

WebSphere Business Integration Adapters: An Adapter Development and WebSphere Business Integration Solution

Incorporate business-to-business exchanges
using WebSphere BI Connect

Use adapters with Message Broker
and Server Foundation

Develop and deploy a custom adapter



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



International Technical Support Organization

**WebSphere Business Integration Adapters: An
Adapter Development and WebSphere Business
Integration Solution**

July 2005

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

First Edition (July 2005)

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

© 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	xv
Trademarks	xvi
Preface	xvii
The team that wrote this redbook	xviii
Become a published author	xx
Comments welcome	xx
Part 1. Adapter development: The theory	1
Chapter 1. Adapter development	3
1.1 Technical assessment	4
1.2 Getting started with the development	5
1.2.1 Understanding the application	5
1.2.2 Identifying the directionality of the connector	5
1.2.3 Identifying the application-specific business objects	5
1.2.4 Investigating the application data interaction interface	6
1.2.5 Investigating the event management and notification mechanism ...	7
1.2.6 Investigating communication across operating systems	7
1.3 Getting started with the development project	7
Chapter 2. Business objects	11
2.1 Overview of business objects	12
2.2 Types of business objects	14
2.2.1 Application-specific business object	14
2.2.2 Generic business object	15
2.3 Business object design	15
2.3.1 Flat business object	15
2.3.2 Hierarchical business object	16
2.3.3 Business object structure	16
2.3.4 Wrapper business objects	18
2.4 Application-specific information and metadata	19
2.5 Metadata and metaobjects	21
Chapter 3. Object Discovery Agents	23
3.1 Overview of ODA	24
3.2 Design considerations	26
3.3 Setting up the development environment	26
3.4 Object Discovery Agent process flows	26

3.4.1	Obtaining ODA configuration properties	27
3.4.2	Selecting and confirming source data	29
3.4.3	Generating content	29
3.4.4	Saving content	31
3.5	Object Discovery Agent data sources	32
Chapter 4.	Agent initialization and termination	33
4.1	Connector startup	34
4.1.1	WebSphere InterChange Server	34
4.1.2	Business Integration Message Broker or Application Server	34
4.2	Connector initialization	35
4.2.1	Establishing a connection	35
4.2.2	Checking the connector version	36
4.2.3	Recovering In-Progress events	36
4.3	Obtaining the business object handler	37
4.4	Connector termination	39
4.5	Extending the connector base class	39
Chapter 5.	Data handlers and name handlers	41
5.1	Design decisions for data handlers	42
5.1.1	Determining whether the data is metadata-driven	42
5.2	The development process	43
5.3	Creating the custom data handler	44
5.4	Converting data	44
5.5	Name handler development overview	45
5.6	Implementing a name handler	46
5.7	Name handler requirements	47
5.8	Reuse	47
5.9	Configuration requirements	47
Chapter 6.	Business object handlers	49
6.1	Overview of request processing	50
6.2	Metadata	52
6.2.1	Metadata-driven connectors	52
6.2.2	Partially metadata-driven connectors	53
6.2.3	Connectors that do not use metadata	55
6.3	One generic business object handler for all business objects	56
6.4	Multiple business object handlers	57
6.5	Designing the doVerbFor() method	57
6.5.1	Performing the verb action	59
6.5.2	Handling the Create verb	60
6.5.3	Handling the Retrieve verb	62
6.5.4	Handling the RetrieveByContent verb	65
6.6	Handling the Update verb	66

6.6.1 Handling the Delete verb.	69
6.7 Business object processing and cardinality.	70
6.7.1 Processing flat business objects.	70
6.7.2 Processing hierarchical business objects	72
6.8 Custom business object handlers	75
6.8.1 Creating the class for the custom business object handler.	76
6.8.2 Implementing the doVerbForCustom() method.	76
Chapter 7. Connector configuration properties	79
7.1 Overview of connector configuration property.	80
7.1.1 Standard connector configuration properties	80
7.1.2 Connector-specific configuration properties	80
7.2 Using properties in your adapter	81
7.3 Retrieving properties	81
7.3.1 Single-valued simple properties	81
7.3.2 Hierarchical properties	82
7.4 Some last thoughts about standard properties	85
7.4.1 RepositoryDirectory.	85
7.4.2 DeliveryTransport	85
Chapter 8. Asynchronous event processing and notification.	89
8.1 Overview of event notification	90
8.1.1 Subscription to a business object	92
8.2 Event and archive stores.	92
8.2.1 Content of an event store	93
8.2.2 Implementing an event store.	96
8.2.3 Accessing the event store	101
8.2.4 Creating an archive store	107
8.3 Polling	109
8.3.1 Standard behavior.	110
8.3.2 Basic logic for the poll method	112
8.3.3 Archiving events	113
8.3.4 Multithreaded	114
8.3.5 Processing events by event priority	114
8.3.6 Event distribution.	115
8.4 Custom pollForEvents() method	115
8.4.1 Accessing a subscription manager	117
8.4.2 Verifying the connection	117
8.4.3 Retrieving event records	118
8.4.4 Getting the business object name, verb, and key	120
8.4.5 Checking for subscriptions to the event	121
8.4.6 Retrieving application data	123
8.4.7 Sending the business object to Adapter Framework.	125

8.4.8	Completing the processing of an event	130
8.4.9	Archiving the event	131
8.4.10	Releasing event store resources	133
8.5	Processing delete events	134
8.5.1	Setting the verb in the event record	135
8.5.2	Setting the verb in the business object	135
8.5.3	Setting the verb during mapping	136
8.6	Guaranteed event delivery	136
8.6.1	Container-managed events	137
8.6.2	Duplicate event elimination	139
Chapter 9.	Synchronous event processing (callback)	143
9.1	Overview of the executeCollaboration method	144
9.1.1	Parameters	144
9.2	Using the executeCollaboration method	145
9.3	Single or multithreaded support for sequential event processing	146
9.4	Adapter termination and sequential event processing	147
9.4.1	Normal termination	147
9.4.2	Abnormal termination or no neat adapter shutdown situations	147
Chapter 10.	Local versus remote deployment	149
10.1	Overview of adapter deployment	150
10.2	Local deployment	151
10.2.1	Configuring startup in the Windows environment	152
10.2.2	Configuring startup in a UNIX environment	153
10.3	Remote deployment	153
10.4	Setting up the communication between a remote agent and a broker	155
10.4.1	Native WebSphere MQ	156
10.4.2	HTTP/HTTPS	158
10.4.3	Configuring the adapter startup in remote environments	161
Chapter 11.	Component deployment	163
11.1	Specifics about component deployment	164
11.1.1	WebSphere InterChange Server	164
11.1.2	WebSphere Business Integration Message Broker	165
11.1.3	WebSphere Application Server	167
Part 2.	Developing our custom adapter	169
Chapter 12.	Setting up the development environment	171
12.1	Installing the necessary components	172
12.1.1	Installing WebSphere Studio Application Developer Integration Edition V5.0.1	172
12.1.2	Installing IBM JDK 1.3.1	172

12.1.3	Installing WebSphere MQ v5.3.0.2	172
12.1.4	Installing WebSphere Business Integration Adapter Framework V2.4	173
12.1.5	Installing WebSphere Business Integration Adapter Development Kit V2.4	184
12.2	Developing the adapter	189
12.2.1	Adapter Framework	190
12.2.2	Application-specific component	193
12.2.3	Developing the application-specific component	193
Chapter 13.	Adapter design and environment	195
13.1	Understanding our scenario environment	196
13.1.1	The requirements	197
13.2	Examining the application environment	198
13.3	Examining the API	199
13.4	Designing application-specific business objects	201
13.5	Conclusion	201
Chapter 14.	Object Discovery Agent	203
14.1	Developing the ODA	204
14.1.1	Setting up the development environment	204
14.1.2	Naming conventions for the ODA	206
14.2	Implementing the ODA	206
14.2.1	Extending the ODA base class	206
14.2.2	Obtaining the handle to the ODKUtility object	207
14.2.3	Initializing the configuration-property array	207
14.2.4	Initializing ODA metadata	208
14.2.5	Initializing the ODA startup	209
14.2.6	Choosing the ODA content protocol	210
14.2.7	Generating source nodes	211
14.2.8	Generating business object definitions	212
14.2.9	Providing access to generated business object definitions	215
14.3	Testing the ODA	216
14.3.1	Setting up the test environment	217
14.4	Deploying the ODA	221
14.4.1	Exporting the ODA	221
14.4.2	Creating the startup scripts	224
14.5	Generating business objects using the ODA	225
14.5.1	Completing the business objects	232
Chapter 15.	Initializing and terminating the adapter agent	239
15.1	Extending the connector base class	240
15.2	Initializing the adapter agent	241
15.2.1	Retrieving connector configuration properties	241

15.2.2	Establishing a connection to the application	245
15.2.3	Checking the connector version	252
15.2.4	Terminating the adapter agent	252
Chapter 16.	Implementing a business object handler	255
16.1	Extending the business-object-handler base class	256
16.2	Implementing the doVerbFor() method	256
16.2.1	Obtaining the active verb	257
16.2.2	Verifying the connection before processing the verb	258
16.2.3	Branching on the active verb.	260
16.3	Performing the verb operation.	262
16.3.1	Accessing the business object	262
16.3.2	Implementing our verb operation	267
Chapter 17.	Implementing event notification	277
17.1	Extending the event store class	278
17.1.1	Implementing methods to in the CWConnectorEventStore.	279
17.2	Implementing the fetchEvents() method	281
17.3	Implementing the deleteEvents() method	283
17.4	Implementing the setEventStatus() method	285
17.5	Implementing the archiveEvents() method	286
17.6	Implementing the recoverInProgress() method.	287
Chapter 18.	Polling for events	291
18.1	Using the pollForEvents() method.	292
Part 3.	Setting up applications for our scenario	305
Chapter 19.	Overview of our scenario and applications	307
19.1	Understanding our scenario environment	308
19.2	The requirements	309
Chapter 20.	Installing and configuring the scenario infrastructure	311
20.1	Messaging infrastructure	312
20.2	The RedTenant Web application.	312
20.3	Installing the RedTenant application	315
20.4	Preparing the environment	316
20.4.1	Creating the RedTenant application server.	316
20.4.2	Configuring resources for the WebSphere MQ JMS provider.	317
20.4.3	Deploying the application	323
20.4.4	A quick test	326
20.5	Create the back-end application database	328
20.6	Create the back-end application	330
Part 4.	Configuring and testing adapters and business objects	333

Chapter 21. Creating the business objects and connector	335
21.1 Front-end business objects	336
21.1.1 Creating business objects with the ODA	336
21.1.2 Modifying business objects for use	343
21.2 JMSConnector	355
21.2.1 Creating a JMS Connector	355
21.3 Connector metaobjects for the data handler	359
21.3.1 Creating metaobjects	360
21.4 Queue connection factory and queue objects	362
21.5 Supported business objects for the connector	365
21.6 Starting the connector	366
21.7 Unit Testing	367
 Chapter 22. Object Discovery Agent	 379
22.1 Setting up the custom ODA	380
22.2 Generating business objects using the ODA	381
22.2.1 Completing the business objects	389
 Chapter 23. Packaging the custom adapter for distribution	 397
23.1 Connector naming conventions	398
23.2 Defining the connector	398
23.2.1 Defining the connector with the Connector Configurator	399
23.2.2 Defining the connector configuration properties	401
23.3 Creating the WebSphere MQ objects	405
23.4 Preparing the connector's directory	405
23.5 Creating the startup script	406
23.5.1 Startup script for our connector	407
23.6 Creating the Windows shortcut	410
 Chapter 24. Unit testing the connector	 411
24.1 Start the connector	412
24.1.1 Troubleshooting start errors	412
24.2 Testing the interaction between the connector and business objects	414
24.2.1 Adding supported business objects	414
24.2.2 Starting the Test Connector	416
24.2.3 Preparing test data for request processing (RM_Tenant.Retrieve)	419
24.2.4 Sending test data to the Test Connector (RM_Tenant.Retrieve)	428
24.2.5 Sending a response to the request (RM_Tenant.Retrieve)	430
24.2.6 Preparing test data for processing (RM_Maintenance.Create)	436
24.2.7 Sending test data (RM_Maintenance.Create)	437
24.3 Testing the interaction between the connector and the application	439
24.3.1 Sending a request business object (RM_Tenant.Retrieve)	440
24.3.2 Checking the response message received	443
24.3.3 Sending a request business object (RM_Maintenance.Create)	443

24.3.4	Creating an event	444
24.3.5	Checking the event message that is sent	445
24.4	Conclusions on unit testing	448
Part 5.	Message Broker components	449
Chapter 25.	Deploying business objects to the Message Broker	451
25.1	Importing front-end business objects	452
25.1.1	Defining the Message Set.	452
25.1.2	Importing schema definitions and creating message definitions. .	454
25.2	Importing back-end business objects	463
Chapter 26.	Building and testing message flow for Retrieve	469
26.1	Retrieve processing scenario	470
26.2	Creating a Message Flow project and Message Flows.....	471
26.3	RedTenant_to_RedMaint_Request.....	472
26.4	Deploying Message Sets and Message Flows	484
26.5	Unit testing the flow	492
26.6	RedMaint_to_RedTenant_Response	496
26.7	ResponseRetrieve.....	498
26.8	Deploying the Message Flows	511
26.9	Unit testing the flow	511
26.10	End-to-end testing.....	514
Chapter 27.	Building and testing message flows for Create.....	519
27.1	Creating a processing scenario.....	520
27.2	RedTenant_to_RedMaint_Request.....	521
27.3	Deploying the Message Flows	538
27.4	Unit testing the flow	539
27.5	RedMaint_RetrieveTenantByContent	541
27.6	Deploying RedMaint_RetrieveTenantByContent.....	547
27.7	Unit testing RedMaint_RetrieveTenantByContent.....	547
27.8	RedMaint_to_RedTenant_Response	549
27.9	Deploying RedMaint_to_RedTenant_Response.....	554
27.10	Unit testing RedMaint_to_RedTenant_Response.....	554
27.11	End-to-end testing.....	555
Chapter 28.	Building and testing message flow for Update	557
28.1	RedTenant_to_RedMaint_Request.....	558
28.2	RedMaint_to_RedTenant_Response	561
Part 6.	Business-to-business components	563
Chapter 29.	Integrating external contractors using WebSphere BI Connect	565

29.1	Initial configuration of the Express server	567
29.2	Initial configuration of the Advanced server	573
29.2.1	Updating the hubadmin profile	573
29.2.2	Configuring the community manager's profile	593
29.2.3	Configuring the community participant's profile	601
29.2.4	Creating the participant connection.	605
29.3	Configuring the Express server of RedContract	610
29.4	Trading documents	615
29.5	JMS setup for WebSphere Business Integration Connect	621
29.5.1	Defining MQ resources	621
29.5.2	Enabling JMS	622
29.5.3	Creating the JMS gateway	624
29.5.4	Creating the JMS target	626
29.5.5	Validation of the JMS and MQ configuration.	628
Part 7.	Server Foundation components	631
Chapter 30.	Adding human interaction using WebSphere BI Server	
	Foundation	633
30.1	Developing the services	634
30.1.1	Step 1: Developing the InhouseSend service	634
30.1.2	Exploring the WSDL editor	639
30.1.3	Step 2: Developing the InhouseSend service	642
30.1.4	Generating deploy code for the JMS service	655
30.1.5	Deploying and testing the new service	662
30.2	Developing the adapter send update service	670
30.2.1	Creating artifacts and deploying adapter in System Manager	670
30.2.2	Importing the deployed connector in a service project	675
30.2.3	Defining resources in the test server.	680
30.2.4	Testing the adapter service using the test client	689
30.2.5	Developing the WSDL for the staff activity	694
30.3	Developing a business process.	697
30.3.1	Creating a business process.	697
30.3.2	Overview of the BPEL editor.	699
30.3.3	Anatomy of the BPEL editor	699
30.3.4	Activities and their icons	701
30.3.5	Tasks.	703
30.3.6	Developing the business process	708
30.3.7	Correcting some errors	721
30.4	Deploying and testing	727
30.5	Testing the process flow	728
30.6	Extending the business process	740
Chapter 31.	Integrating and automating the process	759

31.1 Developing the WebSphere Business Integration Connect send service . .	760
31.1.1 Define JNDI resources	760
31.1.2 Building the send service	761
31.1.3 Adding the send service to the process model	771
31.1.4 Testing the extended business process	778
31.2 Developing a process start bean	784
31.2.1 Generating the EJB that is called by an MDB	785
31.2.2 Generating the MDB	794
31.2.3 Defining resources	801
31.2.4 Testing the MDB and EJB	803
31.2.5 Generating proxy for process	808
31.2.6 Adapting the process model	810
31.2.7 Testing the proxy and the adapted process model	812
31.2.8 Adapting the EJB	818
31.2.9 Deploying and testing the changes	819
31.3 Building the receive process	821
31.3.1 Creating the service WBICReceive	821
31.3.2 Building the EJB to process the incoming message	821
31.3.3 Building the message-driven bean	821
31.3.4 Generating proxy for adapter	822
31.3.5 Updating EJB to call adapter proxy	822
31.3.6 Deploying and testing	822
Part 8. Looking after the run-time environment	823
Chapter 32. Gathering data from the run-time	825
32.1 Using WebSphere Business Integration Monitor	826
32.2 Our business process	830
32.3 Setting up the Monitor environment	837
32.4 Setting up the Message Broker environment	843
32.5 Using the Monitor data	857
32.5.1 Workflow Dashboard	857
32.5.2 Business Dashboard	863
Appendix A. Unit test traces	867
InputMessageFromFrontEnd.txt	867
RedTenantDelivery_Retrieve.txt	870
RedMaintRequest.txt	872
ResponseFromBackEnd_WhatIsTheVerb.txt	874
ResponseMessageFromBackEnd.txt	874
ResponseToBeSentToFrontEndAdapter.txt	881
Appendix B. Additional material	887

Locating the Web material	887
Using the Web material	888
System requirements for downloading the Web material	888
How to use the Web material	888
Related publications	889
IBM Redbooks	889
How to get IBM Redbooks	889
Help from IBM	889
Index	891

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:

@server®

@server®

Redbooks (logo) ™

CrossWorlds®

CICS®

Domino®

DB2®

IBM®

IMS™

Redbooks™

SupportPac™

WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

Preface

WebSphere® Business Integration is the IBM® business integration solution for process integration, workforce management, and enterprise application connectivity. WebSphere Business Integration helps you to create and deploy new business processes, synchronize business information in multiple business applications on diverse platforms, and transform message formats en-route between applications.

This IBM Redbook takes you through the full life cycle of an adapter development project, from design considerations, building, and testing through deployment and implementation on multiple broker types (using both an out-of-the-box technology adapter and the custom adapter for our development project).

For this redbook, we designed a scenario that mirrors many of the issues that real-life integration projects can face. The scenario starts by integrating custom enterprise applications. It then integrates those applications into the business-to-business world by extending the infrastructure. Finally, it adds a human interaction component which determines whether to take the internal route or external route (via trading partners) to application integration. Using many of the components within the WebSphere Business Integration family of products, this book includes a range of integration options that are available to implement this scenario.

The chapters in this book are divided into the following parts:

- ▶ Part 1, “Adapter development: The theory” on page 1 discusses the theory of adapter development and outlines the design considerations and the steps that are involved in adapter development.
- ▶ Part 2, “Developing our custom adapter” on page 169 details the development of our custom adapter.
- ▶ Part 3, “Setting up applications for our scenario” on page 305, Part 4, “Configuring and testing adapters and business objects” on page 333, and Part 5, “Message Broker components” on page 449 describe the integration of the custom enterprise applications using a combination of adapters and the WebSphere Business Integration Message Broker.
- ▶ Part 6, “Business-to-business components” on page 563 extends our reach to interact with our trading partners using WebSphere Business Integration Connect.

- ▶ Part 7, “Server Foundation components” on page 631 discusses business processing and logic to allow for human interaction in the overall infrastructure.
- ▶ Part 8, “Looking after the run-time environment” on page 823 includes some of the options that are available for monitoring the solution. Using a combination of the WebSphere Business Integration Modeler and Monitor, you can track business processes. Using the tools that are available in the WebSphere Business Integration Framework, you can track the availability of components from a technology perspective.

The team that wrote this redbook

This redbook 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.

Geert Van de Putte is an IT Specialist at the ITSO, Raleigh Center. He is a subject matter expert for messaging and business integration and has eight years of experience in the design and implementation of WebSphere Business Integration solutions. He has published several IBM Redbooks™ about messaging and business integration solutions. Before joining the ITSO, Geert worked at IBM Global Services, Belgium, where he designed and implemented EAI solutions for customers in many industries. Geert holds a Master of Information Technology degree from the University of Ghent in Belgium.

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 of experience in IT 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 Masters 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 Computing Science degree 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:

Margaret Ticknor
ITSO, Raleigh Center

A special thanks to all of the students who attended and worked through our *WebSphere Business Integration - Making It All Work Together* workshops worldwide.

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, IBM Business Partners, or customers.

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

Obtain more information 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 Redbooks to be as helpful as possible. Send us your comments about this 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 e-mail to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. 8IB Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195



Part 1

Adapter development: The theory

Adapter development

This chapter provides an overview of analysis and design issues to consider when planning a connector development project. It presents topics that can help you judge the complexity of building a connector for your application or technology.

As with most software development projects, careful planning early in the connector development cycle helps prevent problems during later implementation phases.

1.1 Technical assessment

The first step in adapter development is determining if you truly need to build an adapter. Perhaps using a technology adapter is a better approach. Whether to build a custom adapter is not an easy decision, nor one that should be taken lightly. Although there are many factors in determining whether to build a custom adapter, you can divide these factors into two broad categories:

- ▶ Technical drivers

Before you can decide to build a custom adapter, you need to determine whether it is even technically possible to do so. For example, does your application have an application programming interface (API) that can be used by the adapter to connect to the application and to send and receive data?

- ▶ Business drivers

The final decision of building a custom adapter is often determined by the business drivers, rather than technical drivers. Even though it might be technically feasible to build a custom adapter, business needs might determine that this is not the best direction for your company to take.

Examples of business drivers are:

- Time to market
- Cost of implementation or maintenance
- Product direction to open standards, such as Web services

Ultimately, whether to build a custom adapter is up to you. We hope that the information provided in this book can help make the decision process a bit easier.

Begin the technical assessment by first determining your business drivers. Because the business drivers are the deciding factors in your decision, it is vitally important that you know what they are before you begin the technical assessment.

1.2 Getting started with the development

The following sections provide a sample set of planning questions that should be addressed prior to undertaking your adapter development project.

1.2.1 Understanding the application

Sample planning questions to understand the application include:

1. What is the application operating system?
2. What programming languages were used to create the application?
3. What is the execution architecture of the application?
4. Is there a central database for application data? What type of database is it?
5. Is the application or its database distributed across multiple servers?

1.2.2 Identifying the directionality of the connector

To identify the directionality of the connector consider whether the connector needs to send data, receive data, or both?

1.2.3 Identifying the application-specific business objects

Sample planning questions to identify the application-specific business objects include:

1. Do application entities have contained entities?
2. Are there application business entities that are the same type but have different physical representations in the application?
3. Are there application entities that reside in more than one location in the database but correspond to the same logical entity?
4. Are there batch processes associated with the creation of application entities?

1.2.4 Investigating the application data interaction interface

Sample planning questions to investigate the application data interaction interface include:

1. Have there been any other efforts to integrate with this application?
 - a. What was the purpose of the integration?
 - b. Does the integration use interfaces that modify or retrieve information?
 - c. If the integration is able to process an event generated in the application, what is the mechanism used to trigger event processing?
 - d. Will your connector replace the preexisting integration?
2. Is the application data shared by other applications?
 - a. Do other applications create, retrieve, update, or delete this application's data?
 - b. What is the mechanism used by other applications to gain access to the data?
 - c. Is there object-specific business logic used by other applications?
3. Is there a mechanism that the connector can use to communicate with the application?
 - a. Does the API allow access for create, retrieve, update, and delete operations?
 - b. Does the API provide access to all data entity attributes?
 - c. Does the API allow access to the application for event detection?
 - d. Are there inconsistencies in the API implementation?
 - e. Describe the transaction behavior of the API.
 - f. Is the API suited for metadata design?

A metadata-driven connector handles each business object that it supports based on metadata encoded in the business object definition rather than on instructions hard-coded in the connector. An example of this is to execute a business function or update a business component. If the connector is using an API to handle updates to the application database, for example, the application-specific information can provide the connector with information to run an API.
 - g. Does the API enforce application business rules?
4. Are there batch clean-up or merge programs used to purge redundant or invalid data?

1.2.5 Investigating the event management and notification mechanism

Sample planning questions to investigate the event management and notification mechanism include:

1. Describe the event management mechanism.
2. Does it provide the necessary granularity to establish the distinct object and verb?
3. Does event notification occur at a level that can support application business logic?
4. Can the event management mechanism persist event records?

