WebSphere Business Integration Adapter Development: An Integration Broker Deployment Example

- Developing, deploying and testing a custom adapter
- Developing, deploying and testing a technology adapter
- Building message sets and flows for adapters

Lee Gavin
Jeffrey Blight
Antti Lundstrom
Girolamo Palumbo
Liz Savoie
Chris Sparshott
Susumu Sugihara
Alberto Tatarano

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (March 2005)

This edition applies to Version 2, Release 4 of WebSphere Business Integration Adapters (product number 5724-G83).

This document created or updated on March 28, 2005.

© Copyright International Business Machines Corporation 2005. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Notices .................................................. ix
Trademarks ............................................. x

Preface .................................................... xi
The team that wrote this Redpaper ................................ xi
Become a published author ..................................... xiii
Comments welcome ......................................... xiii

Chapter 1. Overview of our scenario and applications ............ 1
1.1 Scenario background ..................................... 2
1.2 The requirements ........................................ 3

Chapter 2. Install and configure the scenario infrastructure ...... 7
2.1 Messaging infrastructure .................................... 8
2.2 The RedTenant Web application ............................. 8
2.3 Installing the RedTenant application ......................... 11
  2.3.1 Preparing the environment ............................. 12
  2.3.2 Creating the RedTenant application server .......... 13
  2.3.3 Configuring resources for the WebSphere MQ JMS provider .... 14
  2.3.4 Deploying the application ............................. 19
  2.3.5 A quick test ........................................... 21
2.4 Create the back-end application database ...................... 24
2.5 Create the back-end application ............................. 25

Part 1. Deploy and test a technology adapter ....................... 29

Chapter 3. Create the business objects and connector .......... 31
3.1 Front-end business objects .................................. 32
  3.1.1 Creating business objects with the ODA ............... 32
  3.1.2 Modifying business objects for use ................... 39
3.2 JMSCconnector ............................................. 51
  3.2.1 Create a JMS Connector ............................... 51
3.3 Connector Metaobjects for the data handler ................... 54
  3.3.1 Creating Metaobjects .................................. 56
3.4 Queue connection factory and queue objects ................ 58
3.5 Supported business objects for the connector ............... 61
3.6 Start the connector ....................................... 62
3.7 Unit test ................................................ 63
Part 2. Deploy and test a custom adapter for the back-end .......................... 75

Chapter 4. Object Discovery Agent .......................................................... 77
  4.1 Set up the custom ODA ................................................................. 78
  4.2 Generate business objects using the ODA ....................................... 79
    4.2.1 Completing the business objects ........................................... 87

Chapter 5. Package the custom adapter for distribution .............................. 95
  5.1 Connector naming conventions ...................................................... 96
  5.2 Define the connector ................................................................. 96
    5.2.1 Define with the Connector Configurator ................................... 97
    5.2.2 Define the connector configuration properties .......................... 99
  5.3 Create the WebSphere MQ objects ............................................... 103
  5.4 Create the connector environment ............................................... 103
    5.4.1 Preparing the connector’s directory ...................................... 104
  5.5 Create the startup script ......................................................... 104
    5.5.1 Startup script for our connector ......................................... 105
  5.6 Create the Windows shortcut ..................................................... 108

Chapter 6. Unit testing the connector ................................................... 111
  6.1 Start the connector ................................................................. 112
    6.1.1 Troubleshooting start up errors .......................................... 112
  6.2 Test interaction between connector and business objects .................. 114
    6.2.1 Add supported business objects .......................................... 114
    6.2.2 Using the Visual Test Connector ......................................... 115
    6.2.3 Prepare test data for request processing (RM_Tenant.Retrieve) .... 119
    6.2.4 Send test data to the test connector (RM_Tenant.Retrieve) .......... 128
    6.2.5 Send response to request (RM_Tenant.Retrieve) ....................... 130
    6.2.6 Prepare test data for processing (RM_Maintenance.Create) ......... 136
    6.2.7 Send test data (RM_Maintenance.Create) ............................... 137
  6.3 Test interaction between connector and application ........................ 140
    6.3.1 Send request business object (RM_Tenant.Retrieve) ................. 140
    6.3.2 Check response message received ....................................... 143
    6.3.3 Send request business object (RM_Maintenance.Create) ............ 144
    6.3.4 Create event ............................................................... 146
    6.3.5 Check event message sent .............................................. 147
  6.4 Conclusions on unit test .......................................................... 150

Part 3. Build and test the integration ..................................................... 151

Chapter 7. Deploy business objects to the Message Broker ....................... 153
  7.1 Front-end business objects ...................................................... 154
    7.1.1 Define Message Set ........................................................ 154
    7.1.2 Import schema definitions and create message definitions .......... 156
7.2 Back-end business objects ................................................. 166

Chapter 8. Build and test message flow for Retrieve .................... 173
  8.1 Retrieve processing scenario ........................................... 174
  8.2 Create a Message Flow project and Message Flows ................. 174
  8.3 RedTenant_to_RedMaint_Request ...................................... 176
  8.4 Deploy ................................................................. 188
  8.5 Unit test flow .......................................................... 196
  8.6 RedMaint_to_RedTenant_Response .................................... 200
  8.7 ResponseRetrieve .................................................... 202
  8.8 Deploy the Message Flows ............................................ 215
  8.9 Unit test flow .......................................................... 215
  8.10 End-to-end test ....................................................... 218

Chapter 9. Build and test message flows for Create ...................... 223
  9.1 Create a processing scenario ......................................... 224
  9.2 RedTenant_to_RedMaint_Request ...................................... 225
  9.3 Deploy the Message Flows ............................................ 243
  9.4 Unit test the flow ..................................................... 244
  9.5 RedMaint_RetrieveTenantByContent .................................. 246
  9.6 Deploy RedMaint_RetrieveTenantByContent .......................... 251
  9.7 Unit test RedMaint_RetrieveTenantByContent ....................... 252
  9.8 RedMaint_to_RedTenant_Response .................................... 253
  9.9 Deploy RedMaint_to_RedTenant_Response ............................ 260
  9.10 Unit test RedMaint_to_RedTenant_Response ......................... 260
  9.11 End-to-end test ....................................................... 260

Chapter 10. Build and test message flow for Update ..................... 263
  10.1 RedTenant_to_RedMaint_Request ..................................... 264
  10.2 RedMaint_to_RedTenant_Response ................................... 267

Chapter 11. Gathering data from the runtime ............................. 269
  11.1 Using WebSphere Business Integration Monitor .................... 270
  11.2 Our business process ................................................ 274
  11.3 Setting up the Monitor environment ................................ 281
  11.4 Setting up the Message Broker environment ....................... 287
  11.5 Using the Monitor data .............................................. 301
      11.5.1 Workflow Dashboard ............................................ 301
      11.5.2 Business Dashboard ............................................ 307

Appendix A. Unit test traces .................................................. 313

Appendix B. Where are my messages? ...................................... 333
Some cheat sheets ............................................................... 334
Appendix C. Additional material ........................................... 337
Locating the Web material ...................................................... 337
Using the Web material ......................................................... 337
  System requirements for downloading the Web material .......... 338
  How to use the Web material ............................................... 338

Related publications .......................................................... 339
IBM Redbooks ........................................................................ 339
How to get IBM Redbooks ..................................................... 339
Help from IBM ..................................................................... 339
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 illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.
Trademarks

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

- CICS®
- CrossWorlds®
- DB2®
- Domino®
- @server®
- ♫server♫
- ETE™
- ibm.com®
- IBM®
- IMS™
- Lotus®
- Notes®
- Redbooks (logo)™
- Redbooks (logo) ♫™
- Redbooks™
- RMF™
- SupportPac™
- Tivoli®
- WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

This IBM Redpaper is the final in a series of Redpapers called *WebSphere Business Integration Adapter Development*. This series discusses the development and deployment of WebSphere® Business Integration Adapters.

In this final Redpaper, *An Integration Broker Deployment Example*, we take the reader through the final phase of the development life cycle of a custom adapter: final testing, then deploying to and testing with an Integration Broker. In our scenario we use the WebSphere Business Integration Message Broker as our Integration Broker. We include sample code for you to download, as well as discuss our development, initial testing and deployment steps.

Look for the other IBM Redpapers in this series:

- *WebSphere Business Integration Adapter Development: An Introduction to the Basics*, REDP-9119
- *WebSphere Business Integration Adapter Development: A Development Example*, REDP-9120

The team that wrote this Redpaper

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

**Lee Gavin** is a Consulting IT Specialist at the ITSO, Raleigh Center. She writes extensively and teaches IBM classes worldwide on all areas of the WebSphere® MQ Family, WebSphere Business Integration and Business Process Management. Before joining ITSO in May 2001, Lee worked for IBM Global Services, Australia.

**Jeffrey Blight** is an Senior IT Specialist from the IBM Hursley Lab. He joined IBM seven years ago and is currently working in the World Wide Web WebSphere Business Integration Competitive POC team. His areas of expertise include the EAI and Business to Business with the WebSphere Business Integration product portfolio. Previously he has worked within EMEA and UK Technical Pre-Sales, specializing in J2EE, Java™, XML and Internet technologies.
Antti Lundstrom is a Certified IT Specialist working the Software Group of IBM Finland. He currently specializes in Technical Pre-Sales Support for WebSphere Business Integration products. He joined IBM in 1997. Prior to that, he worked for a major Finnish bank.

Girolamo Palumbo is an IT Specialist at the Java Technology Center, IBM Semea Sud of Bari, Italy, which is organized under the IBM Business Consulting Services division. He has five years of experience in Web application integration in the banking industry. He holds a degree in computer science from the University of Bari, Italy. His areas of expertise include Java, J2EE Application Development, XML and related technologies, and the integration of enterprise systems (CICS®, IMS™).

Liz Savoie is an Advisory Software Engineer at the IBM WebSphere Business Integration Competency Center in Grand Rapids, Michigan. She has seven years experience in the field of information technology and has been working with WebSphere Business Integration solutions involving WebSphere MQ, WebSphere InterChange Server, and WebSphere Business Integration Adapters for the last four and a half years. She is an IBM CrossWorlds® V4.0 certified solutions expert. She holds a Bachelor of Science degree in computer science from Michigan Technological University.

Chris Sparshott is an IT specialist working as a WebSphere Technical Sales Representative. He has focused on the communication sector for the last two years and is based out of North Harbour, Portsmouth. He has eight years of experience within IT, including Service Management, Domino® Administration, and Pervasive. He has been involved in middleware for the last three years.

Susumu Sugihara is a Software Engineer at the On Demand Solution Center in Yamato, Japan. He has been working for the Proof-Of-Concept demonstration creation for the telecommunication and electronics industries with WebSphere InterChange Server and WebSphere Business Integration Adapters for the last two years. He holds a Master’s degree in electrical engineering from Kyoto University.

Alberto Tatarano is an IT Specialist in the IBM SEMEA Sud Java Technology Center in Bari, Italy, part of the IBM Business Consulting Services division. He has six years of experience in designing and developing e-business applications by using J2EE Object-Oriented technologies. He has client project experience in several sectors and application areas. He holds a degree in computing science from the University of Bari, Italy. Currently, he is involved with JTC Enterprise Application Integration.
Thanks to the following people for their contributions to this project:

Geert Van de Putte
Margaret Ticknor
ITSO, Raleigh Center
IBM US

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll 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 papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks™ in one of the following ways:

► Use the online Contact us review redbook form found at:
   ibm.com/redbooks

► Send your comments in an email to:
   redbook@us.ibm.com

► Mail your comments to:
   IBM® Corporation, International Technical Support Organization
   Dept. HZ8  Building 662
   P.O. Box 12195
   Research Triangle Park, NC 27709-2195
Overview of our scenario and applications

In this chapter, we give some background information about our scenario. We outline the current, fictitious environment and the integration project that our scenario addresses.

One of the most important initial tasks when undertaking an adapter development is to understand the environment and applications in that environment. It is also crucial to understand the requirements for integration.

Our company is a rental property management company that currently runs all of the operations for tenant maintenance through a call center. The management has decided to scale back call center operations because they are proving to be costly.
1.1 Scenario background

The current situation has the call center operators accessing each of the supporting applications separately. This causes delays, potential inconsistencies and redundant data storage. The RedTenant application stores data regarding tenants, as does the RedMaintenance application. As a result, apart from the duplication of data, they have the potential problem of inconsistent data across the two applications.

Currently when maintenance requests are created, if the work is not able to be carried out by the company's own tradespeople, the call-center operators must contact the external contracting company and manually update the details in the RedMaintenance application. This is time-consuming and potentially inconsistent. The call center operators must contact the in-house maintenance teams and external contractors to update the progress of maintenance requests. This is displayed in Figure 1-1.
1.2 The requirements

To rectify the situation, the management have tasked us with:

- Providing as much automation and synchronization as possible
- Giving the tenants the ability to query and submit their own maintenance requests using the RedTenant application with the Internet
- Making it flexible enough so that, at some point in the future:
  - Tenants will be able to use e-mail to request maintenance and receive updates.
  - We can use business-to-business (B2B) technologies to enable automated interaction with external contractors when the maintenance cannot be performed in-house.

The management have also tasked us with abiding by these restrictions:

- Not altering the existing application RedMaintenance application
- Minimal change to the RedTenant application
- Using an integration broker for any data modification, enhancement or semantic mediation or workflows requiring staff interactions, thus enabling an extensible hub and spoke architecture

  This shields each application from the complexities of the others, internal and external.

We have two applications in our scenario. The main application, the back-end, is the RedMaintenance application. With this application the staff of the management company currently maintain tenant and apartment information by directly inputting and updating maintenance requests taken by the call center.

The front-end application is the RedTenant application. With this browser-based application the call center operator can obtain details of a tenant and their apartment. The tenant identifies themselves by their Tenant Id. The call center operator can check the status of any current requests and enter the details of any newly created requests, based on the details from the RedMaintenance system.

The two applications were developed independently. The back-end is a packaged application. The front-end was developed by a small software company. The RedTenant application was developed in readiness for an integration project and is messaging aware. That is, it sends and receives its queries with queues. Prior to the integration project, a series of programs were used to access the database in which all of the information for this application was stored. The programs were written to format the data correctly for the RedTenant queues.
The two applications will be integrated using a combination of adapters and the WebSphere integration brokers.

The RedTenant application will require no more modification than a change of queue destination, which will send the requests to the hub, using an adapter for processing. The application will send data through a JMS adapter to the Broker, where the data will be transformed from the business object data sent from the front-end connector to the business object format required by the back-end connector.

Because the back-end application has an exposed API, we will use a custom adapter which was built for the back-end application. This adapter functions in a similar fashion to any other application adapter, in the sense that it is created specifically for use with this particular application utilizing the expose API.

**Note:** Details of this adapter and the development of it can be found in:

*WebSphere Business Integration Adapter Development: A Development Example*, REDP-9120

The complete integration scenario for our company is shown in Figure 1-2 on page 5. In this paper, we discuss the Message Broker components only. The other IBM Redpapers in this series are:

- *WebSphere Business Integration Adapter Development: An Introduction to the Basics*, REDP-9119
- *WebSphere Business Integration Adapter Development: A Development Example*, REDP-9120

The full integration solution will be detailed in a soon-to-be published IBM Redbook:

*WebSphere Business Integration Adapters: An Adapter Development and WebSphere Business Integration Solution*, SG24-6245.
Chapter 1. Overview of our scenario and applications

Figure 1-2  Complete integration solution
Install and configure the scenario infrastructure

In this chapter, we discuss the installation and configuration of the front-end and back-end applications we use in our scenario.

Important: Our sample applications are intended for illustrative purposes only and are not necessarily fully functional applications. Please keep in mind that the front-end application has no correlation of request and response messages. This means that the application will simply take the first response message it finds on the queue when a request has been made. It is vital when conducting end-to-end testing that you ensure all the required queues are empty, with no left-over messages from unit testing.
2.1 Messaging infrastructure

The queue managers for our applications are:

- REDBROKER, the broker queue manager listening on port 1421
- REDTENANT, the front-end queue manager listening on port 1416

2.2 The RedTenant Web application

The RedTenant Web application is the front-end of the RedMaintenance application, a stand-alone Java application for property management. With this Web interface, a tenant can retrieve his or her apartment’s maintenance information.

To use the application, the tenant does the following:

1. The application starts with the login page. The URL for the application is:

   http://localhost:9080/RedTenant/index.jsp

   The tenant has to enter a valid tenant ID in order to retrieve the apartment details and maintenance list. Figure 2-1 shows the login page.

   ![Figure 2-1 The login page](image)

   If an error occurs, the application returns the login page with an error message as shown in the Figure 2-2 on page 9.
2. If a valid Tenant ID exists into the database, the application will display a page, as shown in the Figure 2-3 on page 10, with a list of the tenant's apartments. For each apartment, it shows the maintenance list.
3. To see the maintenance details, the tenant selects the maintenance instances from the combination box. Clicking the **details** button opens a pop-up with the requested information, as in Figure 2-4.

![Completed maintenance information pop-up window](image)

**Figure 2-4** Completed maintenance information pop-up window
4. Finally, the tenant can create a new maintenance request for a given apartment by clicking the **new** button in Figure 2-3 on page 10. A pop-up window will appear with a text area for the maintenance description and a button to send the request. See Figure 2-6.

2.3 Installing the RedTenant application

The RedTenant application is a Web application (level 1.3) which is compliant with J2EE and based on the Apache Struts framework (version 1.1). This application accesses the back-end system through a business integration...
system, sending XML messages to JMS message queues. We need four connections to the broker, two for the tenant information retrieval (request and response) and the other two connection for the new maintenance creation (request and response).

The software requirements for the RedTenant installation are:

- WebSphere Application Server V5.1
- WebSphere MQ V5.3.0.2

**Note:** If you want to install IBM WebSphere Application Server on the same host as WebSphere MQ, you can install WebSphere MQ, with the Server and Java Messaging features, then install WebSphere Application Server without the Embedded Messaging Server option.

We used WebSphere Studio Application Developer Version 5.0.1 to develop our Web application. All the phases of the RedTenant application are built in a single Web project named RedTenantWEB. The entire project is exported to an EAR file named RedTenantEAR.ear. We use this file to deploy the Web application on WebSphere Application Server.

**Note:** The RedTenantEAR.ear file can be found in the Additional Materials in the RedTenant folder.

The Administrative Console is the simplest way to install an application in WebSphere. It provides the necessary wizards to create, remove, start and stop applications.

To launch the Administrative Console, select **Start → Programs → IBM WebSphere → Application Server v5.1 → Administrative Console**. If WebSphere global security is enabled, enter a valid User ID. If security is not enabled, do not enter anything.

**Note:** In this section, we assume that a connection is made to the Administrative Console installed in the Network Deployment node.

### 2.3.1 Preparing the environment

To prepare for the RedTenant application installation, do the following:

- Create an application server to host the application.
- Configure the necessary resources for the WebSphere MQ JMS Provider.
2.3.2 Creating the RedTenant application server

When WebSphere Application Server is installed, a default server, server1, is created automatically. Although the RedTenant application could be deployed to this default server, we recommend that you create a separate server for the RedTenant application. The default server is intended to run the WebSphere samples and serve as the server template.

To create the application server, do the following:

1. Select **Servers → Application Servers** in the navigation pane.

2. Click the **New** button.

3. Provide the application server name, for example RedTenantServer, and click **Next**.

4. The next window, Figure 2-7, lets you review the current application server settings. From there, you can go back to the previous page or confirm the settings and complete the application server creation by clicking **Finish**.

![Figure 2-7 Creating the RedTenant application server](image-url)
2.3.3 Configuring resources for the WebSphere MQ JMS provider

The next step is to create the necessary JMS resources for the WebSphere MQ JMS Provider. For the RedTenant application, we need four connections to the back-end system. We use two connections, request and response, for the tenant information retrieval and the other two connections for the new maintenance creation. Five resources must be created: a queue connection factory and four queues. JMS resources can be created using the Administrative Console.

To create the connection factory, complete the following steps:

1. Select **Resources → WebSphere MQ JMS Provider** in the navigation pane.
2. In the content pane, select **WebSphere Queue Connection Factories** in the Additional Properties table as shown in Figure 2-8.

3. Click **New** and provide the information specified in Table 2-1 on page 15.
Table 2-1  Queue connection factory properties for RedTenantCF

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>RedTenantCF</td>
<td>The queue connection factory name</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jms/RedTenantCF</td>
<td>The JNDI name for the resource</td>
</tr>
<tr>
<td>Queue Manager</td>
<td>REDTENANT</td>
<td>The name of the WebSphere MQ queue manager for this connection factory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connections created by this factory connect to that queue manager.</td>
</tr>
</tbody>
</table>

Figure 2-9 shows the New Connection Factory table.

Figure 2-9  Creating a connection factory

4. Click **OK**.

The WebSphere MQ Queue Connection Factories table will show the following information in Figure 2-10 on page 16.
To create the first queue destination, do the following:

1. Select Resources → WebSphere MQ JMS Provider entry in the navigation pane.
2. In the content pane, select WebSphere Queue Destinations in the Additional Properties table. See the Figure 2-8 on page 14.
3. Click New and provide the information specified in Table 2-2.

### Table 2-2  Queue destination properties for RTReq1Q

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>RTReq1Q</td>
<td>The queue destination name</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jms/RTReq1Q</td>
<td>The JNDI name for the resource.</td>
</tr>
<tr>
<td>Base Queue Name</td>
<td>RTREQ1Q</td>
<td>The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.</td>
</tr>
<tr>
<td>Base Queue Manager Name</td>
<td>REDTENANT</td>
<td>The name of the WebSphere MQ queue manager to which messages are sent.</td>
</tr>
</tbody>
</table>
4. Click **OK**.

5. Complete the configuration of the other three queue destinations by providing the information specified in the Table 2-3, Table 2-4 on page 18, and Table 2-5 on page 18.

**Table 2-3  Queue destination properties for RTRes1Q**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>RTRes1Q</td>
<td>The queue destination name</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jms/RTRes1Q</td>
<td>The JNDI name for the resource.</td>
</tr>
<tr>
<td>Base Queue Name</td>
<td>RTRES1Q</td>
<td>The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Base Queue Manager Name</td>
<td>REDTENANT</td>
<td>The name of the WebSphere MQ queue manager to which messages are sent.</td>
</tr>
</tbody>
</table>

Table 2-4  Queue destination properties for RTReq2Q

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>RTReq2Q</td>
<td>The queue destination name</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jms/RTReq2Q</td>
<td>The JNDI name for the resource.</td>
</tr>
<tr>
<td>Base Queue Name</td>
<td>RTREQ2Q</td>
<td>The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.</td>
</tr>
<tr>
<td>Base Queue Manager Name</td>
<td>REDTENANT</td>
<td>The name of the WebSphere MQ queue manager to which messages are sent.</td>
</tr>
</tbody>
</table>

Table 2-5  Queue destination properties for RTRes2Q

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>RTRes2Q</td>
<td>The queue destination name</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jms/RTRes2Q</td>
<td>The JNDI name for the resource.</td>
</tr>
<tr>
<td>Base Queue Name</td>
<td>RTRES2Q</td>
<td>The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.</td>
</tr>
</tbody>
</table>
6. Once you have completed all of the queue destination entries, the WebSphere MQ Queue Destinations table shows the following information in Figure 2-12.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Queue Manager Name</td>
<td>REDTENANT</td>
<td>The name of the WebSphere MQ queue manager to which messages are sent.</td>
</tr>
</tbody>
</table>

![Table of WebSphere MQ Queue Destinations]

Figure 2-12 The WebSphere MQ Queue Destinations

### 2.3.4 Deploying the application

We have created the JMS resources the RedTenant application requires. You can now follow the steps below to deploy application using the Administrative Console.

1. Select **Applications → Install New Application** in the navigation pane. See Figure 2-13 on page 20.
2. Enter the full path name of the enterprise application file, for example: C:\SG249345\RedTenantEAR.ear, and click **Next**.

3. Click **Next** in the next two pages to complete Step 1. In our example, we take the default options for the installation.

4. The next two panes show the mapping between resource references and resources. We have already mapped the JMS resources used by the Web module. Click **Next** in both panes.

5. Choose the Virtual Host for the RedTenant Web module, or take the default. Click **Next**.

6. Select the **RedTenantWEB** module, select **RedTenantServer** as shown in Figure 2-14 on page 21 and click **Apply**. Click **Next**.
Figure 2-14  Mapping the RedTenantWEB module to the RedTenantServer

7. Click **Finish** to complete the deployment.

You can now start the RedTenantServer application server and test the RedTenant Web application by invoking `http://<host_name>/RedTenant`. Make sure that the WebSphere MQ queue manager has been started.

**2.3.5 A quick test**

Start with the login screen in Figure 2-15 on page 22. To test the application, do the following:
1. If you use the setup materials for the database for the back-end application, discussed in 2.4, “Create the back-end application database” on page 24, you will see from the sample database entries that there is a tenant ID of 100 in the database. This is the tenant ID that we will be using for all of the testing.

2. Enter the tenant ID into the front-end application, as in Figure 2-16.

3. Because we currently do not have any connectivity to any of the adapters or the other applications, this request will time out and error due to the lack of
response from a back-end. However, it will put a request message on to a queue that will normally be sent.

4. Use RHFUTIL, which is provided in the Additional Materials. See Appendix C, “Additional material” on page 337 for information about how to obtain the materials for this paper.

**Note:** If you are not familiar with RHFUTIL, it is, quite simply, the handiest tool around for handling WebSphere MQ messages.

5. Enter the queue manager, REDTENANT and queue name, RETREQ1Q. Later, this local queue will change to point to the EVENT queue for the JMS connector, but for now we leave it.

6. Select **read message**. Do not browse because we do not want to retain this message.

7. Look at the message data in XML format to see that a message has been created, ready for sending.

---

![Figure 2-17 Message data on queue from the front-end](image)

---

Chapter 2. Install and configure the scenario infrastructure  23
2.4 Create the back-end application database

In this section, we deploy the back-end database and application.

Copy the folder SG246345 from the Additional Materials to your machine.

Our database for the back-end is SG246345. To create and populate the database with sample data, do the following:

1. Open a DB2® command window.
2. Navigate to the schemas folder of SG246345.
3. Enter:

   \texttt{CreateSG246345DB DB2 db2admin itso4you}

   \texttt{DB2} is the DB2 instance name. \texttt{db2admin} is the db2 administrator user ID and \texttt{itso4you} is the db2 administrator password.

4. Once you have created the database, you can choose to populate the database with the sample data we use for the scenario. Using your favorite DB2 tools, use the PopulateSampleData.txt file.

   We imported the file as a script into the DB2 Command Center and ran it from there. See Figure 2-18.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{import_script.png}
\caption{Import script}
\end{figure}
5. Using your favorite DB2 tool, check that the database exists, with the correct tables and the sample data. We used the DB2 Command Center in Figure 2-19.

<table>
<thead>
<tr>
<th>TENANTID</th>
<th>NAME</th>
<th>EMAILID</th>
<th>STATUS</th>
<th>CREATION</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Lee Gavin</td>
<td><a href="mailto:leegavin@uk.ibm.com">leegavin@uk.ibm.com</a></td>
<td>A</td>
<td>Aug 25, 2001</td>
<td>Aug 25, 2001</td>
</tr>
<tr>
<td>120</td>
<td>Redbook Reader</td>
<td><a href="mailto:redbookreader@us.ibm.com">redbookreader@us.ibm.com</a></td>
<td>A</td>
<td>Aug 25, 2001</td>
<td>Aug 25, 2001</td>
</tr>
</tbody>
</table>

Figure 2-19   Some sample data

### 2.5 Create the back-end application

Our back-end application is a Java application which utilizes RMI. To create the back-end application, do the following:

1. The first thing we need to do is to ensure that the application configuration properties are correct.

2. In the RedMaintenance folder of the SG246345 directory, open the file named config with a text editor. See Example 2-1.

