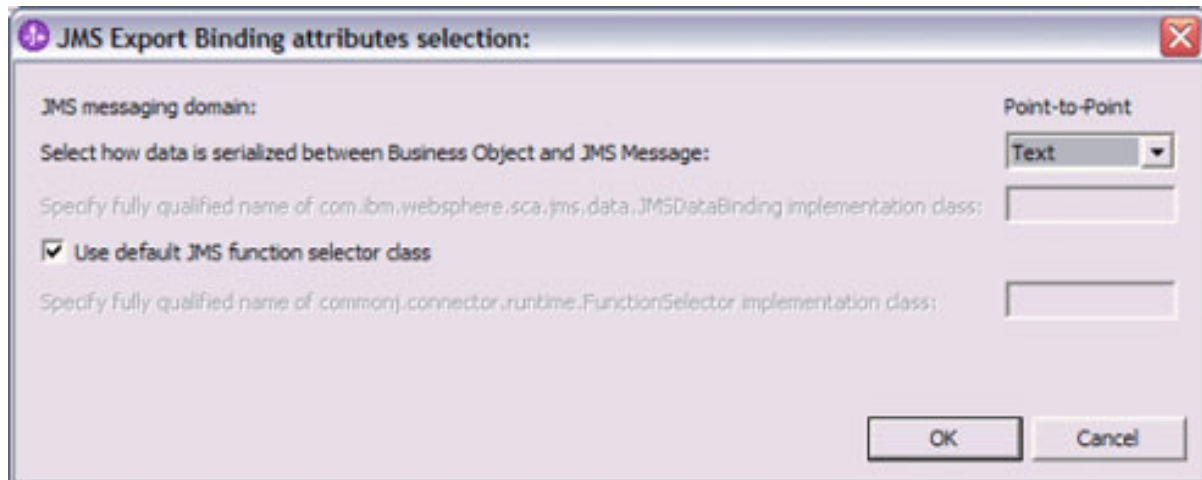
Figure 29. Dragging JMSOrderService



3. Click **OK**.
4. Right click on the newly created **Export1** and select **Generate Binding... > JMS Binding**.

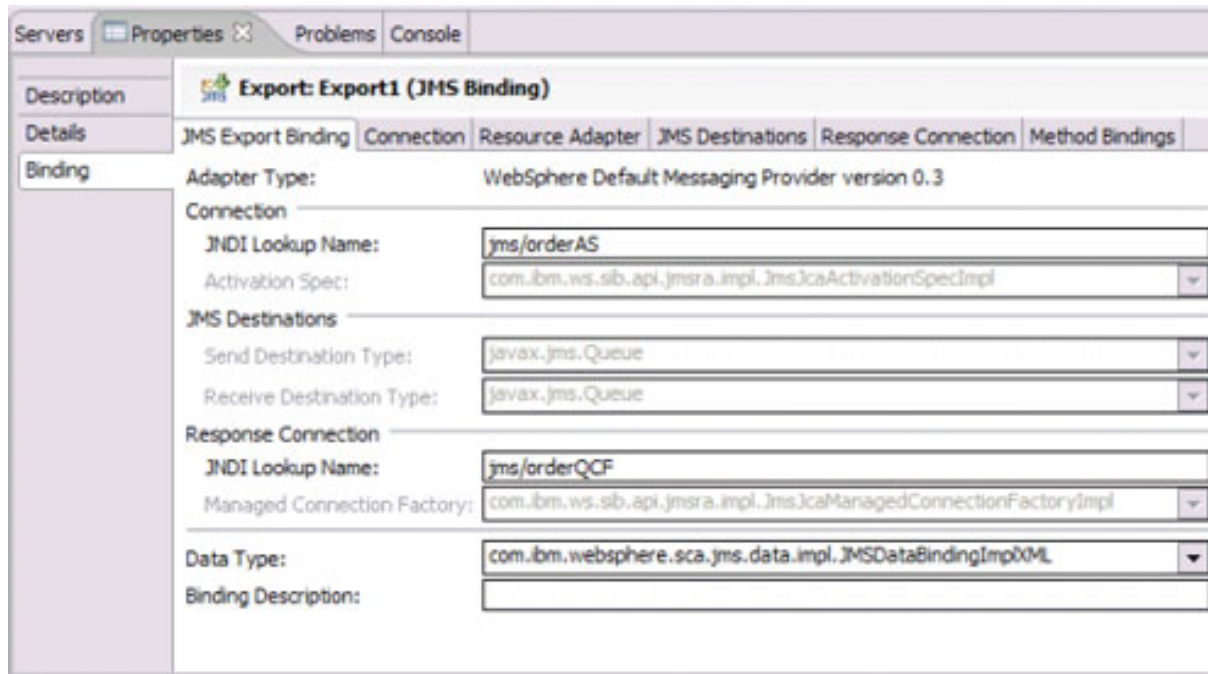
5. The **JMS binding attributes selection** dialog will open. From the **Select how data is serialized between Business Object and JMS Message** drop down box select **Text**, as shown in [Figure 30](#):