1.2.6 Investigating communication across operating systems

Sample planning questions to investigate the communication across operating systems include:

1. Does the API handle the communication mechanism between the application operating system and the connector operating system?
2. If not, is there a mechanism available to handle communication across operating systems?

1.3 Getting started with the development project

As you assemble the answers to these questions, you acquire essential information about application data entities, business object processing, and event management. These findings become the basis for a high-level architecture for the connector.

When you have determined the entities that your connector can support and have examined the application functionality for database interaction and event notification, you should have a clear understanding of the scope of the connector development project. At this point, you can continue with the next phases of connector development.

As part of the connector development process, you code the application-specific component of the connector and then compile and link the connector source files. In addition, the overall process of developing a connector includes other tasks, such as developing application-specific business objects. A quick checklist or overview of the tasks in the connector development process includes the following:

1. Identify the application entities that the connector will make available to other applications, and investigate the integration features that are provided by the application.

Note: If your business integration system uses InterChange Server, identify generic business objects that the connector will support, and define application-specific business objects that correspond to the generic objects.

If your business integration system uses InterChange Server, analyze the relationship between the generic business objects and the application-specific business objects and implement the mapping between them.

These activities are not, strictly speaking, part of adapter development but are essential to keep in mind for the overall project.

2. Define a connector base class for the application-specific component, and implement functions to initialize and terminate the connector.
3. Define a business object handler class, and code one or more business object handlers to handle requests.
4. Define a mechanism to detect events in the application, and implement the mechanism to support event subscriptions.
5. Implement error and message handling for all connector methods.
6. Build the connector.
7. Configure the connector.
8. If your business integration system uses InterChange Server, use the Connector Configurator tool to create the connector definition and save it in the InterChange Server repository. You can call Connector Configurator from System Manager.
9. If your business integration system uses a WebSphere message broker (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, use Connector Configurator to define and create the connector configuration file.

- 10.If WebSphere MQ will be used for messaging between connector components, add message queues for the connector.
- 11.Create a script to start the new connector.
- 12.Test and debug the connector, recoding as necessary.

Archived

Business objects

This chapter describes business objects, including:

- ▶ Information about the structure of business objects for the WebSphere Business Integration system
- ▶ Considerations for business object design

A business integration system uses business objects to carry data and processing instructions between an integration broker and adapters. Business objects represent a request from an integration broker, an event in an application or Web server, or a call from an external site.

2.1 Overview of business objects

A business object definition represents a template for data that can be treated as a collective unit. It contains a business object header, which specifies the name and version of the business object definition. In addition, the business object definition contains the following information:

- ▶ Business object attributes and attribute properties (Name, Type, Cardinality, Key, Foreign key, Required, AppSpecificInfo, Max length, Default value, Comments)
- ▶ Business object verbs
- ▶ Business object application-specific information

Figure 2-1 shows the parts of a business object definition.

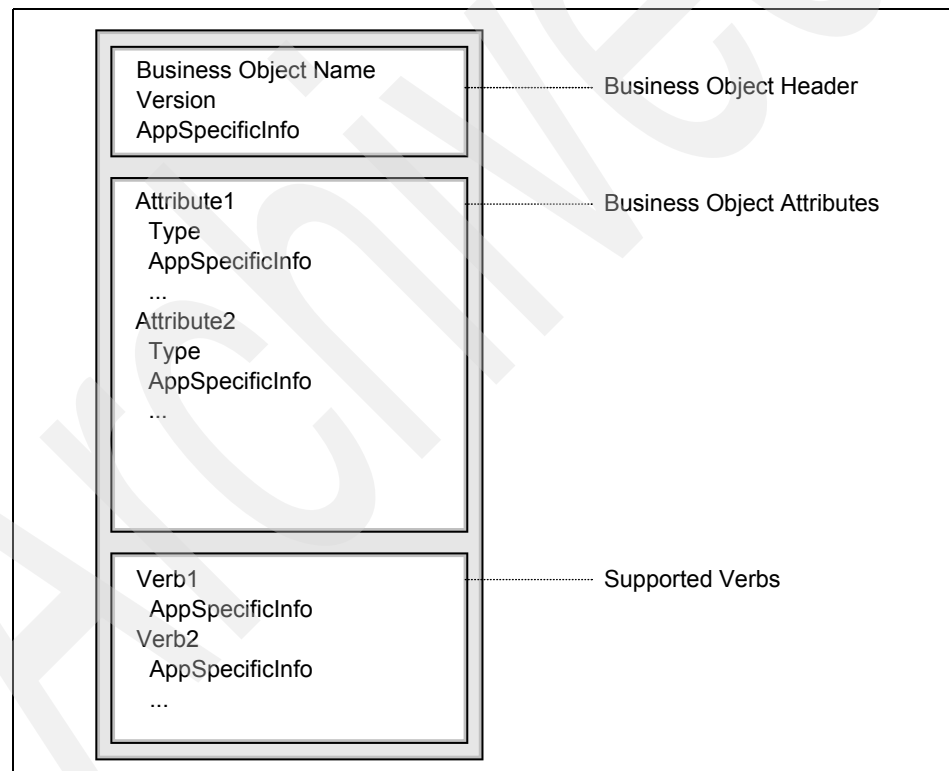


Figure 2-1 Business object definition parts

While the business object definition represents the template for a collection of data, a business object instance (often just called a *business object*) is the run-time entity that contains the actual data. The business object is what is

passed between components of the business integration system. The business object contains the following information:

- ▶ Attributes, each of which contains the data for the associated business object. One of the attributes is usually a key attribute, which contains a value that uniquely identifies this business object among all business objects of the same definition.
- ▶ An active verb, which should be one of the supported verbs for the business object definition.

Figure 2-2 shows the Customer business object and a corresponding business object instance for this definition.

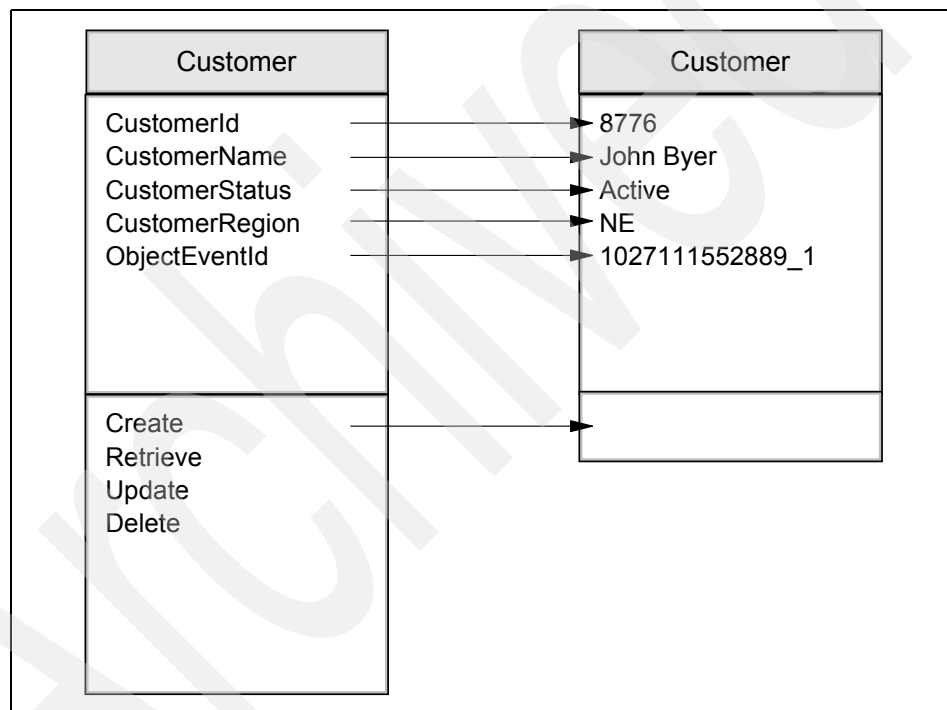


Figure 2-2 Business object definition and sample business object

2.2 Types of business objects

There are essentially two types of business objects: application-specific object and generic business object. Depending on which type of integration broker you choose, you will use one or both of these business object types.

The key to the design of business objects is to develop a business object definition that models as closely (and efficiently) as possible the data that needs to be transmitted between components of the business integration system, for example:

For data that is transferred between a connector and an integration broker, you design application-specific business objects that model the appropriate application entities. These entities might correspond to data structures or technology standards that are used by a particular application, or to specific technology standards that are used by a Web server.

2.2.1 Application-specific business object

An *application-specific business object* is used between an adapter and an integration broker. It models the appropriate application entities for each adapter. An application-specific business object is the unit of work that is triggered within the application, created and processed by the connector, and sent to the integration broker. A connector uses the application-specific business object to export data from its application to other applications and to import data from other applications.

The connector exposes all the information about an application entity that is necessary to allow other applications to share the data. After the connector makes the entity available to other applications, the integration broker can route the data to any number of other applications through their connectors.

Designing the relationship between the connector and its supported application-specific business objects is one of the tasks in connector development. Application-specific business object design can generate requirements for connector programming logic that must be integrated into the connector development process. Therefore, business object and connector developers must work together to develop specifications for the connector and its business objects.

Careful application-specific business object design for both metadata syntax and business object structure is essential for adapter development, regardless of the integration broker that you are using. For more information about metadata syntax, see 2.4, “Application-specific information and metadata” on page 19 and 2.5, “Metadata and metaobjects” on page 21. For more information about business object structure, see 2.3, “Business object design” on page 15.

2.2.2 Generic business object

Generic business object (GBO) design is not actually part of adapter development. However, you must consider it later if you are using the InterChange Server for collaborations and mapping.

2.3 Business object design

Business objects can be either flat or hierarchical.

2.3.1 Flat business object

A flat business object contains one or more simple attributes and a list of supported verbs. A simple attribute represents one value, such as a String or Integer or Date. All simple attributes have single cardinality. In the case of an application-specific business object, a flat business object can represent one entity in an application or in a technology standard.

Use of flat business objects can simplify corresponding connector design in the following ways:

- ▶ On a *Create* operation, the connector might cycle through the attributes, extracting the non-key attribute values from the business object instance and extracting processing instructions from the business object definition. When it has assembled the information it needs to process the business object, the connector might start an application function call or SQL statement to create a new row for the record in the table. The connector then returns a value for the key to the business integration system.
- ▶ On a *Retrieve* operation, the connector might extract the primary key from the business object request, use the key value to retrieve the current set of data for the row, and return a business object with the complete set of values. This type of business object is straightforward in its design and in the connector logic required to process it.

Typically, however, application entities are more complex and include information that is stored in other objects.

2.3.2 Hierarchical business object

Hierarchical business object definitions define the structure of multiple related entities, encapsulating not only each individual entity but also aspects of the relationship between entities. In addition to containing at least one simple attribute, a hierarchical business object has one or more attributes that are complex (that is, the attribute itself contains one or more business objects, called *child business objects*). The business object that contains the complex attribute is called the *parent business object*.

There are two types of relationships between parent and child business objects:

Single cardinality When an attribute in a parent business object represents a single child business object. The type of the attribute is set to the name of the child business object, and the cardinality is set to one.

Multiple cardinality When an attribute in the parent business object represents an array of child business objects. The type of the attribute is set to the name of the child business object, and the cardinality is set to *n*.

In turn, each child business object can contain attributes that contain a child business object, or an array of business objects, and so on. The business object at the top of the hierarchy, which itself does not have a parent, is called the *top-level business object*. Any single business object, independent of its child business objects that it might contain (or that might contain it), is called an *individual business object*.

2.3.3 Business object structure

The way a connector or data handler processes business objects is determined in part by the structure of the business objects that it supports.

As you design the structure of an application-specific business object, you need to determine what structure best represents a particular application entity and how this structure affects the design of connector and data handler logic or how the structure is processed by an existing connector or data handler. Although a goal of connector and data handler design is to code a connector or data handler so that it can handle new and changed business objects without modification, it is difficult to create a connector or data handler that can handle any possible business object.

Typically, a connector or data handler is designed to make assumptions about the structure of its business objects, the relationships between parent and child business objects, and the possible application representation of business

objects. If designing for an existing connector or data handler, your task is to understand these assumptions and design business objects accordingly.

A beginning set of questions to consider about the structure of an application-specific business object is:

- ▶ What is the organization or database schema for the application entity that will be encapsulated in the business object?
- ▶ Does the application entity represent hierarchical data or one-to-many relationships?
- ▶ Does a business object represent one application entity or more than one application entity? In other words, can attribute values in an individual business object be stored in different application entities?
- ▶ What kind of relationships between business objects does the connector or data handler handle?
- ▶ How are the relationships modeled in the business objects, or how are they processed by the connector or data handler?

The simplest business object design is a flat business object that represents one entity. All the attributes of a flat business object are simple.

A business object can represent application entities that include data from other entities, as shown in Table 2-1.

Table 2-1 Representing multiple entities

Structure of business object	Type of data organization	Type of parent/child relationship
Parent business object can have one or more child business objects that represent the other entities.	One-to-one One-to-many	Structural
Parent business object can have one or more foreign-key attributes that reference other top-level business objects that represent the other entities.	One-to-one One-to-many	Semantic
If the application and its interface permit, a flat business object can include attributes that directly reference other entities.	One-to-one	None

When deciding how to structure business objects that represent multiple entities, consider these guidelines:

- ▶ If the relationship between entities is a one-to-many relationship, represent the data in the subordinate entities as child business objects.
- ▶ If the relationship between entities is a many-to-many relationship, represent the data in the related entities as top-level business objects that are referenced by the parent rather than contained by the parent.
- ▶ If a business object definition for an entity includes many attributes from another entity, and the attributes from the second entity form a logical grouping, you might want to create a child business object definition for the second entity rather than locate all the attributes for both entities in the same business object definition.
- ▶ If an existing business object already contains other child business objects, creating one or more child business objects that represent new entities makes the business object structure consistent.

2.3.4 Wrapper business objects

Your business object might need to have multiple, different objects inside that have different uses. A request/response might be one scenario (such as a Web service or workflow operation). Another might be where an adapter (JDBC™, for example) contains multiple tables that are separate business objects but do not have any direct relationships. Another is a wrapper object that contains both the business data (for a data handler, for example) and a dynamic child metaobject that contains run-time parameters.

This can be achieved by using a *top-level business object*. Figure 2-3 on page 19 shows a top-level business object for an existing adapter, which has different request and response objects but also contains run-time configuration information such as a URL for a request and Multipurpose Internet Mail Extensions (MIME) type for the data.

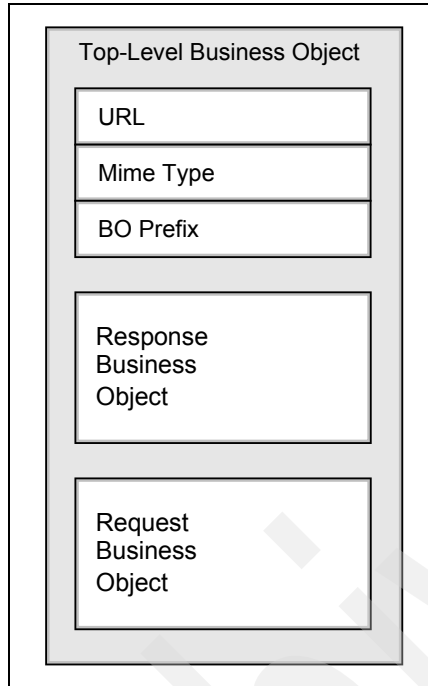


Figure 2-3 Top-level business object with different request and response objects

While designing the application-specific business object, keep in mind that its primary role is to model the entity in the data source. It is also important to identify how its associated connector or data handler handles its processing and what are the requirements of the business process in which it participates.

2.4 Application-specific information and metadata

A business object definition can provide application-specific information, whose content provides metadata to the component that processes the business object. Within a business object definition, you can provide application-specific information at one of the following levels:

- ▶ The business object definition
- ▶ An attribute within the business object definition
- ▶ The business object verb

A common use of application-specific information is to provide a connector or data handler with application-dependent instructions about how to process the business object. The application-specific information is a string that is entered

during business object design and read at run-time by a connector or data handler.

One of the key pieces of information that you provide in the application-specific information is information regarding the structure of the business object itself. For example:

An SAP ALE object represents an IDoc, the child business objects represent the IDoc segments, and the attributes are the fields within the segments. The structure of the IDoc is all important. The application-specific information provides the connector with the structure of the IDoc object and how the SAP RFC layer (API) should treat it.

Conversely, a business object for use with the JDBC connector represents a table or view and the attributes are the columns in the table. The application-specific information provides the connector with the knowledge it requires to put an SQL string together to communicate with the database.

Connectors that are designed to use the application-specific information in definitions of their application-specific business objects are called metadata-driven connectors. An application-specific component that is metadata-driven is flexible because it has no hard-coded instructions for each type of business object that it supports. Without recoding or recompiling, the application-specific component automatically supports new business object definitions, as long as the corresponding application data can be accurately described by the adapter's metadata syntax.

When designing business objects to maximize the metadata-driven behavior of a connector, follow these general recommendations for storing application-specific information in the business object definition:

- ▶ Store entity names, such as table or form names, in the business object-level `AppSpecificInfo` property of a top-level business object.
- ▶ Store subform names or table names in the business object-level `AppSpecificInfo` property of a child business object.
- ▶ Store field names, column names, and other information related to business object attributes in the attribute `AppSpecificInfo` property.
- ▶ Store verb processing instructions in the verb `AppSpecificInfo` property.

The careful use of the `AppSpecificInfo` property enables a connector to handle a variety of business objects in the same way. If an application is consistent in how it handles data operations, and if for all operations the connector performs consistent tasks, the business object can be designed to enable a completely metadata-driven connector.

2.5 Metadata and metaobjects

So, the recommended approach to designing the relationship between business objects and connectors is to store information in the business object definition that helps a connector interact with an application or data source, attribute, and business object verb. However, this information is always *static* information because it is stored in the business object *definition*, and it is about the business object.

It is also possible to use metaobjects that carry run-time data values to specify adapter behavior. The types of values where this might be useful are:

- ▶ Destination file names or directories
- ▶ URLs for a request
- ▶ MIME type
- ▶ Destination or reply queue names

A metaobject is a business object that holds metadata. Connectors can recognize and read two types of metaobjects:

- ▶ Static connector metaobject

Connectors that use metaobjects have as a connector-specific property the metaobjects that it supports, such as:

- DataHandlerConfigMO
- DataHandlerMimeType
- DataHandlerClassName
- Protocol Config MO (for the new HTTP Adapter)

- ▶ Dynamic child metaobject

If it is difficult or unfeasible to specify the necessary metadata through a static metaobject, the connector can optionally accept metadata that is delivered at run-time for each business object instance.

Dynamic metaobjects allow you to change the metadata used by the connector to process a business object on a per-request basis during request processing and to retrieve information about an event message during event processing.

The attribute values of the dynamic child metaobject duplicate and override those of the static metaobject.

Object Discovery Agents

This chapter describes how to develop an Object Discovery Agent (ODA) by using classes that are defined in the Object Discovery Agent Development Kit (ODK) API. An ODA works with Business Object Designer's Business Object Wizard to develop business object definitions for a specific connector or data handler.

3.1 Overview of ODA

You can create application-specific business object definitions for an adapter by using its ODA. The ODA is an optional component of an adapter. When you install a predefined adapter that has an ODA, its ODA is installed automatically. When you are developing a custom adapter and you want to use an ODA to create business object definitions, you can develop it by using the Object Discovery Agent Development Kit (ODK).

At run-time, running an ODA involves the following components:

- ▶ Business Object Designer provides a graphical interface in the form of a wizard to interact with the ODA: Business Object Wizard. The wizard displays a series of dialog boxes to obtain information that the ODA needs to generate the content.
- ▶ The ODA run-time is the intermediary component between Business Object Wizard and the ODA. It uses the classes of the ODK API and the ODK infrastructure to communicate with the ODA. It is the ODA run-time that you start with the ODA startup script.
- ▶ The ODA is the component that discovers source nodes in the data source and generates the content. The ODA receives information in the dialog boxes of Business Object Wizard from the ODA run-time. It then sends information (such as the generated content) to the ODA run-time, which sends it to Business Object Wizard.

Figure 3-1 on page 25 shows the components of the ODA run-time architecture.

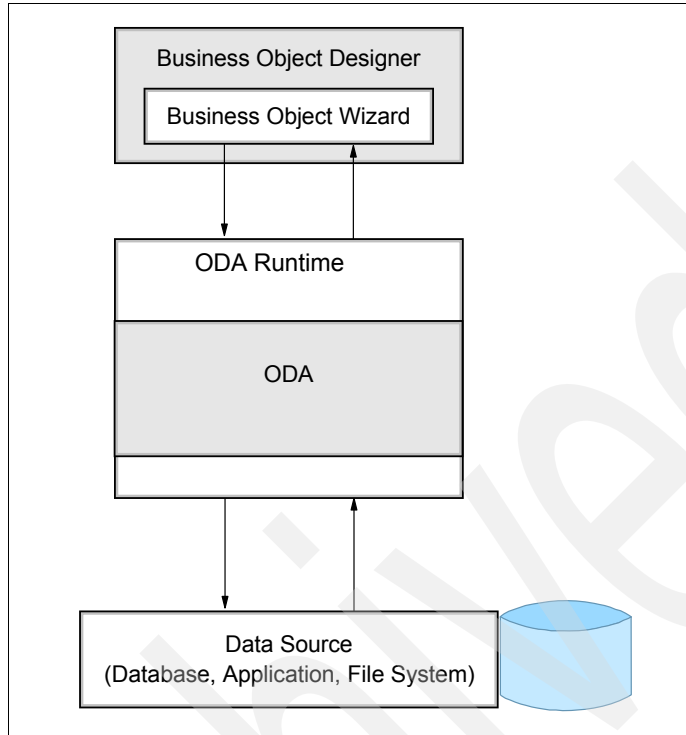


Figure 3-1 ODA architecture

To generate the business object definitions, the ODA:

1. Obtains values for the ODA configuration properties (such as user name and database type) that the ODA requires to connect to the data source (such as an application, database, or file system).
2. Uses these configuration properties to connect to the data source.
3. Obtains the list of *available* source nodes for which business object definitions are to be created, and presents them to the user.
4. Discovers the requirements for the data-source entity that are underlying the source node that is *selected by the user from the list presented* (as defined by an application, database table, file system, or DTD).
5. Generates business object definitions that meet the requirements of the WebSphere Business Integration system and the component that processes the business object, and returns the business object definitions to users.

3.2 Design considerations

To develop an ODA:

1. Extend the ODA base class, `ODKAgentBase2`, to create your ODA class.
2. Implement the methods of the ODA class, which provides the means of starting the ODA.
3. Design and implement the ODA content.
4. Implement error and message handling for all ODA methods.
5. Create any class that needed to handle data-source interactions.
6. Build the ODA.
7. Create a startup script for the new ODA.
8. Test and debug the ODA.

3.3 Setting up the development environment

You need to set up the development environment for the ODA as follows:

1. Install WebSphere Business Integration system software. The ODA library (`CwODK.jar`) is necessary to run ODA. Also, Business Object Designer is necessary to create application-specific business objects by using ODA.
 - If you use WebSphere InterChange Server, the ODA library and Business Object Designer are installed as part of the InterChange Server software.
 - If you use WebSphere Business Integration Message Broker or WebSphere Application Server, you must install the WebSphere Business Integration Adapters product for the ODA library and Business Object Designer.
2. Install the Java Development Kit (JDK™) or a JDK-compliant development product (such as WebSphere Studio Application Development).

3.4 Object Discovery Agent process flows

When a user chooses **File** → **New Using ODA** (to create a business object), the Business Object Designer invokes the Business Object Wizard to run the ODA. The Business Object Wizard then displays the Select Agent dialog box, which provides graphical access to all available Object Discovery Agents. From this dialog box, users select the ODA to run.

The Business Object Wizard connects to this ODA with the following steps:

1. Instantiates an ODA object, which is an object of the ODA class. The ODA class is the extension of the ODA base class, `ODKAgentBase2`. It defines the behavior of the ODA.
2. Obtains a handle to the ODA object, which can be used to access this object when started.

Note: An ODA must already be started for the Business Object Wizard to list it as an ODA available to run.

3.4.1 Obtaining ODA configuration properties

The Business Object Wizard displays the Configure Agent dialog box, which shows the ODA configuration properties. Configuration properties are those properties that the ODA needs to be able to begin running. The ODK API represents a configuration property as an agent-property (`AgentProperty`) object. In this step, the wizard displays the configuration properties, allows you to update them, and then writes the user-initialized properties into the ODA run-time memory.