   **Example 2-1  Config settings**

   ```
   # Configuration properties for AppStart
   databaseMaxConn=5
   databaseURL=jdbc:db2:SG246345
   databaseUserName=db2admin
   databaseUserPwd=itso4you
   jdbcDriverClass=COM.ibm.db2.jdbc.app.DB2Driver
   ```

   3. Modify the database password to ensure that it is correct.

   4. Save and close this file.

   5. We now need to start the rmiregistry.
6. Open a command window and type:
   \texttt{rmiregistry}

7. Leave this window open in the background.

8. Create a shortcut on the desktop for starting the application engine, as shown in Figure 2-20. Ensure that the start directory is RedMaintenance.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{redmaint_shortcut.png}
\caption{RedMaint shortcut}
\end{figure}

\textbf{Note:} The \texttt{rmiregistry} command creates and starts a remote object registry on the specified port on the current host. If the port is omitted, the registry is started on port 1099. The \texttt{rmiregistry} command produces no output and is typically run in the background. A remote object registry is a bootstrap naming service that is used by RMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations.

The registry is typically used to locate the first remote object on which an application needs to invoke methods. That object in turn will provide application-specific support for finding other objects.
9. Using this shortcut, start the application.

10. You will see in the command window the startup parameters we are using for our application. See Example 2-2. Leave this window open.

Example 2-2  Start window

C:\SG246345\RedMaintenance>start_engine
Config file is:.\Config:
Starting Auto thread
-- listing properties --
databaseUserName=db2admin
databaseMaxConn=5
jdbcDriverClass=COM.ibm.db2.jdbc.app.DB2Driver
databaseURL=jdbc:db2:SG246345
databaseUserPwd=itso4you
***** Driver Class is:COM.ibm.db2.jdbc.app.DB2Driver:

11. You will also see a GUI window, Figure 2-21, that can is used for tracing application activity and for problem determination.

Figure 2-21  GUI log

We now have our basic application components in place, so we can move on to the integration of these components and initial testing of our adapter environments.
Deploy and test a technology adapter

In this part of the book, we deploy and test an adapter for the front-end application, RedTenant. We also develop business objects for the front-end.
Create the business objects and connector

In this chapter we create the business objects required for the front-end application and JMS connector. These business objects will transport data between the front-end application and the Message Broker.

We installed the JMS Adapter and the XML DataHandler and the XML ODA for use in this chapter.

Note: We have included repos files of the components in this step for you to use as an alternative to manually creating all of the business objects. These can be found in the Additional Materials in the ReposJarFiles folder. See Appendix C, “Additional material” on page 337 for information on how to obtain the materials for this book.
3.1 Front-end business objects

The front-end Web application, RedTenant, uses schema-based XML documents for its messages. We can use the XML ODA to assist in building the application-specific business objects for these messages by utilizing the schema definitions for these messages.

**Note:** The repos file WebTenant.jar contains these business objects.

3.1.1 Creating business objects with the ODA

To create business objects do the following:

1. Start the ODA. From the start menu select:
   - IBM WebSphere Business Integration Adapters → Adapters → Object Discovery Agent → XML Object Discovery Agent
2. Navigate to the System Manager.
3. Right-click **Integration Component Libraries** (ICL) and select Create new Integration Component Library.
4. Give this new ICL a name. We chose RedMaintenance.
5. In your ICL, right-click and select **Business Objects** → Create New Business Object, as in Figure 3-1.

![Figure 3-1 New business object](image)

6. Close the new business object pop-up and select **File → New Using ODA**. See Figure 3-2 on page 33.
7. If your ODA has started successfully, you will see it in the Located agents window.

If you do not see the XMLODA, check the command window for the ODA to find the error and restart the ODA.
8. Highlight the ODA and click **Next**, as in Figure 3-3.
9. Enter the ODA properties as shown in Figure 3-4. The schema definition for the RedTenant messages is in the RedTenant folder in the Additional Materials.

10. Enter a BOPrefix of Web.

11. Click **Next**. See Figure 3-5 on page 36.
12. The ODA will find a top-level object for this XML schema. Highlight this top-level object, as in Figure 3-5.

13. Click **Next**.
Figure 3-6  Confirm source

14. Select the top-level object again to confirm, as in Figure 3-6.
15. Click Next.

See Figure 3-7 on page 38.
16. Click **OK** to confirm that we will take all of the usual, supported verbs. This is the default setting.

See Figure 3-8 on page 39.
3.1.2 Modifying business objects for use

As often happens when you have created business objects with an ODA, you need to check them, modify them for anything specific that was not picked up by the ODA and save them, both to the project and to the repository. We will now do this for our new XML business objects.

The XML ODA generates the name for a business object definition from the following information:

- The value of the BOPrefix ODA configuration property.
- The value of the TopLevel ODA configuration property.
- The name of the XML element that the business object definition represents.

It separates each of these values with an underscore (\_) character.
Therefore, the name it generates has the following format:

BOPrefix_TopLevel_XMLelement

It also adds a suffix to the business object name of the child objects, as can be seen in Figure 3-9.

These names will end up resulting in some very long ESQL statements when we use the Message Broker for our flows, so we will save each of them with a shortened name for later ease of use. Because most of these objects are also child objects, we will need to modify the parents to ensure we are pointing to the correct children.

Validating "Web_tenant..."
Validating "Web_tenant_P0690378762_tenant..."
Validating "Web_tenant_apartments_P0690378762..."
Validating "Web_tenant_apartment_P0690378762..."
Validating "Web_tenant_maintenances_P0690378762..."
Validating "Web_tenant_maintenance_P0690378762..."

Error: There are no key attributes in the business object "Web_tenant"
Failed saving "Web_tenant"

Figure 3-9  BO log

Also, as can be seen in Figure 3-9, the ODA has not correctly set the key attribute on the main business object. We will also have to resolve this.

Important: If you intend to use the completed ESQL included in the Additional Materials, it is important that you use the same business object names as we do here.

Note: Scroll upwards if you do not see the error. It is definitely there!

To resolve the key attribute for the main business object, do the following:

1. Open the Web_tenant business object.
2. As we can see from Figure 3-10 on page 41, all of the child business objects are incorporated into the TopLevel business object. We must work from the bottom up to modify and save our business objects.
3. Eliminate the error stopping the validation of the TopLevel business object. This is the key field, as we see in the error message in Figure 3-9.
4. Select the XMLDeclaration and ROOT business objects as the key attributes and save the Web_tenant business object.
5. Open the **Web_tenamt_maintenance_P0690378762** business object. This business object is correct as defined.

6. Select **File → Save As**.

7. In the pop-up box, remove the number suffix so that our business object is named **Web_tenamt_maintenance**, as shown in Figure 3-11.
8. Click OK.

9. Save this new business object again, but this time save a copy to file. The directory we need to save this to is the repository directory. The repository directory is where the connector will get its runtime business object definitions.

10. The directory for our repository is `C:\IBM\WebSphereAdapters\repository`. The business object will be saved with its name and an XSD schema file type, as shown in Figure 3-12.

11. Save this business object and close it.

![Figure 3-12 Save to file](image)

12. Open the `Web_tenant_maintenances_P0690378762` business object.

13. As we can see, it contains a child object. Using the drop-down list, change the child object to be the newly saved `Web_tenant_maintenance` object, as shown in Figure 3-13 on page 43.

**Tip:** If you do not see this new business object in the drop-down list, close the Business Object Designer and return to the System Manager. Right-click the ICL and select Refresh View. Return to the Business Object Designer and proceed.
14. Save this business object to your ICL with a name of `Web_tenant_maintenances`.

15. Also save this to a repository file.

16. Open the `Web_tenant_apartment_P0690378762` object and change the child object for maintenances using the new `Web_tenant_maintenances` as shown in Figure 3-14.

17. Save this business object to the ICL and to the repository as `Web_tenant_apartment`.

**Figure 3-13  Change BO type in maintenances**

<table>
<thead>
<tr>
<th>Pos</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>maintenance</td>
<td><code>Web_tenant_maintenance_P0690378762</code></td>
</tr>
<tr>
<td>2</td>
<td>ObjectEventId</td>
<td>Float, Double, String, Date, LongText</td>
</tr>
<tr>
<td>3</td>
<td>Web_tenant</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Web_tenant</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Web_tenant_maintenances</td>
<td><code>Web_tenant_maintenances_P0690378762</code></td>
</tr>
</tbody>
</table>

**Figure 3-14  Change BO type in apartment**

<table>
<thead>
<tr>
<th>Pos</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
<td>String</td>
</tr>
<tr>
<td>2</td>
<td>address</td>
<td>String</td>
</tr>
<tr>
<td>3</td>
<td>number</td>
<td>String</td>
</tr>
<tr>
<td>4</td>
<td>postcode</td>
<td>String</td>
</tr>
<tr>
<td>5</td>
<td>maintenances</td>
<td><code>Web_tenant_maintenances_P0690378762</code></td>
</tr>
<tr>
<td>6</td>
<td>ObjectEventId</td>
<td>Double, String, Date, LongText</td>
</tr>
<tr>
<td>7</td>
<td>Web_tenant</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Web_tenant</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Web_tenant_maintenances</td>
<td><code>Web_tenant_maintenances_P0690378762</code></td>
</tr>
</tbody>
</table>

Chapter 3. Create the business objects and connector  43
18. Open **Web_tenant_apartments_P0690378762**.

19. Change the child business object for apartment to **Web_tenant_apartment** as shown in Figure 3-15.

![Figure 3-15 Change BO in apartments](image)

20. Save this business object to the ICL and to the repository as **Web_tenant_apartments**.

21. Open the **Web_tenant_P0690378762_tenant** business object.

22. Change the child business object for apartments to **Web_tenant_apartments** as shown in Figure 3-16 on page 45.
23. The attribute schemaLocation is of the type xsischemalocation. Set a default value and make it a required value to ensure that the default will be picked up, if required.

24. Insert the correct values as shown in Figure 3-17.

**Important:** Note that there is a space between the words RedTenant and tenant.xsd in the schema location.
25. Save this business object to the ICL and the repository as Web_tenant_tenant.

26. We now need to modify the top level object, Web_tenant.

27. Open the **Web_tenant** business object.

28. Change the child object which forms the ROOT element of the business object to reflect the new Web_tenant_tenant business object as shown in Figure 3-18.

![Figure 3-18 Change BO](image)

**Figure 3-18 Change BO**

29. Click the **General** tab. We can see the schema namespace information for the business object and the supported verbs.

![Figure 3-19 Supported verbs](image)

**Figure 3-19 Supported verbs**

30. Click the **Attributes** tab.

For our business object to represent a schema document, there are requirements for at least two business objects:

- The top-level business object represents the information that defines the schema document and must contain the following:
  - An attribute named XMLDeclaration to represent the XML version
This attribute must have the type=pi tag in its application-specific information.

- An attribute to represent the root element in the schema document

This attribute must have as its type a single-cardinality business object, whose type is the business object definition for the root element of the schema document.

The XML ODA obtains the name of this root element from the Root ODA configuration property.

The application-specific information must list the name of this element with the elem_name tag.

- A root-element business object definition represents the XML-definition document’s root element. It contains an attribute for each of the XML components in the root element.

31. Check that these two conditions are met as shown in Figure 3-20.

<table>
<thead>
<tr>
<th>General</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>XMLDeclaration</td>
</tr>
<tr>
<td>2</td>
<td>ROOT</td>
</tr>
<tr>
<td>3</td>
<td>ObjectEventId</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-20  TopLevel business object

32. Our business object attribute named XMLDeclaration represents the XML declaration in the prolog, this attribute has a type of pi (which indicates XML processing instructions), we will set a default value that may be used.

We give our attribute the value:

```xml
xml version = "1.0"
```

the XML data handler generates the following XML:

```xml
<?xml version="1.0"?>
```
33. Expand all of the child objects within the business object, as shown in Figure 3-21. Check that all of your changes have been made and saved correctly.

34. Save the Web_tenant business object to the ICL and to the repository.

**Note:** When you go back to the System Manager after closing the Business Object Designer, it is a good idea to right-click the ICL and refresh the view to ensure that you are seeing the most recent copies of everything.
Figure 3-22  ODA settings for maintenance

Repeat the steps we have taken up until now to create the Web_newMaintenance business object using the XML ODA and the maintenance schema in the RedMaintenance folder in the Additional Materials.

Note: If you experience errors with the ODA, check that the ODA is still running. If it is not, close the DOS window from the previous ODA and restart it. The usual process is for the ODA to shutdown each time it has finished its discovery.

Continue with Figure 3-23 on page 50.
35. After creating the business objects, add a key to the Web_newMaintenance business object and save it.

36. Open the **Web_newMaintenance__N2116280397_newMaintenance business** object.

37. Save this object to the project and to file as
   **Web_newMaintenance_newMaintenance**. See Figure 3-24.
38. Open the **Web_newMaintenance** business object.

39. Change the ROOT element to use the **Web_NewMaintenance_newMaintenance** business object.

40. Save the **Web_newMaintenance** business object to the project and to file.

**Important:** Check the repository directory to ensure that each of your business objects has actually been saved to file.

### 3.2 JMSConnector

Because the front-end application is messaging aware and uses XML data, a JMS connector will be used to transport the business objects between the broker and the front-end application. We now need to create a JMS connector and configure it.

We will import the connector and its metaobjects from repository export files. We will create a connector from the beginning for the custom adapter in later chapters.

**Note:** The Repos file, JMSConnector.jar, in the JMSConnector folder in the Additional Materials contains the configured JMS connector.

#### 3.2.1 Create a JMS Connector

To create a JMS Connector, do the following:

1. Go to System Manager.
2. Right-click your ICL.
3. Select **Import from Repository File**.
4. Select your ICL.
   - Select **Import from Repository Files Directory**.

Continue with Figure 3-25 on page 52.
5. Navigate to the **Additional Materials** → **JMSConnector** folder within it.

6. Click **Finish**. This will create JMSConnector definitions.

7. Navigate back to the System Manager and refresh the view.

8. Double-click the new **JMSConnector** to open the Connector Configuration properties.

9. Select the **Standard Properties** tab. See Figure 3-26 on page 53.
   
   There are properties that are standard for all connectors. Some of these will differ slightly, depending on what integration broker you are using.
Some of the properties are specific to using a connector with the Message Broker and some need to be checked for correctness.

- When using the Message Broker, the only Delivery Transport type to use is JMS.

- When using the Message Broker, the Broker Type is WMQI.

- The queues that are required will depend on the transport type. For JMS transport type, all of these queues are required. Your VMWare image already has these queues defined.

- The JMS factory classname must be supplied and correct.

- The JMS Message Broker Name is the name of our queue manager.
- When using the Message Broker, the Repository directory must point to the directory where the schema representations of the business objects are stored.

- When using the Message Broker, the RFH2 Message Domain is `mrm`, Message Repository Manager.

- When using the Message Broker the Wire Format is `CwXML`, this will be the XML wire format that we create in the Message Set definitions.

- When using the Message Broker, the XML Name Space Format can be short or long. We have chosen long.

10. Switch to the **Connector-Specific** Properties tab, as in Figure 3-27.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Encrypt</th>
<th>Update Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHandlerConfigMO</td>
<td>JMS_MO_DataHandler</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>LogFileName</td>
<td>STDOUT</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>DataHandlerMimeType</td>
<td>text/xml</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>DataHandlerClassEle</td>
<td>com.crossworlds.DataHandlers.text.xml</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>ConfigurationMetaObj</td>
<td>JMS_MO_Config</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>TraceFileName</td>
<td>STDCUT</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>InDoubtEvents</td>
<td>Reprocess</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>ErrorQueue</td>
<td>queue:jrebroker/mettenant.error</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>InProgressQueue</td>
<td>queue:jrebroker/mettenant.inprogress</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>InputQueue</td>
<td>queue:jrebroker/mettenant.event</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>UnsubscribeQueue</td>
<td>queue:jrebroker/mettenant.unsubscribed</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>MessageResponseRes</td>
<td>queue:jrebroker/mettenant.result</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>CTX_initialContextFactory</td>
<td>com.sun.jndi.fscontext.RETFSCContextFactory</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>CTX_ProviderURL</td>
<td>file:/C:temp</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>QueueConnectionFactor</td>
<td>MyQCF</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>DefaultVerb</td>
<td>Retrieve</td>
<td></td>
<td>agent restart</td>
</tr>
<tr>
<td>UseDefaults</td>
<td>true</td>
<td></td>
<td>agent restart</td>
</tr>
</tbody>
</table>

**Figure 3-27  Connector-specific properties**

11. In Figure 3-27 we see the configuration properties specific to this type of connector. We will investigate these as we progress.

12. Normally, you would save this configuration to file. However, this has already been done in your VMWare image.

### 3.3 Connector metaobjects for the data handler

**Note:** Repos file: MetaObjects.jar in the Additional Materials contains the JMS connector metaobjects.
A data handler is a Java class instance that converts between a particular serialized format and a business object. Data handlers are used by components of a business integration system that transfer information between a business integration broker and some external process, such as our front-end application.

Often, the external process uses some common format such as XML for its native serialized data. Rather than have every adapter handle the transformation between these common formats and business objects, the WebSphere business integration system provides several delivered data handlers. In addition, you can create custom data handlers to handle conversions between your own native formats. The adapter then call the appropriate data handler to perform the data conversion based on the Multipurpose Internet Mail Extensions (MIME) type of the serialized data, XML in our case.

A connector instantiates a data handler based on the MIME type of an input file or the MIME type specified in a business object request. A data-handler metaobject is a hierarchical business object that can contain any number of child objects.

The data-handler configuration information is arranged in the following hierarchy:

- The top-level metaobject contains information about the MIME types that the different data handlers can support. Each top-level attribute is a cardinality 1 attribute referencing a child metaobject for a data handler instance. Each attribute represents one MIME type and indicates which data handler can manipulate it.
- The child metaobject contains the actual configuration information for a particular data handler. Each attribute represents a configuration property and provides information such as its default value and type.

Note: A data handler is not required to use metaobjects to hold configuration information. However, all delivered data handlers are designed to use metaobjects for their configuration information. Data-handler metaobjects allow a connector to instantiate a data handler based on the MIME type of an input file or the MIME type specified in a business object request.

To configure a data handler, you must ensure that its metaobjects are correctly initialized and available to the caller (the connector).

Note: Each delivered data handler uses configuration properties defined in data-handler metaobjects.
3.3.1 Creating metaobjects

To create a metaobject, do the following:

1. Navigate to System Manager.
2. Right-click your ICL.
3. Select Import from Repository File.
4. Select your ICL.
5. Select Import from Repository Files Directory.
6. Navigate to the Additional Materials folder and the MetaObjects folder within it.
7. Click Finish.
8. This creates a metaobject business object definition.
9. Navigate back to the System Manager and refresh the view.
10. Double-click the new JMSConnector to open the Connector Configuration properties. See Figure 3-28.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataHandlerConfigMO</td>
<td>JMS_MO_DataHandler</td>
</tr>
<tr>
<td>LogFileName</td>
<td>STDOUT</td>
</tr>
<tr>
<td>DataHandlerMimeType</td>
<td>text/xml</td>
</tr>
<tr>
<td>DataHandlerClassName</td>
<td>com.crosswordit.DataHandlers.text.xml</td>
</tr>
<tr>
<td>ConfigurationMetaObject</td>
<td>JMS_MO_Config</td>
</tr>
<tr>
<td>TraceFileName</td>
<td>STDOUT</td>
</tr>
<tr>
<td>InDoubleEvents</td>
<td>Reprocess</td>
</tr>
<tr>
<td>ErrorQueue</td>
<td>queue://REDBROKER/WEBTENANT.ERROR</td>
</tr>
<tr>
<td>InProgressQueue</td>
<td>queue://REDBROKER/WEBTENANT.INPROGRESS</td>
</tr>
<tr>
<td>InputQueue</td>
<td>queue://REDBROKER/WEBTENANT.EVENT</td>
</tr>
<tr>
<td>UnsubscribedQueue</td>
<td>queue://REDBROKER/WEBTENANT.UNSUBSCRIBED</td>
</tr>
<tr>
<td>ReplyToQueue</td>
<td>queue://REDBROKER/WEBTENANT.RESULT</td>
</tr>
<tr>
<td>MessageResponseResultProperty</td>
<td>queue://REDBROKER/WEBTENANT.RESULT</td>
</tr>
<tr>
<td>CTX_InitialContextFactory</td>
<td>com.sun.jndi.fscontext.RefSCtxtFactory</td>
</tr>
<tr>
<td>CTX_ProviderURL</td>
<td>file:/C:femp</td>
</tr>
<tr>
<td>QueueConnectionFactoryName</td>
<td>MyGCF</td>
</tr>
<tr>
<td>DefaultVerb</td>
<td>Retrieve</td>
</tr>
<tr>
<td>UseDefaults</td>
<td>true</td>
</tr>
</tbody>
</table>

Figure 3-28  Connector properties

In the Connector-specific properties, we see the names of our metaobjects:

- DataHandlerConfigMO

This is the metaobject passed to the data handler to provide configuration information) and is called JMS_MO_DataHandler.
11. From the System Manager, open the **JMS_MO_DataHandler**. See Figure 3-29.

**Figure 3-29  Data handler metaobject**

12. The first thing we notice is that this object is based on the default XML data handler metaobject. We have merely made some modifications for our specific instance. The BOPrefix property matches the BO prefix of our business objects (Web). This enables the data handler to know which business object it needs.

13. Open the **JMS_MO_Config** business object. See Figure 3-30.

**Figure 3-30  JMS_MO_Config business object**

14. This business object, contains the processing rules for the connector to follow. Review the following properties, for example:

   - **Web_tenant_Retrieve** indicates what the connector should do when a business object of Web_tenant with a verb of Retrieve is received by the connector.
The Application Specific Information (ASI) indicates:

\[\text{InputFormat}=\text{MQSTR};\text{OutputFormat}=\text{MQSTR};\text{OutputDestination}=\text{queue://REDTENANT/RTRES1Q}\]

- The input format of the inbound, or event, message is MQSTR, which means it is string data.
- The output format, how the connector should pass the data to the application, is MQSTR and the destination of that message will be a queue named RTRES1Q at queue manager REDTENANT. This is our application’s response message queue for retrieve queries.

There are many options available, but these alone will suit our purposes.

- The Web_newMaintenance_Create indicates the application response queue for the create requests.

15. Save these business objects to the repository directory and to file, close them.

### 3.4 Queue connection factory and queue objects

Because we are using queue based messaging, we must perform some additional configuration.

1. Open the **Connector Configurator** for the JMSConnector.
2. Select the **Standard Properties** tab.
3. The jms.MessageBrokerName indicates the queue manager to which the connector will connect.
4. Ensure that each of the queues listed in Figure 3-31 on page 59, either exists at that queue manager, or is known to that queue manager. Use the MQ Explorer shortcut on the desktop.

**Note:** These queue names are as they would be known to WebSphere MQ.
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminInQueue</td>
<td>WSEBENTANT_ADMININ</td>
</tr>
<tr>
<td>AdminOutQueue</td>
<td>WSEBENTANT_ADMINOUT</td>
</tr>
<tr>
<td>AgentTraceLevel</td>
<td>5</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>JMSConnector</td>
</tr>
<tr>
<td>BrokerType</td>
<td>VMGii</td>
</tr>
<tr>
<td>CharacterEncoding</td>
<td>ascii7</td>
</tr>
<tr>
<td>ContainerManagedEvents</td>
<td></td>
</tr>
<tr>
<td>DeliveryQueue</td>
<td>WSEBENTANT_DELIVERY</td>
</tr>
<tr>
<td>DeliveryTransport</td>
<td>JMS</td>
</tr>
<tr>
<td>DuplicateEventElimination</td>
<td>false</td>
</tr>
<tr>
<td>FaultQueue</td>
<td>WSEBENTANT_FAULT</td>
</tr>
<tr>
<td>jms.FactoryClassName</td>
<td>CxCommon.Messaging.jms.JBMG2SeriesFactory</td>
</tr>
<tr>
<td>jms.MessageBrokerName</td>
<td>REDBROKER</td>
</tr>
<tr>
<td>jms.NumConcurrentRequests</td>
<td>10</td>
</tr>
<tr>
<td>jms.Password</td>
<td>*******</td>
</tr>
<tr>
<td>jms.UserId</td>
<td></td>
</tr>
<tr>
<td>Locale</td>
<td>en_US</td>
</tr>
<tr>
<td>MessageFile Name</td>
<td>JMSConnector.txt</td>
</tr>
<tr>
<td>PollEnd Time</td>
<td>HH:MM</td>
</tr>
<tr>
<td>PollFrequency</td>
<td>5000</td>
</tr>
<tr>
<td>PollStart Time</td>
<td>HH:MM</td>
</tr>
<tr>
<td>RepositoryDirectory</td>
<td>C:\IBMWbSphereAdapters\repository</td>
</tr>
<tr>
<td>RequestQueue</td>
<td>WSEBENTANT_REQUEST</td>
</tr>
<tr>
<td>ResponseQueue</td>
<td>WSEBENTANT_RESPONSE</td>
</tr>
<tr>
<td>RestartRetryCount</td>
<td>3</td>
</tr>
<tr>
<td>RestartRetryInterval</td>
<td>1</td>
</tr>
<tr>
<td>RFH2MessageDomain</td>
<td>mm</td>
</tr>
<tr>
<td>SynchronousRequestQueue</td>
<td>WSEBENTANT_SYNCHRONOUSREQUEST</td>
</tr>
<tr>
<td>SynchronousRequestTimeout</td>
<td>0</td>
</tr>
<tr>
<td>SynchronousResponseQueue</td>
<td>WSEBENTANT_SYNCHRONOUSRESPONSE</td>
</tr>
<tr>
<td>WireFormat</td>
<td>CwXML</td>
</tr>
<tr>
<td>XMLNameSpaceFormat</td>
<td>long</td>
</tr>
</tbody>
</table>

Figure 3-31 Standard properties

5. Click the **Connector-Specific Properties**, shown in Figure 3-32 on page 60.
6. Because we are using JMS as our transport, we need to ensure that certain JMS related properties are correct and the configuration is in place.

   - QueueConnectionFactoryName is the JMS queue connection factory object defined in the JNDI store that the connector should retrieve and use for establishing a connection to the JMS provider. When looking up this name, the connector uses the initial JNDI context established by the CTX_InitialContextFactory and CTX_ProviderURL properties.
   - CTX_InitialContextFactory is the name of the factory class used to establish an initial JNDI context.
   - CTX_ProviderURL is a fully-qualified URL identifying JNDI context where the connection factor is located. This value is passed to the context factor.

Update the JMS configuration by doing the following:

7. Open `c:\WMQ\Java/bin/jmsAdmin.config` and set the following properties:
   a. INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefsContextFactory
   b. PROVIDER_URL=file:/c:/temp
   c. SECURITY_AUTHENTICATION=none

8. Create a file named MyJNDI.txt containing the following in Example 3-1 on page 61.
Example 3-1  JNDI

```
DEFINE QCF(MyQCF) HOST(student) PORT(1421) CHANNEL(CHANNEL1) QMGR(REDBROKER) 
TRAN(client)
```

9. Change the directory to:
   
   `C:\WMQ\Java\bin`

10. Bind objects to JNDI names by running:
    
    `jmsAdmin.bat < MyJNDI.txt`
    
    This now matches our configuration properties.

11. Ensure that all the queues listed exist or are known to the queue manager.

**Note:** In these connector properties, the queues are referred to by the names with which JMS will know them.

### 3.5 Supported business objects for the connector

For a connector to be able to process business objects, their definitions must be known to the connector. This is known as *business object support*.