Figure 30. Generating JMS Binding



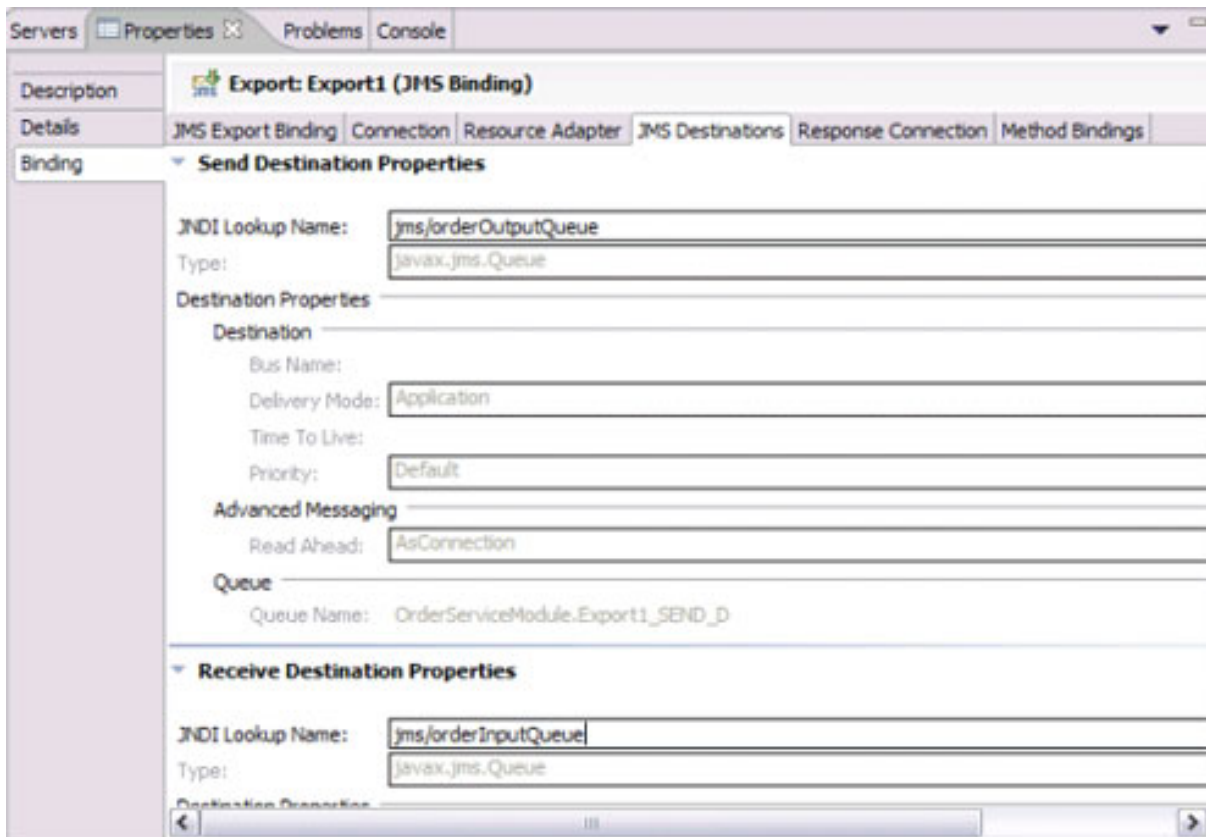
6. In the properties view of **Export1** select the **Binding** tab.
7. On this panel select the **JMS Export Binding** tab.
8. In the **Connection > JNDI Lookup Name** field enter `jms/orderAS`.
9. In the **Response Connection > JNDI Lookup Name** field enter `jms/orderQCF`.
10. The properties panel should look like [Figure 31](#):