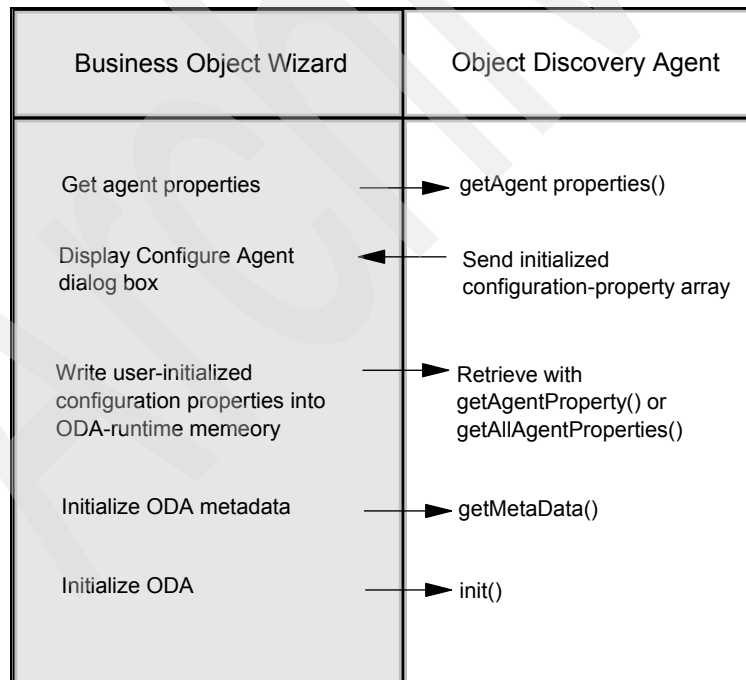


Figure 3-2 Configure agent of Business Object Wizard

As 3.4, “Object Discovery Agent process flows” on page 26 shows, the Business Object Wizard:

1. Obtains the configuration properties from the selected ODA and displays them in the Configure Agent dialog box. To obtain the configuration properties from the ODA, the wizard calls the `getAgentProperties()` method, which is defined in the ODA base class, `ODKAgentBase2`. This method is an abstract method that the ODA developer must implement as part of the ODA class. It returns the ODA's configuration properties to Business Object Wizard as an array of `AgentProperty` objects. These configuration properties can include the names, types, any valid values, descriptions, input restrictions, and any default values. In addition to the configuration properties that the `getAgentProperties()` method provides, Business Object Wizard always provides a set of standard configuration properties, which are common to all ODAs:
 - `MessageFile`
 - `TraceLevel`
 - `TraceFileName`
2. From the Configure Agent dialog box, accepts the entered values or changes for the configuration properties. The wizard sends the user-initialized configuration properties to the ODA. The Business Object Wizard saves these properties in the ODA run-time memory. Within the ODA, you can access these properties through an instance of the `ODKUtility` class, which provides the `getAgentProperty()` and `getAllAgentProperties()` methods for this purpose.
3. Initializes the ODA's metadata, which provides information about the ODA and its capabilities. After it calls the `getAgentProperties()` method, Business Object Wizard calls the `getMetaData()` method of the ODA base class, `ODKAgentBase2`. This method is an abstract method that the ODA developer must implement as part of the ODA class. It returns an initialized `AgentMetaData` object that contains the ODA metadata.
4. Initializes the ODA based on the user-initialized startup properties. To initialize the ODA, the wizard calls the `init()` method of the ODA base class, `ODKAgentBase2`. This method is an abstract method that the ODA developer must implement as part of the ODA class. It performs initialization tasks such as resource allocation and creating a connection to the data source.

3.4.2 Selecting and confirming source data

Business Object Wizard displays the Select Source dialog box, which displays the source nodes of the data source. The source nodes are arranged in the source-node hierarchy. Each source node is the name of an object that the ODA has discovered in the data source. It can either be expanded to display other child nodes or selected for generation into content. Users can expand this source-node hierarchy to choose objects in the data source for conversion to content.

The wizard takes the following actions:

1. Obtains the source-node hierarchy from the selected ODA and displays it top level in the Select Source dialog box. To obtain the source-node hierarchy, the wizard calls the `getTreeNodees()` method of the `IGeneratesBoDefs` interface. The ODA developer must implement this method as part of the ODA class implementation of the `IGeneratesBoDefs` interface. It searches the data source to “discover” source nodes and returns these source nodes to Business Object Wizard as an array of `TreeNode` objects. When users expand a node for the first time, the wizard calls the `getTreeNodees()` method to display that particular level in the source-node hierarchy. Users can traverse this hierarchy to select the level of detail.
2. From the Select Source dialog box, keeps track of the names of the source nodes in the hierarchy that you select for content generation. The wizard generates an array that contains the names of the selected source nodes.

The Business Object Wizard displays the Confirm Source Nodes dialog box, which displays the selected source nodes. You can either confirm the selections or go back to the Select Source dialog box to reselect source nodes. When you click **Next**, the wizard begins the content generation.

3.4.3 Generating content

You can write an ODA to generate one or both of the content types of business object definitions and binary files. The content type determines the structure of the data that the ODA generates. For an ODA to support a particular content, it must implement the appropriate content-generation interface for the ODA.

After source nodes are selected and confirmed, the Business Object Wizard enters the content generation. It displays the Generating Business Objects screen and passes the array of user-selected source nodes to the ODA by calling the content-generation method for business object definitions, `generateBoDefs()`. This method generates the corresponding business object definitions for the selected source nodes. Because an ODA must support the generation of business object definitions in the on-request content protocol,

Business Object Wizard always calls the `generateBoDefs()` method. Therefore, the ODA developer must implement this method as part of the ODA's implementation of the `IGeneratesBoDefs` interface.

Whether the ODA generates file content depends on whether it implements the `IGeneratesBinFiles` interface. If the ODA class implements this interface, the method that actually provides the generated content depends on the content protocol that the ODA uses for the file content type, as follows:

- ▶ If the ODA uses the on-request content protocol to generate content, Business Object Wizard initiates content generation by calling the content-generation method, `generateBinFiles()`. It passes to this method the array of user-selected source nodes. Therefore, for the ODA to support file content, the ODA developer must implement this method as part of the ODA's implementation of the `IGeneratesBinFiles` interface.
- ▶ If the ODA uses the callback content protocol to generate content, the ODA (or some external process) initiates content generation by calling a user-defined method. The ODA developer must implement a mechanism to generate the files.

Therefore, whether Business Object Wizard calls the content-generation method for files, `generateBinFiles()`, depends on the following:

- ▶ Whether the ODA implements that `IGeneratesBinFiles` interface.
- ▶ If it implements `IGeneratesBinFiles`, which content protocol the ODA uses to generate files.

Regardless of the content protocol used, the generation of content involves the following steps:

1. Optionally, obtaining any additional information, such as verb values, as business-object properties.

Often the ODA needs additional information before it can generate the business object definitions. The ODA can request this additional information by defining business-object properties. The ODK API represents a business-object property as an agent-property (`AgentProperty`) object. To collect business-object properties, the ODA can have Business Object Wizard display the Business Object Properties dialog box. In this dialog box, the wizard displays the business-object properties, allows updates, and writes the user-initialized properties into the ODA run-time memory.

2. Generating the requested content and saving it in the generated-content structure in ODA memory.

The ODA provides its generated content to Business Object Wizard in two parts:

- The content metadata

A content metadata (ContentMetaData) object contains information about the ODA's generated content. The Business Object Wizard uses this information to determine which content-retrieval method to use to retrieve the generated content.

- The content itself

The ODA writes the generated content to a generated-content structure, somewhere that is accessible by the methods of the ODA class. For example, it could write the content to an array that is a member variable of the ODA class.

The method that provides the generated content depends on the content protocol that the ODA uses for a particular content type, as follows:

- If the ODA uses the on-request content protocol to generate content, it is the content-generation method that populates the generated-content structure and returns a content metadata object to Business Object Wizard. The Business Object Wizard invokes the content-generation method based on the content type, as follows:
 - For business object definitions, the generateBoDefs() method in the IGeneratesBoDefs interface
 - For files, the generateBinFiles() method in the IGenerateBinFiles interface
- If the ODA uses the callback content protocol to generate content, it is a user-defined method that populates the generated-content structure and sends a content metadata object to Business Object Wizard.

3.4.4 Saving content

The Business Object Wizard displays the Save Business Objects dialog box, which provides options for saving the generated business object definitions. The Business Object Wizard provides the ability to save generated content to an Integration Component Library (ICL) project or a file, or to open each business object definition in the Business Object Designer. To save the generated business object definitions in the specified format, the Business Object Wizard must access the generated content.

3.5 Object Discovery Agent data sources

The ODA data sources can be applications, database tables, file systems, or URIs. Because the ODA needs to get metadata of business objects, the data source provides an API or some other way for this to be achieved.

Note: For more information, see the *IBM WebSphere Business Integration Adapters Business Object Development Guide* at:

<http://www.ibm.com/software/integration/wbiadapters/library/infocenter/index.html>



Agent initialization and termination

This chapter addresses the requirements for an adapter for its initialization and termination process.

4.1 Connector startup

The connector startup begins execution by invoking the Adapter Framework. After the framework is executing, it performs steps to invoke the application-specific component of the connector, based on the integration broker.

4.1.1 WebSphere InterChange Server

If the WebSphere InterChange Server is the integration broker, the framework takes the following steps to invoke the application-specific component:

1. Uses the Object Request Broker (ORB) to establish contact with the InterChange Server.
2. From the repository, loads the connector definition information into memory for the connector process:
 - Connector configuration properties
 - A list of supported business object definitions
3. Begins the execution of the connector application-specific component by instantiating the connector base class and calling methods of this base class that initialize the application-specific component (see 4.2, “Connector initialization” on page 35 and 4.3, “Obtaining the business object handler” on page 37).

After these methods have been called, the connector is active and operational.

4. Contacts the Connector Controller to obtain the subscription list (that is, business objects to which collaborations have subscribed).

4.1.2 Business Integration Message Broker or Application Server

If WebSphere Business Integration Message Broker or WebSphere Application Server is the integration broker, the framework takes the following steps to invoke the application-specific component:

1. From the local repository, loads the following connector definition information into memory for the connector process:
 - Connector configuration properties
 - A list of supported business object definitions

2. Begins the execution of the connector application-specific component by instantiating the connector base class and calling methods of this base class that initialize the application-specific component (see 4.2, “Connector initialization” on page 35 and 4.3, “Obtaining the business object handler” on page 37).

4.2 Connector initialization

To commence connector initialization, the Adapter Framework calls the initialization method of the connector base class:

Class	CWConnectorAgent
Method	agentInit()

As part of the implementation of the connector class, you *must* implement an initialization method for your adapter. The main tasks that the initialization method should perform are:

- ▶ Establishing a connection.
- ▶ Checking the connector version.
- ▶ Recovering In-Progress events.

Important: During the execution of the initialization method, business object definitions and the subscription list are not yet available.

4.2.1 Establishing a connection

The main task that the initialization method must perform is to establish a connection to the application. To do this, the initialization method should:

1. Read the configuration properties that provide information such as `ApplicationUserId` and `ApplicationPassword` that are used to send the logon information to the application.
2. Use the `getConfigProperty()` method to obtain the value of these configuration properties.
3. If a required value is empty, provide a default value (if appropriate).
4. Obtain any connections or files that are required. The initialization method opens a connection with the application. It returns “success” if it succeeds in opening the connection.
5. In a Java adapter, the `agentInit()` method should throw the following:
 - `ConnectionFailureException` if the connection fails.
 - `LogonFailureException` if the connector is unable to log on to the application.

4.2.2 Checking the connector version

The `getVersion()` method returns the version of the adapter. It is called in both of these instances:

- ▶ The initialization method should call `getVersion()` to check the adapter version.

Note: An adapter should also keep track of which versions of an application it supports. It should check the application version when it logs on to the application.

- ▶ The Adapter Framework calls the `getVersion()` method when it needs to get a version for the adapter.

4.2.3 Recovering In-Progress events

During event notification, processing an event includes:

1. Updating the event status as In-Progress.
2. Performing a retrieve for the required application entity.
3. Creating a new business object.
4. Sending the business object to the Adapter Framework.
5. Updating the event status (as sent or failed).

If the connector terminates while processing an event (before updating the event status), an In-Progress event will remain in the event store. When the connector is restarted, it should check the event store for any events that have a status of In-Progress.

This behavior should be configurable. Many adapters use the `InDoubtEvents` configuration property for this purpose. Figure 4-1 on page 37 shows the settings for the use of this property.

For a Java adapter, the `CWConnectorEventStore` class provides the `recoverInProgressEvents()` method to obtain event records with an In-Progress status. As an adapter developer, you should implement this method to take actions that are based on the `InDoubtEvents` configuration property. In this method, the developer can also change the status of In-Progress events to a Ready-To-Poll status, as also shown in Figure 4-1.

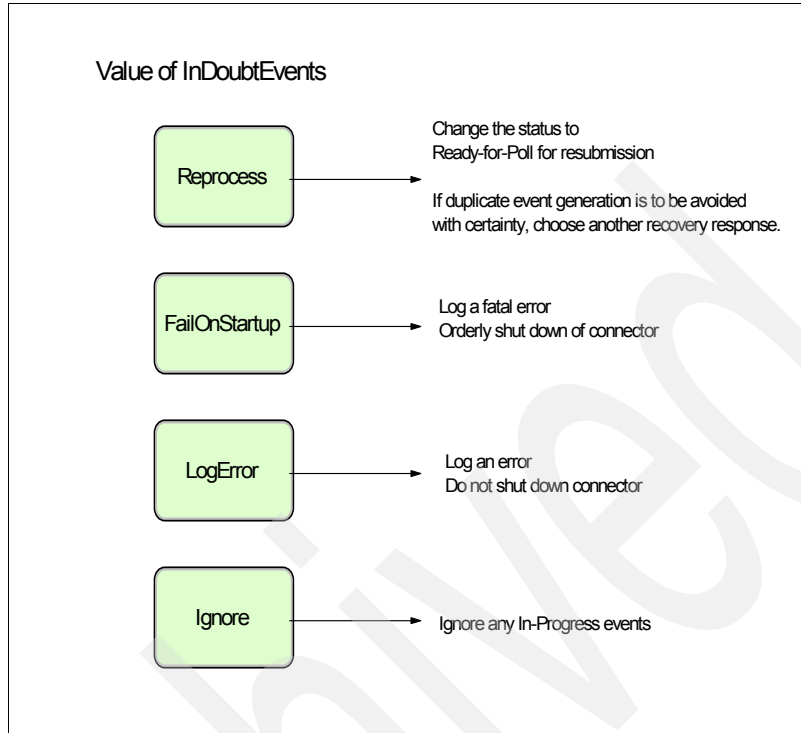


Figure 4-1 Actions to recover In-Progress events

4.3 Obtaining the business object handler

In the final step in connector initialization, the framework obtains the business object handler for each business object that the connector supports. A business object handler receives request business objects from the Adapter Framework and performs verb operations that are defined in the business objects (see Chapter 6, “Business object handlers” on page 49 for more information about business object handlers).

In a Java adapter, the business object handler base class is `CWConnectorBOHandler`. To obtain an instance of a business object handler for a supported business object, the Adapter Framework calls the `getConnectorBOHandlerForBO()` method, which is defined as part of the `CWConnectorAgent` class.

Every adapter *must* have a `getConnectorBOHandlerForBO()` method defined in its connector base class to retrieve the business object handler. This method

returns a reference to the business object handler for a specified business object definition.

The default implementation of `getConnectorBOHandlerForBO()` in the `CWConnectorAgent` class returns a business object handler for a business-object-handler base class named `ConnectorBOHandler`.

If you name your extended business-object-handler base class `ConnectorBOHandler`, you do not need to override the `getConnectorBOHandlerForBO()` method. However, if you name your extended business-object-handler base class something other than `ConnectorBOHandler`, you must override `getConnectorBOHandlerForBO()` to return an instance of your extended business-object-handler base class.

To instantiate the business object handler, the Adapter Framework processes as follows:

1. During initialization, the Adapter Framework receives a list of business object definitions that the connector supports.
2. The Adapter Framework calls the `getConnectorBOHandlerForBO()` method.

Note: This is called once for every supported business object.

3. The `getConnectorBOHandlerForBO()` method instantiates the correct business object handler for that business object. This is based on the name of the business object definition it receives as an argument.
4. The business object handler is returned to the Adapter Framework.
5. The Adapter Framework stores the reference to the business object handlers in the associated business object definition, in the memory of the connector's process.

The number of business object handlers that the Adapter Framework obtains through its calls to the `getConnectorBOHandlerForBO()` method depends on the overall design for business object handling in your adapter:

- ▶ If the adapter is metadata-driven, it can be designed to use a generic, metadata-driven business object handler.
- ▶ If some or all application-specific business objects require special processing, you must set up multiple business object handlers for those objects.

(For more information regarding business object design and metadata-driven business objects, see 2.3, “Business object design” on page 15 and 2.5, “Metadata and metaobjects” on page 21).

Important: During execution of the `getConnectorBOHandlerForBO()` method, the business object class methods are not available.

Example 4-1 contains an implementation of the `getConnectorBOHandlerForBO()` method that returns a metadata-driven business object handler. It calls the constructor for the `ExampleBOHandler` class, which instantiates a single business-object-handler base class that handles all the business objects supported by the connector.

Example 4-1 getConnectorBOHandlerForBO

```
public CWConnectorBOHandler getConnectorBOHandlerForBO(String BOName){ return
new ExampleConnectorBOHandler();
}
```

4.4 Connector termination

The Adapter Framework calls the `terminate()` method when the connector is shutting down. In your implementation of this method, it is good practice to free all the memory and log off from the application. You must implement this method for the adapter.

The `CWConnectorAgent` class does not provide a default implementation for the `terminate()` method. Therefore, the connector class *must* implement this method if resource clean-up is required.

4.5 Extending the connector base class

For initialization and termination of a connector, the following methods of the `CWConnectorAgent` base class *must* be implemented in your adapter:

- ▶ `agentInit()`
- ▶ `getVersion()`
- ▶ `getConnectorBOHandlerForBO()`
- ▶ `terminate()`

Data handlers and name handlers

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 WebSphere business integration broker and some external process.

A data handler calls a *name handler* to extract the name of a business object definition from the serialized data. This task is needed during string-to-business-object conversion when the caller of the data handler does not pass a business object to be populated with the serialized data. In this case, the data handler must create the business object before it can populate it. To create the business object, the data handler must know the name of the associated business object definition. It is the name handler that obtains this business object name.

This chapter describes the design decisions, development process, and implementation for data handlers and name handlers.

5.1 Design decisions for data handlers

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, there are several delivered data handlers. In addition, you can create custom data handlers to handle conversion between your own native format. The adapter can then call the appropriate data handler to convert the data based on the MIME type of the serialized data.

Before you begin to write a custom data handler, we recommend that you understand clearly:

- ▶ The data format of the files that the data handler will be converting.
- ▶ The business object model.
- ▶ How to:
 - Extract values from a business object instance.
 - Build a business object instance and populate it with values from a file.

A data handler is implemented in a Java class named `DataHandler`. This class is an abstract class that the data-handler developer extends to implement a data handler instance.

5.1.1 Determining whether the data is metadata-driven

Metadata is data about the business object that is stored in business object definitions. Metadata in a business object definition provides information that describes the data in an instance of the business object. In general, business object metadata includes the structure of the business object, the settings of its attribute properties, and the content of its application-specific information. It also provides instructions about how to process the data.

A metadata-driven data handler handles each business object that it supports based on metadata that is encoded in the business object definition, rather than on information hard-coded in the data handler. Therefore, a data handler can process new or modified business objects without requiring modifications to the data handler code.

One of the design decisions you need to make is whether your data handler will use metaobjects to initialize its configuration. You should consider the following when deciding whether to use metaobjects:

Metaobjects allow a data handler to be configured dynamically. This design strategy creates a more flexible data handler, one that can be configured based on the context in which it is called. To call a data handler that uses metaobjects, the caller passes the data handler's associated MIME type into the `createHandler()` method. When called with a MIME type, the `createHandler()` method initializes a newly instantiated data handler with the configuration information in the child metaobject.

There is overhead that is associated with the accessing and searching of metaobjects. Your data handler might want to avoid metaobjects if its configuration information does not change (it can be hard-coded) or if it converts data that does not have an associated MIME type. To call a data handler that does not use metaobjects, the caller passes only the data handler's class name into the `createHandler()` method. When called with a class name, the `createHandler()` method instantiates a data handler of that class. It does not search for associated metaobjects. If you design your custom data handler to use metaobjects, you need to create these metaobjects as part of your data handler implementation.

For your custom data handler to be metadata-driven, it must specify information dynamically that identifies the data handler to use.

5.2 The development process

To develop a custom data handler, you code the data handler source file and complete other tasks, such as developing a metaobject for the data handler. To create a custom data handler:

1. Design the data handler, based on the format of the serialized data and structure of the business objects it converts.
2. Create a class that extends the `com.crossworlds.DataHandlers.DataHandler` class.
3. Implement the abstract methods that convert data between specific data format and business objects.
4. Compile the class and add it to the `CustDataHandler.jar` file.
5. Create the data-handler metaobjects.
6. Develop business object definitions that conform to the requirements of the data handler, as well as to the requirements of the caller.

5.3 Creating the custom data handler

To create a custom data handler, you extend the data-handler base class (`DataHandler`) to include your own data-handler class. The `DataHandler` class contains methods that perform the conversions (string-to-business-object and business-object-to-string) as well as utility methods to assist in development.

The ADK contains stub code and makefiles for a custom data handler. The stub file contains Java code that defines an empty class listing all the methods that you must implement. You can use the stub file as a template to generate a custom data handler.

To create a data handler source file using the stub file:

1. Copy the `StubDataHandler.java` file and rename it so that its name matches the name of the data-handler class that it defines.

This file includes import statements that import the data handler package `com.crossworlds.DataHandlers`. It also imports some classes from the Java Connector Development Kit.

2. Change the `StubDataHandler` keyword to the name of the class that implements your custom data handler.

For example, the following line extends the `DataHandler` class to create a custom data-handler class called `ItsoDataHandler`:

```
public class ItsoDataHandler extends DataHandler
```

To develop a data handler, implement the following methods of the `DataHandler` class:

- ▶ The abstract `DataHandler` methods (required)
- ▶ The public `DataHandler` methods (optional)

Note: If the caller reuses a cached instance of the `DataHandler` class over multiple threads, you might need to make the class thread-safe. To determine whether this is required, see the *Connector Development Guide for Java* for more details about the threading model.

5.4 Converting data

A data handler is responsible for converting business objects into serialized data and for converting serialized data into business objects. This serialized data is in an application-readable format (string or input stream).

The data handler base class provides the following abstract methods that you must implement in the `DataHandler` class for your custom data handler:

► String-to-business-object conversion:

`getB0()` - abstract Converts serialized data access through a `Reader` object to a business object.

► Business-object-to-string conversion:

`getStreamFromB0()` Converts a business object to an `InputStream` object.

`getStringFromB0` Converts a business object to a `String` object.

`getByteArrayFromB0()` Converts a business object to a byte array.

Note: For more information, see the *IBM WebSphere Business Integration Data Handler Guide* at:

<http://www.ibm.com/software/integration/wbiadapters/library/infocenter/index.html>

5.5 Name handler development overview

To create a custom name handler:

1. Create a class that extends the `NameHandler` class.
2. Implement the abstract `getB0Name()` method that reads the serialized data and returns the business object name.
3. Compile the class and add it to the `DataHandlers\CustDataHandler.jar` file.
4. Set the default value of the `NameHandler` class attribute in the metaobject for the data handler.

5.6 Implementing a name handler

To implement a custom name handler, you extend the name-handler base class (NameHandler) to include your own name-handler class. The NameHandler class contains the method to extract the name of a business object from serialized data. The package for this name-handler base class is:

```
com.crossworlds.DataHandlers.NameHandler
```

To derive a name-handler class:

1. Create a name-handler class that extends the NameHandler class.
2. Ensure that the name-handler class file imports the classes of the NameHandler package: import.
3. Implement the `getBOName()` method, which is the abstract method in the NameHandler class. Name handlers implement the `getBOName()` method, which reads the serialized data and resolves the BOName based on the contents of the data.

The `getBOName()` method creates an instance of a name handler to extract the name of the business object definition from the serialized data. It instantiates this name-handler object based on the value of the `NameHandlerClass` metaobject attribute. The name handler builds the business object name based on the contents of a message.

The `getBOName()` method has the following forms:

- ▶ The first form returns the business object name based on the `BOPrefix` metaobject attribute (if one exists) as well as on the return value from the NameHandler class.
- ▶ The second form creates a Reader object from the String and then calls the first form.
- ▶ The third form creates a Reader object from the InputStream and then calls the first form.

5.7 Name handler requirements

Name handlers are used to resolve the business object name from the serialized data stream. This is only applicable when the adapters receive information from the applications. (For more information about events, see Chapter 8, “Asynchronous event processing and notification” on page 89.) The request processing does not require the use of name handlers, because the information is coming from a broker. This information should include the business object information. The broker should know which business object it is sending to the adapter.