When using the Message Broker with adapters, not only do you need to add support for your business objects and metaobjects, you also need to add the name of the Message Broker Message Set that contains the definition of the business object. When an event message is delivered from the connector to the broker, the RFH2 header will contain the correct information to allow the Broker to identify and parse the message. The Message Set name will be the one from our connector configuration and the Message will be the name of our business object.

1. Click the **Supported Business Objects** tab. See Figure 3-33 on page 62.

2. We have added the business objects that we require, and the Message Set ID. Actually, it is the name, but that is OK because both ID and name are supported by the Broker. When we define our Message Sets, WebTenant will be our message set for Web_tenant business objects and messages.
   
   Check that ALL of the message sets are WebTenant. Modify those that are not.
3.6 Start the connector

We have now completed our connector configuration. All that remains is to start it and run a few simple tests to verify that everything works correctly.

Before we do this, we will make one minor adjustment to our connector configuration. The PollFrequency standard property indicates how often the connector should poll for new events in the event store. The current value of this property is 5000 milliseconds.

In our case, the event store is a queue, WEBTENANT.EVENT, which is the InputQueue property for the connector. The front-end application sends its retrieve request messages to a queue named RTREQ1Q.

Navigate to the MQ Explorer and delete this local queue and the RTREQ2Q on the REDTENANT queue manager. Redfine both of these queues as remote queue definitions which resolve to the WEBTENANT.EVENT queue on the REDBROKER queue manager.

From this queue the connector will take the ‘raw’ application data and send it to the DeliveryQueue (WEBTENANT.DELIVERY) as a business object ready for processing.

For the initial testing, switch the polling to manual so that you can see the contents of the messages at each step.

1. Click the Connector Configurator.
2. Select the Standard Properties tab.
3. Change the value of the PollFrequency to key.
4. Save the connector configuration to the project and to file, ignoring the warnings about the Trace and Log files. The file location is the JMS folder in the connectors subdirectory for the adapters.

5. Start the connector:
   
   **Start → Programs → IBM WebSphere Business Integration Adapters → Adapters → Connectors → JMSCConnector**

6. When the connector has completed its initialization, you will see a message indicating it is ready to accept polling requests.

7. Start the WebSphere Application Server that runs our front-end application. Do this from the First Steps menu of WebSphere.

8. When the server is initialized, open for e-business, open a Web browser for the RedTenant application.

### 3.7 Unit test

To perform the first test, do the following:

1. Enter an ID for the application logon. Use 100 as in Figure 3-34 because it is one that exists and is also one for which we have created some sample messages for testing.

![Redtenant home page](image)

**Figure 3-34 Redtenant home page**

2. You will receive an error from the server, because there is currently nothing processing our requests.

4. Select the `WEBTENANT.EVENT` queue on `REDBROKER` queue manager. See Figure 3-35.

![Figure 3-35](image)

5. Browse the message on the queue using the `BrowseQ` button, not the `Read Q` button.

6. Switch to the **Data** tab and select the **XML** Data Format of the message as in Figure 3-37 on page 66. This is our raw data from the front-end application.
7. Switch to the **RFH** tab and have a look at the RFH header information as in Figure 3-37 on page 66.
8. Go to the connector DOS window and enter `p` to begin polling for an event.

9. You should see the connector pick up the message from the event queue, transform it to a business object and put the message on the delivery queue for delivery to the Message Broker.

10. Go back to rfhutil, and get the message from the WEBTENANT.DELIVERY queue. There is no need to browse it, because we do not currently need this message.

   See Figure 3-38 on page 67.
11. Select the **Data** tab to see the transformed data.

12. Select the RFH tab as in Figure 3-39 on page 68, and notice that now the information that we have provided with our connector configuration and metaobjects, has cause the RHF2 header to populated correctly for the Broker to interpret.

   - **Message Domain** = mrm
   - **Message Set** = WebTenant
   - **Message Type** = Web_tenant
   - **Message Format** = CwXML
   - **Data Format** = MQSTR
Our outbound request looks good. We will try one further test, sending a response to the connector. In the Additional Materials folder is folder called RedTenantTestMessages. We will use one of these messages to simulate a message coming from the Broker back to the connector to check that our configuration is sound in both directions.

13. Open an instance of rfhutil.

14. Select queue WEBSITE.TENANT.REQUEST at queue manager REDBROKER.

15. Using the Read File button, locate the test message named Web_tenant_Request, meaning a message that is sent to the connector which requests an action from the application.

16. Select the Data tab to see the message contents. It will be similar to that shown in Figure 3-40 on page 69.
17. Select the RFH tab to see that the RFH2 header information is in place. See Figure 3-41 on page 70.
18. Write the `Web_tenant_Request` to the queue.

19. The connector picks up this message and transforms it from business object format to application format.

20. Using `rfhutil`, browse the contents of the message on the RTRES1Q queue on queue manager REDTENANT.

21. Select the **Data** tab to see the application representation of our business object. It will be similar to that shown in Figure 3-42 on page 71.
22. Go back to the browser and resubmit the query. This will not be an actual request and reply because we have already loaded the queue. However, we will get to see whether the application can handle our message data.
23. Select one of the maintenance records to see that the data we expect to see is there. It will be similar to that shown in Figure 3-44. Verify this by checking in the back-end application database.
Once we have our JMSConnector and business objects for our front-end application in place and working as expected, we can create all of the components required for our custom-built back-end application connector.

Remember to clear out all of the queues after your unit testing.

**Tip:** If you are confused about what messages should be when and in what format, Appendix B, “Where are my messages?” on page 333 has some cheat sheets for you.
Deploy and test a custom adapter for the back-end

In this part of the paper, we develop and test a custom adapter for the back-end application, RedMaintenance. We also develop business objects for this application.
Object Discovery Agent

To create the business objects, we will deploy an Object Discovery Agent (ODA), written for our custom-built adapter. We will use this ODA to assist in building the required business objects for the back-end application.
4.1 Set up the custom ODA

The custom ODA consists of several components:

![File Table]

- The common.jar file facilitates communication with the back-end application.
- ODARedMaintenance.jar is the ODA itself, build using the Development Kit.
- policy.txt contains the Java security policy settings.
- startODARedMaintenance.bat is the startup batch file.

Move these files into their correct place and create the start mechanism.

1. Navigate to the ODA directory of the WebSphere Business Integration Adapters directory:
   
   C:\IBM\WebSphereAdapters\ODA

2. Within this directory, create a new folder named RM.

3. Navigate to the Additional Materials and a folder named ODA.

4. Copy the contents of this directory to the newly created RM ODA folder.

5. Create a windows shortcut to start the ODA. The startup file for the ODA is:
   
   start_ODARedMaintenance.bat

6. Create the shortcut in
   
   C:\Documents and Settings\All Users\Start Menu\Programs\IBM WebSphere Business Integration Adapters\Adapters\Object Discovery Agent

   Putting the shortcut in this directory ensures that it can be started from the Programs menu.

For the ODA to be able to communicate with the back-end application, the application must be running. If it is not:

7. From a command prompt, execute the rmiregistry command. Leave this window minimized.


9. Using the shortcut you created for the ODA, start the RedMaintenance ODA.
4.2 Generate business objects using the ODA

To generate business objects with the ODA, do the following:

1. Navigate to the System Manager.
2. From your RedMaintenance ICL, right-click Business Objects and select Create New Business Object. This starts the Business Object Designer.
3. Close the new business object pop-up window.

4. Select File → New Using ODA, as in Figure 4-2, bringing up the first window of the Business Object Wizard. See Figure 4-3 on page 80.
5. Click **Find Agent** to find our ODA.

Do not worry if you see the name of the XML ODA in the list. It is just remembering the previous list.

If you have a large network of ODAs to search through, it might be quicker to simply type in the name of the ODA you expect to be running. Remember that this is using the ORB to find the Agents that are running. See Figure 4-4.

6. Having chosen to find the agent, we now wait while the search is carried out.
7. In Figure 4-5, our ODA has been located.

**Note:** If the ODA is not found, check that it is running. Another thing to be careful of is that you do not have multiple ODAs of the same name running on the same subnet. Remember that the ORB is in use.

8. Select this agent and click **Next**. See Figure 4-6 on page 82.
9. In Figure 4-6, enter required property values:
   - Enter the Host name of the back-end application to which to connect as student.
   - The TraceFileName is RedMaintenanceODA.txt.
   - The required TraceLevel is 5. We chose 5 for maximum trace information.
   - The MessageFile which contains the messages for our ODA is the default message file, UserAgentMessages.txt, because we do not have any ODA specific messages.

10. Click Next. See Figure 4-7 on page 83.
11. The components that have been discovered from the back-end application will be displayed.

   If you do not see these components, check your ODA settings and check that the back-end application is running correctly.

12. Select the nodes from which ODA will generate business objects. We selected all of them.

13. Click **Next**. See Figure 4-8 on page 84.
You are asked for confirmation before you click **Next**. See Figure 4-9 on page 85.
15. We now must enter any additional properties that are needed for the BO generation.

16. Enter a prefix for the business object names that are to be generated. We selected **RM_**.

17. Click **OK**. The business object definition generation begins.
18. We are now asked to save the business objects, as in Figure 4-10. Select the **Shutdown ODA RedMaintenance** button. We no longer need the ODA.

19. Click **Finish**.

In the Business Object Designer in Figure 4-11, we see the newly created business object definitions. The asterisk next to each object indicates that they are unsaved. We will discuss this later.

20. You can also see from the GUI activity window of the back-end application that the ODA has successfully contacted the application. See Figure 4-12 on page 87.
4.2.1 Completing the business objects

When an ODA discovers the application business objects, quite often what is passed back is only part of the story of the application entities. That is, the ODA can only generate definitions based on what the application or database broadcasts to it or might incorrectly report column attributes of a database schema based on the way that it interacts with the database. This is the case with our ODA. The back-end application has told the ODA about the different components that it contains, but has not told us about the relationships or the dependances that exist between these components.

Important: It is here that, in the real world, you would require the assistance of someone who has a detailed knowledge of the application itself. Do not skip over this step because it is crucial that the business object design be verified against the application with which it is interacting.

First look at the business objects that were generated by the ODA. We will then detail what amendments need to be made to the business objects to allow our adapter to interact successfully with the API.
The business objects generated for the application objects are all correct in terms of the supported verbs and the application object names, as in Figure 4-13 for the Tenant business object, for the Application-specific Information (ASI).

What we see in Figure 4-14 is the attributes as generated by the ODA. The ASI is correct in that the attribute names of the application object have been generated correctly. We also see key fields and mandatory fields reported, however we need to check whether they are correct. The ApartmentId is not a primary key of this object, more likely it is a foreign key. Moreover, the tenant business object has been discovered as a flat object with no relationships to the other object. This is not accurate because the tenant application object is a parent object which contains apartment and possibly maintenance child objects. The ASI on the EMail attribute is also incorrect. We will update all of these as we continue.
We see in Figure 4-15 that the apartment business object appears to be an accurate representation of the application object. It is a flat object with the ID as a key, mandatory attribute.

Figure 4-16 shows the following errors that need correcting:

- The maintenance business object also has multiple keys flagged. This is not the case and must be corrected.

- The generated business object has an attribute of StatusDescription, but from our discussions with the application experts, we also know that this business object should have an attribute of Status as well. We will have to correct this.

- The ODA has also interpreted the dates as string data when in fact they are dates. This also will require correcting.
We now move on to modifying the generated business objects to accurately reflect the application view of the objects.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Name</th>
<th>Type</th>
<th>Key</th>
<th>Foreign</th>
<th>Required</th>
<th>Card</th>
<th>Maximum Length</th>
<th>Default</th>
<th>App Spec Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Id</td>
<td>Integer</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td>attr=Id</td>
</tr>
<tr>
<td>2</td>
<td>ApartmentNumber</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>attr=ApartmentNumber</td>
</tr>
<tr>
<td>3</td>
<td>AddressLine1</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td>attr=AddressLine1</td>
</tr>
<tr>
<td>4</td>
<td>AddressLine2</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td>attr=AddressLine2</td>
</tr>
<tr>
<td>5</td>
<td>AddressLine3</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td>attr=AddressLine3</td>
</tr>
<tr>
<td>6</td>
<td>AddressLine4</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td>attr=AddressLine4</td>
</tr>
<tr>
<td>7</td>
<td>PostCode</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td>attr=PostCode</td>
</tr>
<tr>
<td>8</td>
<td>ObjectEventId</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4-17 Correct apartment business object**

21. The apartment business object in Figure 4-17 is correct as discovered. Save this business object

**Note:** When you save each of the business objects, save a copy of the business object to a repository file. The schema files generated will be needed later for import to the Message Broker and for connector start up.

22. The maintenance business object must be modified to reflect that it contains foreign key values and column values which are contained in other business objects.

23. The ASI `chfk` reflects that the value for the attribute of this child object is to be obtained from an attribute contained in the parent object.

As you can see in Figure 4-18 on page 91, the value for:

– ApartmentId comes from the ApartmentId attribute of the RM_Tenant object.

– TenantId comes from the Id attribute of the RM_tenant object.

When our adapter is performing a retrieve operation for all of the maintenance records for a particular tenant, these values will be passed to the application for the search.

24. Modify the maintenance business object to add the ASI to these attributes as shown in Figure 4-18 on page 91.

25. Modify the business object to reflect that the ApartmentId and TenantId are foreign keys.
26. Remove the key value flags.

27. Change the type for the date fields from String to Date.

28. Insert an attribute of Status with a type of String above the TenantId and ASI of attr=Status.

29. Save the business object.

30. Re-open the RM_Tenant business object.

As we mentioned earlier, the tenant business object is a parent object of one apartment and 0 - n maintenance objects. We need to reflect this.

31. Highlight the row for ObjectEventId by clicking the far left number.

32. Right-click and select Insert Above. See Figure 4-19 on page 92.

33. In the Type column select (from the drop-down box) the RM_Apartment type.

34. Name this Attribute, RM_Apartment.

35. Set the Cardinality to 1, meaning only one occurrence of this child object is allowed. Each tenant has only one apartment record.

36. Repeat this for the RM_Maintenance, setting the Cardinality to N, meaning for an apartment, there may be multiple maintenance records.

37. Set the Foreign Key flag on for the ApartmentId attribute.

38. Update the ASI to indicate that this attribute is part of a foreign key, with the foreign key attribute being RM_Apartment.Id.

39. Change the ASI on the EMail attribute from EMail to Email. All the ASI are case-sensitive.
Figure 4-19 Correct tenant business object

40. Save this business object.

See Figure 4-20 on page 93.
If we now expand the child objects within the parent object, we see the correct hierarchical structure of our tenant object with all of the keys and foreign keys set correctly.

Double-check that all business objects have been saved to the project (ICL) and to the repository files.

Our business objects are now ready for use.
Package the custom adapter for distribution

To run in the IBM WebSphere business Integration System, a connector must have a definition. Predefined adapters which are provided by WebSphere Business Integration Adapters, have predefined connector definitions in the repository. A system administrator need only configure the application and set the connector's configuration properties to run the connector.

For the IBM WebSphere Business Integration System to be able to use the connector that we have developed, we must take the following steps:

1. Create the connector definition.

2. Because WebSphere MQ will be used for messaging between connector components, add message queues for the connector.

3. Create the connector's initial configuration file.

4. Create the connector's startup script.

**Note:** All of the files relating to the RM custom adapter can be found in the Additional Materials in the RM folder.
5.1 Connector naming conventions

Naming conventions provide a way to make our connector files easier to locate and identify. Table 5-1 summarizes the naming conventions we have used for connector files. Many of these files are based on the connector name, which should uniquely identify it within the WebSphere Business Integration System. This name, `connName`, can identify the application or technology with which the connector communicates. The name we gave to our adapter was RMAadapter, using the usual naming convention, we will name our runtime connector RMConnector.

Table 5-1  Our naming conventions

<table>
<thead>
<tr>
<th>Connector file</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connector definition</td>
<td>RMConnector</td>
</tr>
<tr>
<td>Connector directory</td>
<td>ProductDir/connectors/RM</td>
</tr>
<tr>
<td>Connector configuration file</td>
<td>ProductDir/connectors/RM/RMConnector.cfg</td>
</tr>
<tr>
<td>Connector class</td>
<td>RMAgent.java</td>
</tr>
<tr>
<td>Connector library</td>
<td></td>
</tr>
<tr>
<td>Java jar file: connDir/RMAdapter.jar</td>
<td></td>
</tr>
<tr>
<td>Connector startup script</td>
<td></td>
</tr>
<tr>
<td>Windows platforms:</td>
<td>connDir\start_RMConnector.bat</td>
</tr>
<tr>
<td>UNIX®-based platforms:</td>
<td>connDir/connector_manager_RMConnector.sh</td>
</tr>
<tr>
<td>where connDir is the name of the connector directory, as defined above.</td>
<td></td>
</tr>
</tbody>
</table>

5.2 Define the connector

For our custom adapter we must create a connector definition.

A connector obtains it configuration values at start up. In this section we define these standard connector property values.

The connector uses the following order to determine a property value, where the highest number overrides other values:
1. Default
2. Repository (only if the InterChange Server is the integration broker)
3. Local configuration file
4. Command line.

For our exercise with the Message Broker, the properties we configure will form part of the local configuration file.

5.2.1 Define with the Connector Configurator

To define the connector we create a **connector definition**. This connector definition includes the following information to define the connector in the repository. For initial deployment of our customer adapter, we will create a connector definition with a set of standard properties, the properties which are common for most connectors.

**Note:** Not every connector makes use of all these standard properties. When you select an integration broker from Connector Configurator, you see a list of the standard properties that you need to configure for your adapter running with that particular broker type.

We will also create the WebSphere MQ objects required for our connector.

We will then conduct some basic tests to ensure that the connector starts up correctly before proceeding to testing with some of the business objects that we have previously created.

A tool called Connector Configurator collects this information and stores it in the repository.
**Note:** Here are a few words about the repository and deployment.

If your integration broker is the InterChange Server, the connector and its configuration must be deployed to the repository. This is a database that InterChange Server communicates with to obtain information about components in the WebSphere Business Integration System. Connector definitions must be deployed to the repository. These connector definitions include both standard and connector-specific connector configuration properties that the connector controller and the client Adapter Framework require. The connector can also have a local configuration file, which provides configuration information for the connector locally. When a local configuration file exists, it usually takes precedence over the information in the InterChange Server repository. Deployment for the InterChange Server will be detailed fully in a future paper.

If your integration broker is WebSphere, there are deployment steps that you must take for integration of the artifacts for your adapter service.

The most straightforward connector deployment is using the Message Broker as the connector will read the business object definitions from the local file system. For our basic testing at this stage, there is no deployment involved as such. We can save our configuration in a file and test immediately.

1. Create the directory which will hold our completed connector configuration (as shown in Table 5-1 on page 96. For our example that is:

   C:\IBM\WebSphereAdapters\connectors\RM

2. From the System Manager, start the Connector Configurator:

   **Tools → Connector Configurator.**

3. Enter the properties for the new connector as shown in Figure 5-1 on page 99.
4. Name the connector. We are using RMConnector.

5. Select the broker, WebSphere Message Broker.

6. Select None for the template name. This is a new custom connector and as such, there is no template.

7. Click OK.

5.2.2 Define the connector configuration properties

The connector definition also contains the connector configuration properties. To initialize these properties, we must take the following steps:

We are now presented with a base connector configuration. From here, we tailor the properties accordingly.

8. Modify the Standard Properties as shown in Figure 5-2 on page 100.
Adapters: Integration Broker deployment

9. The properties that can differ depending on your broker choice are in Table 5-2:

Table 5-2 Properties with broker dependencies

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrokerType</td>
<td>Select ICS / WMQI or WAS</td>
</tr>
<tr>
<td>DeliveryTransport</td>
<td>JMS is the transport for WMQI and WAS, however it is a good idea to use JMS for ICS also. See the previous Redpaper for a discussion on delivery transports.</td>
</tr>
</tbody>
</table>
Chapter 5. Package the custom adapter for distribution

10.Ensure that you have the queue names defined to match those that you will define later for your connector. Remember that WebSphere MQ object names are case-sensitive. We have defined our queues as shown below in Figure 5-3 and Table 5-3. It is a good idea to give your queues names that are meaningful because you might have many sets of queues for multiple adapters. We have prefixed all of the queues for our RMConnector with REDMAINT which indicates easily that these are the queues servicing the REDMAINT application and its connector.

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepositoryDirectory</td>
<td>For ICS, set this to remote. For WMQI and WAS, set this to: C:\IBM\WebSphereAdapters\Repository</td>
</tr>
<tr>
<td>WireFormat</td>
<td>For ICS, set this to CwBO. For WMQI and WAS, set this to CwXML.</td>
</tr>
<tr>
<td>XMLNameSpaceFormat</td>
<td>Not used for ICS</td>
</tr>
</tbody>
</table>

**Important:** This is not a comprehensive list of all of the properties which might differ between broker types. For more information, look up the Standard Configuration Properties section in any of the delivered Adapter Guides.

10. Ensure that you have the queue names defined to match those that you will define later for your connector. Remember that WebSphere MQ object names are case-sensitive. We have defined our queues as shown below in Figure 5-3 and Table 5-3. It is a good idea to give your queues names that are meaningful because you might have many sets of queues for multiple adapters. We have prefixed all of the queues for our RMConnector with REDMAINT which indicates easily that these are the queues servicing the REDMAINT application and its connector.

![Queue Statistics](image)

**Figure 5-3   RMConnector queues**

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDMAINT.ADMININQUEUE</td>
<td>Used by the broker to send admin messages to the connector</td>
</tr>
<tr>
<td>REDMAINT.ADMINOUTQUEUE</td>
<td>Used by the connector to send messages to the broker</td>
</tr>
</tbody>
</table>

**Table 5-3   Queues and usage**
11. There are other queues available which are used for JMS when specifying ContainerManagedEvents. This provides for guaranteed delivery over JMS. We will not be using them because they would override the pollForEvents, which we want to use as part of our connector functionality.

We now need to enter any application-specific properties for our connector. These are the properties that uniquely define the information required to communicate with our application. If you are not the adapter developer, ask the developer which properties have been included in the adapter and the values you need to specify for each. See Figure 5-4.

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDMAINT.DELIVERYQUEUE</td>
<td>Used by the connector to send business objects to the broker (JMS only).</td>
</tr>
<tr>
<td>REDMAINT.FAULTQUEUE</td>
<td>Used by the connector to forward messages if a fault occurs.</td>
</tr>
<tr>
<td>REDMAINT.REQUESTQUEUE</td>
<td>Used by broker to send business objects to the connector.</td>
</tr>
<tr>
<td>REDMAINT.RESPONSEQUEUE</td>
<td>Used for delivery of response messages from the Adapter Framework to the broker (ICS and JMS only).</td>
</tr>
<tr>
<td>REDMAINT.SYNCHRONOUSREQUESTQUEUE</td>
<td>Used for request messages from the Adapter Framework to the broker which require a synchronous response (JMS only).</td>
</tr>
<tr>
<td>REDMAINT.SYNCHRONOUS.RESPONSEQUEUE</td>
<td>Used for synchronous response messages from the broker to the Adapter Framework (JMS only).</td>
</tr>
</tbody>
</table>

Figure 5-4  Application-specific properties
12. Create the application-specific properties as shown in Figure 5-4 on page 102. The properties we require are:

- The name of the host machine of the application and the RMI registered name of the application
- Maximum number of connections
- The Callback properties relate to an ICS broker implementation only

13. Go to the Logging and Tracing properties and preset any values you require. We have chosen to send all logging and tracing information to both STDOUT and to a file, the same file. It makes problem determination much easier.

14. Save this connector definition to your project.

15. Save this connector definition to file. The file name will be RMConnector.cfg in the directory that you have previously created for the connector.

16. Copy the message resource file: RMConnectorMessages.txt into this directory. It is in the Additional Materials in the RM folder.

### 5.3 Create the WebSphere MQ objects

Using your favorite queue tool, create the required queues as shown in Table 5-3 on page 101.

### 5.4 Create the connector environment

To start up the connector, we execute a connector startup script.
5.4.1 Preparing the connector’s directory

The connector directory contains the runtime files for our connector. To prepare the connector directory, do the following:

1. We have previously created a connector directory for the new connector under the connectors subdirectory of the product directory:

   ProductDir\connectors\connName

   By convention, this directory name matches the connector name connName. The connector name is a string that uniquely identifies the connector.

   **Note:** It is worth noticing here the difference between some of the different names. The connName in our case is RM. However, the connector definition name is RMConnector. This is the same as the ApplicationName property in the connector configuration.

2. Move our connector’s library file to this connector directory.

   A Java connector’s library file is a Java archive (jar) file. We created this jar file when we compiled the connector. (our jar file is RMAdapter.jar).

5.5 Create the startup script

A connector requires a startup script for the system administrator to start execution of the connector process. The startup script to use depends on the operating system on which we are developing your connector.

Figure 5-5 on page 105 shows the steps to start a connector on a Windows® system.
5.5.1 Startup script for our connector

We have tailored a standard startup script for use with our custom adapter. Example 5-1 shows this startup script.

1. Copy this script into the RM connector directory. It is in the Additional Materials in the RM folder.

Example 5-1  RMConnector startup script

```bash
REM @echo off
call "%CROSSWORLDS%"\bin\CWWsdEnv

REM set the directory where the specific connector resides
set CONNDIR="%CROSSWORLDS%\connectors\%1"

REM goto the connector specific drive & directory
cd /d %CONNDIR%

REM set the name to be the application connector that is starting
set CONNAME=%1

REM set the package name to the connector package name
set CONNPACKAGENAME=com.ibm.itso.RedHouse.RM.Adapter.RMAgent

REM set the server name
```

Figure 5-5  Starting a connector on a Windows System
set SERVER=%2

set DataHandler_JAR="%CROSSWORLDS%\DataHandlers\CwDataHandler.jar;"%CROSSWORLDS%\DataHandlers\CustDataHandler.jar

if %CWVERSION%=="4.X"  set XML_PARSER="%CROSSWORLDS%\lib\xerces.jar
if %CWVERSION%=="3.X"  set XML_PARSER="%CROSSWORLDS%\DataHandlers\Dependencies\IBM\xml4j.jar

set VBJTOOLS="%CROSSWORLDS%\lib\vbjtools.jar

set AGENT="%CONNDIR%\RMAAdapter.jar
set RED_MAINT="%CONNDIR%\common.jar

set JCLASSES=.;%JCLASSES%;%RED_MAINT%;%XML_PARSER%;%DataHandler_JAR%;%VBJTOOLS%;%AGENT%

REM start the Java connector under the Java Application End
%CWJAVA% -version
%CWJAVA% -mx128m %ORB_PROPERTY% -Djava.security.policy=policy.txt
-Djava.ext.dirs="%MQ_LIB%";%JRE_EXT_DIRS%
-Djava.library.path="%CROSSWORLDS%\bin;%CONNDIR%;%MQ_LIB%";%JRE_EXT_DIRS%
-Duser.home="%CROSSWORLDS%" -cp %JCLASSES%;%CONNDIR%;CONNNAME%.jar;
AppEndWrapper
-1%CONNPACKAGENAME% -n%CONNNAME%Connector -s%SERVER% %3 %4 %5

endlocal
pause

There are several things worth mentioning in our script.

- Calling the environment file
  There is nothing specific or different for our connector.

- Moving into the connector directory
  There is nothing specific or different for our connector.