Figure 31. JMS Export Binding



11. In the properties view select the **JMS Destinations** tab.
12. Expand **Send Destination Properties**.
13. In the **JNDI Lookup Name** field enter `.jms/orderOutputQueue`.
14. Expand **Receive Destination Properties**.
15. In the **JNDI Lookup Name** field enter `.jms/orderInputQueue`.
16. The properties panel should look like [Figure 32](#):

Figure 32. JMS Export Binding Destinations



17. Save the changes.

Wiring the module

1. Wire **Export1** to **Mediation1**.
2. Wire **Mediation1** to **Import1**.
3. OrderServiceModule should now look like [Figure 33](#):

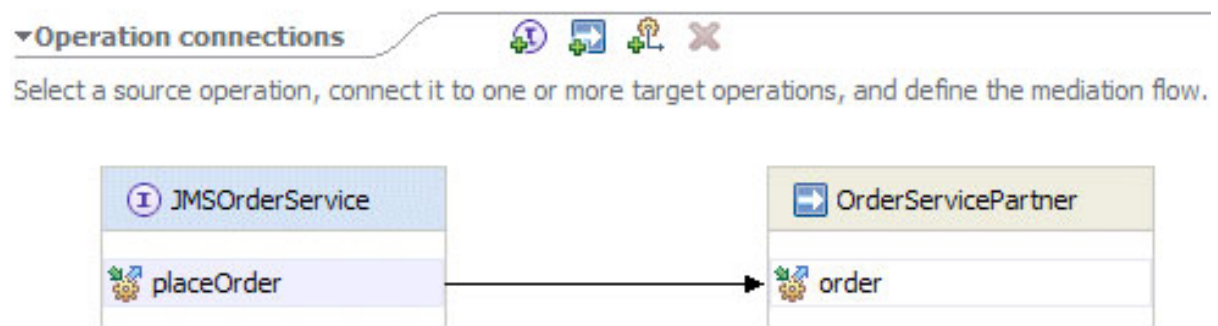
Figure 33. Order service module



Creating a mediation flow

1. Right click **Mediation1** and select **Generate Implementation**.
2. In the **Generate Implementation** dialog click **OK** to store the mediation flow in the default location.
3. In the **Operation connections** panel, wire **placeOrder** to **order**, as shown in [Figure 34](#):

Figure 34. Wiring operations



4. In the **Request** panel add a **XSLTransformation** mediation primitive to the palette.
5. Wire **JMSOrderService_placeOrder_Input** to the input terminal on **XSLTransformation1**.
6. Wire the output terminal on **XSLTransformation1** to **OrderServicePartner_order_Callout**.
7. The mediation flow should now look like [Figure 35](#):