5.8 Reuse

The data handler determines which name handler to invoke by using the value of the `NameHandlerClass` attribute that is stored in the data-handler metaobject. You can use a name handler lookup file to specify parameters that control the behavior of the name handler. When the name handler has extracted the required value from the serialized data, a lookup file provides the name of the business object that matches the value.

5.9 Configuration requirements

To tell a data handler to use a custom name handler, you must set the Default Value property of a metaobject attribute to the full class name. The data handler can then obtain the class name at run-time from one of its configuration options. By default, this metaobject attribute is called `NameHandlerClass`. (As an example, the child metaobject associated with both the XML and EDI data handlers include this attribute.)

The delivered default value for this attribute specifies the name of the default name handler class. To have a data handler use a custom name handler, make sure that you update the Default Value property for the `NameHandlerClass` attribute in the child metaobject that is associated with the data handler that you extend.

Business object handlers

This chapter presents information about how to provide *request processing* in a connector. Request processing implements a mechanism to receive requests, in the form of request business objects, from an integration broker and to initiate the appropriate changes in the application business entities. The mechanism for implementing request processing is a *business object handler*, which contains methods that interact with an application to transform request business objects into requests for application operations.

6.1 Overview of request processing

In addition to detecting application events, another role of a connector is to respond to requests from the integration broker. A connector receives a request business object from an integration broker when the broker requests a change to the connector's application or needs information from the connector's application. In general, connectors perform create, retrieve, and update operations on application data in response to requests from a collaboration. Depending on the application's policies, the connector might also support delete operations. This role is called *request processing*.

Request processing involves the following steps:

1. An integration broker initiates request processing by sending a request to the Adapter Framework. This request is in the form of a business object (called the *request business object*) and a verb.
2. The Adapter Framework has the task of determining which business object handler in the application-specific component should process the request business object.
3. The Adapter Framework passes the request business object to the business object handler defined for it in its business object definition. The Adapter Framework does this by calling the `doVerbFor()` method that is defined in the business object class and passing in the request business object. The business object handler then processes the business object, converting it to one or more application requests.
4. When the business object handler completes the interaction with the application, it returns a return-status descriptor and possibly a response business object to the Adapter Framework.

A business object handler is the Java class that is responsible for transforming the request business object into a request for the appropriate application operation. Each business object definition refers to a business object handler, which contains a set of methods to perform the tasks for the verbs that the business object definition supports. An application-specific component includes one or more business object handlers to perform tasks for the verbs in the connector's supported business objects.

Depending on the active verb, a business object handler can insert the data associated with a business object into an application, update an object, retrieve the object, delete it, or perform another task.

The Adapter Framework obtains the correct business object handler in the following way:

1. When the connector starts up, the Adapter Framework receives from the Connector Controller the list of business objects that the connector supports.
2. The Adapter Framework calls the `getConnectorBOHandlerForBO()` method (defined in the connector base class) to instantiate one or more business object handlers.
3. For each supported business object, the `getConnectorBOHandlerForBO()` method returns a reference to a business object handler, and this reference is stored in the business object definition in the memory of the connector process.

All conversions between business objects and application operations take place within the business object handler (or handlers).

Defining and coding of business object handlers is one of the primary tasks in connector development. The way to implement a business object handler depends on the API that you are using and how this interface exposes application entities. To determine how many business object handlers your connector requires, you need to take a look at the application with which the connector will interact and consider the following:

- ▶ If the application is form-based, table-based, or object-based and has a standard access method across entities, you might be able to design business objects that store information about application entities. The business object handler can process the application entities in a metadata-driven business object handler. You can derive one generic business object handler class to implement a metadata-driven business object handler, which handles processing of all business objects.
- ▶ If the application has different access methods for different kinds of entities, some or all of the application entities might require individual business object handlers. You can:
 - Derive a generic business object handler class to implement a metadata-driven business object handler for some business objects, and separate business object handler classes to implement business object handlers for other business objects.
 - Derive multiple business object handler classes, one for each business object definition that the connector supports.

Another consideration in the design of a business object handler is whether you need to have separate processing for certain verbs of the business object. If some verb (or verbs) requires special processing, you can create a custom business object handler for the verb.

6.2 Metadata

In addition to its structure and attributes, a business object definition can contain application-specific information, which can provide processing instructions or information about how the business object is represented in the application. Such information is called *metadata*.

Metadata can include any information that the connector needs in its interactions with the application. For example, if a business object definition for a table-based application includes metadata that provides the application table and column names, the connector can locate requested data using this information, and the application column names do not need to be encoded in the connector. Because the connector has access to its supported business object definitions at run-time, it can use the metadata in the business object definition to dynamically determine how to process a particular business object.

Depending on the application and its programming interface (API), you might design a connector and its business objects based on the ability to support the use of metadata, as shown in Table 6-1.

Table 6-1 Connector support for metadata

Connector's use of metadata	Business object handler required
Entirely driven by the processing instructions in the metadata of its business object definitions	One generic metadata-driven business object handler
Partially driven by the metadata in its business object definitions	One partially metadata-driven business object handler
Cannot use metadata	Separate business object handler for each business object that does not use metadata

Although some application interfaces have constraints that restrict the use of metadata in connector and business object design, a worthwhile goal for connector development is to make the connector as metadata-driven as possible. The following sections discuss the advantages and disadvantages of the approaches that are shown in Table 6-1.

6.2.1 Metadata-driven connectors

To be able to support metadata-driven design, the API must be able to specify what objects in the application are to be acted upon. In general, this means that you can use the business object metadata to provide information about the application entity to be acted upon and the attribute data as the values for that object. A *metadata-driven connector* can then use the business object values and the metadata (the application-specific information that the business object

definition contains) to build the appropriate application function calls or SQL statements to access the entity. The function calls perform the required changes in the application for the business object and verb the connector is processing.

Applications that are based on forms, tables, or objects are well suited for metadata-driven connectors. For example, applications that are forms-based consist of named forms. Programmatic interaction with a forms-based application consists of opening a form, reading or writing fields on the form, and then saving or dismissing the form. The connector for such an application can be driven directly by the business object definitions that the connector supports.

The main benefit to a metadata-driven connector is that the connector can use one generic business object handler for all business objects. In this approach, the business object definition contains all the information that the connector needs to process the business object. Because the business object itself contains the application-specific information, the connector can handle new or modified business objects without requiring modifications to the connector source code. The connector can be written in a generic manner, with a single metadata-driven business object handler, which does not contain hard coded logic for processing-specific business objects.

Note: Business object names should not have semantic value to the metadata-driven connector. The connector should process identically two business objects with the same structure, data, and application-specific information with different names.

Because a metadata-driven connector derives its processing instructions from its application-specific business objects, the business objects must be designed with this type of processing in mind. This approach to connector and business object design provides flexibility and easy extensibility, but it requires more planning in the design phase. When connectors are designed to work with business object metadata, the business object itself can be changed without requiring corresponding changes in the connector.

6.2.2 Partially metadata-driven connectors

We encourage you to use the metadata approach for designing connectors and application-specific business object definitions. However, some applications might not be suited for this approach. APIs that are specific for each entity in an application make it more difficult to write a metadata-driven connector. Often the issue is that the call itself differs between objects in some structural way, rather than just in the name of the method or the data that is passed.

Sometimes, you can still drive a connector with metadata, though this metadata does not contain the actual processing instructions. This partially metadata-driven connector can use the metadata in the business object definition or attributes to help determine what processing to perform. For example, an application that has a large amount of business logic embedded in its user interface might have restrictions about how an external program, such as a connector, can get information into and out of its database. In some cases, it might be necessary to provide an extension to the application using the application environment and application programming interface. You might need to add object-specific modules to the application to handle the processing for each business object. The application might require the use of its application environment and interface to ensure that the application business logic is enforced and not bypassed.

Figure 6-1 illustrates an application extension that is responsible for handling requests from the connector. The extension contains separate modules for each business object supported by the connector.

In this case, the business object and attribute application-specific information can still contain metadata for the connector. This metadata specifies the name of the module or API call needed to perform operations for the business object in the application. The connector can still be implemented with a single business object handler, but it is a *partially metadata-driven business object handler*, because this metadata does not contain the processing instructions.

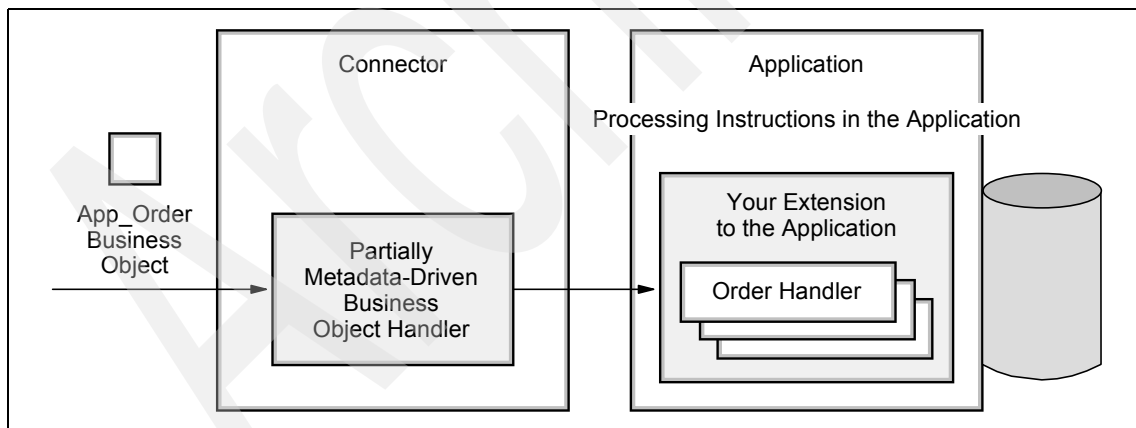


Figure 6-1 Partially metadata-driven connectors interacting with the application extension

The partially metadata-driven connector still uses just one business object handler. However, unlike with a metadata-driven connector, there is coding to do when new business objects are created for the connector. In this case, new

object functions must be written and added to the application, but the connector does not need to be recorded or recompiled.

6.2.3 Connectors that do not use metadata

If the API does not provide the ability to specify what entities in the application are to be acted upon, the connector cannot use metadata to support a single business object handler. Instead, it must provide multiple business object handlers, one for each business object the connector supports. In this approach, each business object handler contains specific logic and code to process a particular business object.

In Figure 6-2, the connector has multiple, object-specific business object handlers. When the connector receives a business object, it calls the appropriate business object handler for that business object.

The drawback of this non-metadata approach is that when a business object is changed or a new business object is added, this type of connector must be recorded to handle the new or changed business object.

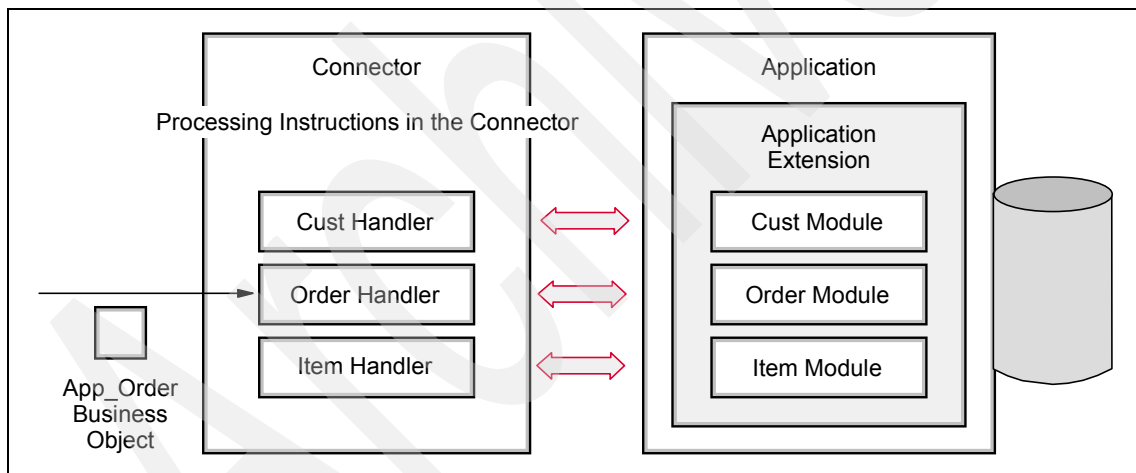


Figure 6-2 Request processing with connector non-metadata-driven

6.3 One generic business object handler for all business objects

If the API is suitable for a metadata-driven connector and if you design business object definitions to include metadata, you can implement a metadata-driven business object handler. This business object handler uses the metadata to process all requests. A business object handler can be completely metadata-driven if the application is consistent in its design, and the metadata follows a consistent syntax for each supported business object.

Business object definitions have specific locations for different types of application-specific data. For example, business object attributes have a set of properties, such as Key, Foreign Key, Required, and Type, that provide the business object handler with information that it can use to drive business object processing. In addition, the `AppSpecificInfo` property can provide the business object handler with application-specific information, which can specify how to access data in the application and how to process application entities.

The `AppSpecificInfo` property is available for the business object definition, attributes, and verbs. You can use the `AppSpecificInfo` property to:

- ▶ Discover the operation to perform on the application entity that examines the verb which is passed in the business object.
- ▶ Obtain the application entity name (such as application table or form) that examines the business object metadata.
- ▶ Obtain information about the back-end application properties (such as attributes, column names, or other information) that examine the metadata of the business object attributes.

If a business object definition contains the table name and column names, you do not have to code those names explicitly in the business object handler.

Encoding specific application information in a business object accomplishes the following:

- ▶ One business object handler class can perform all operations for all business objects supported by the connector. You do not have to code a separate business object handler for each supported business object.
- ▶ Changes to a business object definition do not require recoding the connector, as long as the changes conform to existing metadata syntax. Thus, you can add attributes to a business object definition, remove attributes, or reorder attributes without recompiling or recoding the connector.

If information about application entities is encoded consistently in the business object definition, all request business objects can be handled by a single business object handler class in the connector. Also, you need to implement only a single `getConnectorBOHandlerForBO()` method to return the single business object handler and a single `doVerbFor()` method to implement this business object handler. This approach is recommended for connector development because it provides flexibility and automatic support for new business object attributes.

6.4 Multiple business object handlers

For each business object definition that does not encapsulate all the metadata and business logic for an application entity, you need a separate business object handler class. You can derive separate handler classes directly from the business object handler base class, or you can derive a single utility class and derive subclasses as needed. You must then implement the `getConnectorBOHandlerForBO()` method to return business object handler that corresponds to particular business object definitions.

Each business object handler must contain a `doVerbFor()` method. If you implement multiple business object handlers, you must implement a `doVerbFor()` method for each business object handler class. In each `doVerbFor()` method, include code to handle any parts of the application entity or operations on the application entity that the business object definition does not describe.

This approach results in higher maintenance requirements and longer development time than designing a single business object handler for a metadata-driven adapter.

6.5 Designing the `doVerbFor()` method

For a Java connector, the `CWConnectorBOHandler` class defines the `doVerbFor()` method, which is an abstract method. The `doVerbFor()` method typically follows a basic logic for request processing.

Figure 6-3 on page 58 shows a flow chart of the method's basic logic.

When the Adapter Framework receives a request, it calls the `doVerbFor()` method for the business object handler class that is associated with the business object definition of the request business object. To this `doVerbFor()` method, the Adapter Framework passes the request business object. The tasks that the

doVerbFor() method performs after it has received a request business object from the Adapter Framework are:

- Determines which verb to perform, based on the supported verb in the request business object definition.
- Obtains information from the request business object to build the suitable interaction with the application.

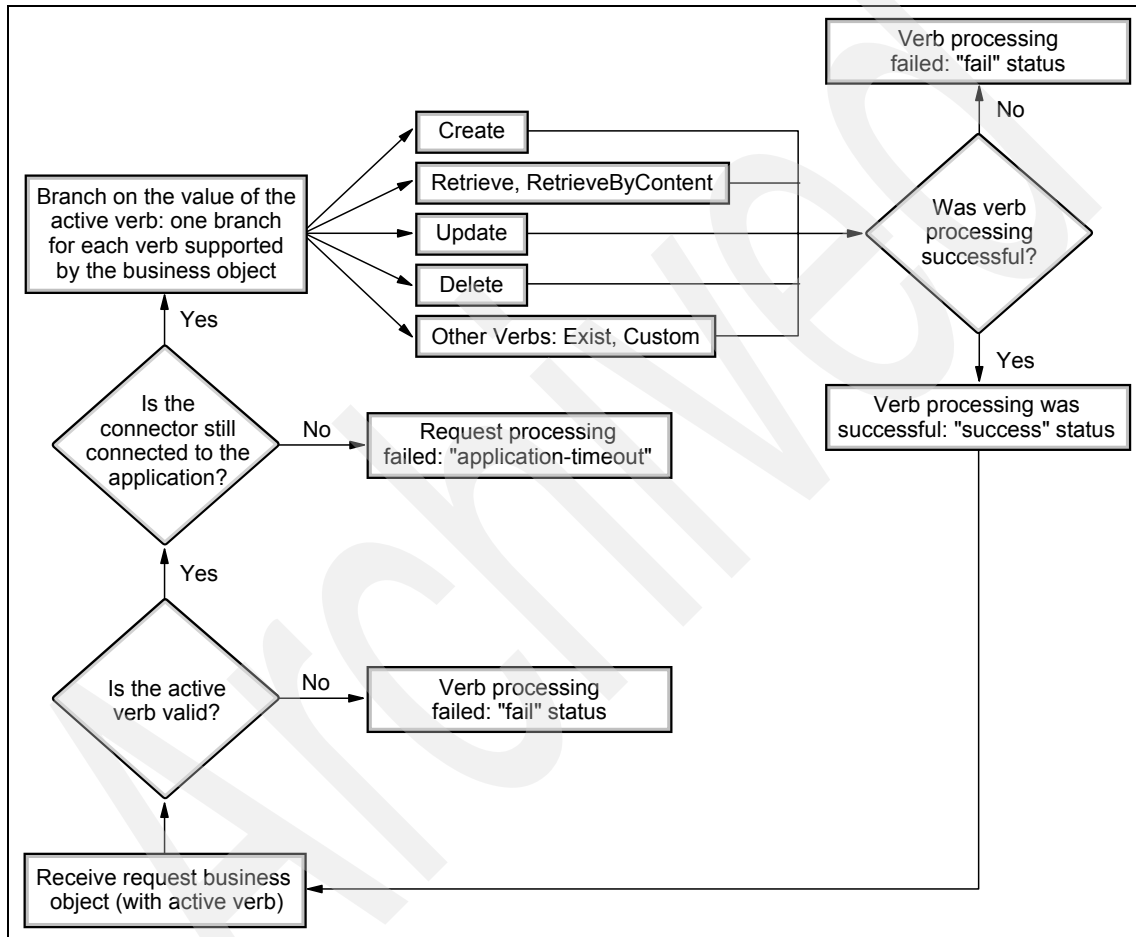


Figure 6-3 Flowchart for basic logic of the `doVerbFor()` method

6.5.1 Performing the verb action

The standard verbs that the IBM WebSphere Business Integration system expects to handle are Create, Retrieve, Update, and Delete. Table 6-1 defines these standard verbs. Your `doVerbFor()` method should implement the verbs that are appropriate for its application.

Table 6-2 Verbs implemented by the `doVerbFor()` method

Verb	Description
Create	Make a new entity in the application.
Retrieve	Using key values, retrieve a complete business object.
Update	Change the value in one or more fields of the application.
Delete	Remove the entity from the application.
RetrieveByValue	Using a non-key value, return a complete business object.
Custom verb	Perform an application-specific operation.

Note: After the Adapter Framework passes a request business object to your connector's `doVerbFor()` method, the `doVerbFor()` method can implement verb processing in any way that is necessary. Your verb processing code is not limited to the suggestions presented here.

Note: When InterChange Server is the integration broker and you design your own collaborations, you can implement any custom verbs that you need. Your collaborations and connectors are not limited to the standard list of verbs.

This basic verb-processing logic consists of the following steps:

1. Gets the verb from the request business object. The `doVerbFor()` method must first retrieve the active verb from the business object with the `getVerb()` method. For a Java connector, the `getVerb()` method is defined in the `CWConnectorBusObj` class.
2. Performs the verb operation. In the business object handler, you can design the `doVerbFor()` method in either of the following ways:
 - Implements verb processing for each supported verb directly within the `doVerbFor()` method. You can modularize the verb processing so that each verb operation is implemented in a separate verb method called from `doVerbFor()`. The method should also take appropriate action if the verb is

not a supported verb by returning a message in the return-status descriptor and a “fail” status.

- Handles all verb processing in the same method using a metadata-driven `doVerbFor()` method.

6.5.2 Handling the Create verb

When the business object handler obtains a Create verb from the request business object, it must ensure that a new application entity, whose type is indicated by the business object definition, is created as follows:

- ▶ For a flat business object, the Create verb indicates that the specified entity must be created.
- ▶ For a hierarchical business object, the Create verb indicates that one or more application entities (to match the entire business object) must be created.

The business object handler must set all the values in the new application entities to the attribute values in the request business object. To ensure that all required attributes in the request business object have values assigned, you can call the `initAndValidateAttributes()` method, which assigns the attribute's default value to each required attribute that does not have its value set (when the `UseDefaults` connector configuration property is set to true). The `initAndValidateAttributes()` method is defined in the `CWConnectorUtil` class. Call the `initAndValidateAttributes()` method before performing the Create operation in the application.

Note: For a table-based application, the entire application entity must be created in the application database, usually as a new row to the database table that is associated with the business object definition of the request business object.

Standard processing for the Create verb

The following steps outline the standard processing for a Create verb:

1. Creates the application entity corresponding to the top-level business object.
2. Handles the primary key or keys for the application entity:
 - If the application generates its own primary key (or keys), gets these key values for insertion in the top-level business object.
 - If the application does not generate its own primary key (or keys), inserts the key values from the request business object into the appropriate key column (or columns) of the application entity.

3. Sets foreign key attributes in any first-level child business objects to the value of the top-level primary key.
4. Recursively creates the application entities that correspond to the first-level child business objects, and continues recursively creating all child business objects at all subsequent levels in the business object hierarchy.

Note: The way that the main method provides primary key values to the submethod can vary. For example, the main Create method might pass the parent business object to the submethod, and the submethod can then retrieve the primary key from the parent business object to set the foreign key in the child business object. Alternatively, the main method might traverse the parent object, find first-level children, set the foreign key attributes in the child business objects, and then call the submethod on each child.

Outcome status for Create verb processing

Table 6-3 shows the outcome status that Create operation should return.

Table 6-3 Possible outcome status for Create verb processing

Create condition	Outcome status
If the Create operation is successful and the application generates new key values, the connector returns the “Value Changed” outcome status to indicate that the connector has changed the business object.	VALCHANGE
If the Create operation is successful, and the application does not generate new key values, the connector can simply return “Success”.	SUCCEED
If the application entity already exists, the connector can return either of the following status: <ul style="list-style-type: none"> ► Fails the Create operation. ► Returns an outcome status that indicates the application entity already exists. 	FAIL VALDUPLES
If the Create operation fails, the connector returns the Fail outcome status.	FAIL

6.5.3 Handling the Retrieve verb

When the business object handler obtains a Retrieve verb from the request business object, it must ensure that an existing application entity, whose type is indicated by the business object definition, is retrieved as follows:

- ▶ For a flat business object, the Retrieve verb indicates that the specified entity is retrieved by its key values. The verb operation returns a business object that contains the current values for the application entity.
- ▶ For a hierarchical business object, the Retrieve verb indicates that one or more application entities (to match the entire business object) are retrieved by the key values of the top-level business object. The verb operation returns a business object in which all simple attributes of each business object in the hierarchy match the values of the corresponding entity attributes, and the number of individual business objects in each child business object array matches the number of child entities in the application.

For the Retrieve verb, the business object handler obtains the key value (or values) from the request business object. These key values uniquely identify an application entity. The business object handler then uses these key values to retrieve all the data associated with an application entity. If we have a hierarchical business object, the connector retrieves the entire hierarchical image of the entity, including all child objects.

Important: All business object handlers must implement a `doVerbFor()` method with verb processing for the Retrieve verb. This requirement holds even if your connector will not perform request processing.

Standard processing for the Retrieve verb

The following steps outline the standard processing for a Retrieve verb:

1. Creates a new business object of the same type as the request business object. This new business object is the *response business object*, which will hold the retrieved copy of the request business object.
2. Sets the primary keys in the new top-level business object to the values of the top-level keys in the request business object.
3. Retrieves the application data for the top-level business object and fills the response top-level business object's simple attributes.
4. Retrieves all the application data associated with the top-level entity, and creates and fills child business objects as needed.