- Setting the environment variables
  In the start_connName.bat script, we must provide any of the
  connector-specific information that the environment variables listed in
  Table 5-4 on page 107 specify.
Table 5-4  Environment variables in the connector startup script

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtDirs</td>
<td>None</td>
</tr>
<tr>
<td>JCLASS</td>
<td>Specify any application-specific jar files. Jar files are separated with a semicolon (;). See below</td>
</tr>
<tr>
<td>JVMArgs</td>
<td>None</td>
</tr>
<tr>
<td>LibPath</td>
<td>None</td>
</tr>
</tbody>
</table>

- Here we have our own environment variable of RED_MAINT. This is for the library files for the back-end adapter API. Ensure that this variable is concatenated with the JCLASS.

- We also specify the language-specific startup parameters required for a Java connector to specify connector-specific classes (CONNPACKAGENAME):

  ```java
  com.ibm.itso.RedHouse.RM.Adapter.RMAgent
  ```

- Invoking the connector

  To actually invoke the connector within the JVM, the `start_connName.bat` script usually calls the `start_adapter.bat` script. The `start_adapter.bat` script provides information to initialize the necessary environment for the connector runtime (which includes the Adapter Framework) with its startup parameters.

  We do not have any of the external variables that require setup. Those variables that are not already currently set as part of our system environment are set by the call to the CWConnEnv environment setup utility.

  We are making the call to start the adapter from within our `start_RMConnector.bat` file as in Example 5-2.

Example 5-2  Adapter start

```bash
%CWJAVA% -mx128m %ORB_PROPERTY% -Djava.security.policy=policy.txt
-Djava.ext.dirs="%MQ_LIB%";%JRE_EXT_DIRS%
-Djava.library.path="%CROSSWORLDS%\bin;%CONNDIR%;%MQ_LIB%";%JRE_EXT_DIRS%
-Duser.home="%CROSSWORLDS%" -cp %JCLASS%;%CONNDIR%\%CONNAME%.jar;
AppEndWrapper -l%CONNPACKAGENAME% -n%CONNAME%Connector -s%SERVER%  %3 %4 %5
```

- Note that the Java security policy parameter uses a file named `policy.txt`. This file sets the security permissions and is included in the Additional Materials. Copy this file to the RM connector directory with all of the other required files.

When we start the connector, we will be passing the following startup parameters from the shortcut we will create next:
The name of connector definition: -n (RM)

The name of the broker instance: -s (WMQI_WAS)

Because this is a Message Broker connector, we must also specify the location of the local connector configuration file: -c

C:\IBM\WebSphereAdapters\connectors\RM\RMConnector.cfg

5.6 Create the Windows shortcut

The easiest way to start a connector running on a windows environment is either have it running as a service:

- If you are running an ICS broker, the installer has a component which handles this for you.
- If you are not running an ICS broker, you can choose to use the SVRANY.exe utility and create the service definition.
- You can use a shortcut in the Start Programs folder.

To get started quickly and enable a small test, we will create a shortcut.

1. Create a shortcut to the start_RMConnector.bat file in the following directory:

   C:\Documents and Settings\All Users\Start Menu\Programs\IBM WebSphere Business Integration Adapters\Adapters\Connectors

2. Ensure that the shortcut has the following properties:

   The target has our start parameters as follows

   C:\IBM\WebSphereAdapters\connectors\RM\start_RMConnector.bat RM WMQI_WAS -c C:\IBM\WebSphereAdapters\connectors\RM\RMConnector.cfg

   See Figure 5-6 on page 109.
3. We also create a start_Command.bat file that can be used to start the connector. This is not necessary, but is useful as part of the packaged up connector files we will need for deployment.

4. Before we test the startup of our connector, check all of the following files are in the RM folder of the connectors directory:
   
   - Application interface files:
     - common.jar
     - policy.txt
   - Adapter files:
     - RMAAdapter.jar
     - RMConnector.cfg
   - Command files:
     - start_RMConnector.bat
     - start_Command.bat

   In the messages folder of the connectors directory:
   - RMConnectorMessages.txt.
Unit testing the connector

Before we move on to the deployment and specific building of artifacts for the broker, we need to check that the adapter works correctly with our infrastructure.
6.1 Start the connector

The first thing we need to verify is that the connector starts correctly. Using the windows shortcut or the start_Command.bat file, start the connector.

The first thing that the connector does on startup is parse all of the configuration parameters. Next it loads the connector agent and initializes the connector. It then attempts to make contact with all of the required components:

- The delivery transport mechanism is verified.
- Supported business objects are verified.

If all goes well, the agent is then in an Active state and ready to go.

6.1.1 Troubleshooting start up errors

There are many things that might go wrong upon startup. In this section, we list a few of them. This is when a log or trace file becomes useful.

1. Configuration file error as in Example 6-1

Example 6-1  Configuration file error

```
[Msg: A fatal error was encountered while reading the configuration file. Configuration file C:\IBM\WebSphereAdapters\connectors\RM\RMConnector.cfg is not found. Shutting down the process.]
```

A configuration file of this name does not exist in the stated location. Check that local configuration file has been saved to the correct directory, with the correct name.

2. API error as in Example 6-2.

Example 6-2  API not found

```
[System: Server] [Thread: wbia_main (#1692490151)] [Msg: java.lang.NoClassDefFoundError: com/ibm/itso/rm/common/RMRMIRegisterException
 at java.lang.Class.forName1(Native Method)
 at java.lang.Class.forName(Class.java:142)
 at AppSide_Connector.BusObjJavaInterface.<init>(BusObjJavaInterface.java:82)
 at AppSide_Connector.AgentBusinessObjectManager.setupNativeInterface(AgentBusinessObjectManager.java:482)
 at AppSide_Connector.AgentBusinessObjectManager.<init>(AgentBusinessObjectManager.java:208)
 at AppSide_Connector.AppEnd.run(AppEnd.java:1191)
 at AppSide_Connector.AppEnd.init(AppEnd.java:304)
```
This seems to indicate that we are missing the piece that tells us about our application API and how to communicate with the application. Check that the common.jar file exists in the connector directory.

3. Queue Manager error as in Example 6-3

**Example 6-3   No connection to the Queue Manager**

Can NOT create JMS connection to queue manager -- REDBROKER

[Type: Error] [MsgID: 9052] [Mesg: Unable to get a MQ series Queue Manager or Queue Connection. Reason: failed to create connection --javax.jms.JMSException: MQJMS2005: failed to create MQQueueManager for 'REDBROKER'.]

The queue manager name is wrong. Check that the name of the queue manager in the configuration matches the actual name of the queue manager.

The queue manager name is correct, but it is not running. Start the queue manager.

4. Invalid Queue as in Example 6-4

**Example 6-4   No Queue connection**


[System: ConnectorAgent] [SS: RMConnector] [Thread: wbia_main (#1692489475)]


[System: ConnectorAgent] [SS: RMConnector] [Thread: wbia_main (#1692489475)]

[Type: Error] [MsgID: 9052] [Mesg: Unable to get a MQ series Queue Manager or Queue Connection. Reason: failed to create sessionMQJMS2008.]

One of the required queues has not been created or has been created with an incorrect name. Check that the names of the queues in the configuration all exist and that they have the correct names.

**Tip:** For those of you know already are familiar with WebSphere MQ, the 2085 will be a dead giveaway. If you are not familiar with WebSphere MQ reason codes, navigate to a command prompt and type `MQRC xxxx`, where `xxxx` is the reason code. In this case it is 2085.
Once we have the connector successfully started, we now need to run a quick unit test to verify that the connector can communicate with the application using our configuration.

### 6.2 Test interaction between connector and business objects

To test interaction with the connector, we will use the Visual Test connector. This tool is useful in the early stages of integration development. Using the Message Broker as our Integration Broker means that we can get this happening relatively quickly and easily because we do not require any deployment activities. We will create some test data and feed it to the connector using the Test Connector. Using the Test Connector, we will simulate a response from the back-end application and a response business object.

Once we have verified that this interaction is working correctly, we will use our test data to actually send a request to the back-end application to verify that we get the anticipated response.

When we have verified that the request processing is functioning correctly, we will run a similar series of tests to verify that the event processing is functioning correctly.

#### 6.2.1 Add supported business objects

To enable our test connector to send or receive business objects, we need to add support for the business objects to the connector definition.

1. Navigate to the System Manager.
2. Double-click the **RMConnector** to open the Connector Configurator.

![Figure 6-1 Add business object support](image)

3. Select the **Supported Business Objects** tab, as in Figure 6-1.
4. Add each of the RM_ business objects.

5. Add the message set identifier RedTenant to each.

6. Save the connector configuration to both the project and the connector configuration file.

**Important:** When using the Message Broker as the integration broker, message sets and messages are used. The messages that are created by the broker will have the message set identifier and message type in the JMS folder. This must match the message set identifier and the business object name in the business object definition.

Messages that are created by the connector will also have the message set identifier and message type in the JMS folder, taken from the business object definition. This is required to match the message set identifier and message type as known to the broker for the message to be correctly parsed.

At the moment we are not at a point where any broker is involved. However it is required that a message set identifier be part of the supported business objects attributes because we nominated at Adapter Framework installation time that we were using the Message Broker as our integration broker.

At this point, so long as what is in the messages and what is in the business object definitions match, it is not important what value is here. The value becomes more relevant when we start using the broker.

7. Stop the connector and restart it to check our business object support.

8. If you see any errors similar to Example 6-5, it might indicate that your business objects have not been saved to file to the correct directory. Check that your business objects are in the correct directory. If they are not, move the files to the correct directory, stop and restart the connector.

*Example 6-5  Business object not found*

```plaintext
[MESSG: c:\IBM\WebSphereAdapters\repository\RM_Apartment.xsd (The system cannot find the file specified)
  c:\IBM\WebSphereAdapters\repository\RM_Apartment.xsd]
```

9. Stop the connector.

### 6.2.2 Using the Visual Test Connector

We will use the Visual Test Connector for part of our testing, but first we need to set up the correct configuration.
Start the Test Connector
To start the Test Connector, do the following:

1. Start the Visual Test Connector.

   Start → Programs → IBM WebSphere Business Integration
   Adapters → Tools → Visual Test Connector

2. We need to create a profile which will allow us to emulate the RM Connector. Select File → Create / Select Profile, as in Figure 6-2.

   ![Figure 6-2 Select new profile]

4. Enter the profile details as shown in Figure 6-3:
   – Connector configuration file is the path to the saved configuration file for the RMConnector.
   – Connector name is RMConnector. This is case-sensitive and must match exactly the ApplicationName connector property.
   – Broker type is WMQI.
5. Click OK. See Figure 6-4.
6. Highlight the new profile for the RMConnector.

7. Click **OK**.

8. From the front screen, select **File → Connect**, as in Figure 6-5.

   This starts up the RMConnector. The difference between running the test connector and running the real connector is that we are not connecting to the application. We are only checking that the connector knows about all of the business objects and that they look as we expect.

   What we can do is emulate the behavior of the connector in different testing situations, without any danger or disruption to the real application.

   ![Figure 6-5 Connect](image)

   Once the connector has initialized. We will be able to check for our list of supported business objects and prepare the test data.
6.2.3 Prepare test data for request processing (RM_Tenant.Retrieve)

To prepare the test data, do the following:

1. Select **RM_Tenant** from the BO Type drop-down list, as in Figure 6-6.
2. Next to BO Instance, click **Create**.

3. Give this new instance a name, as in Figure 6-7. What it is, is not really important.
4. Click **OK**.

We are now presented with a template business object to fill with data for testing. See Figure 6-8 on page 120.
5. Using the drop-down list, select a verb of **Retrieve**. Retrieving data will be our first test.

   See Figure 6-9 on page 121.
6. Enter a tenant Id. If you are using our sample data, 100 is a tenant Id that we created for testing.

**Tip:** There are several ways that you can test from this point forward. We have chosen to use a method using rfhutil. Once we have set up our test data, we have a reusable test scenario. We are going to send this BO to the connector. However, because of our JMS messaging configuration, the message will not really go to the queue we need for sending in a request. The message will go to the DELIVERYQUEUE. However, using rfhutil allows us to capture the message, complete with all of the correct MQMD and JMS information and replay it to the correct location. We will explain as we go along. Our testing method allows us to perform initial unit tests for the connector, without the use of any other feeder connector, any broker or any connected application.
7. Select **Request → Send**, as in Figure 6-10. Doing this sends the business object.

8. Start up an instance of rfhutil, included in the Additional Materials.

9. Enter the queue manager REDBROKER, if you are using our sample configuration, and queue REDMAINT.DELIVERYQUEUE.

10. Select **Read Q**.
    
    This will read the message from the queue. Do not use browse because we do not need to retain this message for future use.

    See Figure 6-11 on page 123.
Figure 6-11  Select correct queue

11. Select the Data tab. See Figure 6-12 on page 124.

12. Select the XML Data Format radio button.
13. We now see the business object ready to be sent. Scroll to the right and check that the verb is Retrieve.

14. Select the MQMD tab. We now see the correctly formatted message descriptor. See Figure 6-13 on page 125.
15. Select the RFH tab. We now see all the correct RFH header showing the correct information for JMS. See Figure 6-14 on page 126.
On the RFH header tab, note the following:

Message Domain: mrm
Message Set: RedTenant
Message Type: RM_Tenant
Output Format: CwXML

All of these values appear to be correct for a business object that will be used by the Message Broker as the integration broker, based on the combination of properties in our Connector Configurator.

We now need to save this message. The great thing about using rfhutil is that when we save the message, we are also saving all of the message header information for later use.

See Figure 6-15 on page 127.
Chapter 6. Unit testing the connector

17. Select **Write File**, **as in Figure 6-15**.
18. Save this file to a location for later use.

**Note:** Just a word of caution, you cannot edit this file with a text editor now because it will mess up the headers.

We now have a message, complete with all the header information required by the connector, that we can send in to emulate an inbound request to the connector.
6.2.4 Send test data to the test connector (RM_Tenant.Retrieve)

To send test data to the connector, do the following:

1. Open another instance of rfhutil.

2. Enter the Queue Manager REDBROKER and queue REDMAINT.REQUESTQUEUE. This is the inbound request queue for the connector.

3. Select Read File and read the file that was created in the previous steps. See Figure 6-17 on page 129.

Figure 6-16 Read message file
4. Select **Write Q** to put a message on to the queue.

5. Go back to the Test Connector DOS window.

6. You will see the arrival of the RM_Tenant.Retrieve business object as shown in Example 6-6. You might have to scroll back quite a bit.

**Example 6-6  Test Connector log**

Received request RM_Tenant.Retrieve
Received BO <StartHeader>
VERSION = 3.0>
<EndHeader>
<StartBO:RM_Tenant>

   BusinessObject = RM_Tenant
   Verb = Retrieve
   Locale = en_US

   Id = 100
   Name = CxIgnore
7. Go to the Test Connector GUI screen. You will see the arrival of the business object in the BO Request List on the right. See Figure 6-18.

![Figure 6-18 BO request list]

6.2.5 Send response to request (RM_Tenant.Retrieve)

To send a test response, do the following:

1. Double-click this business object to open it.
2. Fill in some details for the RM_Tenant object.
Figure 6-19  Response BO

3. Right-click the **RM_Apartment** object and select **Add Instance**, as in Figure 6-19.

4. Add some apartment data.

5. Right click the **RM_Maintenance** object and select **Add Instance**.

6. Add some maintenance data. See Figure 6-20 on page 132.
7. When you have completed the response data, click **OK**.
8. Select Request, as in Figure 6-21.

9. Select Reply → Success.

10. Check the Test Connector DOS window. You will see a message similar to:

    [Mesg::Since replyToQueue is not set, Receiver.sendResponse returns w/o sending back response.]

    The connector is telling us that although it has processed our response message successfully, the request message did not contain the information that the connector needs to know where to send the response \textit{bomo} (Business Object Message Object). In other words, there is no reply to queue destination.

11. Go back to rfhutil for our REQUESTQUEUE.

12. Select the MQMD tab.

    See Figure 6-22 on page 134.
13. Enter the Reply to Queue Manager as REDBROKER and Reply to Queue as REDMAINT.RESPONSEQUEUE. This queue will deliver the response message to the Brokers.

14. Send the message again.
15. Complete a new response BO as before. See Figure 6-23.
16. Send a reply of Success.
17. Check the Test Connector DOS window. You will see a message similar to:

```
[Msg: :sending the bomo --
CxCommon.Messaging.BusObjMsgObject@5999b71e to the replyToQueue --
queue://REDBROKER/REDMAINT.RESPONSEQUEUE]
```
18. Go back to the instance of rfhutil which is looking at the RESPONSEQUEUE.
19. Select **Read Q**.
20. Select the **Data** tab and **XML Data Format**.

At this point, we are expecting the message data to be a correct XML schema representation of our business object, similar to that shown in Figure 6-24.

![Figure 6-24  Response message](image)

**6.2.6 Prepare test data for processing (RM_Maintenance.Create)**

To prepare the test data for processing, do the following:

1. Using the method we have described, create a test message to use with an `RM_Maintenance.Create` request.
2. Send this business object.
3. Save the message on the DELIVERYQUEUE to a file for future use.

6.2.7 Send test data (RM_Maintenance.Create)

To send the test data, do the following:

1. Using the method described previously, put the test message on to the REQUESTQUEUE, remembering to add the ReplyToQueue.

2. Go back to the Test Connector and check that the request business object has been received.

   See Figure 6-26 on page 138.
3. Open the request business object.

4. Enter data that emulates a response from the application. In our example, we use a maintenance request Id of 4 because we know that it is the next that will be created in our database. We give it a status of A, which indicates an active, or open, request.

See Figure 6-27 on page 139.
Chapter 6. Unit testing the connector

5. Send this as a SUCCESS reply.

6. Check the Test Connector log for the response message to be sent, as in Example 6-7.

**Example 6-7 Test connector log**

```
[Msg: <StartBO:RM_Maintenance>]
[Msg:   BusinessObject = RM_Maintenance]
[Msg:   Verb = Create]
[Msg:   Locale = en_US]
[Msg:   Id = 4]
[Msg:   ApartmentId = 1]
[Msg:   Status = A]
[Msg:   TenantId = 100]
[Msg:   ProblemDescription = Test for a new maintenance request]
[Msg:   StatusDescription = CxIgnore]
[Msg:   ExpectedCompletion = CxIgnore]
[Msg:   ActualCompletion = CxIgnore]
[Msg:   ObjectEventId = RMConnector_1093618747016_2]
```

7. Go to rfhutil and check the contents of the message on the RESPONSEQUEUE. See Figure 6-28 on page 140.
6.3 Test interaction between connector and application

Now that we have established that the connector can process the business objects correctly, we need to test that the interaction between the connector and the application for business object handling works as we expect. We need to verify that when using the actual connector and back-end application, that the results we receive for request messages accurately reflect the application data. We also need to check that requests for creation of objects in the back-end application result in correct application objects being created, and correct events being generated.

Essentially, we will repeat the tests that we have performed in 6.2, “Test interaction between connector and business objects” on page 114. This time, however, we use the real connector so that we now interact with the real back-end application.

6.3.1 Send request business object (RM_Tenant.Retrieve)

To send a request BO to the RM_Tenant.Retrieve, do the following:
1. Stop the Test Connector.
2. Start the RedMaintenance engine.
3. Start the RMConnector.
4. Using the file from the RM_Tenant.Retrieve test, put a message on to the REDMAINT.REQUESTQUEUE using rfhutil as before. Remember to add the ReplyToQueue details to the Message Descriptor.
5. Check the RMConnector log. You will see the details of the business object being sent, similar to that shown in Example 6-8.

Example 6-8  Response business object

```
[Mesg: <Version = 3.0>]
[Mesg: <EndHeader>]
[Mesg: <StartBO:RM_Tenant>]
[Mesg:    BusinessObject = RM_Tenant]
[Mesg:    Verb = Retrieve]
[Mesg:    Locale = en_US]
[Mesg:    Id = 100]
[Mesg:    Name = Lee Gavin]
[Mesg:    ApartmentId = 1]
[Mesg:    EMail = leegavin@uk.ibm.com]
[Mesg: <StartChild>]
[Mesg:     RM_Apartment = 1]
[Mesg: <StartBO:RM_Apartment>]
[Mesg:     BusinessObject = RM_Apartment]
[Mesg:     Verb = Retrieve]
[Mesg:     Locale = en_US]
[Mesg:     Id = 1]
[Mesg:     ApartmentNumber = 135]
[Mesg:     AddressLine1 = IBM Hursley Park]
[Mesg:     AddressLine2 = Winchester]
[Mesg:     AddressLine3 = Hampshire]
[Mesg:     AddressLine4 = United Kingdom]
[Mesg:     PostCode = SO21 2JN]
[Mesg:     ObjectEventId = CxIgnore]
[Mesg: <EndBO:RM_Apartment>]
[Mesg: <EndChild>]
[Mesg: <StartChild>]
[Mesg:     RM_Maintenance = 3]
[Mesg: <StartBO:RM_Maintenance>]
[Mesg:     BusinessObject = RM_Maintenance]
[Mesg:     Verb = CxBlank]
[Mesg:     Locale = en_US]
[Mesg:     Id = 1]
[Mesg:     ApartmentId = 1]
[Mesg:     Status = C]
[Mesg: <EndBO:RM_Maintenance>]
[Mesg: <EndChild>]
```

Chapter 6. Unit testing the connector  141
6. Check the application log to ensure that the application is reporting successful processing. See Figure 6-29 on page 143.
Figure 6-29  Application log

It all looks good so far.

6.3.2 Check response message received

To check the response message, do the following:

1. Using rfhutil, read the message that was sent to the REDMAINT.RESPONSEQUEUE.

   See Figure 6-30 on page 144.
2. If you are using our test data, the message will be similar to that in Figure 6-30.

6.3.3 Send request business object (RM_Maintenance.Create)

The send the request BO, do the following:

1. Using the file from the RM_Maintenance.Create test, put a message on to the REDMAINT.REQUESTQUEUE using rfhutil as before. Remember to add the ReplyToQueue details to the Message Descriptor).
2. Check the log of the RMConnector. You will see the details of the business object being sent.

3. Check that the message on the RESPONSEQUEUE contains the newly created maintenance record. Verify this by checking that there is a new maintenance record in the database with the same key. See Figure 6-32 on page 146.
6.3.4 Create event

All events are stored in the back-end system. They are treated in the same way as all other objects such as tenant. The adapter calls to the back-end to manipulate the event. For example, when building the event store, the adapter calls out to the back-end to ask for events, which are retuned one at a time. On the GUI, you see this as multiple calls to retrieveObject. The adapter loops for a predefined number of events defined in the adapter properties. If there are not enough events to satisfy the adapter number to process, you see a failed retrieveObject, marked with a red square, in the back-end GUI. As events are processed, the adapter makes calls to the back-end to archive and delete events. These are again in the GUI.
To create the event, we send in a create request for a new maintenance record. This causes an event record to be written to the database by the back-end application.

1. Using your favorite DB2 tool, verify that the event record exists in the database. We used the DB2 Command Center.
   - The database name is SG246345.
   - The table name is itso.events.

![Command Center Image]

**Figure 6-33 Event record**

The event object matches the key value of the newly created maintenance record.

### 6.3.5 Check event message sent

One of the connector configuration properties that controls the polling for events is PollFrequency. This determines how often the connector polls for new events. In our connector, our configured value for this property is key. This enables us to control the polling while we test, so that the connector only polls when we enter `p` in the connector command window. The following message at connector startup reads:

```
[Msg: Press p to poll application.]
```

To poll while we test, do the following:

1. Enter `p` to start a poll for events.
2. Check the connector log to see that the event has been picked up and processed. See Figure 6-34 on page 148.

![Event Record Table]

![Figure 6-34 Event record](image-url)
Check the application log. As we mentioned earlier, the connector polls for a predetermined number of events. The GUI log records these events:

1. Indicates that the new event object is retrieved when one exists.
2. When it cannot find any more new events, an error will be flagged.
3. The event being processed is updated to indicate that it is now in process.
4. The newly created maintenance object is retrieved to be passed to the connector.
5. The event object is updated to indicate that it was successfully processed and passed on to the connector.
6. The event object is then retrieved for archive processing.
7. The event object is written to the archived events table for historical purposes.
8. The processed event object record is deleted from the events table.

At this point, we can check that the message for the Create event has been delivered to the DELIVERYQUEUE by the connector. See Figure 6-35 on page 149.
9. We should also double-check the archive events table to see that the archival process has been completed successfully. This table is itso.archive_events. See Figure 6-36.
6.4 Conclusions on unit test

As you can see, we have been able to test and verify the base functionality of our custom adapter without having to go through the process of creating message sets or message flows for a Message Broker.

All we have used are the tools provided and a few handy little tricks.

We are now ready to build all of the pieces in the Message Broker and tie all of our applications and runtime connectors together. The implementation for our solution of each integration broker is different based on the functionality. However, the one thing they will have in common is the basic infrastructure that we now have in place.
Build and test the integration

In this part of the Redpaper, we put into practice all of the steps we have performed throughout using the WebSphere Business Integration Message Broker.
Chapter 7. Deploy business objects to the Message Broker

In this chapter, we discuss the steps involved in deploying the business objects required for our scenario from the System Manager to the Message Broker Toolkit. We then migrate these message sets from the Message Broker Toolkit to our runtime Broker.

There are several ways that a business object definition can be imported from the System Manager and repository to the Message Broker:

- Using a tool from the System Manager
- Using the command line utility `mqsicreatemsgsets`
- From the Message Broker Toolkit, using a schema importer

Each of these ways has advantages and it is possible to use a combination of more than one method.
7.1 Front-end business objects

In this section, we first import our front-end business object definitions. These are the most complex when you look at them, so it is a good place to start.

7.1.1 Define Message Set

To define the Message Set, do the following

1. Open the Message Broker Toolkit. Use the desktop shortcut.
2. Select the Broker Application Development Perspective.
3. Select File → New → Message Set Project. See Figure 7-1.
4. Enter a Project Name of WebTenant.
5. Select Next.

6. Enter a Message Set Name of WebTenant. This matches our supported BO information in the connector configuration.
7. Check the Use namespaces box.
8. Select Next. See Figure 7-2 on page 155.
9. Select the **XML Wire Format Name** check box.

10. Enter a wire format name of *CwXML*. This matches our connector configuration property.

11. Select **Finish**.

12. Open the new message set, the *.mset* file.

13. Select the CwXML physical properties and ensure that the following properties are set as shown in Figure 7-3 on page 156:
   - Uncheck **Suppress XML Declaration**
   - Check **Suppress DOCTYPE**
   - Remove *mrm* as the Root Tag Name.

14. Save the Message Set by pressing Ctrl-S.
15. Save the Message Set definition and close the .mset file.


7.1.2 Import schema definitions and create message definitions

We can now import the schema files from the business object repository and create message definitions from these schemas.

1. Open the System Manager. First we must set our preferences for the Importer.

2. Select **Window → Preferences**.

3. Now select **System Manager Preferences → Broker Preferences**.
Figure 7-4  Set preferences

4. Set the preferences as shown in Figure 7-4.
   
   – Message Broker importer path
     C:\IBM\WBIMB501\eclipse\mqscreatemsgdefs.exe
   
   – Message Broker workspace directory
     C:\IBM\WBIMB501\eclipse\workspace

5. Click Apply.

Our completed artifacts can only be deployed to an Integration Broker from the appropriate type of User Project. We now need to create a User Project for deployment.

See Figure 7-5 on page 158.
6. From the System Manager, expand the **User Projects** folder.

7. Right-click **Message Broker Projects** and select **New Message Broker project**.

   See Figure 7-7 on page 159.
8. Name the new project. We called ours DeployToRedbroker.

9. Click the **RedMaintenance ICL** to add pointers to the objects in this ICL. Alternatively, you can select only the RM objects and the renamed Web objects from the RedMaintenance ICL, as we did.