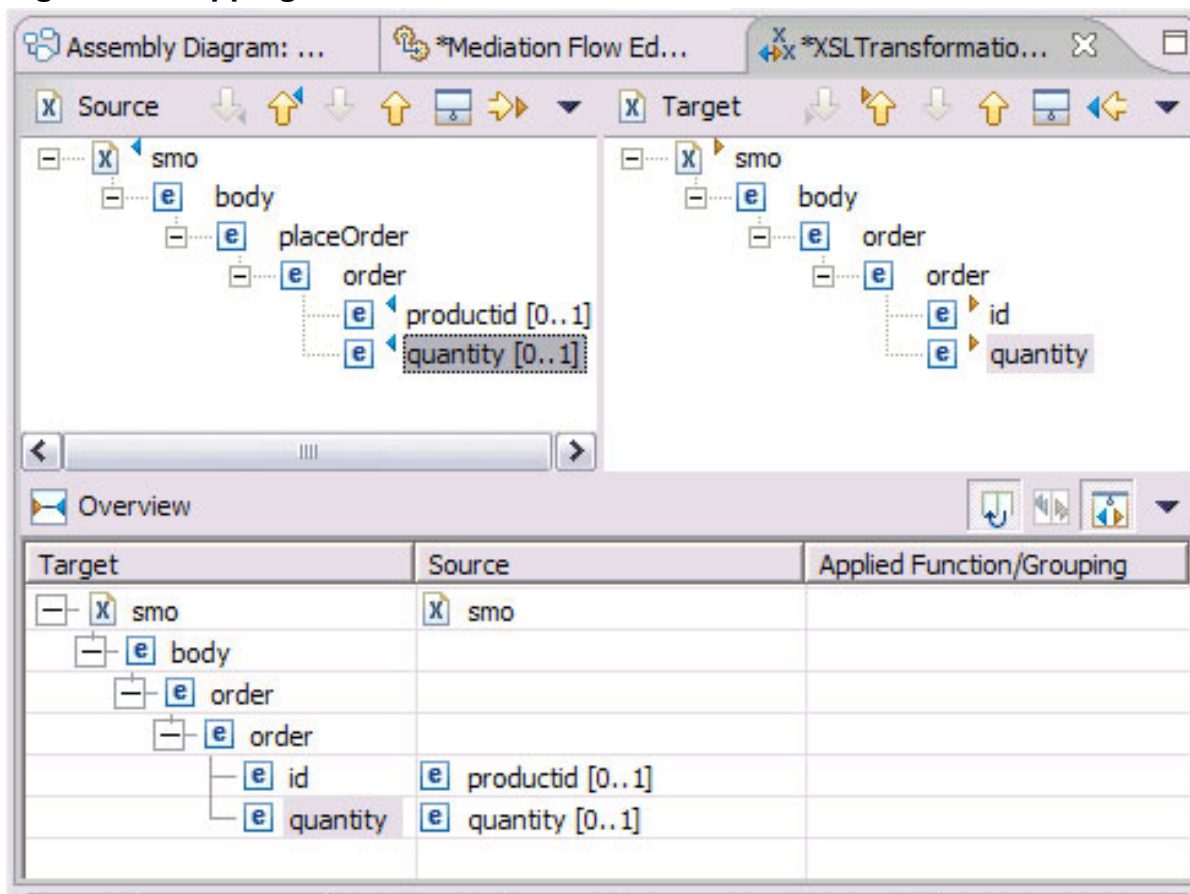
Figure 35. Wiring the request mediation flow



8. On the properties panel for **XSLTransformation1** mediation primitive,

- select the **Details** tab.
9. Press the **New...** button to create a new mapping file.
 10. On the **New XSLT Mapping** dialog click **Finish**. This opens the mapping editor.
 11. Expand the **body** tree on both message types until you can see the contents of the order.
 12. Drag **productid** from the source message to **id** on the target message.
 13. Drag **quantity** from the source message to **quantity** on the target message.
 14. The mapping editor should now look like [Figure 36](#):