Note: By default, the Retrieve method fails if it cannot retrieve application data for all the child objects in a hierarchical business object.

The goal of this approach is to start with the top-level business object and rebuild the complete business object hierarchy. This type of implementation ensures that all children in the request business object that are no longer in the database are removed and are not passed back in the response business object. This implementation also ensures that the hierarchical response business object exactly matches the database state of the application entity. At each level, the Retrieve operation rebuilds the request business object so that it accurately reflects the current database representation of the entity.

Retrieving child objects

To retrieve entities that are associated with the top-level entity, the Retrieve operation might be able to use the API:

- ▶ Ideally, the API will navigate the relationships between application entities and return all related data. The verb operation can then encapsulate the related data as child business objects.
- ▶ If there are not any APIs that provide information about associated entities, you might need to access the application (for example, with generated SQL statements) to retrieve related data. The SQL statements might use foreign keys to navigate through application tables.

Tip: If the attribute application-specific information in the business object definition contains information about foreign keys, the verb operation can use this information to generate a command to access the application (such as SQL statements).

For example, if an attribute contains a child business object, it might use application-specific information, as in the following example:

```
fkey=child_business_object_name.attribute_name
```

Using this application-specific information, the verb operation can find the table name and the column name for the foreign key that is related to the child business object.

We might use the same approach from the child business object perspective and set up the application-specific information into the child attribute that is related to the foreign key column of the child table.

If a Retrieve operation returns multiple rows, each row becomes a child business object. The verb operation might process retrieved rows as follows:

1. For each row, creates a new child business object of the correct type.
2. Sets attributes in the new child business object based on the values that a SELECT statement (or API call) returns for the associated row.
3. Recursively retrieves all children of the child business object, creating the business object and setting the attributes for each one.
4. Inserts the array of child business objects into the multiple-cardinality attribute in the parent business object.

Configuring a Retrieve to ignore missing child objects

By default, the Retrieve operation should fail if it cannot retrieve application data for the complete set of child business objects in a hierarchical business object. However, you can implement the verb operation so that the behavior of the connector is configurable when one or more of the children in a business object are not found in the application.

To do this, define a connector-specific configuration property that is named `IgnoreMissingChildObject`, whose values are `True` and `False`. The Retrieve operation obtains the value of this property to determine how to handle missing child business objects. When the property is `True`, the Retrieve operation should simply move on to the next child in the array if it fails to find a child business object. In this case, the verb operation should return `VALCHANGE` if it is successful in retrieving the top-level object, regardless of whether it is successful in retrieving its children.

Outcome status for Retrieve verb processing

The Retrieve operation should return one of the outcome status values as shown in Table 6-4.

Table 6-4 Possible outcome status for the Retrieve operation

Retrieve condition	Outcome status
When the Retrieve operation is successful, it returns the "Value Changed" outcome status to indicate that the connector has changed the business object.	VALCHANGE
If the <code>IgnoreMissingChildObject</code> connector property is <code>True</code> , the Retrieve operation returns the "Value Changed" outcome status for the business object if it is successful in retrieving the top-level object, regardless of whether it is successful in retrieving its children.	VALCHANGE

Retrieve condition	Outcome status
If the entity that the business object represents does not exist in the application, the connector returns a special outcome status instead of “Fail”.	BO_DOES_NOT_EXIST
<p>If the request business object does not provide a key for the top-level business object, the Retrieve operation can take either of the following actions:</p> <ul style="list-style-type: none"> ► Fills a return-status descriptor with information about the cause of Request failure and returns a “Fail” outcome status. ► Calls the RetrieveByContent method to retrieve using the content of the top-level business object. 	FAIL

6.5.4 Handling the RetrieveByContent verb

An integration broker might need to retrieve a business object for which it has a set of attribute values without having the key attribute (or attributes) that uniquely identifies an application entity. Such a retrieve is called *retrieve by non-key values* or *retrieve by content*.

As an example, if a business object handler receives a Customer business object with the verb RetrieveByContent and with the non-key attributes Name and City set to Smith and San Diego, the RetrieveByContent operation can attempt to retrieve a customer entity that matches the values of the Name and City attributes.

When the business object handler obtains a RetrieveByContent verb from the request business object, it must ensure that an existing application entity, whose type is indicated by the business object definition, is retrieved as follows:

- For a flat business object, the RetrieveByContent verb indicates that the specified entity is retrieved by its non-key values. The verb operation returns a business object that contains the current values for the application entity.
- For a hierarchical business object, the RetrieveByContent verb indicates that one or more application entities (to match the entire business object) are retrieved by the non-key values of the top-level business object. The verb operation returns a business object in which all simple attributes of each business object in the hierarchy match the values of the corresponding entity attributes, and the number of individual business objects in each child business object array matches the number of child entities in the application.

Outcome status for RetrieveByContent processing

The RetrieveByContent operation should return one of the outcome status values as shown in Table 6-5.

Table 6-5 Possible outcome status for the RetrieveByContent operation

RetrieveByContent condition	Outcome status
If the RetrieveByContent operation finds a single entity that matches the query, it returns a “Value Changed” outcome status.	VALCHANGE
If the IgnoreMissingChildObject is True, the RetrieveByContent operation returns the “Value Changed” outcome status for the business object if it is successful in retrieving the top-level object, regardless of whether it is successful in retrieving its children.	VALCHANGE
<p>If the RetrieveByContent operation finds multiple entries that match the query, it:</p> <ul style="list-style-type: none">▶ Retrieves only the first occurrence of the match; this business object is returned to the Adapter Framework through the request business object parameter.▶ Fills a return-status descriptor with further information about the search.▶ Returns a status of “Multiple Hits” to notify the Adapter Framework that there are additional records that match the specification.	MULTIPLE_HITS
<p>If the RetrieveByContent operation does not find matches for retrieve by non-key values, it:</p> <ul style="list-style-type: none">▶ Fills a return-status descriptor containing additional information about the cause of the RetrieveByContent error.▶ Returns a “RetrieveByContent Failed” outcome status.	RETRIEVEBYCONTENT_FAILED

6.6 Handling the Update verb

When the business object handler obtains an Update verb from the request business object, it must ensure that an existing application entity, whose type is indicated by the business object definition, is updated as follows:

- ▶ For a flat business object, the Update verb indicates that the data in the specified entity must be modified as necessary until the application entity exactly matches the request business object.
- ▶ For a hierarchical business object, the Update verb indicates that updates to the application entity must be made to match the entire business object

hierarchy. To do this, the connector might need to create, update, and delete application entities as follows:

- If child entities exist in the application, they are modified as needed.
- Any child business objects contained in the hierarchical business object that do not have corresponding entities in the application are added to the application.
- Any child entities that exist in the application but that are not contained in the business object are deleted from the application.

Standard processing for the Update verb

The following steps outline the standard processing for an Update verb:

1. Creates a new business object of the same type as the request business object. This new business object is the response business object, which will hold the retrieved copy of the request business object.
2. Retrieves a copy of the request business object from the application. Recursively retrieves the data for the entire entity from the application using the primary keys from the request business object:
 - For a flat business object, retrieves the single application entity.
 - For a hierarchical business object, uses the Retrieve operation to descend into the application business object, expanding all paths in the business object hierarchy.
3. Places the retrieved data in the response business object. This response business object is now a representation of the current state of the entity in the application. The Update operation can now compare the two hierarchical business objects and update the application entity appropriately.
4. Updates the simple attributes in the application entity to correspond to the top-level source business object.
5. Compares the response business object (that were created in step 2) with the request business object. Performs this comparison down to the lowest level of the business object hierarchy. Recursively updates the children of the top-level business object, following these rules:
 - If a child business object is present in both the response business object and the request business object, recursively updates the child by performing the Update operation.
 - If a child business object is present in the request business object, but not in the response business object, recursively creates the child by performing the Create operation.
 - If a child business object is not present in the request business object, but is present in the response business object, recursively deletes the child

using either the Delete operation (physical) or a logical delete, depending on the functionality of the connector and the application.

Note: Only the existence or non-existence of the child objects are compared, not the attributes of the child business objects.

Note: The Update operation should fail if an application entity does not exist for any foreign key (Foreign Key is set to true) that is referenced in the request business object. The connector should verify that the foreign key is a valid key (it references an existing application entity). If the foreign key is invalid, the Update operation should fail. A foreign key is assumed to be present in the application, and the connector should never try to create an application object marked as a foreign key.

Outcome status for Update verb processing

The Update operation should return one of the outcome status values as shown in Table 6-6.

Table 6-6 Possible outcome status for the Update operation

Update condition	Outcome status
If the application entity exists, the Update operation returns a “Success” outcome status.	SUCCESS
If a row or entity does not exist, the Update operation: <ul style="list-style-type: none">▶ Creates the application entity.▶ Returns the “Value Changed” outcome status to indicate that the connector has changed the business object.	VALCHANGE
If the Update operation is unable to create the application entity, it: <ul style="list-style-type: none">▶ Fills a return-status descriptor with information about the cause of the update error.▶ Returns a “Fail” outcome status.	FAIL
If any object identified as a foreign key is missing from the application, the Update operation: <ul style="list-style-type: none">▶ Fills a return-status descriptor with information about the cause of the update error.▶ Returns a “Fail” outcome status.	FAIL

6.6.1 Handling the Delete verb

When the business object handler obtains a Delete verb from the request business object, it must ensure that a physical delete is performed (that is, the application deletes the application entity whose type is indicated by the business object definition) as follows:

- ▶ For a flat business object, the Delete verb indicates that the specified entity must be deleted.
- ▶ For a hierarchical business object, the Delete verb indicates that the top-level business object must be deleted. Depending on the application policies, it might delete associated entities representing child business objects.

Standard processing for the Delete verb

The following steps outline the standard processing for a Delete verb:

1. Performs a recursive retrieve on the request business object to get all data in the application that is associated with the top-level business object.
2. Performs a recursive delete on the entities represented by the request business object, starting from the lowest level entities and ascending to the top-level entity.

Note: Delete operations might be limited by application functionality. For example, cascading deletes might not always be the desired operation. If you are using an API, it might complete the delete operation automatically. If you are not using an API, you might need to determine whether the connector should delete child entities in the application. If a child entity is referenced by other entities, it might not be appropriate to delete it.

Outcome status for Delete verb processing

The Delete operation should return one of the outcome status values as shown in Table 6-7.

Table 6-7 Possible outcome status for Delete verb processing

Delete condition	Outcome status
InterChange Server only: In most cases, the connector returns a “Value Changed” outcome status to enable the system to clean up the relationship tables after a delete operation.	VALCHANGE
All integration brokers: If the Delete operation is unsuccessful, it: <ul style="list-style-type: none">► Fills a return-status descriptor with additional information about the cause of the delete error.► Returns a “Fail” outcome status.	FAIL

6.7 Business object processing and cardinality

A business object handler’s role is to deconstruct a request business object, process the request, and perform the requested operation in the application. To do this, a business object handler extracts verb and attribute information from the request business object and generates an API call, SQL statement, or other type of application interaction to perform the operation.

Basic business object processing involves extracting metadata from the business object’s application-specific information (if it exists) and accessing the attribute values. The actions to take on the attribute value depend on whether the business object is *flat* or *hierarchical*. This section provides an overview of how a business object handler can process these types of business objects.

6.7.1 Processing flat business objects

If a business object does *not* contain any other business objects (called child business objects), it is called a *flat business object*. All the attributes in a flat business object are simple attributes (that is, each attribute contains an actual value, not a reference to another business object).

After the verb operation has accessed the information that it needs within the business object definition, it often needs to access information about attributes. Attribute properties include the cardinality, key or foreign key designation, and maximum length. For example, the Create method might need to obtain the attribute’s application-specific information. A connector business object handler

typically uses the attribute properties to decide how to process the attribute value.

For a *database-oriented application*, the business object handler that handles a flat business object, deconstructing a business object includes the following steps:

1. Extracts the table and column names from the application-specific information in the business object definition.
2. Extracts the values of the attributes from the business object instance.

As Figure 6-4 shows, the Customer business object definition is designed for a metadata-driven connector. Its business object definition includes application-specific information that the verb operation uses to locate the application entity upon which operates.

Important: Application-specific information is also used to store information about foreign keys and other types of relationships between entities in the application database. A metadata-driven connector can use this information to build an SQL statement or an API call.

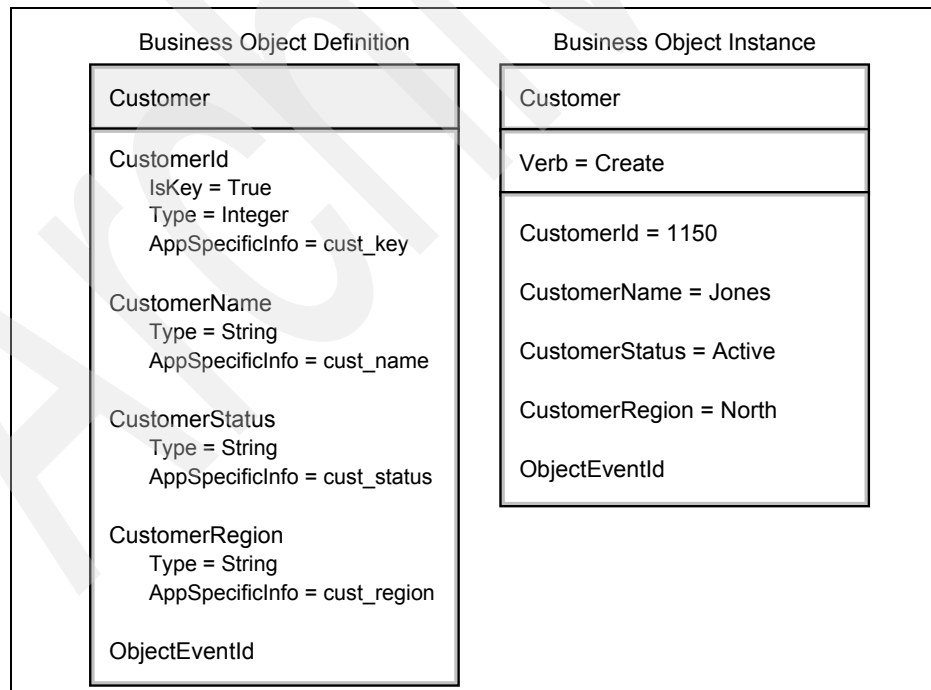


Figure 6-4 Business object definition and business object instance

6.7.2 Processing hierarchical business objects

Business objects are hierarchical when parent business objects can contain child business objects, which can, in turn, contain child business objects, and so on. A hierarchical business object is composed of a *top-level business object*, which is the business object at the very top of the hierarchy, and *child business objects*, which are all business objects under the top-level business object. A child business object is contained in a parent object as an attribute.

There are two types of containment relationships between parent and child business objects:

- ▶ Cardinality 1 containment: The attribute contains a single child business object.
- ▶ Cardinality n containment: The attribute contains several child business objects in a structure called a business object array.

Figure 6-5 shows a typical hierarchical business object. The top-level business object has both cardinality 1 and cardinality n relationships with child business objects.

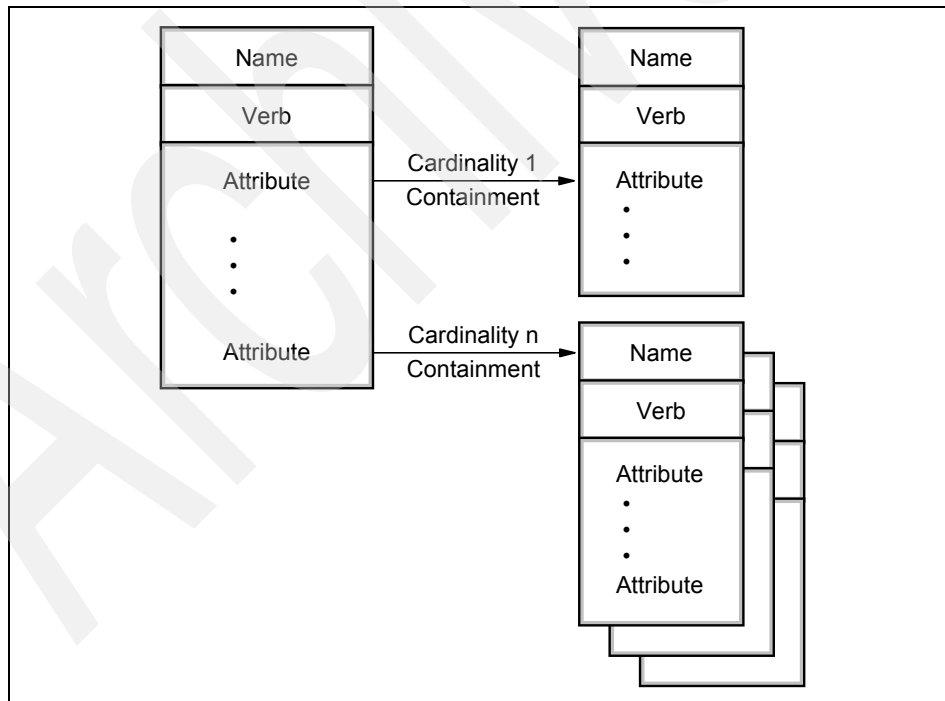


Figure 6-5 Hierarchical business object

In a typical table-based application, relationships between entities are represented by primary keys and foreign keys in the database, where the parent entity contains the primary keys and the child entity contains the foreign keys. An hierarchical business object can be organized in a similar way:

- In a cardinality 1 (single cardinality) type of relationship, each parent business object relates to a single child business object. The child business object typically contains one or more foreign keys whose values are the same as the corresponding primary keys in the parent business object. Although applications might structure the relationships between entities in different ways, a single cardinality relationship for an application that uses foreign keys might be represented as shown in Figure 6-6.
- In a cardinality n (multiple cardinality) type relationship, each parent business object can relate to zero or more child business objects in an array of child business objects. Each child business object within the array contains foreign key attributes whose values are the same as the corresponding values in the primary key attributes of the parent business object. A multiple cardinality relationship might be represented as shown in Figure 6-7 on page 74.

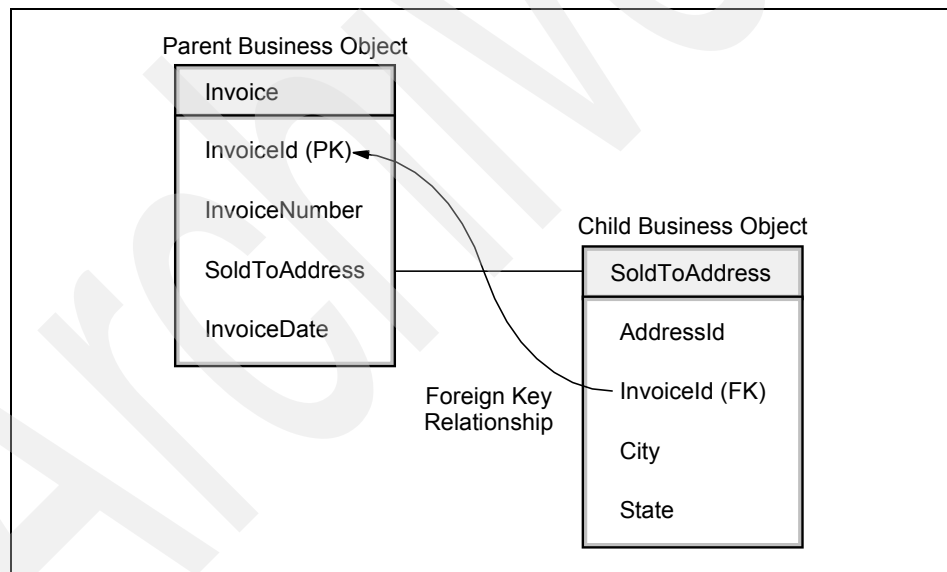


Figure 6-6 Business object with single cardinality

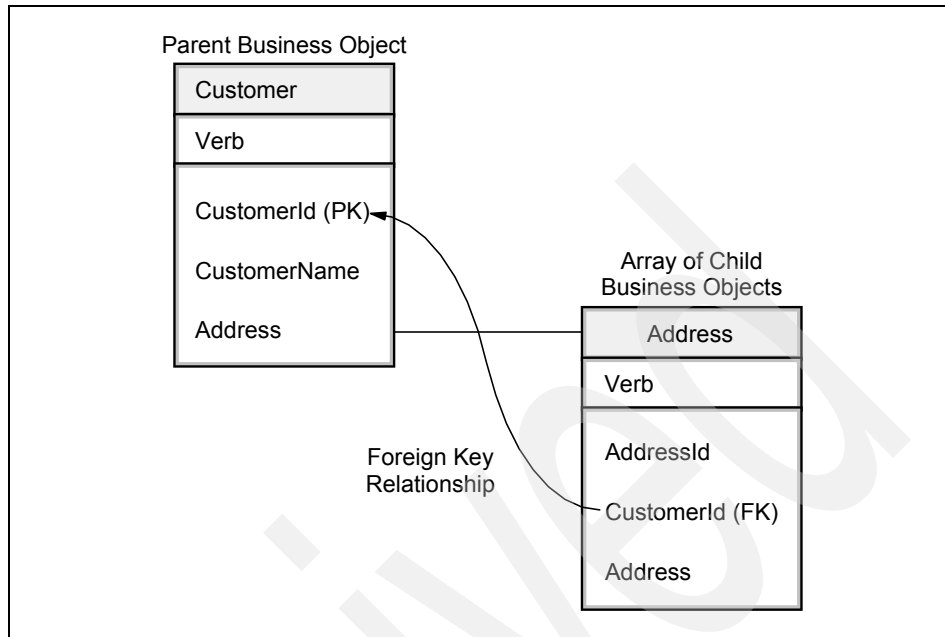


Figure 6-7 Business object with multiple cardinality

As part of its verb processing, the `doVerbFor()` method needs to handle any hierarchical business objects. To process a hierarchical business object, the `doVerbFor()` method takes the same basic steps as it does to process a flat business object: It obtains any application-specific information and then accesses the attribute. However, if the attribute contains a child business object, the `doVerbFor()` method must take the following steps to access the child business object:

1. Determines whether the attribute type is an object by calling the `isObjectType()` method. The `isObjectType()` method returns True if an attribute is complex, that is, if it contains a business object.
2. When the `doVerbFor()` method finds an attribute contains a business object, it checks the cardinality of the attribute using the `isMultipleCard()` method.

If the attribute has single cardinality (cardinality 1), the method can perform the requested operation on the child. One way to perform an operation on a child business object is to call recursively the `doVerbFor()` method or a verb method on the child object.

However, such a recursive call assumes that the child business object is set as follows:

- ▶ If the verb on a child business object is set, the method should perform the specified operation.
- ▶ If the verb on the child business object is not set, the verb method should set the verb in the child business object to the verb in the top-level business object before calling another method on the child.

If an attribute has multiple cardinality (cardinality n), the attribute contains an array of child business objects. In this case, the connector must access the contents of the array before it can process individual child business objects. From the array, the `doVerbFor()` method can access individual business objects:

- ▶ To access individual business objects, the method can get the number of child business objects in the array with the `getObjectCount()` method and then iterate through the objects.
- ▶ To get an individual child business object, the method can obtain the business object at one element of the array.

After the `doVerbFor()` method has access to a child business object, it can process the child recursively as needed.

Note: A connector should never create arrays for child business objects. An array is always associated with a business object definition when cardinality is n .

6.8 Custom business object handlers

The Adapter Framework calls the `doVerbFor()` method in the `CWConnectorBOHandler` class (which implements the business object handler) for all verbs that a particular business object supports. Therefore, all verbs in a business object are processed in one standard way (although they can initiate different actions within the application). However, if your connector supports a business object that requires different processing for some particular verb, you can create a custom business object handler to handle that verb for the business object.

6.8.1 Creating the class for the custom business object handler

To create a custom business object handler, you must create a class that implements the `CWCustomBOHandler` interface. The `CWCustomBOHandler` interface provides the `doVerbForCustom()` method, which you must implement to define a custom business object handler. To create a custom business object handler class for a Java connector:

1. Create a class that implements the `CWCustomBOHandler` interface.
2. Implement the `doVerbForCustom()` method to define the behavior of the business object handler.

6.8.2 Implementing the `doVerbForCustom()` method

The `doVerbForCustom()` method provides the functionality for the custom business object handler. The Adapter Framework calls the low-level `doVerbFor()` method (defined in the `BOHandlerBase` class) for the appropriate business object handler when it receives a request business object. This low-level `doVerbFor()` method determines which business object handler to call as follows:

- ▶ If the business object's verb has the `CB0H` tag in its application-specific information, it calls the `doVerbForCustom()` method.

The `CB0H` tag specifies the full name (including the package name) of the custom business object handler class, which implements the `CWCustomBOHandlerInterface` interface and its `doVerbForCustom()` method.