10. Click **Finish**.

   See Figure 7-8 on page 160.
11. Right-click this new User Project. In our case, it is **DeployToRedbroker**.

12. Select **Deploy to Message Broker workspace**.

**Important:** You must not have the Message Broker Workspace open while performing the deployment.

See Figure 7-9 on page 161.
13. Select all of the business objects, our renamed Web_objects, for our front-end application.

14. Click **Next**.

    See Figure 7-10 on page 162.
15. Enter the parameters for the import.
   - Message set name is WebTenant.
   - Check **Deploy in verbose mode**, for logging.
   - Select **long** namespace support.

16. Click **Finish** to start the import.

**Important:** When the import has completed, the DOS window simply closes without a pop-up. This is usually OK. Cancel out of the GUI utility and check the log, as in step 17.

17. Check the log for successful completion. The log is usually located in the bin folder of the WBI Adapters installation directory and is named `mqsicreatemsgdefs.report.txt`. Example 7-1 on page 163 shows a sample of a successful import.
Example 7-1  Import report

Parameter -p (message set project) is "\\WebTenant\\WebTenant"
Parameter -d (directory of source files) is
"C:\IBM\WebSphereAdapters\Tools\WSWB203\Workspace\1106952327016"
Parameter -opt (options file) is "C:\DOCUME~1\db2admin\LOCALS~1\Temp\Broker40670tmp"

Importing file \WebTenant\importFiles\Web_newMaintenance.xsd

Creating message "Web_newMaintenance" from global element "Web_newMaintenance".
Changing schema location for import statement
"../webnewmaintenancenewmaintenance/Web_newMaintenance_newMaintenance.mxsd".
File "\\WebTenant\importFiles\Web_newMaintenance.xsd" imported successfully.

Elapsed time processing this message definition file: 12.031 seconds
Number of warnings for this message definition file: 0

Importing file \WebTenant\importFiles\Web_newMaintenance_newMaintenance.xsd

Creating message "Web_newMaintenance_newMaintenance" from global element
"Web_newMaintenance_newMaintenance".
File "\\WebTenant\importFiles\Web_newMaintenance_newMaintenance.xsd" imported successfully.

Elapsed time processing this message definition file: 1.75 seconds
Number of warnings for this message definition file: 0

Importing file \WebTenant\importFiles\Web_tenant.xsd

Creating message "Web_tenant" from global element "Web_tenant".
Changing schema location for import statement "../webtenanttenant/Web_tenant_tenant.mxsd".
File "\\WebTenant\importFiles\Web_tenant.xsd" imported successfully.

Elapsed time processing this message definition file: 2.985 seconds
Number of warnings for this message definition file: 0

Importing file \WebTenant\importFiles\Web_tenant_apartment.xsd

Creating message "Web_tenant_apartment" from global element "Web_tenant_apartment".
Changing schema location for import statement "../webtenantmaintenances/Web_tenant_maintenances.mxsd".
File "\\WebTenant\importFiles\Web_tenant_apartment.xsd" imported successfully.

Elapsed time processing this message definition file: 2.141 seconds
Number of warnings for this message definition file: 0
Importing file \\WebTenant\importFiles\Web_tenant_apartments.xsd

Creating message "Web_tenant_apartments" from global element "Web_tenant_apartments".
Changing schema location for import statement
"../webtenantapartment/Web_tenant_apartment.mxsd".
File "\\WebTenant\importFiles\Web_tenant_apartments.xsd" imported successfully.

Elapsed time processing this message definition file: 1.374 seconds
Number of warnings for this message definition file: 0

Importing file \\WebTenant\importFiles\Web_tenant_maintenance.xsd

Creating message "Web_tenant_maintenance" from global element "Web_tenant_maintenance".
File "\\WebTenant\importFiles\Web_tenant_maintenance.xsd" imported successfully.

Elapsed time processing this message definition file: 0.75 seconds
Number of warnings for this message definition file: 0

Importing file \\WebTenant\importFiles\Web_tenant_maintenances.xsd

Creating message "Web_tenant_maintenances" from global element "Web_tenant_maintenances".
Changing schema location for import statement
"../webtenantmaintenance/Web_tenant_maintenance.mxsd".
File "\\WebTenant\importFiles\Web_tenant_maintenances.xsd" imported successfully.

Elapsed time processing this message definition file: 0.907 seconds
Number of warnings for this message definition file: 0

Importing file \\WebTenant\importFiles\Web_tenant_tenant.xsd

Creating message "Web_tenant_tenant" from global element "Web_tenant_tenant".
Changing schema location for import statement
"../webtenantapartments/Web_tenant_apartments.mxsd".
File "\\WebTenant\importFiles\Web_tenant_tenant.xsd" imported successfully.

Elapsed time processing this message definition file: 2.969 seconds
Number of warnings for this message definition file: 0

Number of files imported: 8
18. Open the **Message Brokers Toolkit**.
19. Right-click the **WebTenant** project.
20. Select **Rebuild Project**.
21. Select **Refresh**.
22. You can now open up the messages to see the imported message definitions. Figure 7-11 shows the Web_tenant message.

![Figure 7-11 Web_tenant message](image-url)
23. We need to add a namespace prefix for each of our business object schemas as shown in Figure 7-12. This is done on the CwXML physical format on the message set.

<table>
<thead>
<tr>
<th>Namespace URI</th>
<th>Prefix</th>
</tr>
</thead>
</table>

Figure 7-12 Namespace settings

24. Add the namespace prefixes and save the Message Set. We chose to set the prefixes to the same as the business object names.

7.2 Back-end business objects

We now need to do the same for our back-end application business objects.

1. Message Set project name is RedMaintenance.
2. Message Set name is RedTenant. Note that these two names are not the same. This is intentional.
3. CwXML properties are the same as for our Web messages, including DTD and root tag settings.
4. Save the Message Set Project and close the Message Broker Toolkit.
5. Go back to the System Manager and, as before, import the business object definitions.
   See Figure 7-13 on page 167.
6. This time we select all of the business objects for our Red Maintenance application.

See Figure 7-14 on page 168.
7. Our destination project is the RedMaintenance project. Example 7-2 shows the successful import.

Example 7-2 Import report
Parameter -p (message set project) is "\RedMaintenance\RedTenant"
Parameter -d (directory of source files) is
"C:\IBM\WebSphereAdapters\Tools\WSWB203\Workspace\1106950226562"
Parameter -opt (options file) is "C:\DOCUME~1\db2admin\LOCALS~1\Temp\Broker40669tmp"

Importing file \RedMaintenance\importFiles\RM_Apartment.xsd

Creating message "RM_Apartment" from global element "RM_Apartment".
File "\RedMaintenance\importFiles\RM_Apartment.xsd" imported successfully.

Elapsed time processing this message definition file: 11.172 seconds
Number of warnings for this message definition file: 0
Importing file \RedMaintenance\importFiles\RM_Maintenance.xsd

Creating message "RM_Maintenance" from global element "RM_Maintenance".
File "\RedMaintenance\importFiles\RM_Maintenance.xsd" imported successfully.

Elapsed time processing this message definition file: 2.156 seconds
Number of warnings for this message definition file: 0

Importing file \RedMaintenance\importFiles\RM_PartOrder.xsd

Creating message "RM_PartOrder" from global element "RM_PartOrder".
File "\RedMaintenance\importFiles\RM_PartOrder.xsd" imported successfully.

Elapsed time processing this message definition file: 0.953 seconds
Number of warnings for this message definition file: 0

Importing file \RedMaintenance\importFiles\RM_Tenant.xsd

Creating message "RM_Tenant" from global element "RM_Tenant".
Changing schema location for import statement "../rmapartment/RM_Apartment.mxsd".
Changing schema location for import statement "../rmmaintenance/RM_Maintenance.mxsd".
File "\RedMaintenance\importFiles\RM_Tenant.xsd" imported successfully.

Elapsed time processing this message definition file: 2.5 seconds
Number of warnings for this message definition file: 0

Importing file \RedMaintenance\importFiles\RM_Worker.xsd

Creating message "RM_Worker" from global element "RM_Worker".
File "\RedMaintenance\importFiles\RM_Worker.xsd" imported successfully.

Elapsed time processing this message definition file: 0.625 seconds
Number of warnings for this message definition file: 0

Number of files imported: 5

8. Open the **Message Broker Toolkit**.
9. Rebuild and refresh the RedMaintenance project. Our RM_Tenant message looks as shown in Figure 7-15 on page 170.
10. Namespaces and prefixes should be amended as shown in Figure 7-16 on page 171.

11. Save the Message Set project.
Figure 7-16  Red Maintenance namespaces

Now that we have defined our Message Sets and messages, we can create the message flows for processing them.
Build and test message flow for Retrieve

Now that we have our business objects in place, we will build the message flows required to transform our data from the business objects used by the JMS connector (for the front-end) and the custom RM connector (for use by the back-end). We will discuss the ESQL required to perform the transformation and give some tips for troubleshooting. Finally, we will deploy our message flows to the Broker for our runtime environment.

We will start by building the round trip between the front-end and back-end for the retrieval of tenant and maintenance details.
8.1 Retrieve processing scenario

As shown in Figure 8-1, the retrieve processing and the responses to this processing go through a broker for data transformation and enhancement. In this case, we use the WebSphere Business Integration Message Broker. The figures outline the steps of the processing and which queues and message flows are used for the end-to-end retrieve processing.

Figure 8-1  Processing flow for request

1. Application tenant request           WEBTENANT.EVENT
2. Web_Tenant BO                      WEBTENANT.DELIVERY
3. RM_Tenant.Retrieve                 REDMAINT.REQUESTQUEUE
4. Application retrieve request       -
5. RM_Tenant BO                        REDMAINT.RESPONSEQUEUE
6. Web_Tenant BO                      WEBTENANT.REQUEST
7. Application maintenance response   application response queue
8. Web_Tenant BO                      WEBTENANT.RESULT

Open the Message Broker Toolkit.

8.2 Create a Message Flow project and Message Flows

We do not discuss any tracing in the flows here. However, it is strongly advised that you put trace nodes everywhere you can in the initial phases of developing and testing your flows.
1. Ensure you are in the Broker Application Development perspective.
2. **Select File → New → Message Flow Project.**
3. Name this new project. We chose RedMaintenanceFlows.
4. Right-click the new project and select **New Message Flow.**
5. Name this Message Flow. We chose RedTenant_to_RedMaint_Request.
7. Repeat steps 1-6 to create a second Message Flow. We chose RedMaint_to_RedTenant_Response.
8. Close this Message Flow.
9. Right-click the Message Flow Project and select **Properties → Project References.**
10. Set the project references as shown in Figure 8-2 on page 176 to include the Message Sets defined for the front and back-end.
11. Save the Message Flow Project.

**Tip:** If you are not familiar with the tracing available in the broker, try simply putting trace nodes with FILE trace properties and output to a file in the RedTenantTraces directory that we have created for you.

To see the entire message tree as it passed through the node use a pattern of ${Root}, see Appendix A. Appendix A contains samples of the sorts of valuable information that can be obtained from file traces.
8.3 RedTenant_to_RedMaint_Request

The request from the front-end will go through the JMS connector, where the application message becomes the event which is transformed to a business object and delivered to the broker for processing. The broker will then transform the data to the business object format required by the RM connector and put this on to the request queue of the RM connector.

Figure 8-3 on page 177 shows the fully finished request flow. We will build this up as we go along. You cannot add the Emitter subflow to your message flow yet, but this will be covered in detail when we discuss the monitoring of the runtime environment.
Figure 8-3   Full request message flow

Figure 8-4 shows that portion of the overall flow we will build in this stage.

Figure 8-4   What we build for this stage
The flow for this initial stage contains the following nodes:

- MQInput
- Compute (SetRouting)
- RouteToLabel
- Label (VerbRetrieve)
- Compute (PrepareMessage)
- MQOutput (OutputToRequestQueue)

To create the flow, do the following:

1. In the Message Flow editor for the RedTenant_to_RedMaint_Request flow, drag these nodes to the canvas and rename as in Figure 8-4 on page 177.

2. Wire the nodes together as shown in Figure 8-4 on page 177, omitting the MonitorEmitter subflow.

3. Open the MQInput node. Right-click it and select Properties. Set the properties. See Figure 8-5.

4. Set the input queue name to WEBTENANT.DELIVERY. This is the queue from the JMS connector which is delivering to the broker for processing. See Figure 8-6 on page 179.
5. Set the default Message Domain to **MRM** because we are expecting a message which was modeled in the broker.

6. Select the properties for the compute node and set as in Figure 8-7 on page 180.
   
   a. The ESQL module name is **RequestRouting**.
   
   b. The Compute Mode is **LocalEnvironment and Message**. This is required to allow us to set the routing information in the ESQL.
7. Click **Apply** for these changes.

8. Open the ESQL for the node and write the ESQL as shown in Example 8-1 on page 181. The ESQL does the following:
   a. Copy the entire input message to be transmitted.
      * If the inbound message type is a Web_tenant, then it must be a retrieval request. We set the OutputLocalEnvironment information, which is required for a RouteToLabel to ‘VerbRetrieve’.
      * If the inbound message type is a Web_maintenance, then it must be a create request. However, as the JMS connector uses a default verb (where the application is not able to make a distinction - being nothing more than some application data being dropped on a queue), we need to route to the Create processing. We set the OutputLocalEnvironment information which is required for a RouteToLabel to ‘VerbCreate’.
      * If the inbound message type is RM_Maintenance, then we set the routing for Update.
Note: If you are not familiar with writing ESQL, we have included a text file for each of the ESQL modules. They are in the ESQL folder of the Additional Materials. The name of each file is <ESQL module name>.txt. RequestRouting.txt for example, is shown in Example 8-1.

You can use these as an alternative to writing ESQL. It is worth pointing out, however, that so far we are only doing easy stuff. The programming will get much more complex as we move through the solution. It really is worth the effort to follow what the ESQL is doing, and giving it a try.

Example 8-1 RequestRouting

CREATE COMPUTE MODULE RequestRouting
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

        CALL CopyEntireMessage();

        -- Tenant Retrieving Their Details
        IF InputRoot.Properties.MessageType = 'Web_tenant' THEN
            SET
            OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname='VerbRetrieve';
            END IF;

        -- Due to the default verb on the JMS connector - we need to decide what we are really doing here
        IF InputRoot.Properties.MessageType = 'Web_newMaintenance' THEN
            SET
            OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname='VerbCreate';
            END IF;

        -- This will be the Emitter Pass-Thru for the Contractor Update
        IF InputRoot.Properties.MessageType = 'RM_Maintenance' THEN
            SET
            OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname='VerbUpdate';
            END IF;

        RETURN TRUE;
    END;
CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
        SET OutputRoot.[I] = InputRoot.[I];
        SET I = I + 1;
    END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
END;

END MODULE;

9. Save the ESQL.

10. Open the properties of the RouteToLabel, just for interest. We do not need to modify them at all. See Figure 8-8.

Figure 8-8 VerbRetrieve label
11. Open the properties of the Label (VerbRetrieve) node and set the label name to VerbRetrieve. This must match the VerbRetrieve value that was set in the ESQL:

\[\text{OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname;}\]

12. Open the properties of the PrepareMessage compute node.

13. Set the ESQL module to \textbf{RequestRetrieve}.

14. \textbf{Apply} the change.

15. Complete the ESQL. The ESQL will perform the following:

\begin{enumerate}
\item Copy the message headers from input message to output message.
\item Setup the namespace declarations for the front-end and back-end business object definitions.
\item Set the properties of the output message.
\item Set the message descriptor format to indicate an RFH header is included.
\item Set the RFH header values:
\begin{itemize}
\item Format = string
\item Message Domain = mrm
\end{itemize}
\end{enumerate}
- Message Set = RedTenant
- Message Type = RM_Tenant
- Wire Format = CwXML
- Reply queue = Adapter Response queue

f. Set the business object top-level values:
   - version = 3.0.0
   - delta = FALSE
   - verb = taken from input business object
   - locale = en_US

g. Set the business object data values:
   - Output tenantid id is taken from the input tenant name
   - Output ObjectEventId is taken from the input ObjectEventId

Note: If you were writing this without any instructions such as these, you probably would have taken a few traces to see where in the input and output message body you will find the values that you require. See Example 8-2.

Example 8-2  Without ESQL

(0x0100001B):MRM = (
  (0x0300000B):version = '3.0.0'
  (0x0300000B):verb = 'Retrieve'
  (0x0300000B):locale = 'en_US'
  (0x0300000B):delta = FALSE
)

(0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:XMLDeclaration = "xml version="1.0" encoding="UTF-8" standalone="yes""

(0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:ROOT = (

  (0x0300000B):version = '3.0.0'
  (0x0300000B):verb = ''
  (0x0300000B):locale = 'en_US'
  (0x0300000B):delta = FALSE
)
The trace output in shows the input message. The one tiny piece of information we require at this point is buried quite deeply in the message.

h. Set the last of the message properties:
   - MessageDomain = mrm
   - MessageFormat = CwXML.

Example 8-3 shows the full ESQL.

Example 8-3  RequestRetrieve

CREATE COMPUTE MODULE RequestRetrieve
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN

      CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();

      -- *** Schema Declarations ***

      -- Red Tenant Web Application
      DECLARE Web_tenant NAMESPACE
      DECLARE Web_tenant_tenant NAMESPACE

      -- Red Maintenance Back-end Application
      DECLARE RM_Tenant NAMESPACE
-- ** Properties
SET OutputRoot.Properties.MessageSet = 'RedTenant';
SET OutputRoot.Properties.MessageType = 'RM_Tenant';

-- Enter SQL below this line. SQL above this line might be regenerated, causing any modifications to be lost.
-- ** MQMD
SET "OutputRoot"."MQMD".Format = 'MQHRF2';

-- ** RFH
SET "OutputRoot".MQRFH2.(MQRFH2.Field)Format = 'MQSTR';
SET "OutputRoot".MQRFH2.mcd.Msd = 'mrm';
SET "OutputRoot".MQRFH2.mcd.Set = 'RedTenant';
SET "OutputRoot".MQRFH2.mcd.Type = 'RM_Tenant';
SET "OutputRoot".MQRFH2.mcd.Fmt = 'CwXML';
-- response queue
SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///REDMAINT.RESPONSEQUEUE';

SET OutputRoot.MRM."version" = '3.0.0';
SET OutputRoot.MRM."delta" = FALSE;
set OutputRoot.MRM."verb" = InputRoot.MRM.verb;
SET OutputRoot.MRM."locale" = 'en_US';

-- Set BO fields
SET OutputRoot.MRM.RM_Tenant:ObjectEventId = InputRoot.MRM.Web_tenant:ObjectEventId;
-- ** More properties
SET OutputRoot.Properties.MessageDomain = 'MRM';
SET OutputRoot.Properties.MessageFormat = 'CwXML';

RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
WHILE I < J DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I = I + 1;
END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
SET OutputRoot = InputRoot;
END;
16. Save this ESQL.
17. Open the properties of the MQOutput node. See Figure 8-10.

![MQOutput Node Properties - OutputToRequestQueue](image)

**Figure 8-10** MQOutput node

18. Set the output queue for the RMConnector request queue (REDMAINT.REQUESTQUEUE).
19. Save your message flow.

Your completed flow should look like Figure 8-11 on page 188, plus any additional trace nodes that you have added.
8.4 Deploy

In this section we will deploy our Message Sets and the first of our Message Flows to the Broker.

Before we do this, clear the messages off any of the adapter queues so we have a clean starting point for testing. Ensure that the RMConnector is not running and that the JMSConnector is running either in polling or not polling mode. It does not matter at this point.

1. Navigate to the Message Broker Toolkit.
2. Switch to Broker Administration perspective.
3. In the Domains panel, right-click the Broker Domain name and select Connect to connect to the Configuration Manager.
4. Once you are connected, right-click the Broker name and select New → Execution Group. See Figure 8-12 on page 189.
Figure 8-12   New execution group

5. Enter a name. We chose RedMaintenance.

6. Click **Next**.

7. Enter a description.

8. Click **Finish**.

Figure 8-13   Domains view
9. In the Domains View (Figure 8-13 on page 189), look for the name of the new Execution Group with a yellow exclamation mark next to it. This indicates that there are currently no Message Flows running in this Execution Group.

10. The Alerts view in Figure 8-14 confirms this.

![Figure 8-14  Alerts view](image)

We now need to deploy our Message Sets and Message Flows to the broker.

**Note:** If you are unfamiliar with the latest release of the Message Broker, it is slightly different from prior releases. In Version 5 of the Message Broker, both Message Flows and Message Sets are deployed to the Execution Group in which they will run using a BAR, or Broker Archive file.

11. Select **File → New → Message Broker Archive.** See Figure 8-15 on page 191.
12. Enter a name for the new BAR file. We chose RedMaintenance_Archives.

13. Select **Finish**. The new empty BAR file will be ready for us to add the artifacts.

   See Figure 8-16 on page 192.
14. Click the **Add** button with the green square.

15. Click the boxes for the Message Sets and Message Flows, as in Figure 8-17.
16. Click **OK**. You can see the progress of the operation in Figure 8-18 and the finished state in Figure 8-19.

![Progress window](image1.png)

*Figure 8-18  Progress window*

![Operation completed](image2.png)

*Figure 8-19  Operation complete*

17. When the operation has completed, click **OK**. We now see the contents of the BAR file, as in Figure 8-20 on page 194.

**Important**: If you are not familiar with BAR files, these contain compiled versions of your artifacts. If you make *any* changes to these, you must reimport them to the BAR file and redeploy it. Notice a time and date stamp in Figure 8-20 on page 194. This indicates the last save time for each.
18. Save the BAR file.

**Tip:** If you are not familiar with the Eclipse workspace, be aware that anything in the Editors view, the main view, with an asterisk (*) in front of it as in Figure 8-20, indicates that this resource has unsaved changes.

19. Right-click the BAR file name in the Resource Navigator View and select **Deploy File**.

See Figure 8-21 on page 195.
Chapter 8. Build and test message flow for Retrieve

20. Select the **RedMaintenance Execution Group**.

21. Click **OK**. As in Figure 8-22, you receive a response message from the Configuration Manager.

**Figure 8-21  Select Execution Group**

**Figure 8-22  Configuration Manager response**
This does not mean that your BAR file has successfully deployed. It merely indicates that the Configuration Manager has accepted your request and initiated a deploy.

22. Double-click the Event Log in the Domains view to open the Broker Domain Event Log. It contains messages pertaining to deployment events. Check for a successful deployment message. This is not the same thing as the Windows Event Viewer.

### 8.5 Unit test flow

Now that we have the first of our flows running, we can test the transformation of a front-end request to a back-end business object. Before we start, clear any left-over messages from previous unit testing.

1. Open the browser window for the front-end application.

2. Enter a Tenant Id. We chose 100 or 101 because these have maintenance records in the back-end.

3. If the JMSCconnector is set for manual polling, then instigate a poll cycle.
   
   We are now expecting that the broker has picked the message up from the DELIVERY queue, transformed it and put it to the REQUEST queue for the RMConnector.

4. Check that this message exists by using rfhutil in browse mode.
   
   See Figure 8-23 on page 197.
5. Click the **Data** tab to see the message for the transformed object, as in Figure 8-24 on page 198.
6. Check the traces we have supplied for you if you need some clues:
   - `InputMessageFromFrontEnd` shows the message tree as it was delivered to the broker on the MQInput. See Example A-1 on page 313.
   - `RedTenantDelivery_Retrieve` shows the path that the message took after the `RouteToLabel` and the message tree as it was prior to the Compute. See Example A-2 on page 315.
   - `RedMaintRequest` shows the message tree after the Compute, that is, what should have gone to the REQUEST queue. See Example A-3 on page 318.

**Tip:** If you do not see all of this, check the Windows Event Viewer, not the Domain Event Log. The Windows Event Viewer will contain runtime errors from the broker, while the Event Log only contains deployment time errors.
The final test for our first message flow will be to let the RMConnector pick up our message and process it correctly.

1. Ensure that the RedMaintenance application is running.
2. Start the RMConnector.
3. We should now see the RMConnector process the message from the REQUEST queue and place the result of the retrieve operation on the RESPONSE queue for the broker (REDMAINT.RESPONSEQUEUE).
4. Check the application log for a successful retrieve. See Figure 8-25.

5. Using rfhutil, browse the message from the REDMAINT.RESPONSEQUEUE and check that it resembles Figure 8-26 on page 200.
6. When all of this is correct, we are ready to build the message flow to handle the response.

7. Shut down the JMSConnector.

### 8.6 RedMaint_to_RedTenant_Response

The back-end application transmits the results of the look-up to the RM connector, which transforms the application data to a business object and places it on the response queue for the broker. The broker transforms this data to the business object format required by the JMS connector and puts the message on to the request queue for the JMS connector. The JMS connector transforms the data to the application format required by the front-end application and puts it on to the application’s response queue.

See Figure 8-27 on page 201 for an overview of this process.
Do not concern yourself with the emitter for the time being.

The flow for this initial stage contains the following nodes:

- MQInput
- Compute (SetRouting)
- RouteToLabel
- Label (ResponseRetrieve)
- Compute (PrepareTenantMessage)
- MQOutput (RespondToRequestor)

1. Open the **RedMaint_to_RedTenant_Response** flow.
2. Drag these nodes to the Message Flow Editor canvas, rename them accordingly and wire them together.
8.7 ResponseRetrieve

To create the ResponseRetrieve object, do the following:

1. Open the MQInput node properties.

![MQInput node properties](image)

2. Set the input queue name to `REDMAIN.RESPONSEQUEUE`, the response queue from the RMConnector.

3. Set the default message domain to `mrm`. See Figure 8-29 on page 203.
4. Open the SetRouting compute node properties and set the Compute Mode to `LocalEnvironment and Message`.

5. Set the ESQL Module name to `ResponseRouting`.

6. Create the ESQL as shown in Example 8-4.

   The ESQL will do the following:
   a. Copy the entire input message to be passed on.
   b. Set the routing destination based on the Verb of the message.

```
Example 8-4  ResponseRouting

CREATE COMPUTE MODULE ResponseRouting
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   -- CALL CopyMessageHeaders();
   -- CALL CopyEntireMessage();

   CALL CopyEntireMessage();

   -- RESPONSE FROM TENANT RETRIEVE REQUEST
```
IF InputRoot.MRM.verb = 'Retrieve'
   AND InputRoot.Properties.MessageType = 'RM_Tenant' THEN
   SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname
   = 'ResponseRetrieve';
ELSE
   -- CREATE MAINTENANCE RESPONSE
   IF InputRoot.MRM.verb = 'Create' THEN
      SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname
   = 'ResponseCreate';
   ELSE
      -- RESPONSE FROM CONTRACTOR UPDATE REQUEST
      IF InputRoot.MRM.verb = 'Update' THEN
         SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname
      = 'ResponseUpdate';
      END IF;
   END IF;
END IF;
RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
   DECLARE I INTEGER 1;
   DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
   WHILE I < J DO
      SET OutputRoot.*[I] = InputRoot.*[I];
      SET I = I + 1;
   END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
   SET OutputRoot = InputRoot;
END;
END MODULE;

7. Save this ESQL.
8. Open properties of the ResponseRetrieve label. See Figure 8-30 on page 205.
9. Set the LabelName to match that set in the routing ESQL.
10. Open the properties of the PrepareTenantMessage compute node. See Figure 8-31 on page 206.
11. Set the ESQL Module name to **ResponseRetrieve**.