Figure 36. Mapping editor for XSLTransformation1



15. Save the mapping file and close it.

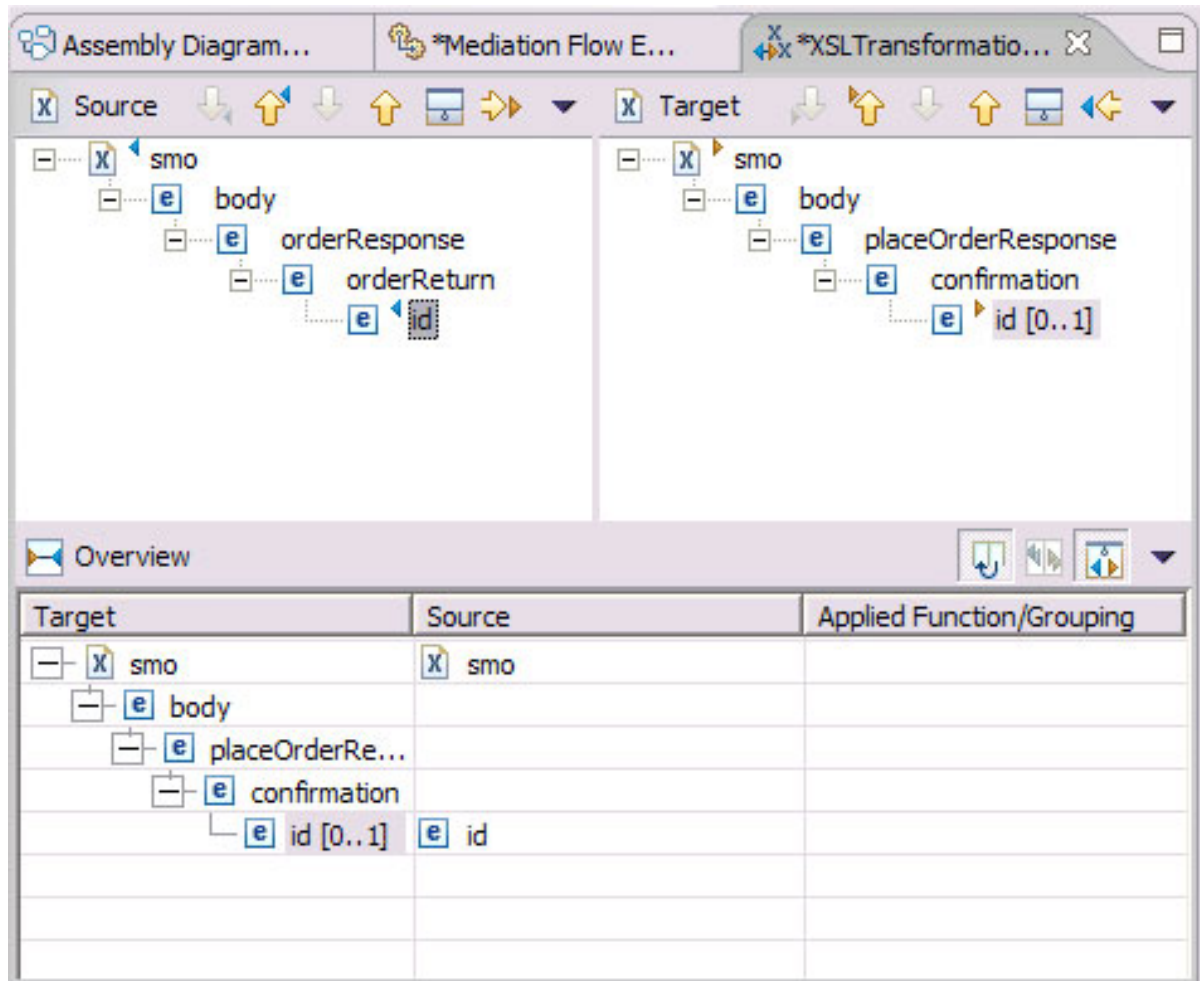
16. On the properties panel for **XSLTransformation1** mediation primitive, select the **Details** tab.
17. Click the **Regenerate XSL** to generate an xsl file from the mapping we just defined.
18. Click **OK**.
19. Switch to the **Response Flow** tab.
20. In the **Response** panel add a XSLTransformation mediation primitive to the palette.
21. Wire **OrderServicePartner_order_CalloutResponse** to the input terminal on **XSLTransformation1**.
22. Wire the output terminal on **XSLTransformation1** to **JMSOrderService_placeOrder_InputResponse**.
23. The mediation flow should now look like [Figure 37](#):