If the `CB0H` tag exists, the low-level `doVerbFor()` method tries to create a new instance of the class that this tag specifies. If this instantiation is successful, the low-level `doVerbFor()` method calls the `doVerbForCustom()` method in this class.

- ▶ Otherwise, it calls the `doVerbFor()` method, which the connector developer must implement as part of the business object handler's `CWConnectorBOHandler` class.

The implementation of the `doVerbForCustom()` method must handle the verb processing of the verb for which its class is specified. In this method, you must customize the behavior of the `doVerbForCustom()` method to meet the special processing needs of your business object's verb.

Note: Unlike the `doVerbFor()` method, the `doVerbForCustom()` method is not invoked directly by the Adapter Framework. Instead, the Adapter Framework invokes the low-level `doVerbFor()` method, which in turn, invokes the `doVerbForCustom()` method. Therefore, the `doVerbForCustom()` method cannot include calls to any methods in the `CWConnectorBOHandler` class.

Archived

Connector configuration properties

This chapter describes a connector configuration property. It also explains how to define and set connector configuration properties and how to use connector configuration properties in your adapter.

7.1 Overview of connector configuration property

A *connector configuration property* (which is also referred to as a connector property) allows you to create place holders, which are similar to variables, that the connector can use to access the information that it requires for processing. There are two categories of properties:

- ▶ Standard connector configuration properties
- ▶ Connector-specific configuration properties

7.1.1 Standard connector configuration properties

Standard configuration properties provide information that is typically used by the Adapter Framework. These properties are *usually* common to all connectors and represent the type of behavior that the Adapter Framework enforces. These properties are discussed in 7.2, “Using properties in your adapter” on page 81.

7.1.2 Connector-specific configuration properties

Connector-specific properties, as the name implies, provide the information that is required by a specific connector at run-time. The properties provide a way of changing information or logic within the connector's application-specific component, without having to recode or rebuild the adapter. Application-specific properties can be used to:

- ▶ Hold the value of constants. These values include the name of a server or database, database drivers, event table names, user IDs, and passwords for connecting to an application and for files that the connector needs to read.
- ▶ Set behavior for the connector in particular situations. A configuration property could be set to indicate that a connector should not fail a business object retrieve operation for a business object if the child object of a hierarchical business object is missing. A configuration property might determine whether the application or the connector should create IDs for a new object in a create operation.

Any number of application-specific properties can be set for a connector. After the required properties have been identified, they are defined as part of the connector configuration process (using the Connector Configurator).

7.2 Using properties in your adapter

Connector configuration properties are passed to the connector as part of connector initialization (see Chapter 4, “Agent initialization and termination” on page 33). The application-specific component of the connector retrieves the values of any configuration properties that it requires for initialization, based on the type of connector property.

A connector can use a connector configuration property that has one of the following types:

- ▶ A simple configuration property, which contains only string values (that is, it does not contain any other properties). A single-valued simple property contains only one string value.
- ▶ A hierarchical configuration property, which contains other properties and values. A connector property of this type can contain multiple values.

C++ connectors do not support hierarchical properties.

In previous releases, configuration properties were only single-value and simple. With the latest releases, Java connectors can support hierarchical properties and multiple values.

7.3 Retrieving properties

When developing an adapter, you should take into account the retrieval of the connector configuration properties.

7.3.1 Single-valued simple properties

The Java connector library provides two methods for retrieving the value of a simple connector configuration property.

Both methods are defined in the `CWConnectorUtil` class and function as follows:

- ▶ `getConfigProp()`
Retrieves the value of a specified simple configuration property.
Takes as input a string for the name of the configuration property and returns the value of the property as a Java `String`.
- ▶ `getAllConnectorAgentProperties()`
Retrieves the values of all configuration properties.

Note: If the method retrieves a multiple value property, it only retrieves the first of the connector property values.

Does not require input arguments and returns the value of all configuration properties in a Java Hashtable.

Example 7-1 uses the `get()` method of the `Hashtable` class to retrieve the value of each property.

Example 7-1 Retrieve all properties

```
connectorProperties = CWConnectorUtil.getAllConnectorAgentProperties();

// get Connector Configuration Properties to establish Connection

String connectString = (String)connectorProperties.get("ConnectString"); String
userName = (String)connectorProperties.get("ApplicationUserName"); String
userPassword = (String)connectorProperties.get("ApplicationPassword");

if(connectString == null || connectString.equals(""))
|| userName==null || userPassword==null )
```

7.3.2 Hierarchical properties

A hierarchical connector property can contain the following values:

- ▶ One or more child properties. Each child property can, in turn, contain its own child properties and string values.
- ▶ One or more string values.

A hierarchical property with more than one string value is called a *multi-valued* property. As mention previously, a property with only one string value is called a *single-valued* property.

The Java connector library represents a hierarchical property with the `CWProperty` class. An object of this class is called a connector-property object and it can represent a simple or a hierarchical (single-valued or multi-valued) configuration property.

The metadata for a hierarchical property that a connector-property object provides is shown in Table 7-1 on page 83.

Table 7-1 Connector-property object metadata

Information	Description	CWProperty method
Name Cardinality	The name of the connector property. Indicates the number of values that the property contains single-valued and multi-valued.	getName() getCardinality()
Property type	Indicates whether the property contains any child properties: <ul style="list-style-type: none"> ▶ A simple property contains no child properties, only string values. ▶ A hierarchical property contains one or more child properties. 	getPropType()
Encryption flag	Whether the property value is to be encrypted.	getEncryptionFlag() setEncryptionFlag()

Retrieving the property value is a two-step process.

1. Retrieve the top-level connector-property object for one or all of the configuration properties.
2. Retrieve the required property value from the connector-property object.

Retrieve the top-level connector-property object

You can use either of the two methods shown in Table 7-2 to retrieve the top-level connector-property object. Both methods are defined in the CWConnectorUtil class and function as shown in the table.

Table 7-2 Top-level methods

Method	Description
getHierachicalConfigProp()	Retrieves the top-level connector-property object of a specified hierarchical configuration property. Takes the name of a configuration property as an argument and returns a single CWProperty object that contains the top-level connector-property object.
getAllConfigProperties()	Retrieves the top-level connector-property objects for all configuration properties, regardless of whether the property is simple, hierarchical, or multi-valued. Returns an array of CWProperty objects, each containing a top-level connector-property object for one of the configuration properties.

Retrieve the connector-property value

After you have retrieved the top-level connector-property object, you can retrieve the values from this connector-property object. There are methods provided in the `CWProperty` class for retrieving:

- ▶ Child properties
- ▶ String values

Retrieve child properties

Table 7-3 shows the methods for retrieving child property values.

Table 7-3 Methods for retrieving child property values

CWProperty method	Description
<code>getHierChildProps()</code>	Obtains all child properties of the hierarchical connector property.
<code>getChildPropsWithPrefix()</code>	Obtains all child properties of the hierarchical connector property that has a specified prefix.
<code>getHierChildProp()</code>	Obtains a single specified child property from the hierarchical connector property.

You can use the `hasChildren()` method to determine whether the current connector-property object contains any child properties.

Retrieve string values

Table 7-4 shows the methods for retrieving string values.

Table 7-4 Methods for retrieving string values

CWProperty method	Description
<code>getStringValues()</code>	Obtains all string values of the hierarchical connector property.
<code>getChildPropValue()</code>	Obtains all string values of a specified child property.

You can use the `hasValue()` method to determine whether the current connector-property object contains any strings.

Note: For more information about connector configuration properties, see the *WebSphere Business Integration Adapters Connector Development Guide for Java* and the individual adapter guides.

7.4 Some last thoughts about standard properties

These next few sections are not, strictly speaking, related directly to your adapter development. However, we thought that there were a few things worth keeping in mind about standard properties as an adapter developer.

7.4.1 RepositoryDirectory

Not all connectors make use of all of the standard properties, and property settings can differ from integration broker to integration broker. Standard property dependencies are based on the *RepositoryDirectory* standard configuration property.

The RepositoryDirectory is the location of the repository from which the connector reads the XML schema documents that store the metadata for business object definitions.

When the integration broker is:

- ▶ ICS, this value must be set to <REMOTE>, because the connector obtains this information from the InterChange Server repository.
- ▶ WebSphere Business Integration Message Broker or WebSphere Application Server, this value must be set to <local directory> (that is, the location of the XML schema documents).

7.4.2 DeliveryTransport

This standard property is vitally important to both the behavior and performance of your connector. It specifies the transport mechanism for the delivery of events.

Possible allowable values are:

- ▶ MQ for WebSphere MQ
- ▶ IDL for CORBA IIOP
- ▶ JMS for Java Messaging Service

If the broker type is:

- ▶ ICS, the value of the DeliveryTransport property can be MQ, IDL, or JMS.

The default is IDL.

- ▶ Not ICS (that is, the RepositoryDirectory is a local directory), the value can only be JMS.

Note: The connector sends service call requests and administrative messages over CORBA IIOP if the value configured for the DeliveryTransport property is MQ or IDL.

WebSphere MQ or IDL (InterChange Server only)

Use WebSphere MQ rather than IIOP (CORBA) for event delivery transport where possible. WebSphere MQ offers the following advantages over IDL:

- ▶ **Asynchronous communication**

WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.

- ▶ **Server side performance**

WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves having to write potentially large events to the repository database.

- ▶ **Agent side performance**

WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connectors polling thread picks up an event, places it in the connector queue, and then picks up the next event. This is faster than IDL, which requires the connector polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, and then pick up the next event.

WebSphere MQ is the best quality of service option on a high-performance network (where the subnet, LAN, and enterprise applications have reasonable response times). All synchronous processing is through IIOP (request - response and admin messages) for low latency, but asynchronous event notification is WebSphere MQ for highest throughput.

IDL, which is actually IIOP (CORBA), should be used as the transport only on a high-performance network (subnet and LAN), where no asynchronous event notification is required. It requires a synchronous call to the InterChange Server, which then has to persist it to the database (because it is not using WebSphere MQ for persistence). This is not the best option for high-throughput asynchronous scenarios.

Java Messaging Service (all broker types)

This option enables communication between the connector and client framework using Java Messaging Service (JMS).

This option uses WebSphere MQ for all messages: synchronous, asynchronous, and admin messages. There is an additional overhead, due to the persistent messaging (versus IIOP), especially for request/response. Also, there is an additional overhead for asynchronous event notification when using the InterChange Server as the broker. The InterChange Server does not rely on WebSphere MQ for persistence and consequently must persist the entire object to the Work In Progress tables (WIP) of the repository database.

However, this is the best transport to use in the case of adapters distributed across a WAN or across a firewall, because WebSphere MQ provides the best quality of service over unreliable networks. This is usually called remote agent deployment (see Chapter 10, “Local versus remote deployment” on page 149) and includes MQ to MQ communication with a WebSphere MQ server in proximity to the connector. WebSphere MQ over SSL or HTTPS can also be used for security.

For the InterChange Server, JMS transport is required for connectors that receive requests from collaborations which implement Long Lived Business Processes (LLBP), whether for asynchronous requests or a synchronous request with time-out.

Asynchronous event processing and notification

One role of an adapter is to detect changes to applications' business entities. When an event that affects an application entity occurs, such as when a user of the application creates, updates, or deletes application data, a connector sends an *event* to the integration broker. This event contains a business object and a verb. This role is called *event notification*. To enable automated integration, event notification is an absolute necessity when building a custom adapter.

An *event-notification mechanism* enables a connector to determine when an entity within an application changes.

This chapter provides information about the event-notification mechanism.

8.1 Overview of event notification

The ways in which application-specific components detect and retrieve events differ from one adapter to another. However, the way in which application-specific components send events to the Adapter Framework and the way in which the Adapter Framework delivers those events to the integration broker is standard across all adapters.

When an event occurs in an application, the connector application-specific component processes the event, retrieves related application data, and returns the data to the integration broker in a business object. The following steps outline the tasks of an event-notification mechanism:

1. An application performs an event and puts an event record into the event store. The *event store* is a persistent cache in the application where event records are saved until the connector can process them. The event record contains information about the change to an event store in the application. This information includes the data that has been created or changed, as well as the operation (such as create, delete, or update) that has been performed on the data.
2. The connector's application-specific component monitors the event store, usually through a polling mechanism, to check for incoming events. When it finds an event, it retrieves its event record from the event store and converts it into an application-specific business object with a verb.
3. Before sending the business object to the integration broker, the application-specific component can verify that the integration broker is interested in receiving the business object.

If you use WebSphere InterChange Server as the business integration system, the Adapter Framework does not assume that the integration broker is always interested in every supported business object. At initialization, the Adapter Framework requests its subscription list from the Connector Controller. At run-time, the application-specific component can query the Adapter Framework to verify that some collaboration subscribes to a particular business object. The application-specific connector component can send the event only if some collaboration is currently subscribed. The application-specific component sends the event, in the form of a business object and a verb, to the Adapter Framework, which in turn sends it to the Connector Controller within InterChange Server.

However, if you use the Message Broker or WebSphere Application Server (Server Foundation) as the business integration system, the Adapter Framework assumes that the integration broker is interested in all the connector's supported business objects. If the application-specific connector component queries the Adapter Framework to verify whether to send the

business object, it will receive a confirmation for every business object that the connector supports.

4. If the integration broker is interested in the business object, the connector application-specific component sends the event, in the form of a business object and a verb, to the Adapter Framework, which in turn sends it to the integration broker.

The implementation of an event-notification mechanism is a three-stage process:

1. Create an event store that the application uses to hold notifications of events that have changed application business entities.
2. Implement an event detection mechanism within the application. Event detection notices a change in an application entity and writes an event record containing information about the change to an event store in the application.
3. Implement an event retrieval mechanism (such as a polling mechanism) within the connector to retrieve events from the event store and take the appropriate action to notify other applications.

Figure 8-1 illustrates the components of the event-notification mechanism. In event notification, the flow of information is from the application to the connector and then to the integration broker.

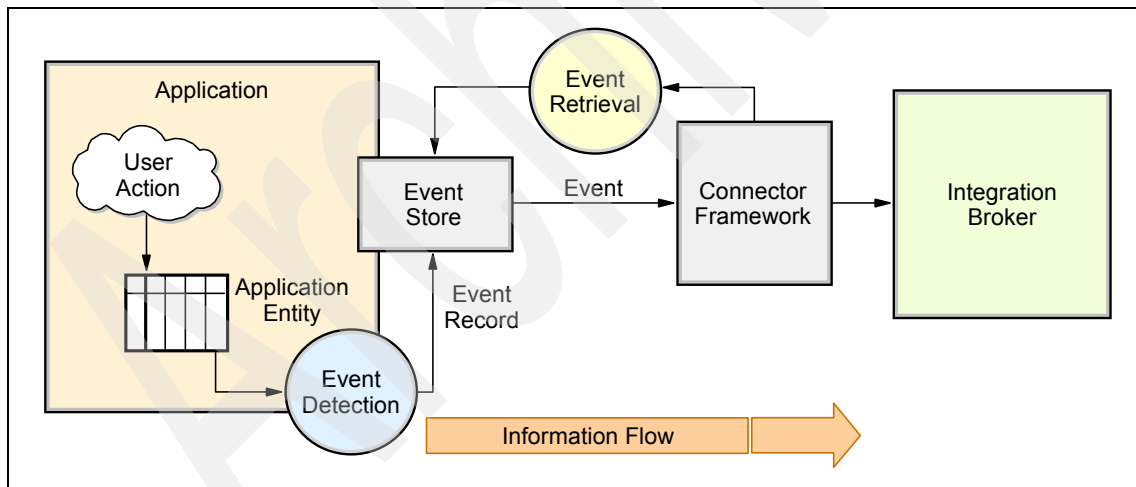


Figure 8-1 Event detection and retrieval mechanism

In many cases, an application must be configured or modified before the connector can use the event-notification mechanism. Typically, this application configuration occurs as part of the installation of the connector's application-specific component. Modifications to the application might include setting up a user account in the application, creating an event store and event

table in the application database, inserting stored procedures in the database, or setting up an inbox. If the application generates event records, it might be necessary to configure the text of the event records.

The connector might also need to be configured to use the event-notification mechanism. For example, a system administrator might need to set connector-specific configuration properties to the names of the event store and event table.

8.1.1 Subscription to a business object

A connector assumes that the business integration system uses a *publish and subscribe model* to move information from an application to an integration broker for processing as follows:

- ▶ An integration broker *subscribes* to a business object that represents an event in an application.
- ▶ A connector uses an event-notification mechanism to monitor when application events occur. When an application event does occur, the connector *publishes* a notification of the event in the form of a business object and a verb. When the integration broker receives an event in the form of the business object to which it has subscribed, it can begin the associated business logic on this data.

Note: If your business integration system uses WebSphere InterChange Server, a collaboration subscribes to a business object that represents an event in an application, and then the collaboration waits. The Connector Controller checks its subscription list when it receives a business object from the Adapter Framework to determine which, if any, collaborations have subscribed to this type of business object. If there is a subscription, it then forwards the business object to the subscribing collaboration. When a collaboration receives the subscribed event, it executes.

8.2 Event and archive stores

An *event store* is a persistent cache in the application where event records are saved until the connector can process them. The data store provides an ordered list of operations that take place in the application. The event store might be a database table, application event queue, e-mail inbox, or any type of persistent store. If the connector is not available, a persistent event store enables the application to detect and save event records until the connector becomes available.

The information in the event store generally includes the business object type, the verb, the key that identifies the changed data entity in the application data store, the time stamp, and the priority. The event store might also contain the complete data for the event itself.

The *archive store* serves the same basic purpose as an event store — it saves archive records in a persistent cache until the connector can process them. Event archives are useful for troubleshooting and record keeping. An archive store contains status information about each event, such as:

- ▶ Successfully sent to the integration broker
- ▶ Processing failed

8.2.1 Content of an event store

Event records must encapsulate everything a connector needs to process an event. Each event record should include enough information so that the connector poll method can retrieve the event data and build a business object that represents the event.

Note: Although different event retrieval mechanisms might exist, this section describes event records in the context of the most common mechanism, which is polling.

If the application provides an event detection mechanism that writes event records to an event store, the event record should provide discrete details about the object and verb. If the application does not provide sufficient detail, it might be possible to configure it to provide this level of detail.

Table 8-1 lists the standard elements of an event record.

Table 8-1 *Standard elements of an event record*

Element	Description
Event identifier	A unique identifier for the event.
Business object name	The name of the business object definition as it appears in the repository.
Verb	The name of the verb, such as Create, Update, or Delete.
Object key	The primary key for the application entity.
Priority	The priority of the event in the range 0 - n, where 0 is the highest priority.

Element	Description
Time stamp	The time at which the application generated the event.
Status	The status of the event. This is used for archiving events.
Description	A text string describing the event.
Connector identifier	An identifier for the connector that will process the event.

Note: A minimal set of information in an event record includes the event ID, business object name, verb, and object key. You might also want to set a priority for an event so that if large numbers of events are queued in the event store, the connector can select events in order of priority.

Event identifier

Each event must have a unique identifier. This identifier can be a number that is generated by the application or a number that is generated by a scheme that your connector uses. As an example of an event ID numbering scheme, the event might generate a sequential identifier, such as 00123, to which the connector adds its name. The resulting object event ID is ConnectorName_00123. Another technique might be to generate a time stamp, resulting in an identifier such as ConnectorName_06139833001001.

Your connector can optionally store the event ID in the ObjectEventId attribute in a business object. The ObjectEventId attribute is a unique value that identifies each event in the IBM WebSphere Business Integration system. Because this attribute is required, the Adapter Framework generates a value for it if the application-specific connector does not provide a value. If no values for ObjectEventIds are provided for hierarchical business objects, the Adapter Framework generates values for the parent business object and for each child. If the connector generates ObjectEventId values for hierarchical business objects, each value must be unique across all business objects in the hierarchy regardless of level.

Business object name

You can use the name of the business object definition to check for event subscriptions. Note that the event record should specify the exact name of the business object definition.

Event verb

The verb represents the kind of event that occurred in the application, such as Create, Update, or Delete. You can use the verb to check for event subscriptions.

Note: Events that represent the deletion of application data should generate event records with the Delete verb, even for logical delete operations where the delete is an update of a status value to inactive. For more details, see 8.5, “Processing delete events” on page 134.

The verb that the connector sets in the business object should be same verb that was specified in the event record.

Object key

The entity’s object key enables the connector to retrieve the full set of entity data if the object has subscribing events.

Note: The only data from the application entity that event records should include are the business object name, active verb, and object key. Storing additional entity data in the event store requires memory and processing time that might be unneeded if no subscriptions exist for the event.

The object key column must use name-value pairs to set data in the event record. For example, if ContractId is the name of an attribute in the business object, the object key field in the event record would be ContractId=45381.

Depending on the application, the object key might be a concatenation of several fields. Therefore, the connector should support multiple name-value pairs that are separated by a delimiter (for example, ContractId=45381:HeaderId=321). The delimiter should be configurable as set by the PollAttributeDelimiter connector configuration property. The default value for the delimiter is a colon (:).

Event status

A Java connector should use the event-status constants, which are defined in the CWConnectorEventStatusConstants class. Table 8-2 lists the event-status constants.

Table 8-2 Event-status values for a Java connector

Event-status constant	Description
READY_FOR_POLL	Ready for poll.
SUCCESS	Sent to the integration broker.

Event-status constant	Description
UNSUBSCRIBED	No subscriptions for event.
IN_PROGRESS	Event is in progress.
ERROR_PROCESSING_EVENT	Error in processing the event. A description of the error can be appended to the event description in the event record.
ERROR_POSTING_EVENT	Error in sending the event to the integration broker. A description of the error can be appended to the event description in the event record.
ERROR_OBJECT_NOT_FOUND	Error in finding the event in the application database.

8.2.2 Implementing an event store

The event store is usually provided with the adapter as an artifact that is specific to the application development environment and that is imported readily into the system. Where the application provides a facility for events, the adapter's event detection mechanism generally leverages it to populate the event store. For such applications, setting up the connector's event-notification mechanism is a normal application setup task.

For example, the application might allow the installation of a script that executes when a particular type of event occurs and the script to place a notification in an event store. This mechanism is frequently referred to as a *user exit*. Another application might have an internal workflow system that can register an event in the application and write to an event store when executed. One example is the IBM WebSphere Business Integration Adapter for mySap.com, which can use the SAP internal workflow event notification mechanism. The adapter provides the link into this SAP internal event notification mechanism.

If the application does not provide native support for events, the event notification mechanism can use any of the following as the event store:

- ▶ Event inbox
- ▶ Event table
- ▶ E-mail
- ▶ Flat files

Note: Some applications might provide multiple ways of keeping track of changes to application entities. For example, an application might provide workflow for some database tables and user exits for other tables. If this is the case, you might have to implement an event notification mechanism that handles events in one way for some business objects and another way for other business objects.

Using an event inbox

Some applications have a built-in inbox mechanism. You can use this inbox mechanism to transfer information about application events to the connector, as follows:

- ▶ For the event detection mechanism, you might need to identify the entities and events that trigger entries in the inbox.
- ▶ For the event retrieval, the connector's application-specific component can retrieve the entries. If an API is available that provides interfaces to access the inbox, the application-specific component can use this API.

Figure 8-2 illustrates this kind of interaction.

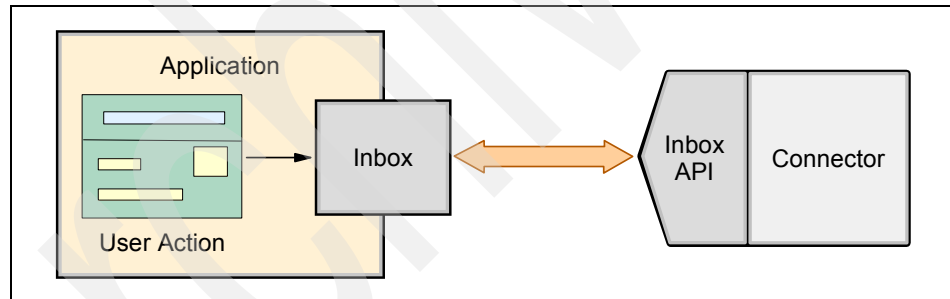


Figure 8-2 An event inbox as an event store

Using a database event table

An application can use its application database to store event information. It can create a special *event table* in this database to use as the event store for event records. This table is created during the installation of the connector. With an event table as an event store, as follows:

- ▶ For the event detection, when an event of interest to the connector occurs, the application places an event record in the event table.
- ▶ For the event retrieval, the connector application-specific component polls the event table periodically and processes any events. Applications often provide database APIs that enable the connector to gain access to the contents of the event table.

Figure 8-3 shows this interaction.