12. Complete the ESQL as shown in Example 8-5 on page 207.

   The ESQL will perform the following:
   
   a. Copy the message headers from input message to output message.
   
   b. Setup the namespace declarations for the front-end and back-end business object definitions.
   
   c. Set the properties of the output message.
   
   d. Set the message descriptor format to indicate an RFH header is included.
   
   e. Set the RFH header values:
      
      - Format = string
      - Message Domain = mrm
      - Message Set = WebTenant
      - Message Type = Web_tenant
      - Wire Format = CwXML
      - Reply queue = Adapter Result queue, eventhough we are not going to do anything with these results.
f. Set the business object top-level values:
   - version = 3.0.0
   - delta = FALSE
   - verb = taken from input business object
   - locale = en_US

  g. Set the business object data values:
   - XMLDeclaration
   - Schema location (The JMS connector uses it for the application data.

  h. Set the tenant object details:
   - The output Id is taken from the input tenant object Id.
   - The output name is taken from the input tenant object name.
   - The output email is taken from the input tenant object email.

  i. Set the address object details.

  j. Loop through and for each input maintenance record, create an output maintenance record. Make it look formatted and pretty.

  k. Set the last of the message properties.

---

Example 8-5  ResponseRetrieve

CREATE COMPUTE MODULE ResponseRetrieve
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   -- (Remove for mapping node
   CALL CopyMessageHeaders();
   -- CALL CopyEntireMessage();

   -- *** Schema Declarations ***

   -- Red Tenant Web Application
   DECLARE Web_tenant NAMESPACE
   DECLARE Web_tenant_tenant NAMESPACE
   DECLARE Web_tenant_maintenance NAMESPACE
   DECLARE Web_tenant_maintenances NAMESPACE
   DECLARE Web_tenant_apartment NAMESPACE
   DECLARE Web_tenant_apartments NAMESPACE

---
-- Red Maintenance Back-end Application

-- ** Properties
SET OutputRoot.Properties.MessageSet = 'WebTenant';
SET OutputRoot.Properties.MessageType = 'Web_tenant';

-- Enter SQL below this line. SQL above this line might be regenerated, causing any modifications to be lost.
-- ** MQMD
SET "OutputRoot"."MQMD".Format = 'MQHRF2  ';
SET OutputRoot.MQMD.MsgType = 8;
SET OutputRoot.MQMD.ReplyToQ = ' ';
SET OutputRoot.MQMD.ReplyToQMgr = ' ';

-- ** RFH
SET "OutputRoot".MQRFH2.(MQRFH2.Field)Format = 'MQSTR   ';
SET "OutputRoot".MQRFH2.mcd.Msd = 'mrm';
SET "OutputRoot".MQRFH2.mcd.Set = 'WebTenant';
SET "OutputRoot".MQRFH2.mcd.Type = 'Web_tenant';
SET "OutputRoot".MQRFH2.mcd.Fmt = 'CwXML';

-- response queue
SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///WEBTENANT.RESULT';

-- Include for mapping mode
--   SET OutputRoot.MRM = InputRoot.MRM ;

-- ** Set Verb and locale information
SET OutputRoot.MRM."version" = '3.0.0';
SET OutputRoot.MRM."delta" = FALSE;
SET OutputRoot.MRM."verb" = InputRoot.MRM.verb;
SET OutputRoot.MRM."locale" = 'en_US';

-- ** Set BO fields

-- Web_tenant:XMLDeclaration
SET OutputRoot.MRM.Web_tenant:XMLDeclaration = 'xml version="1.0" encoding="UTF-8"';

-- ** set the schema location so the JMS connector can fill in the details for the onwards processing to the application
-- 
SET
tion
  = 'http://www.ibm.com/RedTenant_tenant.xsd';

-- ** Set the tenant details

SET
  InputRoot.MRM.RM_Tenant:Id;
SET
  InputRoot.MRM.RM_Tenant:Name;
SET
  InputRoot.MRM.RM_Tenant:EMail;

-- ** Set the address

--
SET
  .Web_tenant_apartments:Web_tenant_apartments:apartment.size = '1';

--
  SET
  = CAST(InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:Id AS CHARACTER) ;

--
  SET
=
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:ApartmentNumber || \
' ' ||
  InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:AddressLine1 || ' ' ||
  InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:AddressLine2 || ' ' ||
  InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:AddressLine3 || ' ' ||
  InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:AddressLine4
;
---

SET
=
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment:PostCode
;
--- *** Set the maintenance records

DECLARE C INTEGER 1;
DECLARE M INTEGER;
SET M = CAST(InputRoot.MRM.RM_Tenant:RM_Maintenance.size AS INTEGER);

SET
= CAST(M AS CHARACTER);

WHILE C <= M DO

  DECLARE MaintInfo CHARACTER;
  SET
CAST(InputRoot.MRM.RM_Tenant:RM_Maintenance:RM_Maintenance:RM_Maintenance[\(\text{C}\)].RM_Maintenance:Id AS CHARACTER);

CASE
  WHEN 'A' THEN
    SET
  WHEN 'P' THEN
    SET
  WHEN 'C' THEN
    SET
  ELSE
    SET
  END CASE;

SET MaintInfo = ' ';

IF
  ELSE
SET MaintInfo = MaintInfo ||
END IF;
IF
ELSE
  SET MaintInfo = MaintInfo || '  CURRENT STATUS INFO: ' ||
END IF;
IF
ELSE
  SET MaintInfo = MaintInfo || '  EXPECTED COMPLETION DATE: ' ||
END IF;
IF
ELSE
  IF
    SET MaintInfo = MaintInfo || '  ACTUAL COMPLETION: ' ||
  END IF;
END IF;
END IF;
SET
.Web_tenant_maintenance:description = MaintInfo;
SET C = C + 1;
END WHILE;

-- ** More properties

SET OutputRoot.Properties.MessageDomain = 'MRM';
SET OutputRoot.Properties.MessageFormat = 'CwXML';

RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I = I + 1;
END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
  SET OutputRoot = InputRoot;
END;
END MODULE;

13. Save the ESQL.

14. Open the properties of the MQOutput node (RespondToRequestor).
15. Set the output queue to WEBTENANT.REQUEST, the request queue for the JMS connector. See Figure 8-32.

Your complete flow up to this point should look similar to Figure 8-33, plus any trace nodes you are using.
As you can see from the ESQL in this flow, this is where the real fun starts! This transformation could have been achieved with a combination of mapping nodes and ESQL. However, we are doing it all in the one node using ESQL.

**Tip:** In the Eclipse toolkit in the ESQL Editor, there is a context-sensitive facility called Content Assist, or Code Assist. When you use the ESQL Editor with predefined messages you can use the Content Assist, invoked by Ctrl+Space, to construct field references.

For example, for our RM_Tenant messages, if you were to type InputRoot.MRM (Ctrl+Space), you would see a drop-down box showing the next level of the message (RM_Tenant:RM_Maintenance for example).

This and a decent trace output of the message tree will help you a lot when you are traversing the multi-level messages.

16. Save this Message Flow.

8.8 Deploy the Message Flows

To deploy the message flows, do the following:

1. Navigate back to the Broker Administration Perspective.
2. Update the BAR file to now include the new Message Flow.
3. Save the BAR file.
4. Redeploy the BAR file to the RedMaintenance Execution Group.
5. We expect to see both Message Flows deployed to the broker.

8.9 Unit test flow

You should already have a message on your REDMAINT.RESPONSEQUEUE from the previous test. If you read the message instead of browsed it, recreate the test.

As soon as the Message Flow is deployed, the message should have been picked up from the queue and processed.

1. It should now be on the WEBTENANT.REQUEST queue in its transformed state.
2. Use rhutil to browse the message on the queue. See Figure 8-34 on page 216.
Finally, we need to test that the JMSConnector will take our business object and turn it into data that is acceptable to the application.

3. Start the JMSConnector.

4. The Message is picked up by the connector, transformed to application format and put on the application response queue, RTRES1Q at queue manager REDTENANT.

5. Use rfhutil to browse this message and verify that it looks like a reasonably correct application message. See Figure 8-35 on page 217.
6. Shutdown the JMSConnector. We no longer need it and will modify it prior to our full end-to-end test.

7. Navigate back to the browser window and reissue the request for tenant 100, or whichever tenant you used. See Figure 8-36 on page 218.
8. We now expect that the message will be picked up from the application response queue and the results displayed correctly as shown in Figure 8-36.

8.10 End-to-end test

Only one thing remains, and that is to put all the pieces together on automatic and see it run from end to end without assistance or manual intervention.

For this we want to ensure that the JMS connector is polling correctly. To start the end to end test, do the following:

1. Check that none of our queues have leftover messages on them from our previous testing.

2. Open the System Manager and Connector Configurator and change the polling frequency for the JMSConnector to 1000, meaning 1000 milliseconds; every one second, our application deserves decent response)

3. Save the change to the project and to file.

4. Start the JMSConnector, ensure that the new polling has taken affect.

5. Ensure that all other components are running:
   - Message Flows
   - RMConnector
   - RedMaintenance application
6. Open a browser window. To prove that we are not cheating, try entering another tenant Id. We use 101 because we know that it exists and has maintenance records.

![Test 101](image)

7. Check for the correct response received, as in Figure 8-38 on page 220.
8. Try another one. Tenant 102 has an apartment, but no maintenance records. Test that our transformations handle this correctly.

10. In Figure 8-39, tenant 102 is returned with the name and apartment details but no maintenance records and no errors.

11. Once more, try the old faithful tenant 100 again from start to finish. Return to the RedTenant home page and enter tenant 100. See Figure 8-40 on page 222.
12. Check a maintenance record, as in Figure 8-41.

We now have the retrieve component of the solution working correctly. Now we move on to the creation of new maintenance requests.
Build and test message flows for Create

Now that we have basic request and response flow in place, we will extend the flows to incorporate the requests for new maintenance orders from the front-end application. We will detail the nodes and ESQL required to perform the transformation. Finally, we will deploy our message flows to the Broker for our runtime environment.
9.1 Create a processing scenario

As shown in Figure 9-1 and Figure 9-2 on page 225, the request processing and the responses to this processing will go through a broker for data transformation and enhancement. In this case we will use the WebSphere Business Integration Message Broker. The figures outline the steps of the processing and which queues and message flows will be used for the end-to-end maintenance create processing.

1. Application maintenance request WEBTENANT.EVENT
2. Web_newMaintenance BO WEBTENANT.DELIVERY
3. RM_Tenant.RetrieveByContent REDMAINT.REQUESTQUEUE
4. Application retrieve request -
5. RM_Tenant BO REDMAINT.SYNCHRONOUSRESPONSEQUEUE
6. RM_Maintenance.Create REDMAINT.REQUESTQUEUE
7. Application maintenance create request -
8. RM_Maintenance BO REDMAINT.RESPONSEQUEUE
9. Web_newMaintenance WEBTENANT.REQUEST
10. Application maintenance response application response queue
11. Web_newMaintenance BO WEBTENANT.RESULT

Figure 9-1  Processing flow request
We first need to extend the message flow for the request processing. When we initially created the flow, our routing catered to Web_newMaintenance requests arriving on the queue. We now need to add the processing to the flow which will cater to the processing of these new messages.

We need to explain the processing that we will be using for the maintenance create. In a perfect world, we would receive all of the details from the front-end application that are required to enable the request passed to the back-end to be processed successfully. Unfortunately, things do not always work out the way that we would like them to, especially when we are dealing with applications that cannot be modified.

In the case of the request for a new maintenance record, the front-end application passes us the tenant name and the details of the new request only. The reason for this is the entire process used to be mostly manual and the call center operator had screens for both applications available. This is not enough information for the back-end application to be able to create a maintenance request. The back-end application requires the tenant Id and the apartment Id because these are mandatory elements of the maintenance object in the application.
We have a fairly straightforward way of accomplishing this when we are using the adapter. The RetrieveByContent verb allows us to retrieve the tenant details using nonkey values because this is an exposed function of the back-end application API. We will store the details of the original maintenance request for later use. This could be done several ways, which we will describe a little later.

We will send a RetrieveByContent based on what we know about the tenant, the tenant name.

We could have easily had the message come back to the main flow by adding a new routing for RetrieveByContent verbs, but we want to show you another message flow. Instead, we will send the message to the synchronous response queue, which is not being used anyway. This will, in turn, start another flow to build the business object data and pass it back to the request queue for adapter processing.

To start building the RedTenant_to=RedMaint-Request Flow, do the following:
1. We add the following nodes to the Message Flow Editor canvas for RedTenant_to_RedMaint_Request:
   - Label (VerbCreate)
   - DataInsert (StoreRequestInfo)
   - Compute (SetUpRetrieveByContent).
2. Connect these nodes together according to Figure 9-3 on page 227.
3. Open the properties of the Label node and set the label to VerbCreate.
   See Figure 9-4 on page 228.
4. We now need to add the database we are using to store the request information.

   **Note:** The ddl for this is in the Additional Materials in the REDHOLD folder.

5. Open the **Data** perspective by selecting **Window → Open Perspective → Data**.

6. Right-click the **DB servers** view of the Data perspective and choose **New Connection** as shown in Figure 9-5 on page 229.
7. Enter properties for the database connection.
   - Connection name is REDHOLD
   - Database name is REDHOLD
   - User id is db2admin
   - Password is the db2admin password

8. Click Browse to set the Class Location for the DB2 JDBC driver, or enter the location. On the student machines this will be C:\sql\lib\java\db2java.zip. Click Finish to complete the connection.

9. After the database connection is made, the DB Servers view shows details of the database including schemas and tables. Right-click the database and choose Import to Folder, as in Figure 9-6 on page 230.
10. Select the **RedMaintenanceFlows** project and click **OK**.

11. Click **Finish**. You see the database in the RedMaintenanceFlows project as shown in Figure 9-7.
12. Navigate back to the Broker Application Development perspective. We can add the database to our DataInsert node and map the insert.

13. Open the properties of the DataInsert node and add the REDHOLD data source, as in Figure 9-8.

![DataInsert Node Properties - StoreRequestInfo](image)

*Figure 9-8  Add Data Source*

14. Click **Apply**.

15. Right-click the Data Insert node and select **Open Mappings**. See Figure 9-9 on page 232.
16. Right-click the Source pane of the Mapping Editor.
17. Select \textbf{Add Message Mapping Input}.

See Figure 9-10 on page 233.
Chapter 9. Build and test message flows for Create  233

18. Select the **Web_newMaintenance** message.

19. Select **Finish**.

   See Figure 9-11 on page 234.
20. Right-click in the **Target** pane of the Mapping Editor.

21. Select → **Add RDB Table Mapping Object**.

   See Figure 9-12 on page 235.
Chapter 9. Build and test message flows for Create 235

Figure 9-12  Add database table

22. Select a database from the workspace.
23. Click Next.
   
   See Figure 9-13 on page 236.
24. Select the REDHOLD database that we added to our Message Flow Project.

25. Click **Finish**.
26. Your Source and Target look as shown in Figure 9-14.
   Mapping can be accomplished fairly easily using a drag and drop function.

27. Drag the ObjectEventId from the Source to the OBJECTID of the Target as in Figure 9-15.

   **Note**: Make sure you take the ObjectEventId at prefix:ObjectEventId, not the one at prefix:ROOT.ObjectEventId.

28. Drag the description from the Source to REQUEST of the Target.

   ![Figure 9-15 Mapping field to field](image)

   This maps from one field to another and creates an insert request for each.
   We need a single, consolidated insert to the database.

29. In the Outline View, highlight both of the mappings as shown in Figure 9-16 on page 238.

30. Right-click and select → **Combine to Same Row**, as in Figure 9-16 on page 238.
31. Select **Remove Selected Mappings**. This leaves us with a single insert and none of the original extraneous mappings.

32. You can check that your mappings are still correct for each field by right-clicking each element in the Overview pane, as shown in Figure 9-17 and Figure 9-18 on page 239.

![Figure 9-16 Combine mappings](image)

**Figure 9-16 Combine mappings**

![Figure 9-17 ObjectEventId](image)

**Figure 9-17 ObjectEventId**
Chapter 9. Build and test message flows for Create

Figure 9-18  REQUEST

33. Save the mappings.

**About using the database for persisting message data:** We have used a database to temporarily store the request information. This is required to persist it over multiple flows, or units of work. We have used this method because it is very simple and quick to set up to demonstrate our solution.

If you have the time and are familiar with the message aggregation function in the Message Broker, you might want to consider it as an alternative to using a database datastore. Basically, you can fan out the messages with the use of two aggregation requests, one for a message to a queue holding the original request data and one for the RetrieveByContent message to the connector queue. The fan-in function takes the message from the held data queue and also the reply message from the RetrieveByContent and allows you to aggregate them into the create request.

Try this if you get a chance. It is very cool.
34. Open the compute node properties. See Figure 9-19.
35. Set the ESQL Module to RetrieveByContent.
36. Set the Compute Mode to LocalEnvironment and Message.
37. Click Apply to save the changes.
38. Open the ESQL for this node.
39. Create the ESQL as shown in Example 9-1 on page 241.

The ESQL will perform the following:

a. Copy the message headers.
b. Declare the required namespaces.
c. Set the properties for Message Set (RedTenant) and Message Type (RM_Tenant).
d. Set the MQMD for an RHF2 header.
e. Set the RFH2 properties:
   • Format = MQSTR
   • Message Domain = mrm
• Message Set = RedTenant
• Message Type = RM_Tenant
• Wire Format = CwXML
• Reply queue = REDMAINT.SYNCHRONOUSRESPONSEQUEUE

f. Set the schema attributes.

g. Set the business object fields:

• Output Tenant Id = 0
  It is a function of the application API that if a mandatory field is not being used in a RetrieveByContent that it still must have a value. Setting this to zero will let the application know that we do not know the value for this element and that it should use the values that we pass as the selection criteria.

• Output Tenant name is the value from the Input Tenant Id
• Output ObjectEventId is the value of the Input ObjectEventId

h. Set the last of the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 9-1  RetrieveByContent

CREATE COMPUTE MODULE RetrieveByContent
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN

    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();

    -- ***  Schema Declarations ***

    -- Red Tenant Web Application
    DECLARE Web_tenant NAMESPACE
    DECLARE Web_tenant_tenant NAMESPACE
    DECLARE ns NAMESPACE
    DECLARE null1 NAMESPACE

    -- Red Maintenance Back-end Application
    DECLARE RM_Tenant NAMESPACE

    -- ** Properties
SET OutputRoot.Properties.MessageSet = 'RedTenant';
SET OutputRoot.Properties.MessageType = 'RM_Tenant';

-- Enter SQL below this line. SQL above this line might be regenerated, causing any modifications to be lost.
-- ** MQMD
SET "OutputRoot"."MQMD".Format = 'MQHRF2';

-- ** RFH
SET "OutputRoot".MQRFH2.(MQRFH2.Field)Format = 'MQSTR';
SET "OutputRoot".MQRFH2.mcd.Msd = 'mrm';
SET "OutputRoot".MQRFH2.mcd.Set = 'RedTenant';
SET "OutputRoot".MQRFH2.mcd.Type = 'RM_Tenant';
SET "OutputRoot".MQRFH2.mcd.Fmt = 'CwXML';
-- response queue
SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///REDMAINT.SYNCHRONOUSRESPONSEQUEUE';

SET OutputRoot.MRM."version" = '3.0.0';
SET OutputRoot.MRM."delta" = FALSE;
SET OutputRoot.MRM."verb" = 'RetrieveByContent';
SET OutputRoot.MRM."locale" = 'en_US';

-- Set BO fields
-- For the retrieve by content we set the ID to zero and allow the name to be the search criteria

SET OutputRoot.MRM.RM_Tenant:Id = 0;
SET OutputRoot.MRM.RM_Tenant:ObjectEventId = InputRoot.MRM.ns:ObjectEventId;
-- ** More properties

SET OutputRoot.Properties.MessageDomain = 'MRM';
SET OutputRoot.Properties.MessageFormat = 'CwXML';

RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I + 1;
    END WHILE;
END;

END MODULE;
40. Save the ESQL.

Figure 9-20  Retrieve flow so far

The Message Flow should now look as shown in Figure 9-20, with any additional tracing nodes you may have added.

41. Save the Message Flow.

9.3 Deploy the Message Flows

To deploy the Message Flows, do the following:

1. Go back to the Broker Administration perspective.
2. Using the technique as described in 8.8, “Deploy the Message Flows” on page 215, add the changed flow to the Broker Archive file.
3. Save this updated BAR file.
9.4 Unit test the flow

We are now ready to test that the RetrieveByContent request message has been correctly created and that the original request data has been safely stored for later retrieval.

1. If it is running, stop the RMConnector.

2. Go to the Broker Administration perspective and stop all of the Message Flows in the RedMaintenance Execution Group.

3. Use one of our testing messages, Web_newMaintenance_DeliveryQueue in the RedTenantTestMessages folder of Additional Materials, to emulate a new create maintenance event from the front-end.

4. Read this file into rfhutil.

5. Check that it looks OK as a Web_newMaintenance BO, Figure 9-21, to be delivered to the broker. Make a note of the tenant name, maintenance description and the ObjectEventId.

6. Put the message to the WEBTENANT.DELIVERY queue.

7. From the Broker Administration perspective, start the RedTenant_to_RedMain_Request Message Flow.

8. Using rfhutil, browse the message on the REDMAINT.REQUESTQUEUE. See Figure 9-22 on page 245.
9. Check that the message has the tenant fields as we set them in the ESQL.
10. Check that the ObjectEventId is the same as the original. We need to preserve this for the complete create process.
11. Check that the verb on the message is RetrieveByContent.
12. Open a DB2 command window, Figure 9-23, and check that the row has been successfully inserted into the REDHOLD table with the correct description and key (ObjectEventId).

<table>
<thead>
<tr>
<th>Interactive</th>
<th>Script</th>
<th>Results</th>
<th>Access Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform required changes and then click the commit update button.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECTID</th>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSConnector_ID:414d512052454454e414e54202007bda3412000413</td>
<td>Test new maintenance for Redbook Reader</td>
</tr>
</tbody>
</table>

Figure 9-23 Inserted row

13. Start the RMConnector.
14. Check the connector log for successful retrieval of the correct tenant details.
15. Using rfhutil, browse the REDMAINT.SYNCHRONOUSRESPONSEQUEUE for the RM_Tenant business object.

See Figure 9-24 on page 246.
If this is all correct, we are now ready to move on from building the Message Flow to reconstruct the create request and send it on to the RMConnector.

16. Stop the RMConnector.

### 9.5 RedMaint_RetrieveTenantByContent

To create Redmaint_RetriveTenantByContent, do the following:

1. Navigate to the Broker Application Development perspective.


3. Drag the following nodes to the canvas and connect as shown in Figure 9-25 on page 247.
   - MQInput
   - Compute (PrepareCreateRequest)
   - MQOutput (SendForCreate)
4. Open the MQInput node properties and set the queue name to REDMAINT.SYNCHRONOUSRESPONSEQUEUE, as in Figure 9-26.

![MQInput Node Properties - MQInput](image)

*Figure 9-26  Input queue*

5. Open the Compute node.

See Figure 9-27 on page 248.
6. We will be performing database operations, so we need to set the Data Source property to REDHOLD.

7. Set the ESQL Module to RequestCreate.

8. **Apply** the changes.

9. Create the ESQL as shown in Example 9-2 on page 249.

   The ESQL will do the following:
   - a. Copy the message headers.
   - b. Set the required namespace declarations.
   - c. Set the properties for Message Set (RedTenant) and Message Type (RM_Maintenance).
   - d. Set the RFH2 header in the Message Descriptor.
   - e. Set the RFH2 header details:
     - Format = MQSTR
     - Message Domain = mrm
     - Message Set = RedTenant
     - Message Type is RM_Maintenance
• Wire Format = CwXML
• Reply queue is REDMAINT.RESPONSEQUEUE
f. Set the schema attributes (remember the verb is now Create).
g. Set the BO data from the message:
   • Output apartment id is the Input apartment id.
   • Output tenant id is the Input tenant id.
h. Set the BO data that was previously store (using the ObjectEventId as the key).
   • Output problem description is the REQUEST column data from the table.
i. Set the Output ObjectEventId from the Input ObjectEventId.
j. Delete the record from the table as it is no longer required.
k. Set the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 9-2  RequestCreate

CREATE COMPUTE MODULE RequestCreate
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   -- CALL CopyMessageHeaders();
   -- CALL CopyEntireMessage();

   -- Prepare the output message

   CALL CopyMessageHeaders();
   -- CALL CopyEntireMessage();

   -- ***  Schema Declarations ***

   -- Red Tenant Web Application

   -- Red Maintenance Back-end Application
   DECLARE RM_Tenant NAMESPACE
   DECLARE RM_Maintenance NAMESPACE

   -- ** Properties
   SET OutputRoot.Properties.MessageSet = 'RedTenant';
   SET OutputRoot.Properties.MessageType = 'RM_Maintenance';

   -- Enter SQL below this line. SQL above this line might be regenerated, causing any modifications to be lost.
```sql
-- ** MQMD
SET "OutputRoot"."MQMD".Format = 'MQRF2  ';

-- ** RFH
SET "OutputRoot".MQRFH2.(MQRFH2.Field)Format = 'MQSTR   ';
SET "OutputRoot".MQRFH2.mcd.Msd = 'mrm';
SET "OutputRoot".MQRFH2.mcd.Set = 'RedTenant';
SET "OutputRoot".MQRFH2.mcd.Type = 'RM_Maintenance';
SET "OutputRoot".MQRFH2.mcdFmt = 'CwXML';
-- response queue
SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///REDMAINT.RESPONSEQUEUE';

SET OutputRoot.MRM."version" = '3.0.0';
SET OutputRoot.MRM."delta" = FALSE;
set OutputRoot.MRM."verb" = 'Create';
SET OutputRoot.MRM."locale" = 'en_US';

-- Set BO fields
SET OutputRoot.MRM.RM_Maintenance:ApartmentId = InputRoot.MRM.RM_Tenant:ApartmentId;
SET OutputRoot.MRM.RM_Maintenance:TenantId = InputRoot.MRM.RM_Tenant:Id;

-- Pickup the description that we saved previously
SET OutputRoot.MRM.RM_Maintenance:ProblemDescription =
THE (SELECT ITEM T.REQUEST FROM Database.REDHOLD as T
WHERE ((T.OBJECTID = InputRoot.MRM.RM_Tenant:ObjectEventId)));

SET OutputRoot.MRM.RM_Maintenance:ObjectEventId = InputRoot.MRM.RM_Tenant:ObjectEventId;

-- Cleanup the table row as we no longer need it
DELETE FROM Database.REDHOLD AS T
WHERE T.OBJECTID = InputRoot.MRM.RM_Tenant:ObjectEventId ;

-- ** More properties
SET OutputRoot.Properties.MessageDomain = 'MRM';
SET OutputRoot.Properties.MessageFormat = 'CwXML';

RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
    SET OutputRoot.*[I] = InputRoot.*[I];
  END;
```

Adapters: Integration Broker deployment
Chapter 9. Build and test message flows for Create

10. Save the ESQL.

11. Open the properties for the MQOutput node, as in Figure 9-28.

   ![MQOutput Node Properties - SendForCreate](image)

   **Figure 9-28  MQOutput node**

   12. Set the queue to the REDMAINT.REQUESTQUEUE.

   13. Save the flow.

9.6 Deploy RedMaint_RetrieveTenantByContent

To deploy this flow, do the following:
1. Go back to the Broker Administration perspective.
2. Using the technique as described in 8.8, “Deploy the Message Flows” on page 215, add the changed flow to the Broker Archive file.
3. Save this updated BAR file.

9.7 Unit test RedMaint_RetrieveTenantByContent

If your Message Flow is functioning correctly, as soon as it was deployed successfully, the message that was on it from the previous test will have been picked up by the flow. The new RM_Maintenance message should now be sitting on the REDMAINT.REQUESTQUEUE waiting for the RMConnector (Figure 9-29).