Figure 37. Wiring the response mediation flow



24. On the properties panel for **XSLTransformation1** mediation primitive, select the **Details** tab.
25. Press the **New...** button to create a new mapping file.
26. On the **New XSLT Mapping** dialog click **Finish**. This opens the mapping editor.
27. Expand the **body** tree on both message types until you can see the contents of the order.
28. Drag **id** from the source message to **id** on the target message.
29. The mapping editor should now look like [Figure 38](#):

Figure 38. Mapping Editor for XSLTransformation1



30. Save the mapping file and close it.
31. On the properties panel for **XSLTransformation1** mediation primitive, select the **Details** tab.
32. Click the **Regenerate XSL** to generate an xsl file from the mapping we just defined.
33. Click **OK**.
34. Save **Mediation1**.

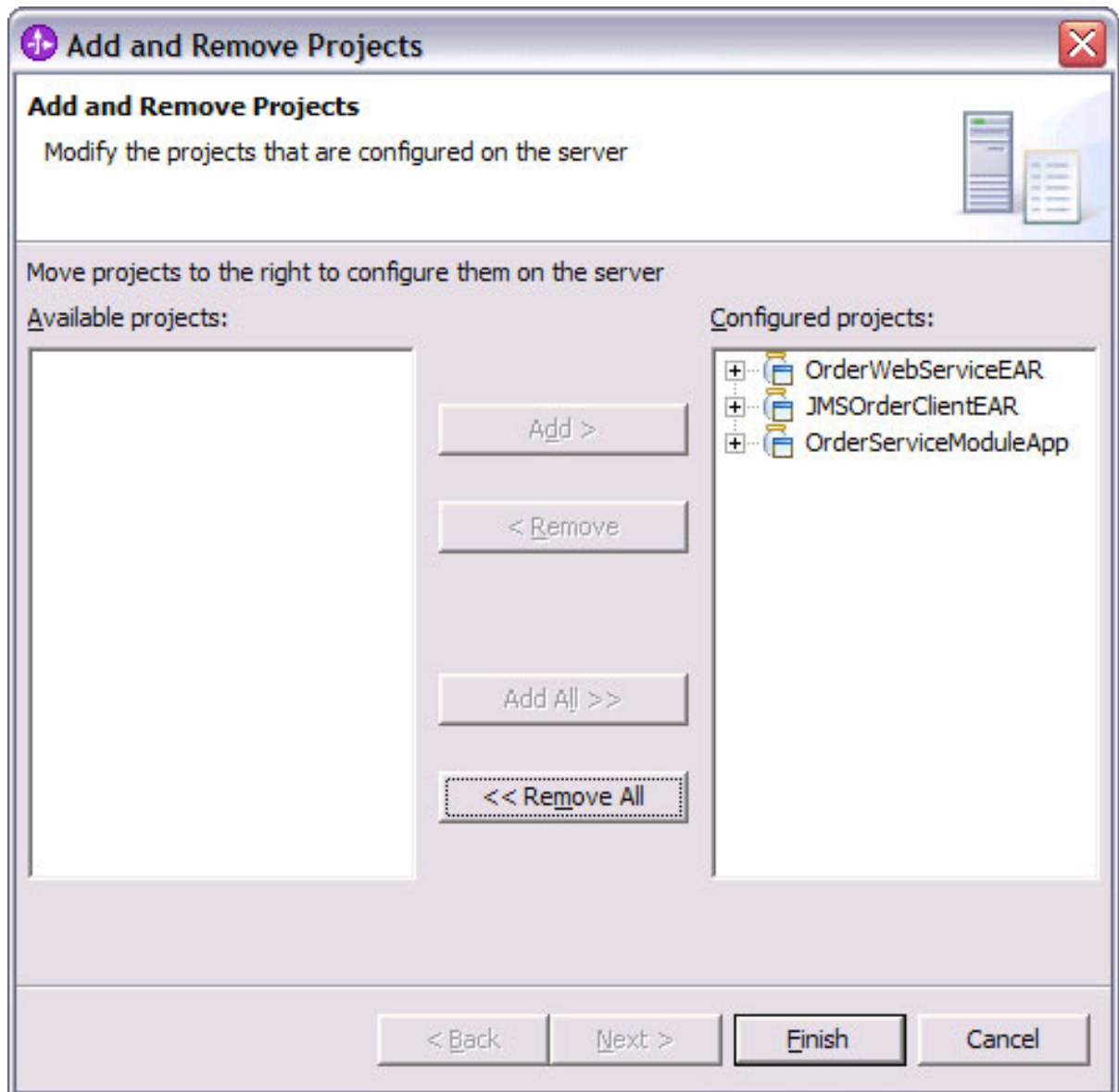
The creation of the Mediation Module is complete. This now transforms a JMS request message to a SOAP request message suitable for invoking the Order Web service.