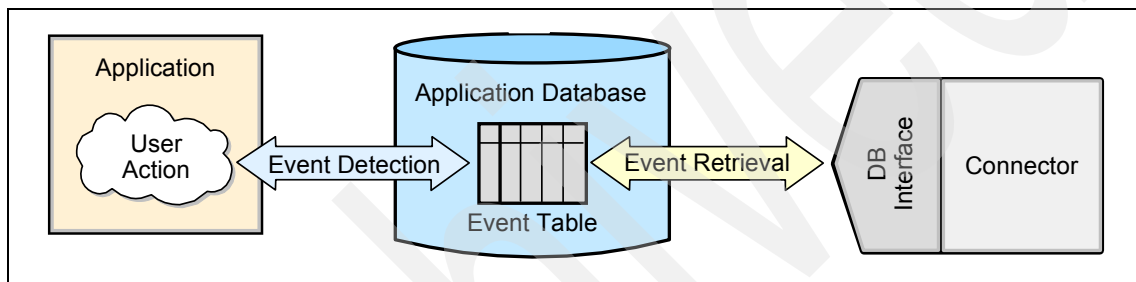


Figure 8-3 An event table as an event store

Note: Avoid full table scans of existing application tables as a way of determining whether application tables have changed. The recommended approach is to populate an event table with event information and poll the event table.

If your connector supports archiving of events, you can also create an archive table in the application database to hold the archived events. Table 8-3 shows a recommended schema for event and archive tables. You can extend this schema as needed for your application.

Table 8-3 Recommended schema for event and archive table

Column name	Type	Description
event_id	Use appropriate type for database	The unique key for the event. System constraints determine the format.
object_name	Char 80	Complete name of the business object.
object_verb	Char 80	Event verb.
object_key	Char 80	The primary key of the object.
event_priority	Integer	The priority of the event, where 0 is the highest priority.
event_time	DateTime	The time stamp for the event (time at which the event occurred).
event_processed	DateTime	For the archive table only. The time at which the event was handed to the Adapter Framework.
event_status	Integer	For possible status values, see Table 8-2 on page 95.
event_description	Char 255	Event description or error string.
connector_id	Integer	ID for the connector (if applicable).

Using an e-mail system as an event store

You can use an e-mail system as an event store, as follows:

- ▶ For event detection, the application sends a message to a mailbox when an application event occurs.
- ▶ For event retrieval, the connector's application-specific component checks the mailbox and retrieves the event message.

Figure 8-4 illustrates this interaction.

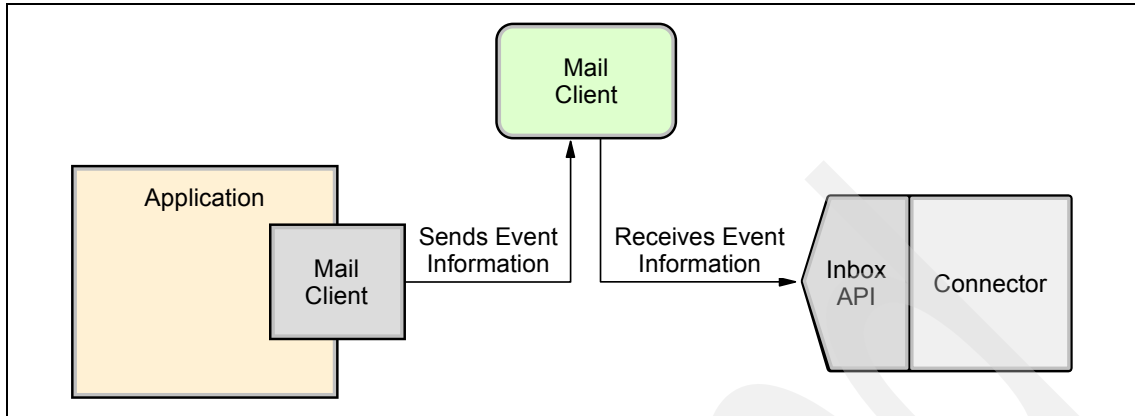


Figure 8-4 A mailbox as an event store

For an e-mail-based event store, the mailbox used for a connector must be configurable, and the actual name of the inbox used should reflect its usage. The following list specifies the format and recommended names for fields in event messages:

- ▶ **Message attributes:** Messages usually have certain attributes, such as a creation date and time, and a priority. You might be able to use these attributes in the event notification mechanism. For example, you might be able to use the date and time attributes to represent the date and time at which the event occurred.
- ▶ **Subject:** The subject of an event message might have the following format:
`object_name object_verb event_id`
 In this example, fields are separated by spaces for human-readability, but connectors can use a different field delimiter. The event_id is the unique key for the event. Depending on the application, the event_id key might or might not be included in the message. The event_id can be derived from a combination of the connector name, business object name, and either the message time stamp or the system time.
- ▶ **Body:** The body of an event message might contain a sequence of key/value pairs that are separated by delimiters. These key/value pairs are the elements of the object key. For example, if a particular customer and address are uniquely identified by the combination of CustomerId and AddrSeqNum, the body of the mail message might look like this:

`CustomerId 34225 AddrSeqNum 2`

The body of the event message can be a list of attribute names for the business object, and the values that should be inserted into those attributes.

Using a flat file

If no other event detection mechanism is available, it might be possible to set up an event store using flat files. With this type of event store:

- ▶ For event detection, the event detection mechanism in the application writes event records to a file.
- ▶ For event retrieval, the connector's application-specific component locates the file and reads the event information.

If the file is not directly accessible by the connector (for example, if it was generated on a mainframe system), the file must be transferred to a location that the connector can access. One way of transferring files is to use File Transfer Protocol (FTP), either internally in the connector or via an external tool to copy the file from one location to another. There are other ways to transfer information between files. The approach that you choose depends on your application and connector.

Figure 8-5 illustrates event detection and retrieval using flat files. In this example, FTP is used to transfer the event information to a location that is accessible by the connector.

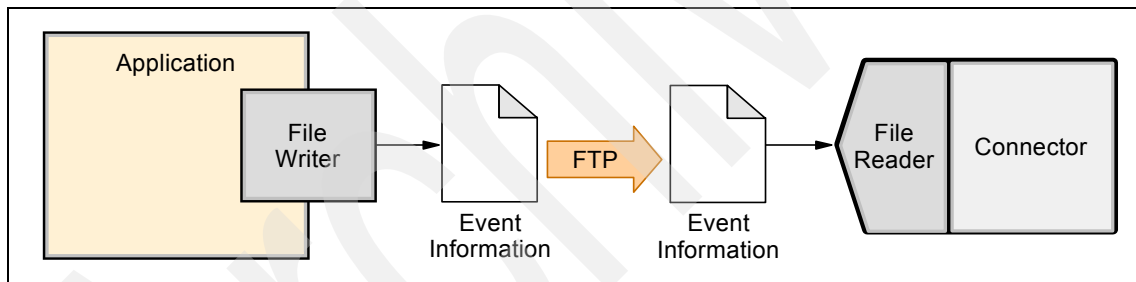


Figure 8-5 Retrieving event records from flat files

8.2.3 Accessing the event store

If a connector is expected to process information that originates in its application, it must obtain access to the application's event store.

Table 8-4 shows the objects that the Java connector library provides in support of obtaining access to an event store from within a Java connector.

Table 8-4 Support for defining access to an event store

Java connector library class	Description
CWConnectorEvent	Represents an event object that provides access to an event record within the Java connector.
CWConnectorEventStoreFactory	Provides a single method that creates an event store object.
CWConnectorEventStore	Represents the event store.
CWConnectorEventStatusConstants	Defines static constants for status values that an event can have.

The event store object

The CWConnectorEventStore class defines an event store. As Table 8-5 shows, this class provides an additional layer for standardizing the event retrieval, processing, and archiving mechanisms.

Table 8-5 Methods of the CWConnectorEventStore class

Event-store task	CWConnectorEventStore method	Implementation status
Event retrieval	fetchEvents()	Must be implemented.
	getB0()	Implementation provided in base class; however, you must override this implementation if your connector does not support the RetrieveByContent verb.
	getNextEvent()	Implementation provided in base class.
Event processing	recoverInProgressEvents()	Must be implemented.
	resubmitArchivedEvents()	Must be implemented.
	setEventStatus()	Must be implemented.
	setEventsToProcess()	Implementation provided in base class.
	updateEventStatus()	Implementation provided in base class.

Event-store task	CWConnectorEventStore method	Implementation status
Archiving	archiveEvent()	Must be implemented if the connector supports archiving.
	deleteEvent()	Must be implemented.
Error processing	getTerminate(), setTerminate()	Implementation provided in base class.
Resource cleanup	cleanupResources()	Not required for the event-store class but must be implemented if resources used to access the event store need to be released.

To define an event store:

1. Extend the CWConnectorEventStore class, naming your new class to identify the event store that your connector accesses.
2. Define any additional data members that your event store might require. The CWConnectorEventStore class contains a single data member: an events vector array called eventsToProcess. Events retrieved from the event store are saved in this Java vector object. Declare any other information that is required to access the application's event and archive stores as data members in your extended CWConnectorEventStore class. This information should include the location of the event and archive stores. For example:
 - In a table-based application, this information might be the event and archive table names and any database connection information.
 - In a file-based event store, this information might include the names of the event and archive directories.
 - An extended event store should also store any metadata information required for accessing or processing the event records. This information might include any “order by” information needed for JDBC queries.
3. Implement the appropriate abstract methods within the CWConnectorEventStore class (see Table 8-5 on page 102) to provide access to the event store. You can implement those CWConnectorEventStore methods that your event store requires, with the following conditions:
 - You must provide implementations for the abstract methods with “Must be implemented” in the Implementation status column of Table 8-5 on page 102. These methods are required to support the default implementation of the pollForEvents() method.

Note: If you override the default implementation of the `pollForEvents()` method, you can define only those `CWConnectorEventStore` methods that your `pollForEvents()` method needs to use.

- The `CWConnectorEventStore` class provides implementations for the methods with “Implementation provided in base class” in the Implementation status column of Table 8-5 on page 102.
- 4. Access to the `CWConnectorEventStore` methods as needed to perform event retrieval, event processing, and archiving from within the `pollForEvents()` poll method. For more information, see 8.4, “Custom `pollForEvents()` method” on page 115.

The `CWConnectorEventStoreFactory` interface defines an event-store factory, which provides a single method to instantiate an event store, the `getEventStore()` method, which must be implemented.

To define an event-store factory:

1. Create a new event-store-factory class to implement the `CWConnectorEventStoreFactory` interface. Name your new class to include the name of the event store that your `CWConnectorEventStore` class accesses.
2. Implement the `getEventStore()` method of the `CWConnectorEventStoreFactory` interface within your event-store-factory class to provide an event-store factory for your extended `CWConnectorEventStore` class.
3. Determine whether to use the default implementation of the `getEventStore()` method in the `CWConnectorAgent` class to instantiate an event store.
 - If you use the default implementation of this `getEventStore()` method, define the `EventStoreFactory` connector configuration property and set it to the entire class name (including its package name) for your event-store-factory class (which implements the `CWConnectorEventStoreFactory` interface).

The `EventStoreFactory` property has the following format:

connectorPackageName.EventStoreFactoryClassName

In this format, the *connectorPackageName* is as follows, where *connectorName* is the name of the connector:

com.crossworlds.connectors.connectorName

The `EventStoreFactoryClassName` is the name of the class that implements the `CWConnectorEventStoreFactory` interface.

Note: The `EventStoreFactory` property is a user-defined property, not a standard property. You must define this property with `Connector Configurator` for any connector that provides an event-store factory.

If the `EventStoreFactory` property is not set, the default implementation of the `getEventStore()` method attempts to generate the name of the event store.

- If the default implementation of the `getEventStore()` method does not adequately address the needs of your connector, you can override it in your connector class. Within this method, you can call some custom event-store constructor.

The event object

The Java connector obtains event records from the event store and encapsulates them as event objects. The event-store class builds event objects for each event record that the connector retrieves from the event store. The information in each event object is then used to build and retrieve the business object that the connector sends to the integration broker.

The default event object that the `CWConnectorEvent` class defines contains the event information in Table 8-1 on page 93. The `CWConnectorEvent` class provides accessor methods for this information, as Table 8-6 shows.

Table 8-6 Methods to retrieve information in an event object

Element	CWConnectorEvent method
Event ID	<code>getEventID()</code>
Business object name	<code>getBusObjName()</code>
Business object verb	<code>getVerb()</code>

Element	CWConnectorEvent method
Object key	<p>getIDValues(), getKeyDelimiter()</p> <p>These CWConnectorEvent methods provide access to the actual data values that identify the business object. The getIDValues() method assumes that this data is a name-value pair. For example, if the object key contains data for the ContractId attribute in the business object, the name-value pair in the business object data would be ContractId=45381. If the object key in the event record contains a concatenation of fields, the getIDValues() method assumes that each name-value pair is separated by a delimiter, which the getKeyDelimiter() method returns. The delimiter should be configurable as set by the PollAttributeDelimiter connector configuration property. The default value for the delimiter is a colon (:).</p>
Priority	getPriority()
Time stamp	getEventTimeStamp()
Status	<p>getStatus()</p> <p>Use the following methods to set event status: getNextEvent(), recoverInProgressEvents(), resubmitArchivedEvents(), setEventStatus(), updateEventStatus().</p>
Description	A text string describing the event.
Connector ID	getConnectorID()

In addition to providing the standard information in an event record, the event object also provides access methods for the information shown in Table 8-7 on page 106.

Table 8-7 Additional event information in the event object

Element	Description	Accessor method
Effective date	Date on which the event becomes active and should be processed. This information might be useful when there is a change to an object in one system that should not be propagated until the date on which it becomes effective (such as a salary change).	getEffectiveDate()

Element	Description	Accessor method
Event source	Source from where the event originated. This information might be needed by a connector that needs to track the event source for archiving.	<code>getEventSource()</code> , <code>setEventSource()</code>
Triggering user	User identifier (ID) associated with the user that triggered this event. This information can be used to avoid synchronization problems between two systems.	<code>getTriggeringUser()</code>

If your event record requires information beyond what the default event class provides (Table 8-6 on page 105 and Table 8-7), you can take the following steps:

1. Extend the `CWConnectorEvent` class, naming your new class to identify the event store whose event records your event class encapsulates.
2. Define any additional data members that your event might require. The `CWConnectorEvent` class contains the data members whose accessor methods are listed in Table 8-6 on page 105 and Table 8-7. Any other information that is required to access the application's event records needs to be declared as data members in your extended `CWConnectorEvent` class.
3. Provide accessor methods for any data members that you add to your extended `CWConnectorEvent` class. To support true encapsulation, your data members should be private members of your extended `CWConnectorEvent` class. To provide access to these data members, you define `Get` methods to retrieve each data member's value. You can also define `Set` methods for those data members that connector developers are allowed to set.

8.2.4 Creating an archive store

If the application provides archiving services, you can use those. Otherwise, an archive store is usually implemented using the same mechanism as the event store:

- For an event-notification mechanism that uses database triggers, one way to set up event archiving is to install a delete trigger on the event table. When the connector's application-specific component deletes a processed or unsubscribed event from the event table, the delete trigger moves the event to the archive table.

Note: If a connector uses an event table, an administrator might need to clean up the archive periodically.

- ▶ With an e-mail event notification scheme, archiving might consist of moving a message to a different folder. A folder called Archive might be used for archiving event messages.

Configuring a connector for archiving

Archiving can have a performance impact in the form of the archive store and moving the event records into this store. Therefore, you might want to design event archiving to be configurable at install time, so that a system administrator can control whether events are archived.

To make archiving configurable, you can create a connector-specific configuration property that specifies whether the connector archives unsubscribed events. We suggest a name of *ArchiveProcessed* for this configuration property. If the configuration property specifies no archiving, the connector application-specific component can delete or ignore the event. If the connector is performance-constrained or if the event volume is extremely high, archiving events is not required.

Accessing the archive store

A connector performs archiving as part of the event processing in its poll method, `pollForEvents()`. After a connector has processed an event, the connector must move the event to an archive store whether or not the event was successfully delivered to the Adapter Framework. Events that have no subscriptions are also moved to the archive. Archiving processed or unsubscribed events ensures that events are not lost.

Your poll method should consider archiving an event when any of the following conditions occur:

- ▶ When the poll method has processed the event and the Adapter Framework has delivered the business object.
- ▶ When no subscriptions exist for the event.

Note: If a connector uses an event table, an administrator might need to clean up the archive periodically. For example, the administrator might need to truncate the archive to free disk space.

8.3 Polling

For most connectors, the application-specific component of the connector implements the event retrieval mechanism. The connector developer does this as part of the connector design and implementation. This mechanism works in conjunction with the event detection mechanism, which detects entity changes and writes event records to the event store. Event retrieval transfers information about application events from the event store to the connector's application-specific component.

Two common mechanisms that are used to retrieve event records from an event store are:

- ▶ *Polling*: The most common type of event retrieval mechanism.
- ▶ *Event callback*: Connectors can be notified of application events through an event-callback mechanism. However, few applications currently provide event-callback APIs for application events.

The polling method is specific to the application, based on the event notification mechanism that the connector uses. Polling behavior is configurable, including poll frequency and the maximum number of events processed per poll call. The interface to the event store can be through the APIs where possible, or it can be provided through database queries to retrieve new events directly from an event table

In a polling mechanism, the application provides a persistent event store, such as an database table or inbox, where it writes event records when changes to application entities occur. The connector periodically checks, or polls, the event store for changes to entities that correspond to business object definitions that the connector supports. In general, the only information about the business object that is kept in the event store is the type of operation and the key values of the application entity. As the connector processes the event, it retrieves the remainder of the application entity data. After the connector has processed the event, it removes the event record from the event store and places it in an archive store.

To implement a polling mechanism to perform event retrieval, the connector's application-specific component uses a *poll method*, called the `pollForEvents()` method. The poll method checks the event store, retrieves new events, and processes each event before returning.

As previously stated, the Adapter Framework calls the poll method at a specified polling interval as defined by the *PollFrequency* connector configuration property. This property is initialized at connector installation time with Connector Configurator. Typically, the polling interval is about 10 seconds.

Note: If your connector does not need to poll to retrieve event information, polling can be turned off by setting the PollFrequency property to zero (0).

Therefore, the Adapter Framework calls the `pollForEvents()` method in either of the following conditions:

- ▶ The PollFrequency is set to a value greater than zero.
- ▶ The connector startup script specifies a value for the `-fPollFreq` option.

8.3.1 Standard behavior

The Adapter Framework generally initiates a poll call at periodic intervals. If there has been a change since the last poll call, the application-specific component determines if a business object definition exists to represent the changed data.

After detecting an event, the adapter's application-specific component:

1. Associates the application event with a business object definition and creates an instance of that business object.
2. Sets the verb and key value attributes in the business object.
3. Retrieves application data and populates the business object's attributes, generally by invoking the same method used for integration server-initiated requests.
4. Forwards the business object to the Adapter Framework.
5. Archives the event (optional).

After the Adapter Framework receives the business object from the application-specific component, it delivers the event to the integration server.

Figure 8-6 illustrates the basic behavior of a poll method.

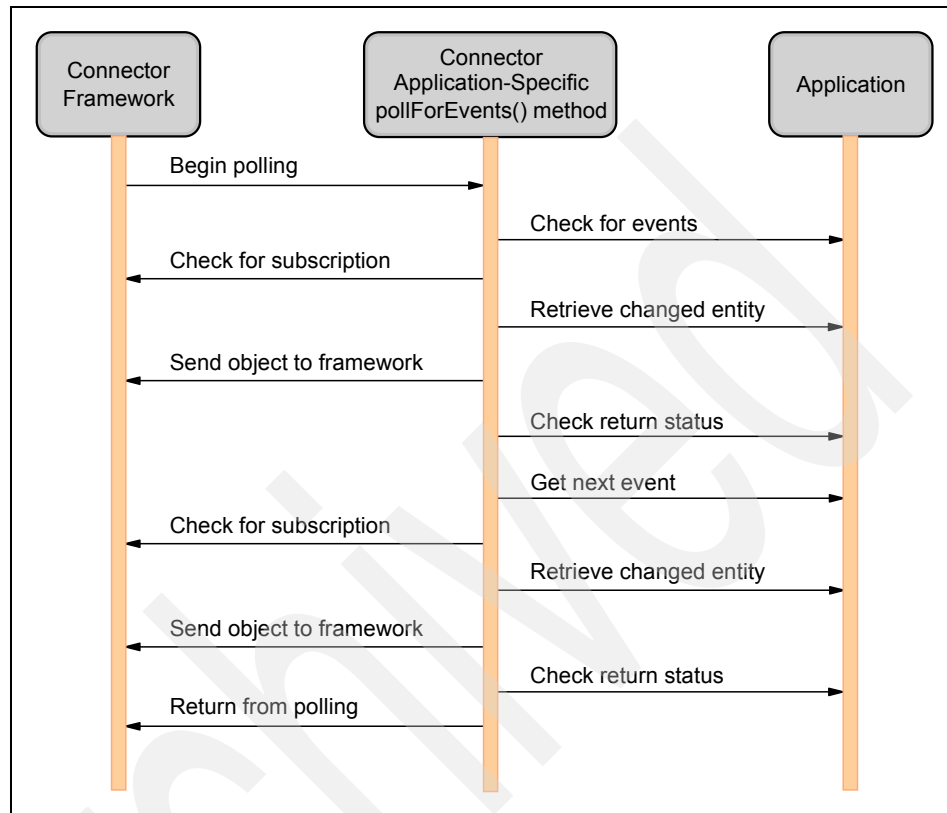


Figure 8-6 Flowchart for basic logic of the `pollForEvents()` method

The steps involved in a poll method include the following:

1. The Adapter Framework calls the application-specific component's `pollForEvents()` method to begin polling.
2. The `pollForEvents()` method checks the event store in the application for new events and retrieves the events.
3. The poll method then queries the Adapter Framework to determine whether an event has subscribers.
4. If an event has subscribers, the poll method retrieves the complete set of data for the business object from the application.
5. The poll method sends the business object to the Adapter Framework, which routes it to its destination (such as InterChange Server).

Each time the poll method is called, it checks for and retrieves new events, determines whether the event has subscribers, retrieves application data for events with subscribers, and sends business objects to InterChange Server.

8.3.2 Basic logic for the poll method

Regardless of whether the application provides an event store in a table, inbox, or other location, the connector must poll periodically to retrieve event information. The connector's poll method, `pollForEvents()`, polls the event store, retrieves event records, and processes events. To process an event, the poll method determines whether the event has subscribers, creates a new business object containing application data that encapsulates the event, and sends the business object to the Adapter Framework.

Note: If your connector will be implementing request processing but not event notification, you might not need to fully implement the `pollForEvents()` method. However, because the poll method is defined with a default implementation in the Java connector library, polling is already implemented. If you want to disable polling, you can implement a stub for this method.

The `pollForEvents()` method typically uses a basic logic for event processing, as shown in the flow chart in Figure 8-7 on page 113.

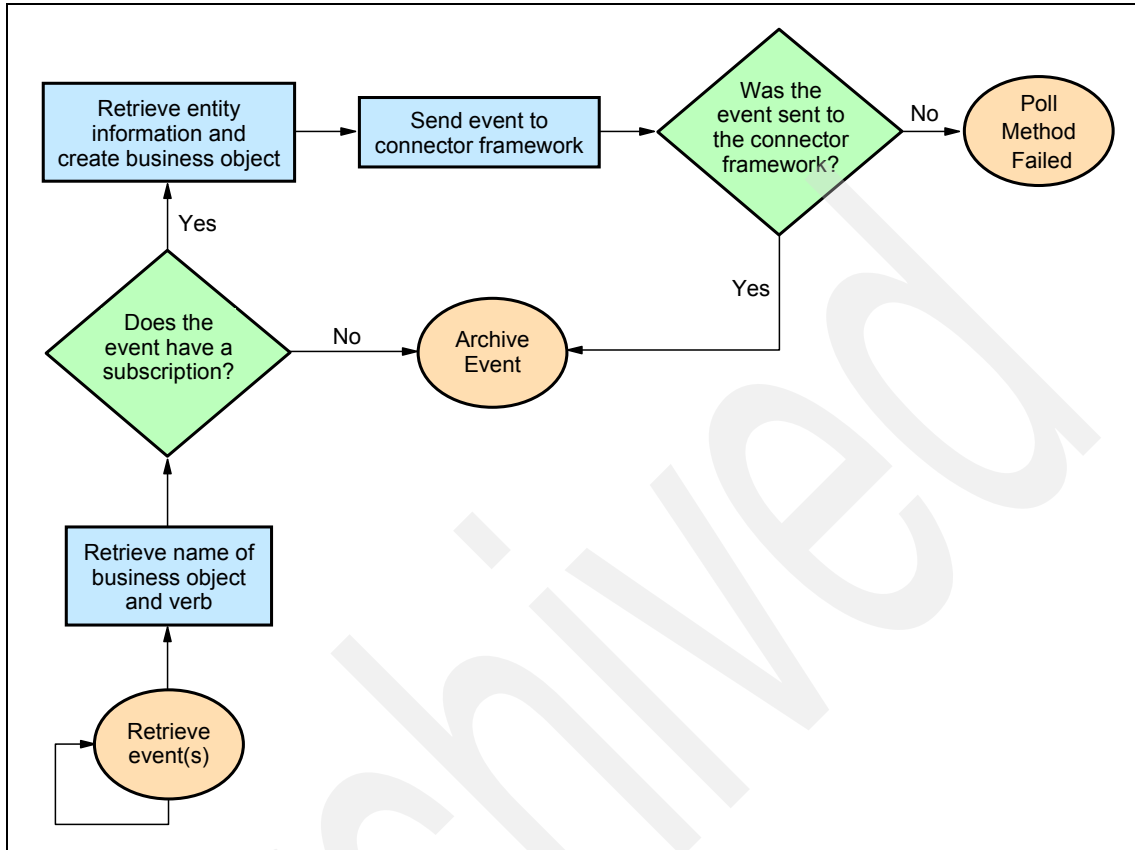


Figure 8-7 Flow chart for basic logic of the `pollForEvents()` method

8.3.3 Archiving events

After a connector has processed an event, it can archive the event. Archiving processed or unsubscribed events ensures that events are not lost.