![Request queue](image)

To test the message flow, do the following:
1. Check the message on the request queue to ensure that it has all of the required data, the correct ObjectEventId and verb (Create).
2. Start the RMConnector.
3. Check the connector log and the application log for a successful creation of a new maintenance record.
4. Check the database for the new row (the dates set by default in the application for a new maintenance for actual and expected completion with be today’s date).
5. Check the message on the response queue for the new maintenance id on the BO. See Figure 9-30 on page 253.
We are now ready to complete the round trip for a creation request.

9.8 RedMaint_to_RedTenant_Response

To create RedMaint_to_RedTenant_Response, do the following:

1. Go to the Broker Development perspective and open the RedMaint_to_RedTenant_Response flow.

   See Figure 9-31 on page 254.
2. Add the following nodes and connect as shown in Figure 9-31.
   - Label (ResponseRetrieve)
   - Compute (PrepareCreateResponse).

   See Figure 9-32 on page 255.
3. Open the properties for the Label node, set the LabelName to ResponseCreate, as in Figure 9-32.
4. Open the properties for the Compute node and set the ESQL module to ResponseCreate, as in Figure 9-33.

5. Apply the change.

6. Complete the ESQL as shown in Example 9-3 on page 257.

   The ESQL will:
   a. Copy the message headers.
   b. Set any required namespace declarations.
   c. Set the properties of Message Set (WebTenant) and Message Type (Web_newMaintenance).
   d. Set the Message Descriptor to contain an RFH2 header.
   e. Set the RFH header properties:
      • Format = MQSTR
      • Message Domain = mrm
      • Message Set = WebTenant
      • Message Type = Web_newMaintenance
      • Wire Format = CwXML
• Reply q is WEBTENANT.RESULT

f. Set the schema attributes.

g. Set the BO data:
   • XLM Declaration
   • Schema location for the application schema
   • Set the Output message maintenance id to the Input message maintenance id.

h. Set the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 9-3 ResponseCreate

CREATE COMPUTE MODULE ResponseCreate
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   CALL CopyMessageHeaders();
   -- CALL CopyEntireMessage();

   -- *** Schema Declarations ***

   -- Red Tenant Web Application

   -- Red Maintenance Back-end Application

   -- ** Properties
   SET OutputRoot.Properties.MessageSet = 'WebTenant';
   SET OutputRoot.Properties.MessageType = 'Web_newMaintenance';

   -- Enter SQL below this line. SQL above this line might be regenerated, causing any modifications to be lost.

   -- ** MQMD
   SET "OutputRoot"."MQMD".Format = 'MQHRF2';
   SET OutputRoot.MQMD.MsgType = 8;
   SET OutputRoot.MQMD.ReplyToQ = ' ';
   SET OutputRoot.MQMD.ReplyToQMgr = ' ';
SET "OutputRoot".MQRFH2.(MQRFH2.Field)Format = 'MQSTR
SET "OutputRoot".MQRFH2.mcd.Msd = 'mrm';
SET "OutputRoot".MQRFH2.mcd.Set = 'WebTenant';
SET "OutputRoot".MQRFH2.mcd.Type = 'Web_newMaintenance';
SET "OutputRoot".MQRFH2.mcd.Fmt = 'CwXML';

-- response queue
SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///WEBTENANT.RESULT';

-- ** Set Verb and locale information

SET OutputRoot.MRM."version" = '3.0.0';
SET OutputRoot.MRM."delta" = FALSE;
SET OutputRoot.MRM."verb" = InputRoot.MRM.verb;
SET OutputRoot.MRM."locale" = 'en_US';

-- ** Set BO fields

-- XMLDeclaration

SET OutputRoot.MRM.Web_newMaintenance:XMLDeclaration = 'xml version="1.0"
encoding="UTF-8"';

-- ** set the schema location so the JMS connector can fill in the details for the
onwards processing to the application

Maintenance.Web_newMaintenance_newMaintenance:schemaLocation =
'http://www.ibm.com/RedMaintenance maintenance.xsd';

-- ** Set the maintenance details

SET
Maintenance.Web_newMaintenance_newMaintenance:id =
InputRoot.MRM.RM_Maintenance:Id;

-- ** More properties

SET OutputRoot.Properties.MessageDomain = 'MRM';
SET OutputRoot.Properties.MessageFormat = 'CwXML';

RETURN TRUE;
CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I + 1;
    END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
END;
END MODULE;

7. Save the ESQL.

Figure 9-34  Message Flow so far

8. The Message Flow should now look similar to that shown in Figure 9-34.

9.9 Deploy RedMaint_to_RedTenant_Response

To deploy Redmaint_to_RedTenant_Response, do the following:
1. Go back to the Broker Administration perspective.
2. Using the technique as described in 8.8, “Deploy the Message Flows” on page 215, add the changed flow to the Broker Archive file.
3. Save this updated BAR file.

9.10 Unit test RedMaint_to_RedTenant_Response

Using the techniques learned so far, unit test the transformation of the response message to the front-end application response queue.

Check the meta-object configuration, if necessary, to see where the JMSConnector should be putting the application response for a create request. This is the JMS_MO_Config business object.

9.11 End-to-end test

We are now ready for the final test of our integration with the front-end application. Our front-end application is responsible for the retrieve and create operations, so we will run through the entire process.
1. Clear all of the queues for the JMSConnector, the RMConnector and the front-end application.
2. Ensure that the front-end application and the back-end application are running.
3. Ensure that all three message flows are running.
4. Start the JMSConnector.
5. Start the RMConnector.
6. At the front-end application home screen (Figure 9-35 on page 261), enter a Tenant ID.
Chapter 9. Build and test message flows for Create

7. Wait for the tenant details in Figure 9-36.

8. When the response from the request returns, create a new maintenance request as in Figure 9-37.
9. Wait for the response from the application to notify that the request has been received, as in Figure 9-38.

![New maintenance - Microsoft Internet Explorer](image)

*Figure 9-38 Request received*

10. You might want to repeat the retrieve to verify that your new maintenance record really is there.

Congratulations. You have concluded the front-end integration side of the solution.
Build and test message flow for Update

The update processing is handled by the WebSphere Server Foundation component of our solution because it handles maintenance update requests from our internal and external contractors.

However, we would like to be able to monitor this component with the WebSphere Business Integration Monitor in the same way that we will be monitoring our retrieve and create processing.

Because the current version of the Monitor does not accept feeds from Server Foundation, we will be using a special node for the Message Broker, which captures information for the Monitor at the start of both our Request and Response flows.

To enable us to capture this information for the update processing, we are going to set up a pass-through of sorts and catch the messages from Server Foundation. This will be explained later. For the moment we will modify our flows to enable this pass-through.
10.1 RedTenant_to_RedMaint_Request

We now need to add the last of the label processing for the request flow.

To add the last label processing, do the following:

1. Open the Broker Application Development perspective.
2. Add the following nodes to the canvas and connect as shown in Figure 10-1
   - Label (VerbUpdate)
   - Compute (RerouteUpdate)
3. Open the Label node and set the Label Name to VerbUpdate, as in Figure 10-2 on page 265.
Chapter 10. Build and test message flow for Update

4. Open the Compute node properties, shown in Figure 10-3.

Figure 10-2  Label node

Figure 10-3  Compute node
5. Set the ESQL Module Name to RequestUpdate.
6. Complete the ESQL as shown in Example 10-1.
   The ESQL will do the following:
   a. Copy the Input message to the Output message
   b. Set the reply queue in the JMS header to REDMAINT.RESPONSEQUEUE

Example 10-1  RequestUpdate

CREATE COMPUTE MODULE RequestUpdate
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    -- CALL CopyMessageHeaders();
    -- Take the entire message and prepare to forward it on.
    CALL CopyEntireMessage();
    -- Ensure that we have the correct response queue to force the loop-back for emitter processing
    SET "OutputRoot".MQRFH2.jms.Rto = 'queue:///REDMAINT.RESPONSEQUEUE';
    RETURN TRUE;
  END;

  CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
      SET OutputRoot.*[I] = InputRoot.*[I];
      SET I = I + 1;
    END WHILE;
  END;

  CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
  END;
END MODULE;

7. Save the Message Flow.
8. Add the modified flow to the BAR file.
10.2 RedMaint_to_RedTenant_Response

In the response flow, we do not need to do anything with the response message from the adapter because the original message was an Agent Delivery from our Server Foundation process. There will be more on this topic later. We merely allow the response to pass through to capture the emitter data we require.

For completeness, we will add a label node, which does not perform any processing.

To add the node, do the following:

1. Open the Label node properties and give the label a name of ResponseUpdate, as in Figure 10-4.

2. Save the Message Flow.

3. Add the modified flow to the BAR file.

4. Deploy the BAR file.

This concludes the building of our Message Broker components.
Gathering data from the runtime

Now that we have successfully deployed our integration solution, we will explore a method of gathering statistics and runtime data for our adapter environment.
11.1 Using WebSphere Business Integration Monitor

WebSphere Business Integration Monitor is a Java based and Web-based client/server application which allows the tracking and monitoring of business processes. This functionality enables statistical reports to be generated, based on real business data, which can then be used to fine-tune these business processes. It is therefore possible, by effective use of the WebSphere Business Integration Monitor, to significantly enhance a business process by providing a sound and realistic basis for business process management.

The core functionality of WebSphere Business Integration Monitor stems from its ability to track data generated from a multitude of sources and then to use this data within a model of the business process. These process models allow the performance of the business process to be analyzed as the process model executes in accordance with the data provided to it. It is therefore possible to establish what activities are being run and where any potential bottlenecks in the business process may arise.

WebSphere Business Integration Monitor consists of two parts:

- A server component
- A client component

We will not be exploring here, in any depth, the complex business analysis functions available using the Monitor. What we will be exploring are the steps involved in enabling data to be sent to the Monitor from our integration brokers, Message Broker and Server Foundation, to facilitate this kind of analysis.
WebSphere Business Integration Monitor (Figure 11-1) client consists of four main components:

- The *Workflow Dashboard* is an HTML-based client that allows the monitorization of active business process.
- The *Business Dashboard* is an HTML-based client that allows the analysis of historical data over a period of time.
- The *Notification* component is an HTML-based client that allows users to view notifications sent from process instances at runtime.
- The *Administration Utility* is an HTML-based client that allows the monitor administrator to import the modeled business processes with their associated business measures as well as it facilitates the tools to maintain the database and status of the services, for example the start and stop of the Event Processor service.

They can be accessed by both Monitor Portlets or by a traditional J2EE client. These components allow the user to track processes and to generate reports.
Workflow Dashboard
With the Workflow Dashboard, you can track processes, work items and business performance measures in real time, while getting the functionality for to make changes and explore what if scenarios. This functionality can be broken down into the following types:

- Track a process and its performance in real-time.
- Analyze both process instances and activity instances.
- Analyze and monitor executing processes and their performance metrics.
- Enhance workflow alerts and monitoring based on real business scenarios.
- Execute both administrative and corrective actions, including load balancing and redirecting.

Business Dashboard
The Business Dashboard allows you to generate analytical reports which comprise of historical data, using a set of statistics-based tools. This functionality can be broken down into the following points:

- Generate detailed and tailored statistics and reports based upon historical process performance.
- Compare and contrast actual and statistical information for the purpose of analysis and possible redesign.
- Produce performance-orientated data to provide a basis for sound and accurate business process management.

Notifications
Notifications is an HTML-based client which allows users to view notifications created by a process instance during its execution.

Administration Utility
The Administration Utility, is a Web-based client which allows users to perform a number of administrative functions. These functions can include manipulation of event triggers and database tables as well as manipulation and termination of process instances. This particular piece of functionality can be broken down into the following points:

- Stop and start Event Queue triggers.
- Drop Monitor and Event Queue database tables.
- Terminate process instances along with their history data.
Import WebSphere Business Integration Workbench XML files that contain data relevant to the processes and business measures defined in WebSphere Business Integration Workbench.

Define new business measures in the runtime environment.

WebSphere Business Integration Message Broker integrates with Monitor through the Emitter Node. This node is part of a support pack (IB01) for WebSphere Business Integration Message Broker. The emitter node integrates with the Monitor by sending data regarding the message flow in which it runs, to the Monitor database.

Typically, to integrate WebSphere Business Integration Message Broker with Monitor, it is necessary to work with the WebSphere Business Integration Modeler. The Modeler is used to construct the high-level view of the business process, which can include the data structures the business process uses. Once this business process has been constructed, it is possible to export various definition files for use with Monitor, Workflow and WebSphere Business Integration Message Broker.

In order for the Monitor to integrate with WebSphere Business Integration Message Broker, it needs a common frame of reference. This is supplied by the export files from Modeler.

In this instance, the XSD file relating to the message structures for business processes will need to be exported from Modeler and into WebSphere Business Integration Message Broker. In the case of the Monitor, an XML file produced by the Modeler containing data on the business process will need to be imported into Monitor.

Once this is complete, data sent from the emitter node can then be interpreted by the Monitor in a meaningful way.

**Important:** For our example we have installed WebSphere Business Integration Monitor v4.2.4 including FixPack3. We have also installed WebSphere Business Integration Modeler v5.1.1 for illustrative purposes, to show our business process which was built for integration the Message Broker. However, at this time it is not possible to export your processes from v5.1.1 of the Modeler for import into the Monitor. A v4.2.4 export is needed. We have included an export of our organization and business process definition which we created and exported from the correct version of the Modeler. It is in the ITSO folder of the Additional Materials.
11.2 Our business process

In our example, the process is defined by the round trip between the requestor and the back-end application. To see the very basic business process, we have created for our Red Tenant process, do the following:

1. Open the WebSphere Business Integration Modeler, as in Figure 11-2.

2. Select File → Import.

   See Figure 11-3 on page 275.
Figure 11-3 Select import type

3. Select **WebSphere Business Integration Modeler Import**. See Figure 11-4 on page 276.
4. Select **WebSphere Business Integration Modeler v4.2.4 (org)** because we will be importing an organization file exported from v4.2.4.

See Figure 11-5 on page 277.
Chapter 11. Gathering data from the runtime

5. Browse the ITSO folder in the Additional Materials. Within this folder is another folder named ITSO, the org file is contained in this sub-folder.

6. Select to **create a new project**, as in Figure 11-6 on page 278. Name it ITSO when requested.
7. The new project will be created for the import.

8. Once the project has been created successfully (Figure 11-7), the import of the org file will continue.
9. As you can see from the drop-down list in Figure 11-8, we have a Business Item named RedTenant. This is the data structure we will populate as part of our process.

10. We can also see a process named RedTenant.

11. Double-click RedTenant to open it.

12. We see in Figure 11-9 that our process definition consists of two activities:
The following figures show elements as they were defined in the original v4.2.4 of the Modeler to illustrate the components that we have created specifically for our process and feeding the Monitor from the Message Broker.

![Image of Process Tree](image)

**Figure 11-10  Process tree**

13. Figure 11-10 shows that our process consists of two tasks, RedTenant_to_RedMaint_Request and RedMaint_to_RedTenant_Response. This corresponds to the name of our two primary Message Flows.

![Image of Data Structure Tree](image)

**Figure 11-11  Data structure**

14. Figure 11-11 above shows the RedTenant data structure we will populate as part of the process. This is the data that we will feed to the Monitor along with our process and task information. The structure will correspond to a message definition we will use in the Message Broker.

15. Figure 11-12 on page 281 shows the business measures we have defined. These business measures are not very realistic in a business sense. They do not provide any real metrics value. However, the purpose of this exercise is not to show how to construct complex expressions or queries for the monitor. Our intention is to show how to have process data from the Message Broker flows to be collected for analysis by the Monitor. So for our purposes, these are sufficient to illustrate the logistics.
16. The file we have exported from the Modeler for use as the import to the Monitor is in the Additional Materials in the ITSO folder and is named RedTenant.xml.

11.3 Setting up the Monitor environment

We need to setup the Monitor database schema prior to importing our business process. This must always be performed by the administrator prior to any use of the Monitor.

To set up the Monitor environment, do the following:

1. Ensure that the Monitor Web application is running.
2. Enter the the Monitor Administration login screen. This is normally:

   http://<host>/monitor/admin/index.cmd
You will be able to see if the database schema has been setup or not. In our example, Figure 11-13, it has not because there is a button to create the database.

Login as the administrator. The method of authentication will vary depending on how you have installed the Monitor. In our case we will use the default, which is to use the workflow administrator user and password (admin / password).

Click the **Create Database** button.
6. Once the database create has been successfully completed, you arrive at the main administration screen, as in Figure 11-14. From here we will perform a little basic set up to get us going.

7. On the left, select **Setup Manager → Event Queue**.

   See Figure 11-15 on page 284.
8. For our exercise, we will start the **WMQI Connector** and the **Event Processor**. Select these and click **Start**.

9. Select the **System Properties**. See Figure 11-16 on page 285.
10. Review the system properties and modify any if required. For our exercise, the defaults are sufficient.

11. We will now import our business process and business measures.

12. Select **Import / Export → Import Organization**.
13. Browse to the RedTenant.xml file in the Additional Materials ITSO folder, as in Figure 11-17.

14. Select Import.

15. When the import has completed successfully, we can verify that our process was imported correctly, as in Figure 11-18.

16. Navigate back to the Cleanup Manager, as in Figure 11-19 on page 287.
17. Select **Process Version**.

18. Use the drop-down list of Process Lists to verify that our RedTenant process is in the database.

We now have the basics in place. From here we will make the required modifications to our Message Flows to enable data to be sent to the Monitor.

## 11.4 Setting up the Message Broker environment

The way in which WebSphere Business Integration Message Broker integrates with the Monitor is through the Emitter Node. This node can be located as part of a support pack (IB01) for the Message Broker. The emitter node integrates with the Monitor by sending data regarding the message flow in which it runs, to the Monitor database.

As we have mentioned earlier, typically in order to integrate WebSphere Business Integration Message Broker with the Monitor, it is necessary to work with the WebSphere Business Integration Modeler. The Modeler is used to construct the high-level view of the business process. This includes the data structures the business process uses. Once this business process has been constructed, it is then possible to export various definition files for use with Monitor, Workflow and WebSphere Business Integration Message Broker.
In order for the Monitor to integrate with WebSphere Business Integration Message Broker, it needs a common frame of reference. This is supplied by the export files from Modeler.

In this instance, the XSD file relating to the message structures for business processes will need to be exported from Modeler and into WebSphere Business Integration Message Broker.

Once this is complete, data sent from the Emitter Node can then be interpreted by the Monitor in a meaningful way. In our example, we have installed the Emitter Node into the Message Broker Toolkit according to the instructions.

**Note:** The message set we use and the subflows for the request and response flows are included in the Message Broker toolkit workspace on your VMWare image.

The EmitterMessages Message Set Project contains the message for the data we will be emitting. Note that the message set name is Customer. The name of the message set is not in anyway directly linked to the Monitor. It is, however, linked to what we build in the Broker. We will discuss this in more detail when we see the ESQL.

1. Import the Message Set into your Broker Workspace.

   **Note:** In your VMWare image, we have done this for you.

2. In your RedMaintenance Message Flow project, add a reference to the EmitterMessages Message Set.

3. Go to the Data perspective and configure the datasource for the WFMDB. This is the default name of the Monitor database.

   **Note:** In your VMWare image, we have done this for you.

See Figure 11-20 on page 289.
4. Copy this data source to your RedMaintenanceFlows project.

5. EmitterFlows, contains the two subflows we have created for handling the emitting of data on the request flow and the response flow. Copy or move these to the RedMaintenanceFlows project.

6. Double-click the MonitorEmitter Message Flow (Figure 11-21) to open it in the Message Flow Editor.
7. Open the Compute node properties, as in Figure 11-22.
8. The Compute Mode must be set to \texttt{LocalEnvironment and Message}.
9. The ESQL Module is \texttt{EmitterRequest}.
10. Open the ESQL in Example 11-1 on page 291.

The ESQL performs the following:

- Declares namespaces for our BO messages.
- Set the Emitter data for each of the different type of messages that will be passing through.
- Pass the Input message to the output message (we do not wish to modify the incoming message in any way, we merely pass it on).
- Set the Emitter values in the Environment.
  
  Set the MEmitter Task value. This will be the process identifier in the Monitor.
Important: This value must be unique for each message that passes through this flow. If the Monitor receives a duplicate task, it will merely ignore it.

However, because we want to record the full round trip, we need to ensure that this value will be correlated with an identical value on the return, response, trip. One way to ensure this is to use the ObjectEventId, which will be unique for every new business object sent by the JMSAdapter.

Set the MEmitter business model solution name (BM_SOLUTIONNAME) to the name of our process as it is known to the Monitor, RedTenant.

Set the MEmitter business model version (BM_VERSION) to match the version of our process model that is currently valid in the Monitor. Refer to Figure 11-19 on page 287.

- Set up the data to be emitted. This will depend entirely on what is available at the time, we fill what we can here, and we fill the rest of the data when it is available in the return message. In the response flow, see Example 11-2 on page 296).

Example 11-1  EmitterRequest

CREATE COMPUTE MODULE EmitterRequest

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- Red Tenant Web Application

    -- Set Emitter Data for a TENANT MESSAGE TYPE from Web Application
    IF InputRoot.Properties.MessageType = 'Web_tenant' THEN
        SET OutputRoot = InputRoot;
        Important:
        This value must be unique for each message that passes through this flow. If the Monitor receives a duplicate task, it will merely ignore it.
        However, because we want to record the full round trip, we need to ensure that this value will be correlated with an identical value on the return, response, trip. One way to ensure this is to use the ObjectEventId, which will be unique for every new business object sent by the JMSAdapter.

        Set the MEmitter business model solution name (BM_SOLUTIONNAME) to the name of our process as it is known to the Monitor, RedTenant.

        Set the MEmitter business model version (BM_VERSION) to match the version of our process model that is currently valid in the Monitor. Refer to Figure 11-19 on page 287.

        - Set up the data to be emitted. This will depend entirely on what is available at the time, we fill what we can here, and we fill the rest of the data when it is available in the return message. In the response flow, see Example 11-2 on page 296).

    END IF;

END;
```sql
SET Environment.MEmitter.Task = InputRoot.MRM.Web_tenant:ObjectEventId;
SET Environment.MEmitter.BM_SOLUTIONNAME = 'RedTenant';
SET Environment.MEmitter.BM_VERSION = '2003/8/31 8:00 AM';

DECLARE generic BLOB;
DECLARE options integer BITOR(RootBitStream, ValidateContent, ValidateValue, ValidateException, ValidateFullConstraints);

-- Using the XSD
CREATE LASTCHILD OF Environment.MEmitter DOMAIN 'MRM' NAME 'System';

--Here we fill monitor data structure values before the start of the flow
SET Environment.MEmitter.System.TenantName = ' ';

IF InputRoot.MRM.verb = 'Retrieve' AND InputRoot.Properties.MessageType = 'Web_newMaintenance' THEN
  SET Environment.MEmitter.System.Verb = 'Create';
ELSE
  SET Environment.MEmitter.System.Verb = InputRoot.MRM.verb;
END IF;

SET generic = ASBITSTREAM(Environment.MEmitter.System Options options SET 'Customer' TYPE 'RedTenant' FORMAT 'XML1');
SET Environment.MEmitter.BusinessData = cast(generic as char ccsid 1208);
END IF;

-- Set Emitter Data for a MAINTENANCE MESSAGE TYPE Type from Web Application
--**************************************************************************

IF InputRoot.Properties.MessageType = 'Web_newMaintenance' THEN

  SET OutputRoot = InputRoot;
  SET Environment.MEmitter.Task = InputRoot.MRM.Web_newMaintenance:ObjectEventId;
  SET Environment.MEmitter.BM_SOLUTIONNAME = 'RedTenant';
  SET Environment.MEmitter.BM_VERSION = '2003/8/31 8:00 AM';

  DECLARE generic BLOB;
  DECLARE options integer BITOR(RootBitStream, ValidateContent, ValidateValue, ValidateException, ValidateFullConstraints);

  -- Using the XSD
```
CREATE LASTCHILD OF Environment.MEmitter DOMAIN 'MRM' NAME 'System';

--Here we fill monitor data structure values before the start of the flow

SET Environment.MEmitter.System.TenantId = 0;
--
InputRoot.MRM.ns:ROOT.null1:Web_newMaintenance_newMaintenance.null1:tenantId;

IF InputRoot.MRM.verb = 'Retrieve' THEN
  SET Environment.MEmitter.System.Verb = 'Create';
ELSE
  set Environment.MEmitter.System.Verb = InputRoot.MRM.verb;
END IF;

SET generic = ASBITSTREAM(Environment.MEmitter.System Options options SET 'Customer' TYPE 'RedTenant' FORMAT 'XML1');
  SET Environment.MEmitter.BusinessData = cast(generic as char ccsid 1208);
END IF;

-- Set Emitter Data for a MAINTENANCE UPDATE Request from Contractor
--******************************************************************

IF InputRoot.Properties.MessageType = 'RM_Maintenance' THEN

SET OutputRoot = InputRoot;
Set Environment.MEmitter.Task = InputRoot.MRM.RM_Maintenance:ObjectEventId;
  Set Environment.MEmitter.BM_SOLUTIONNAME = 'RedTenant';
  Set Environment.MEmitter.BM_VERSION = '2003/8/31 8:00 AM';

declare generic BLOB;
declare options integer BITOR(RootBitStream, ValidateContent, ValidateValue, ValidateException, ValidateFullConstraints);

-- Using the XSD
CREATE LASTCHILD OF Environment.MEmitter DOMAIN 'MRM' NAME 'System';

--Here we fill monitor data structure values before the start of the flow

SET Environment.MEmitter.System.TenantId = InputRoot.MRM.RM_Maintenance:TenantId;
  SET Environment.MEmitter.System.Verb = InputRoot.MRM.verb;
11. Open the **Properties** of the MonitorEmitter node, as in Figure 11-23.

![MonitorEmitter properties](image)

*Figure 11-23  MonitorEmitter properties*

12. The Datasource is the **WFMDB** data source
13. The Schema is **WFM**, the default schema for the Monitor at installation.
14. Select the **Trace** properties.
   
   See Figure 11-24 on page 295.
15. We have chosen to create traces of the Emitter flows, we chose the c:\RedTenantTraces directory as the target for the traces.

16. Select the **EmitterResponse** flow, as in Figure 11-25.
17. We set the Basic properties of the Compute node as before, this time using the **EmitterResponse** ESQL Module.

18. We create the ESQL as before. We fill the values of the Customer message to be emitted with the return values from the back-end application. It is important here that we also remember to set the Emitter task to the ObjectEventId. This will tie the request flow and the response flow together for our business process in the Monitor.

See Example 11-2.