Section 6. Testing the mediation module

Now we have the three components required to test the JMS to SOAP Web service invocation.

1. In the server view, right click on the WebSphere ESB server and select **Add and Remove Projects**.
2. Add all three projects, as shown in [Figure 39](#):

Figure 39. Adding Projects to server



3. Click **Finish**.
4. Restart the WebSphere ESB server, to ensure the activation specification is registered with the mediation module.
5. Open a Web browser and enter the URL `http://localhost:9080/JMSOrderClient/Order.jsp`. Order.jsp will be displayed, as shown in [Figure 40](#):

Figure 40. Order.jsp



6. Enter a Product ID of 12345.
7. Click the **Order** button.

If the Web service was invoked correctly a success message is displayed, as shown in [Figure 41](#):

Figure 41. Success message



Section 7. Conclusion

In this article we have used WebSphere Integration Developer and WebSphere ESB to develop and test:

1. A JMS client that can send JMS TextMessages containing a business object
2. A simple Java Web service that accepts Order business objects and returns Confirmation business objects
3. A mediation module that connects the JMS client to the Web service

Hopefully this article has given you a basic understanding of how to use these two products, to integrate JMS applications with Web services.

Downloads

Description	Name	Size	Download method
Project Interchange for mediation module	OrderServiceModule.zip	15KB	HTTP
Project Interchange for JMS client	JMSOrderClient.zip	9KB	HTTP
Project Interchange for Web service	OrderWebService.zip	18KB	HTTP

[Information about download methods](#)

Resources

- [WebSphere ESB product information](#)
Product descriptions, product news, training information, support information, and more.
- [WebSphere ESB Information Center](#)
A single Eclipse-based Web portal to all WebSphere ESB documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere ESB.
- [WebSphere ESB support](#)
A searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.
- [WebSphere Integration Developer product page](#)
A single Eclipse-based Web portal to all WebSphere Integration Developer documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere Integration Developer environment.
- [WebSphere Integration Developer information center](#)
Product descriptions, product news, training information, support information, and more.
- [WebSphere Integration Developer support](#)
A searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.
- [Getting started with WebSphere Enterprise Service Bus and WebSphere Integration Developer](#)
This article introduces developers to the IBM WebSphere Enterprise Service Bus server and its accompanying tooling, WebSphere Integration Developer.
- [Developing custom mediations for WebSphere Enterprise Service Bus](#)
This article introduces the use of custom mediations using the WebSphere Integration Developer V6 environment for WebSphere Enterprise Service Bus V6.

About the author

Philip Norton



Philip Norton is a software engineer at IBM Hursley Lab. He works on the WebSphere ESB Development team. His expertise includes Java and JMS for WebSphere MQ. He is a certified Java Programmer and he has a degree in Computer Science from Canterbury University. You can reach Philip at nortonp@uk.ibm.com