**Example 11-2  EmitterResponse**

```sql
CREATE COMPUTE MODULE EmitterResponse

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- Red Maintenance Back-end Application
```

-- Set Emitter Data for Response from the RM TENANT RETRIEVE
--*************************************************************************

IF InputRoot.Properties.MessageType = 'RM_Tenant' THEN
  SET OutputRoot = InputRoot;
  Set Environment.MEmitter.Task = InputRoot.MRM.RM_Tenant:ObjectEventId;
  Set Environment.MEmitter.BM_SOLUTIONNAME = 'RedTenant';
  Set Environment.MEmitter.BM_VERSION = '2003/8/31 8:00 AM';

  declare generic BLOB;
  declare options integer BITOR(RootBitStream, ValidateContent, ValidateValue, ValidateException, ValidateFullConstraints);

  -- Using the XSD
  CREATE LASTCHILD OF Environment.MEmitter DOMAIN 'MRM' NAME 'System';

  -- Here we fill monitor data structure values before the start of the flow
  SET Environment.MEmitter.System.TenantId = InputRoot.MRM.RM_Tenant:Id;
  SET Environment.MEmitter.System.TenantName = InputRoot.MRM.RM_Tenant:Name;

  IF InputRoot.MRM.verb = 'Retrieve' AND
    InputRoot.MQRFH2.mcd.Set = 'WebMaintenance' THEN
    SET Environment.MEmitter.System.Verb = 'Create';
  ELSE
    set Environment.MEmitter.System.Verb = InputRoot.MRM.verb;
  END IF;

  SET generic = ASBITSTREAM(Environment.MEmitter.System Options options SET 'Customer' TYPE 'RedTenant' FORMAT 'XML1');
  SET Environment.MEmitter.BusinessData = cast(generic as char ccsid 1208);
END IF;

-- Set Emitter Data for Response from the RM MAINTENANCE REQUEST
--*************************************************************************

IF InputRoot.Properties.MessageType = 'RM_Maintenance' THEN
  SET OutputRoot = InputRoot;
  Set Environment.MEmitter.Task = InputRoot.MRM.RM_Maintenance:ObjectEventId;
  Set Environment.MEmitter.BM_SOLUTIONNAME = 'RedTenant';
  Set Environment.MEmitter.BM_VERSION = '2003/8/31 8:00 AM';

  declare generic BLOB;
declare options integer BITOR(RootBitStream, ValidateContent, ValidateValue, ValidateException, ValidateFullConstraints);

-- Using the XSD
CREATE LASTCHILD OF Environment.MEmitter DOMAIN 'MRM' NAME 'System';

-- Here we fill monitor data structure values before the start of the flow
SET Environment.MEmitter.System.TenantId = InputRoot.MRM.RM_Maintenance:TenantId;
  -- SET Environment.MEmitter.System.Verb = 'Create';

SET generic = ASBITSTREAM(Environment.MEmitter.System Options options SET 'Customer' TYPE 'RedTenant' FORMAT 'XML1');
  SET Environment.MEmitter.BusinessData = cast(generic as char ccsid 1208);

END IF;

END;

END MODULE;

19. The properties for the MonitorEmitter node are as before.

We now need to modify our request and response flows to include the call to the Emitter subflows.
20. Open the RedTenant_to_RedMaint_Request flow.
21. Insert a call to the MonitorEmitter subflow between the MQInput and the SetRouting compute node.
22. Connect the nodes as shown in Figure 11-27.
23. Save the Message Flow.
25. Insert the MonitorEmitterResponse subflow between the MQInput and the SetRouting Compute node.
26. Connect the nodes as shown in Figure 11-28.
27. Save the Message Flow.

As we mentioned previously, we will attempt to collect process flow data coming in from our Server Foundation flow. As we see in the request and response flows, we will trap the inbound rerouted message and gather data for the emitter on the inbound. The Server Foundation process does not require a response to its Agent Delivery request. However, as we have sent the response back through
this flow, our emitter will complete the trip and merely throw the response away, as opposed to the Retrieve and Create, where we respond to the requestor.

11.5 Using the Monitor data

We now need to populate the Monitor database with some data. To do this, use the solution so far to send some retrieves, creates and updates. This will populate our event data in the monitor.

11.5.1 Workflow Dashboard

The Workflow Dashboard allows us to track processes, work items and business performance measures in real time.

To start the Workflow Dashboard, do the following:

1. Ensure that the Monitor is running.
2. Enter the Web address for the WebSphere Business Integration Monitor:
   
   http://localhost/monitor/index.cmd

   See Figure 11-29 on page 302.
3. Logon using administrator privileges and your password.

4. Select the **Workflow Dashboard**. We will look first at the events that we are sending to ensure that they are making it through to the Monitor.
5. Use the drop-down list to find our process, RedTenant.

6. Select Business Measures to determine which business measures we want to see.

See Figure 11-31 on page 304.
Workflow Dashboard - Business Measures

Specify the set of business measures and attributes that you want to display and the order in which they should be displayed.

![Figure 11-31 Select business measures](image)

7. From this window, we selected:
   - TenantId, TenantName and Verb from our emitter message
   - Starting Time
   - Elapsed Time
   - Status
   - Process Diagram

You can select business measures clicking the business measure and then clicking the arrow to select or deselect them. You can also reorder the selected business measures using the up and down arrows.

8. After you have selected the required business measures, click **OK**.

9. We now need to set the filter for the activities we want to see. Click **Set Filter**.

10. In Figure 11-32 on page 305 and Figure 11-33 on page 305, we have selected to see only processes that did not successfully complete. That is, we put a filter on the status and selected only running, terminating and so forth.
11. However, it is a good idea to begin with to see everything. With that in mind, select all statuses. This is done by selecting the values and then adding them to the Filter expression.

12. Click OK.
13. Click the **Save View As** icon and give the view a name. We selected **All** in Figure 11-34.

14. Click the **Go** button to see the number of instances available for viewing.

15. Set the number to view all available.

16. Click **View**. See Figure 11-35.
17. We now see the list of process instances available, based on the filter with the Business Measures we have selected.

18. If we click the **Process Diagram** icon, we see the graphical representation of our process model, as in Figure 11-36. Note this is a v4.2.4 representation.

19. Run some more processes to check the real-time updates from the Emitter.

   We will now go to the Business Dashboard and set up some historical reports.

### 11.5.2 Business Dashboard

To start the Business Dashboard, do the following:

1. Select the **Configuration View** in the Business Dashboard.

   See Figure 11-37 on page 308.
Figure 11-37  Create view

2. Using the drop-down list, select our RedTenant process.
3. Select an analysis type. We chose Basic Analysis.
4. Click the Business Measures button.
5. Select a business measure as the basis for the analysis. We chose Number of new items in Figure 11-38.

Figure 11-38  Select business measure

6. Click the Back button.
7. Select a breakdown attribute. We chose Verb because we want to see analysis of historical data based on the number of new items per day, by verb.
8. Move down the screen (Figure 11-39). Select an interval for reporting. We chose YTD so we can see everything that we have created so far.

9. Select a report type. We chose Table & Graph.

10. Go back up to the top and save this view. We saved it as Basic.

11. Go to the Views view.

12. Select our newly saved view.
13. Click the **View** button to run the query.

14. Once you have fed your Monitor over a period of time, you will see a report similar to that shown in Figure 11-40. This particular report is a snapshot of our testing the emitter over a few days.
15. As before, you can drill down for a more detailed analysis. See Figure 11-41. You can set filters or export the data as a CSV file to be used in reporting packages, spreadsheets and so forth.

Figure 11-41 Report details

While this is not a particularly elegant use of the Monitor, and we do not gain any real business metric benefit from it by our use of business measures, except for a trail of how many tenant and maintenance operations we are processing, we have achieved what we set out to do. We wanted to be able to add our Adapters solution into the Monitor for visibility and we have done that.
Appendix A. Unit test traces

These are samples of the traces that you should expect to see from your unit testing for the message flows.

Example: A-1  InputMessageFromFrontEnd.txt

```plaintext
*** Message tree starts here ***
(  
(0x01000000):Properties = (  
 (0x03000000):MessageSet      = 'WebTenant'  
 (0x03000000):MessageType     = 'Web_tenant'  
 (0x03000000):MessageFormat   = 'CwXML'  
 (0x03000000):Encoding        = 273  
 (0x03000000):CodedCharSetId  = 1208  
 (0x03000000):Transactional   = TRUE  
 (0x03000000):Persistence     = TRUE  
 (0x03000000):CreationTime    = GMTTIMESTAMP '2004-10-08 14:40:47.290'  
 (0x03000000):ExpirationTime  = -1  
 (0x03000000):Priority        = 4  
 (0x03000000):ReplyIdentifier = X'0000000000000000000000000000000000000000000000000'  
 (0x03000000):ReplyProtocol   = 'MQ'  
 (0x03000000):Topic           = NULL  
 )
(0x01000000):MQMD   = (  
 (0x03000000):SourceQueue      = 'WEBTENANT.DELIVERY'  
 (0x03000000):Transactional   = TRUE  
 (0x03000000):Encoding        = 273  
 (0x03000000):CodedCharSetId  = 819  
 (0x03000000):Format           = 'MQHRF2'  
 (0x03000000):Version          = 2  
)
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>0</td>
</tr>
<tr>
<td>Message Type</td>
<td>8</td>
</tr>
<tr>
<td>Expiry</td>
<td>-1</td>
</tr>
<tr>
<td>Feedback</td>
<td>0</td>
</tr>
<tr>
<td>Priority</td>
<td>4</td>
</tr>
<tr>
<td>Persistence</td>
<td>1</td>
</tr>
<tr>
<td>Message ID</td>
<td>X'414d512052454442524f4b45522020207f33654120001b01'</td>
</tr>
<tr>
<td>Correlation ID</td>
<td>X'0000000000000000000000000000000000000000000000000000000000000000'</td>
</tr>
<tr>
<td>Backout Count</td>
<td>0</td>
</tr>
<tr>
<td>Reply To Queue</td>
<td>'REDBROKER</td>
</tr>
<tr>
<td>User Identifier</td>
<td>'db2admin</td>
</tr>
<tr>
<td>Accounting Token</td>
<td>X'160105150000000a837d66532621f2a07e53b2be90300000000000000000b'</td>
</tr>
<tr>
<td>Application Data</td>
<td>'ereAdapters\jre\bin\java.exe'</td>
</tr>
<tr>
<td>Put Application Type</td>
<td>11</td>
</tr>
<tr>
<td>Put Application Name</td>
<td>'ereAdapters\jre\bin\java.exe'</td>
</tr>
<tr>
<td>Put Date</td>
<td>DATE '2004-10-08'</td>
</tr>
<tr>
<td>Put Time</td>
<td>GMTTIME '14:40:47.290'</td>
</tr>
<tr>
<td>Group ID</td>
<td>X'0000000000000000000000000000000000000000000000000000000000000000'</td>
</tr>
<tr>
<td>Message Sequence Number</td>
<td>1</td>
</tr>
<tr>
<td>Original Length</td>
<td>-1</td>
</tr>
</tbody>
</table>

```ruby
(0x01000000):MQRFH2    = (
  (0x03000000):Version  = 2
  (0x03000000):Format   = 'MQSTR'
  (0x03000000):Encoding = 273
  (0x03000000):CodedCharSetId = 1208
  (0x03000000):Flags    = 0
  (0x03000000):NameValueCCSID = 1208
  (0x01000000):mcd      = (
    (0x00000000):Msd     = (  
      (0x00000000): = 'mrm'  
    )
    (0x00000000):Set      = (  
      (0x00000000): = 'WebTenant'
    )
    (0x00000000):Type     = (  
      (0x00000000): = 'Web_tenant'
    )
    (0x00000000):Fmt      = (  
      (0x00000000): = 'CwXML'
    )
  )
  (0x01000000):jms      = (  
    (0x01000000):Dst      = (  
      (0x00000000): = 'queue:///WEBTENANT.DELIVERY'
    )
    (0x01000000):Tms      = (  
      (0x00000000): = '1097246447297'
    )
    (0x01000000):Dlv      = (  
  )
)
```

314  Adapters: Integration Broker deployment
Example: A-2 RedTenantDelivery_Retrieve.txt

*** START OF MESSAGE TREE ***
*** VERB IDENTIFIED AS RETRIEVE ***
(
(0x01000000):Properties = (}
Adapters: Integration Broker deployment

316
Appendix A. Unit test traces

(0x01000000):Msd = (
    (0x02000000): = 'mrm'
)

(0x01000000):Set = (
    (0x02000000): = 'WebTenant'
)

(0x01000000):Type = (
    (0x02000000): = 'Web_tenant'
)

(0x01000000):Fmt = (
    (0x02000000): = 'CwXML'
)

(0x01000000):jms = (
    (0x01000000):Dst = (
        (0x02000000): = 'queue:///WEBTENANT.DELIVERY'
    )
    (0x01000000):Tms = (
        (0x02000000): = '1097246447297'
    )
    (0x01000000):Dlv = (
        (0x02000000): = '2'
    )
)

(0x01000000):usr = (
    (0x01000000):WSDLBinding = (
        (0x02000000): = 'Web_tenantAgentDeliveryBinding'
    )
    (0x01000000):WSDLOperation = (
        (0x02000000): = 'Web_tenantRetrieve'
    )
)

(0x01000021):MRM = (
    (0x0300000B):version = '3.0.0'
    (0x0300000B):verb = 'Retrieve'
    (0x0300000B):locale = 'en_US'
    (0x0300000B):delta = FALSE
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:ROOT = (}


Example: A-3  RedMaintRequest.txt

**** START OF MESSAGE TREE ****

(0x01000000):Properties = {
    (0x03000000):MessageSet = 'RedTenant'
    (0x03000000):MessageType = 'RM_Tenant'
    (0x03000000):MessageFormat = 'CwXML'
    (0x03000000):Encoding = 273
    (0x03000000):CodedCharSetId = 1208
    (0x03000000):Transactional = TRUE
    (0x03000000):Persistence = TRUE
    (0x03000000):CreationTime = GMTTIMESTAMP '2004-10-08 14:40:47.290'
    (0x03000000):ExpirationTime = -1
    (0x03000000):Priority = 4
    (0x03000000):ReplyIdentifier = X'000000000000000000000000000000000000000000000000'
    (0x03000000):ReplyProtocol = 'MQ'
    (0x03000000):Topic = NULL
}

(0x01000000):MQMD = {
    (0x03000000):SourceQueue = 'WEBTENANT.DELIVERY'
    (0x03000000):Transactional = TRUE
    (0x03000000):Encoding = 273
    (0x03000000):CodedCharSetId = 819
    (0x03000000):Format = 'MQHRF2'
    (0x03000000):Version = 2
    (0x03000000):Report = 0
    (0x03000000):MsgType = 8
    (0x03000000):Expiry = -1
    (0x03000000):Feedback = 0
    (0x03000000):Priority = 4
    (0x03000000):Persistence = 1
    (0x03000000):MsgId = X'414d51205245442020209e33654120000504'
    (0x03000000):CorrelId = X'000000000000000000000000000000000000000000000000'
    (0x03000000):BackoutCount = 0
    (0x03000000):ReplyToQ = 'REDBROKER'
    (0x03000000):ReplyToQMgr = 'REDBROKER'
}
Example: A-4  ResponseFromBackEnd_WhatIsTheVerb.txt

*** What is the verb on the response message? ****
'Retrieve'
'ResponseRetrieve'

Example: A-5  ResponseMessageFromBackEnd.txt

*** START OF MESSAGE TREE ****
*** THIS IS A RETRIEVE RESPONSE ****

(0x01000000):Properties = (   
(0x03000000):MessageSet      = 'RedTenant'  
(0x03000000):MessageType     = 'RM_Tenant'  
(0x03000000):MessageFormat   = 'CwXML'  
(0x03000000):Encoding        = 273  
(0x03000000):CodedCharSetId  = 1208  
(0x03000000):Transactional   = TRUE  
(0x03000000):Persistence     = TRUE  
(0x03000000):CreationTime    = GMTTIMESTAMP '2004-10-08 16:52:13.710'  
(0x03000000):ExpirationTime  = -1  
(0x03000000):Priority        = 0  
(0x03000000):ReplyIdentifier = X'414d512052454442524f4b45522020207f33654120002502'  
(0x03000000):ReplyProtocol   = 'MQ'  
(0x03000000):Topic           = NULL 
)

(0x01000000):MQMD       = (   
(0x03000000):SourceQueue      = 'REDMAINT.DELIVERYQUEUE'  
(0x03000000):Transactional   = TRUE  
(0x03000000):Encoding        = 273  
(0x03000000):CodedCharSetId   = 819

320  Adapters: Integration Broker deployment
(0x03000000):Format
(0x03000000):Version
(0x03000000):Report
(0x03000000):MsgType
(0x03000000):Expiry
(0x03000000):Feedback
(0x03000000):Priority
(0x03000000):Persistence
(0x03000000):MsgId
(0x03000000):CorrelId
(0x03000000):BackoutCount
(0x03000000):ReplyToQ
(0x03000000):ReplyToQMgr
(0x03000000):UserIdentifier
(0x03000000):AccountingToken
(0x03000000):ApplIdentityData
(0x03000000):PutApplType
(0x03000000):PutApplName
(0x03000000):PutDate
(0x03000000):PutTime
(0x03000000):ApplOriginData
(0x03000000):GroupId
(0x03000000):MsgSeqNumber
(0x03000000):Offset
(0x03000000):MsgFlags
(0x03000000):OriginalLength

=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=

'MQHRF2 '
2
0
1
-1
0
0
1
X'414d512052454442524f4b45522020207f33654120001e05'
X'414d512052454442524f4b45522020207f33654120002502'
0
'REDMAINT.DELIVERYQUEUE
'
'REDBROKER
'
'db2admin
'
X'16010515000000a837d66532621f2a07e53b2be903000000000000000000000b'
'
'
11
'ereAdapters\jre\bin\java.exe'
DATE '2004-10-08'
GMTTIME '16:52:13.710'
'
'
X'000000000000000000000000000000000000000000000000'
1
0
0
-1

)
(0x01000000):MQRFH2
= (
(0x03000000):Version
= 2
(0x03000000):Format
= 'MQSTR
'
(0x03000000):Encoding
= 273
(0x03000000):CodedCharSetId = 1208
(0x03000000):Flags
= 0
(0x03000000):NameValueCCSID = 1208
(0x01000000):mcd
= (
(0x01000000):Msd = (
(0x02000000): = 'mrm'
)
(0x01000000):Set = (
(0x02000000): = 'RedTenant'
)
(0x01000000):Type = (
(0x02000000): = 'RM_Tenant'
)
(0x01000000):Fmt = (
(0x02000000): = 'CwXML'
)
)
(0x01000000):jms
= (
(0x01000000):Dst = (
(0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'
)
(0x01000000):Rto = (
(0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'

Appendix A. Unit test traces

321


Adapters: Integration Broker deployment

(MRM:version = '3.0.0'
(MRM:verb = 'Retrieve'
(MRM:locale = 'en_US'
(MRM:delta = FALSE
(MRM:RM_Apartment:version = '3.0.0'
(MRM:RM_Apartment:verb = 'Retrieve'
(MRM:RM_Apartment:locale = 'en_US'
(MRM:RM_Apartment:delta = FALSE
(MRM:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:Id = 1
'IBM Hursley Park'

(0x0300000B):size = 7  
(0x0300000B):version = '3.0.0'  
(0x0300000B):verb = ''  
(0x0300000B):locale = 'en_US'  
(0x0300000B):delta = FALSE  
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Id = 1  
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId = 1  
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:TenantId = 100  

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ProblemDescription = 'I have a leaking tap'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:StatusDescription = 'New part installed'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ExpectedCompletion = '0'
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ActualCompletion = '0'

(0x0300000B):version = '3.0.0'  
(0x0300000B):verb = ''  
(0x0300000B):locale = 'en_US'  
(0x0300000B):delta = FALSE  
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Id = 2}
Adapters: Integration Broker deployment
Adapters: Integration Broker deployment

(delta) $\delta = \text{FALSE}$


[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId $\equiv 1$

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status $\equiv \text{'A'}$

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:TenantId $\equiv 100$


[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ExpectedCompletion = '0'

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ActualCompletion $\equiv '0'$


[0x0300000B]:version $\equiv \text{'3.0.0'}$

[0x0300000B]:verb $\equiv ''$

[0x0300000B]:locale $\equiv \text{'en_US'}$

[0x0300000B]:delta $\delta = \text{FALSE}$


[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId $\equiv 1$

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status $\equiv \text{'A'}$

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:TenantId $\equiv 100$


[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ExpectedCompletion = '0'

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ActualCompletion $\equiv '0'$


[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ExpectedCompletion = '0'

[0x0300000B]:http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ActualCompletion $\equiv '0'$

Example: A-6  ResponseToBeSentToFrontEndAdapter.txt

*** START OF MESSAGE TREE ****

(0x01000000):Properties = (  
(0x03000000):MessageSet      = 'WebTenant'  
(0x03000000):MessageType     = 'Web_tenant'  
(0x03000000):MessageFormat   = 'CwXML'  
(0x03000000):Encoding        = 273  
(0x03000000):CodedCharSetId  = 1208  
(0x03000000):Transactional   = TRUE  
(0x03000000):Persistence     = TRUE  
(0x03000000):CreationTime    = GMTTIMESTAMP '2004-10-08 16:52:13.710'  
(0x03000000):ExpirationTime  = -1  
(0x03000000):Priority        = 0  
(0x03000000):ReplyIdentifier = X'414d512052454442524f4554220207f33654120002502'  
(0x03000000):ReplyProtocol   = 'MQ'  
(0x03000000):Topic           = NULL
)

(0x01000000):MQMD      = (  
(0x03000000):SourceQueue      = 'REDMAINT.DELIVERYQUEUE'  
(0x03000000):Transactional    = TRUE  
(0x03000000):Encoding         = 273  
(0x03000000):CodedCharSetId   = 819  
(0x03000000):Format           = 'MQRFH2'  
(0x03000000):Version          = 2  
(0x03000000):Report           = 0  
(0x03000000):MsgType          = 8  
(0x03000000):Expiry           = -1  
(0x03000000):Feedback         = 0  
(0x03000000):Priority         = 0  
(0x03000000):Persistence      = 1  
(0x03000000):MsgId            = X'414d512052454442524f4b4552220207f33654120002502'  
(0x03000000):CorrelId         = X'414d512052454442524f4b4552220207f33654120002502'  
(0x03000000):BackoutCount     = 0  
(0x03000000):ReplyToQ         = ' '  
(0x03000000):ReplyToQMgr      = ' '  
(0x03000000):UserIdentifier   = 'db2admin'  
(0x03000000):AccountingToken  = X'1601051500000009837d66532621f2a07e53b2be9030000000000000000000000b'  
(0x03000000):ApplIdentityData   = ' '  
(0x03000000):PutApplType      = 11  
(0x03000000):PutApplName      = 'ereAdapters\jre\bin\java.exe'  
(0x03000000):PutDate          = DATE '2004-10-08'  
(0x03000000):PutTime          = GMTTIME '16:52:13.710'  
(0x03000000):ApplOriginData   = ' '  
(0x03000000):GroupId          = X'00000000000000000000000000000000000000000000000000000000'  
(0x03000000):MsgSeqNumber     = 1  
(0x03000000):Offset           = 0  
(0x03000000):MsgFlags         = 0  
(0x03000000):OriginalLength   = -1
)

(0x01000000):MQRFH2    = (  
(0x03000000):Version          = 2  
(0x03000000):Format           = 'MQSTR'
)
Adapters: Integration Broker deployment

(0x03000000):Encoding       = 273
(0x03000000):CodedCharSetId = 1208
(0x03000000):Flags          = 0
(0x03000000):NameValueCCSID = 1208
(0x01000000):mcd            = {
    (0x01000000):Msd      = {
        (0x02000000): = 'mrm'
    }
    (0x01000000):Set      = {
        (0x02000000): = 'WebTenant'
    }
    (0x01000000):Type     = {
        (0x02000000): = 'Web_tenant'
    }
    (0x01000000):Fmt      = {
        (0x02000000): = 'CwXML'
    }
    (0x01000000):jms      = {
        (0x01000000):Dst     = {
            (0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'
        }
        (0x01000000):Rto     = {
            (0x02000000): = 'queue:///WEBTENANT.RESULT'
        }
        (0x01000000):Tms     = {
            (0x02000000): = '1097254333719'
        }
        (0x01000000):Pri     = {
            (0x02000000): = '0'
        }
        (0x01000000):Dlv     = {
            (0x02000000): = '2'
        }
    }
    (0x01000000):usr      = {
        (0x01000000):Description   = {
            (0x01000000):WSDLBinding   = {
                (0x01000000): = 'Web_tenantAgentDeliveryBinding'
            }
            (0x01000000):Status        = {
                (0x01000000):WSDLOperation = {
                    (0x01000000): = 'Web_tenantRetrieve'
                }
                (0x01000000):RequestType   = {
                    (0x01000000): = 'Response'
                }
            }
        }
    }
    (0x01000021):MRM            = {
        (0x03000000):version = '3.0.0'
        (0x03000000):delta   = FALSE
    }
}
(0x03000008):verb = 'Retrieve'
(0x03000000):locale = 'en_US'
(0x03000000):http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:XMLDeclaration = 'xml version="1.0" encoding="UTF-8"'
(0x0300000B):http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenants:id = 100
(0x0300000B):http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenants:email = 'leegavin@uk.ibm.com'
(0x01000000):http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment:id = '1'
(0x03000000):http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment:address = '135 IBM Hursley Park Winchester Hampshire United Kingdom'
Adapters: Integration Broker deployment

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'COMPLETED'

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:description = 'I have a leaking tap New part installed Expected completion date: 0'

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = ( 

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '2'
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'ACTIVE'

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = ( 

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '3'
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'ACTIVE'
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:description = 'High speed internet not working Cable company contacted, modem to be replaced. Expected completion date: 0'

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = ( 

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '10'
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'IN PROGRESS'
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:description = 'Test for a new maintenance request Part on order Expected completion date: 0'

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = ( 

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '11'
Appendix A. Unit test traces

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'IN PROGRESS'

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:description = 'Test for a new maintenance request Lee's Test - Number Only Expected completion date: 0'

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = (  
  (0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '12'  
  (0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'ACTIVE'   
  )

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance = (  
  (0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id = '13'  
  (0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:status = 'ACTIVE'   
  )

)
Where are my messages?

In this Appendix we provide some cheat sheets to assist in your unit testing. Use these diagrams to assist in working out where your messages should be at the different points in the flows between the applications and the connectors.
Some cheat sheets

These four figures show the queues that you should expect to see the data on and in what format, for the different stages of the processing between the front and back-end applications.

Figure B-1  RedTenant to Message Broker interaction (request)

Figure B-2  Message Broker to RedMaintenance interaction (request)
Appendix B. Where are my messages?

**Figure B-3**  RedMaintenance to Message Broker interaction (response)

**Figure B-4**  Message Broker to RedTenant interaction (response)
Additional material

This Redpaper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246345

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number, SG246345.

Using the Web material

The additional Web material that accompanies this Redpaper includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG246345.zip</td>
<td>Sample applications and code</td>
</tr>
</tbody>
</table>
System requirements for downloading the Web material

The following system configuration is recommended:

- **Hard disk space:** 2GB
- **Operating System:** Windows
- **Processor:** 1600MHz or higher
- **Memory:** 2GB

How to use the Web material

Create a subdirectory (AdditionalMaterials) on your workstation, and unzip the contents of the Web material zip file into this folder.

All instructions for using the AdditionalMaterials will give a reference to the location within this folder.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 339. Note that some of the documents referenced here may be available in softcopy only.

- WebSphere Business Integration Adapter Development: An Introduction to the Basics, REDP-9119
- WebSphere Business Integration Adapter Development: A Development Example, REDP-9120

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
WebSphere Business Integration Adapter Development: An Integration Broker Deployment Example

Developing, deploying and testing a custom adapter

Developing, deploying and testing a technology adapter

Building message sets and flows for adapters

This IBM Redpaper is the final in a series of Redpapers called WebSphere Business Integration Adapter Development. This series discusses the development and deployment of WebSphere® Business Integration Adapters.

In this final Redpaper, An Integration Broker Deployment Example, we take the reader through the final phase of the development life cycle of a custom adapter: final testing, then deploying to and testing with an Integration Broker. In our scenario we use the WebSphere Business Integration Message Broker as our Integration Broker. We include sample code for you to download, as well as discuss our development, initial testing and deployment steps.

Look for the other IBM Redpapers in this series:

- WebSphere Business Integration Adapter Development: An Introduction to the Basics, REDP-9119
- WebSphere Business Integration Adapter Development: A Development Example, REDP-9120

For more information: ibm.com/redbooks