Archiving usually involves the following steps:

1. Copy the event record from the event store to the archive store. An archive record contains the same basic information as an event record.
2. Update the event status of the event in the archive store. The archive record should be updated with one of the event-status values that are listed in Table 8-8 on page 114.
3. Delete the event record from the event store.

Table 8-8 Event status value in an archive record

Status	Description
Success	The event was detected, and the connector created a business object for the event and sent the business object to the Adapter Framework.
Unsubscribed	The event was detected, but there were no subscriptions for the event, so the event was not sent to the Adapter Framework and on to the integration broker.
Error	The event was detected, but the connector encountered an error when trying to process the event. The error occurred either in the process of building a business object for the event or in sending the business object to Adapter Framework.

8.3.4 Multithreaded

Java connectors must be thread safe. The Adapter Framework can use multiple threads to perform event delivery (execution of the `pollForEvents()` method) and request processing (execution of the `doVerbFor()` method).

8.3.5 Processing events by event priority

Event priority enables the connector poll method to handle situations where the number of events in the event store exceeds the maximum number of events the connector retrieves in a single poll. In this type of polling implementation, the poll method polls and processes events in order of priority. Event priority is defined as an integer value in the range 0 - n, with 0 as the highest priority.

To process events by event priority, the following tasks must be implemented in the event notification mechanism:

- ▶ The event detection mechanism must assign a priority value to an event record when it saves it to the event store.
- ▶ The event retrieval mechanism (the polling mechanism) must specify the order in which it retrieves event records to process, based on the event priority.

Note: As events are picked up, event priority values are not decremented. In rare circumstances, this might lead to low priority events not being picked up.

The SQL statement Example 8-1 shows how a connector might select event records based on event priority. The **SELECT** statement sorts the events by priority, and the connector processes each event in turn.

Example 8-1 Example SQL statement

```
SELECT event_id, object_name, object_verb, object_key
FROM event_table
WHERE event_status = 0 ORDER BY event_priority
```

8.3.6 Event distribution

The polling mechanisms can be implemented so that multiple connectors can poll the same event store. Each connector can be configured to process certain events, create specific business objects, and pass those business objects to broker system. This configuration can streamline the processing of certain types of events and increase the transfer of data out of an application.

To implement event distribution so that multiple connectors can poll the event store, complete the following steps:

1. Add a column to the event record for an integer connector identifier (ID), and design the event detection mechanism to specify which connector will pick up the event. This might be done per application entity. For example, the event detection mechanism might specify that all Customer events be picked up by the connector that has the `connectorId` field set to 4.
2. Add an application-specific connector property named `ConnectorId`. Assign each connector a unique identifier and store this value in its `ConnectorId` property.

Implement the poll method to query for the value of the `ConnectorId` property. If the property is not set, the poll method can retrieve all event records from the event store as usual. If the property is set to a connector identifier value, the poll method retrieves only those events that match the `ConnectorId` property.

8.4 Custom `pollForEvents()` method

For a Java connector, the `CWConnectorAgent` class defines the `pollForEvents()` method. This class provides a default implementation of the `pollForEvents()` method. You can use this default implementation or override the method with your own poll method. However, the `pollForEvents()` method must be implemented.

The Java-based pseudo-code in Example 8-2 shows the basic logic flow for a `pollForEvents()` method (see 8.3.2, “Basic logic for the poll method” on page 112 for more details).

Example 8-2 Basic logic of the `pollForEvents()` method

```
public int pollForEvents() {
    int status = 0;
    get the events from the event store
    for (events 1 to MaxEvents in event store) {
        extract B0Name, verb, and key from the event record
        if (ConnectorBase.isSubscribed(B0Name, B0verb) {
            B0 = JavaConnectorUtil.createBusinessObject(B0Name)
            B0.setAttrValue(key)

            retrieve application data using doVerbFor()
            B0.setVerb(Retrieve)
            B0.doVerbFor()
            B0.setVerb(B0verb)
            status = gotApp1Event(BusinessObject);

            archive event record with success or failure status
        } else {
            archive item with unsubscribed status
        }
    }
    return status;
}
```

The method first retrieves a set of events from the event store. For each event, the method calls the `isSubscribed()` method to determine whether any subscriptions exist for the corresponding business object. If there are subscriptions, the method retrieves the data from the application, creates a new business object, and calls the `gotApp1Event()` method to send the business object to InterChange Server. If there are no subscriptions, the method archives the event record with a status value of unprocessed.

Note: For a flow chart of the poll method’s basic logic, see Figure 8-6 on page 111.

The steps in the basic logic for the event processing that the `pollForEvents()` method typically performs are:

1. Set up a subscription manager for the connector.
2. Verify that the connector still has a valid connection to the event store.

3. Retrieve a specified number of event records from the event store and store them in an events array. Cycle through the events array. For each event, mark the event in the event store as In-Progress and begin processing.
4. Get the business object name, verb, and key data from the event record.
5. Check for subscriptions to the event.
If the event has subscribers:
 - Retrieve the application data and create the business object.
 - Send the business object to the Adapter Framework for event delivery.
 - Complete the event processing.If the event does not have subscribers, update the event status to Unsubscribed.
6. Archive the event.
7. Release resources used to access the event store.

8.4.1 Accessing a subscription manager

As part of connector initialization, the Adapter Framework instantiates a subscription manager. This subscription manager keeps the subscription list current. A connector has access to the subscription manager and the connector subscription list through a *subscription handler*, which is included in the connector base class. It can use the methods of this class to determine whether business objects have subscribers and to send business objects to the Connector Controller.

8.4.2 Verifying the connection

When the `agentInit()` method in the connector class initializes the application-specific component, one of its most common tasks is to establish a connection to the application. The `poll` method requires access to the event store. Therefore, before the `pollForEvents()` method begins processing events, it should verify that the connector is still connected to the application. The way to perform this verification is application-specific. Consult your application documentation for more information.

A good design practice is to code the connector application-specific component so that it shuts down whenever the connection to the application is lost. If the connection has been lost, the connector should not continue with event polling. Instead, it should return `APPRESPONSETIMEOUT` to notify the Adapter Framework of the loss of connection to the application.

Note: To surface an APPRESPONSETIMEOUT outcome status returned by the doVerbFor() method from within the pollForEvents() method, use the getTerminate() method of the CWConnectorEventStore class. For more information, see 8.4.6, “Retrieving application data” on page 123.

8.4.3 Retrieving event records

To send event notifications to the Adapter Framework, the poll method must first retrieve event records from the event store. Table 8-9 lists the methods that the Java connector library provides to retrieve event records from the event store.

Table 8-9 Classes and methods for event retrieval

Java connector library class	Class method
CWConnectorAgent	getEventStore()
CWConnectorEventStoreFactory	getEventStore()
CWConnectorEventStore	fetchEvents(), getNextEvent(), updateEventStatus()

The poll method can retrieve one event record at a time and process it, or it can retrieve a specified number of event records per poll and cache them to an events array. Processing multiple events per poll can improve performance when the application generates large numbers of events.

The number of events picked up in any polling cycle should be configurable using the connector configuration property *PollQuantity*. At installation time, a system administrator sets the value of PollQuantity to an appropriate number, such as 50. The poll method can use the getConfigProp() method to retrieve the value of the PollQuantity property, and then retrieve the specified number of event records and process them in a single poll.

The connector should assign the *In-Progress* status to any event that it has read out of the event store and has started to process. If the connector terminates while processing an event and before updating the event status to indicate that the event was either sent or failed, it will leave an In-Progress event in the table.

The Java connector library provides the `CWConnectorEventStore` class to represent an event store. To retrieve event records from this event store, the `poll` method:

1. Instantiates an event-store object with the `getEventStore()` method that is defined in the `CWConnectorAgent` class. The default implementation of this method calls the `getEventStore()` method of the event-store-factory class named in the `EventStoreFactory` connector configuration property. The event-store-factory class implements the `CWConnectorEventStoreFactory` interface for your event store.
2. Retrieves a specified number of event records from the event store with the `fetchEvents()` method.

You must implement the `fetchEvents()` method as part of the `CWConnectorEventStore` class. This method can use the value of the `PollQuantity` connector configuration property as the number of event records to retrieve. The method must take the following actions:

- Creates a `CWConnectorEvent` event object for each event record that it retrieves. These event records can be ordered by their time stamp. For information about retrieving event records by event priority, see 8.3.5, “Processing events by event priority” on page 114.

Note: If the event store is implemented with an event table in the application database, the `fetchEvents()` method can use JDBC methods to access the event table.

- Puts each event object into the *eventsInProgress* events vector.

The `fetchEvents()` method should throw the `StatusChangeFailedException` exception if the application is unable to fetch events because it is unable to access the event store. When the `pollForEvents()` method catches this exception, it can return the `APPRESPONSETIMEOUT` outcome status to indicate the lack of response from the application’s event store.

3. Loops through the events in the *eventsInProgress* events vector, taking the following actions on each event object:
 - Retrieves the next event object to process with the `getNextEvent()` method.
 - Updates the status of both the event record (in the event store) and the event object (retrieved from the events vector) to `IN_PROGRESS` with the `updateEventStatus()` method.

The `updateEventStatus()` method should throw the `StatusChangeFailedException` exception if the application is unable to change event status because it is unable to access the event store. When the

`pollForEvents()` method catches this exception, it can return the `APPRESPONSETIMEOUT` outcome status to indicate the lack of response from the application's event store.

Setting the event status to `IN_PROGRESS` indicates that the poll method has begun processing on the event. Example 8-3 shows a code fragment that retrieves event records from the event store that accesses each record as an event object.

Example 8-3 Retrieving event records from the event store

```
// Instantiate event store
CWConnectorEventStore evts=getEventStore();
// Fetch PollQuantity number of events from the application.
try {
    evts.fetchEvents();
} catch (StatusChangeFailedException e) {
    // log error message
    return CWConnectorConstant.FAIL;
}
// Get the property values for PollQuantity
int pollQuantity;
String poll=CWConnectorUtil.getConfigProp("PollQuantity");
if (poll == null || poll.equals(""))
    pollQuantity=1;
else
    pollQuantity=Integer.parseInt(poll);
for (int i=0; i < pollQuantity; i++) {
    // Process each event retrieved from the application.
    // Get the next event to be processed.
    evtObj=evts.getNextEvent();
}
```

8.4.4 Getting the business object name, verb, and key

After the connector has retrieved an event, it extracts the event ID, the object key, and the name and verb of the business object from the event record. The connector uses the business object name and verb to determine whether the integration broker is interested in this type of business object. If the business object and its active verb have subscribers, the connector uses the entity key to retrieve the complete set of data.

Table 8-10 lists the method that the Java connector library provides to obtain the name of the business object definition and the verb from the retrieved event records.

Table 8-10 Method for obtaining event information

Java connector library class	Method
CWConnectorEvent getBusObjName()	getVerb()

Important: The connector should send the business object with the same verb that was in the event record.

After the getNextEvent() method has retrieved an event object to be processed, the Java connector can use the appropriate accessor methods of the CWConnectorEvent class to obtain the information needed to check for an event subscription, as shown in Table 8-11.

Table 8-11 Accessors for the event attributes

Event information	Accessor
Event ID	getEventID()
Business object name	getBusObjName()
Verb	getVerb()
Object key	getIDValues()

8.4.5 Checking for subscriptions to the event

To determine whether the integration broker is interested in receiving a particular business object and verb, the poll method calls the isSubscribed() method. The isSubscribed() method takes the name of the current business object and a verb as arguments. The name of the business object and verb must match the name of the business object and verb in the repository

If you use WebSphere InterChange Server as the integration system, the poll method can determine if any collaboration subscribes to the business object with a particular verb. At initialization, the Adapter Framework requests its subscription list from the Connector Controller at connector initialization. At run-time, the application-specific component can use the isSubscribed() method to query the Adapter Framework to verify that some collaboration subscribes to a particular business object. The application-specific connector component can send the event only if some collaboration is currently subscribed.

If you use the Message Broker or WebSphere Application Server (Server Foundation) as the business integration system, the Adapter Framework assumes that the integration broker is interested in all the connector's supported business objects. If the poll method uses the isSubscribed() method to query

the Adapter Framework about subscriptions for a particular business object, the method returns true for every business object that the connector supports.

Table 8-12 on page 122 lists the methods that the Java connector library provides to check for subscriptions to the event.

Table 8-12 Classes and methods for checking subscriptions

Java connector library class	Method
CWConnectorAgent	isSubscribed()
CWConnectorEventStore	updateEventStatus(), archiveEvent(), deleteEvent()

Based on the value that the `isSubscribed()` method returns, the poll method should take one of the following actions:

- ▶ If there are subscribers for an event, the connector takes one of the actions described in “Events that have subscriptions” on page 123.
- ▶ If there are no subscriptions for the event, the connector should take one of the actions described in “Events that do not have subscriptions” on page 123.

For a Java connector, the `isSubscribed()` method is defined in the `CWConnectorAgent` class, because the subscription manager is part of the connector base class. The method returns true if there are subscribers and false if there are no subscribers.

Example 8-4 shows a code fragment that checks for subscriptions in a Java connector.

Example 8-4 Checking for subscriptions in a Java connector

```
if (isSubscribed(evtObj.getBusObjName(), evtObj.getVerb())) {  
    // handle event  
} else {  
    // Update the event status to UNSUBSCRIBED.  
    evts.updateEventStatus(evtObj,  
CWConnectorEventStatusConstants.UNSUBSCRIBED);  
    // Archive the event (if archiving is supported)  
    return CWConnectorConstant.FAIL;  
}
```

If no subscriptions exist for the event, this code fragment uses the `updateEventStatus()` method to update the event's status to `UNSUBSCRIBED` and then archives the event.

Events that have subscriptions

If there are subscribers for an event, the connector:

1. Retrieves the complete set of business object data from the entity in the application database.
2. Sends the business object to the Adapter Framework, which routes it to the integration broker.
3. Completes the processing on the event.
4. Archives the event (if archiving is implemented) in case the integration broker subscribes at a later time.

Events that do not have subscriptions

If there are no subscriptions for the event, the connector:

1. Updates the status of the event to Unsubscribed to indicate that there were no subscribers.
2. Archives the event (if archiving is implemented) in case the integration broker subscribes at a later time. Moving the event record to the archive store prevents the poll method from picking up unsubscribed events. For more information, see 8.4.9, “Archiving the event” on page 131.
3. Returns “fail” (FAIL outcome status for a Java connector) to indicate there are events pending for which no subscriptions currently exist.

We suggest that the connector return “fail” if no subscriptions exist for the event. However, you can return the outcome status that your design dictates.

No other processing should be done with unsubscribed events. If at a later date the integration broker subscribes to these events, a system administrator can move the unsubscribed event records from the archive store back to the event store.

8.4.6 Retrieving application data

If there are subscribers for an event, the poll method must:

1. Retrieve the complete set of data for the entity from the application. To retrieve the complete set of entity data, the poll method must use name of the entity’s key information (which is stored in the event) to locate the entity in the application. The poll method must retrieve the complete set of application data when the event has the following verbs:
 - Create
 - Update
 - Delete event for an application that supports logical deletes

For a Delete event from an application that supports physical deletes, the application might have already deleted the entity from the database, and the connector might not be able to retrieve the entity data. For information about delete processing, see 8.5, “Processing delete events” on page 134.

2. Package the entity data in a business object. After the populated business object exists, the poll method can publish the business object to subscribers.

Table 8-13 lists the method that the Java connector library provides to retrieve entity data from the application database and populate a business object.

Table 8-13 Method for retrieving business object data

Java connector library class	Method
CWConnectorEventStore	getB0()

Note: If the event is a delete operation, and the application supports physical deletions of data, the data has most likely been deleted from the application, and the connector cannot retrieve the data. In this case, the connector simply creates a business object, sets the key from the object key of the event record, and sends the business object.

For a Java connector, the standard way of retrieving application data from within the pollForEvents() method is to use the getB0() method in the CWConnectorEventStore class. This method:

1. Creates a temporary CWConnectorBusObj object to hold the new business object.
2. Populates the CWConnectorBusObj object with the data and key values from the specified event object.
3. If the event's verb is Create or Update, sets the business object's verb to RetrieveByContent and calls the doVerbFor() method to retrieve the remaining attribute values from the application.
4. Returns the populated CWConnectorBusObj object to the caller.

If the call to the getB0() method is successful, it returns the populated CWConnectorBusObj object. The following line shows a call to the getB0() method that returns a populated CWConnectorBusObj object called bo:

```
bo = evts.getB0(evtObj);
```

In case the `getB0()` call is not successful, the poll method should take the following steps:

1. Catches any exceptions that `getB0()` throws.
2. Checks for an `ERROR_OBJECT_NOT_FOUND` status in the event object to determine if the `doVerbFor()` method could not find the business object data in the application.
3. Checks for a null value returned by the `getB0()` method, which indicates that the `doVerbFor()` method was not successful.
4. Uses the `getTerminate()` method to check if the terminate-connector flag has been set, which indicates that the `doVerbFor()` method (called from within the `getB0()` method) returned an `APPRESPONSETIMEOUT` outcome status. If the `getTerminate()` method returns true, the `pollForEvents()` method should return an `APPRESPONSETIMEOUT` outcome status to terminate the connector.

Note: The default implementation of the `getB0()` method checks the outcome status of the `doVerbFor()` method and calls the `setTerminate()` method if the `doVerbFor()` method returns an `APPRESPONSETIMEOUT` outcome status. If you override the default implementation of the `getB0()` method, but still use the default implementation of the `pollForEvents()` method, your `getB0()` method implementation should perform this same task.

The `ObjectEventId` attribute is used in the IBM WebSphere Business Integration system to track the flow of business objects through the system. In addition, it is used to keep track of child business objects across requests and responses, because child business objects in a hierarchical business object request might be reordered in a response business object.

Connectors are not required to populate `ObjectEventId` attributes for either a parent business object or its children. If business objects do not have values for `ObjectEventId` attributes, the business integration system generates values for them. However, if a connector populates child `ObjectEventIds`, the values must be unique across all other `ObjectEventId` values for that particular business object regardless of level of hierarchy. `ObjectEventId` values can be generated as part of the event notification mechanism. For suggestions about how to generate `ObjectEventId` values, see “Event identifier” on page 94.

8.4.7 Sending the business object to Adapter Framework

After the data for the business object has been retrieved, the poll method:

1. Sets the business object verb.
2. Sends the business object.

Table 8-14 lists the methods that the Java connector library provides to perform these tasks.

Table 8-14 *Classes and methods for setting the verb and sending the business object*

Java connector library class	Method
CWConnectorBusObj	setVerb()
CWConnectorEvent	getVerb()
CWConnectorAgent	gotApp1Event()

To set the verb in a business object to the verb specified in the event record, the poll method calls the business object method `setVerb()`. The poll method should set the verb to the same verb that was in the event record in the event store.

Note: If the event is a physical delete, use the object keys from the event record to set the keys in the business object, and set the verb to Delete.

For a Java connector, the populated `CWConnectorBusObj` object that the `getB0()` method returns still has a verb of *RetrieveByContent*. The poll method must set the business object's verb to its original value with the `setVerb()` method of the `CWConnectorBusObj` class, as the following code fragment shows:

```
// Set verb to action as indicated in the event record
busObj.setVerb(evtObj.getVerb());
```

In this code fragment, the poll method uses the `getVerb()` method of the `CWConnectorEvent` class to obtain the verb from the event record. This verb then is copied into the business object with the `setVerb()` method.

To send the business object to the Adapter Framework the poll method uses the method `gotApp1Event()` as follows:

1. Checks that the connector is active.
2. Checks that there are subscriptions for the event.
3. Sends the business object to the Adapter Framework.

The Adapter Framework does some processing on the event object to serialize the data and ensure that it is persisted properly. It then makes sure the event is sent.

Note: If you use WebSphere InterChange Server as the integration system, the Adapter Framework makes sure that the event is either sent to InterChange Server through CORBA IIOP or written to a queue (if you are using queues for event notification). If sending the event to InterChange Server, the Adapter Framework forwards the business object to the Connector Controller, which in turn performs any mapping that is required to transform the application-specific business object to a generic business object. The Connector Controller can then send the generic business object to the appropriate collaboration.

If you use Message Broker as the business integration system, the Adapter Framework makes sure that the event is converted to an XML WebSphere MQ message and written to the appropriate MQ queue.

The poll method should check the return code from the `gotAppEvent()` method to ensure that any error conditions are handled appropriately. For example, until the event delivery is successful, the poll method should not remove the event from the event store. Instead, the poll method should update the event record's status to reflect the results of the event delivery.

Table 8-15 shows the possible event-status values, based on the return code from the `gotAppEvent()` method.

Table 8-15 Possible event status after event delivery with the `gotAppEvent()` method

State of event delivery	Return code of <code>gotAppEvent()</code>	Event status
If the event delivery is successful	SUCCEED	SUCCESS
If no subscription exists for the event	NO_SUBSCRIPTION_FOUND	UNSUBSCRIBED
If the connector has been paused	CONNECTOR_NOT_ACTIVE	READY_FOR_POLL
If the event delivery fails	FAIL	ERROR_POSTING_EVENT

The `gotAppEvent()` method returns SUCCEED if the Adapter Framework successfully delivers the business object. The poll method checks the return code from the `gotAppEvent()` method to ensure that the event record's status is updated appropriately. If the `gotAppEvent()` method returns any return code except FAIL, the poll method returns SUCCEED so that it continues to poll for events. However, on a FAIL return code from the `gotAppEvent()` method, the event delivery has failed, so the poll method logs an error message and fails.

Table 8-16 shows the actions that the `pollForEvents()` method takes based on the `gotAppEvent()` return code.

Table 8-16 Possible `pollForEvents()` actions after event delivery with `gotAppEvent()`

Return code of <code>gotAppEvent()</code>	Actions in <code>pollForEvents()</code>
SUCCEED	<ol style="list-style-type: none"> 1. Resets the event status to SUCCESS. 2. If the <code>ArchiveProcessed</code> connector property is set to true, archives the event and deletes it from the event store. 3. Continues polling.
NO_SUBSCRIPTION_FOUND	<ol style="list-style-type: none"> 1. Logs an error message. 2. Resets the event status to UNSUBSCRIBED. 3. If the <code>ArchiveProcessed</code> connector property is set to true, archives the event and deletes it from the event store. 4. Continues polling.
CONNECTOR_NOT_ACTIVE	<ol style="list-style-type: none"> 1. Logs an informational message at a trace level of 3. 2. Prepares the event for future re-execution: <ul style="list-style-type: none"> – For application adapters, resets the event status to <code>READY_FOR_POLL</code>. – For technology adapters, pushes back the event (if possible). 3. Returns SUCCEED as the <code>pollForEvents()</code> outcome status. In this case, the event is not archived.
FAIL	<ol style="list-style-type: none"> 1. Logs an error message. 2. Resets the event status to <code>ERROR_POSTING_EVENT</code>. 3. If the <code>ArchiveProcessed</code> connector property is set to true, archives the event and delete sit from the event store. 4. Returns FAIL as the <code>pollForEvents()</code> outcome status.

Therefore, the action that the `pollForEvents()` method takes when the `gotAppEvents()` method returns an outcome status of `CONNECTOR_NOT_ACTIVE` depends on the type of connector you have created. For an application connector (in particular a connector whose application uses a database as its

event store), the `pollForEvents()` method should reset the event's status to `READY_FOR_POLL` to revert an event back to its unprocessed state.

However, for technology connectors (in particular, those that do not use event tables and, therefore, cannot always revert an event back to an unprocessed state), the connector can hold the event in memory and return an outcome status of `SUCCEED` from the `pollForEvents()` method, rather than attempting to push the event back. The connector should keep this event in memory until the adapter is re-activated and the `pollForEvents()` method is again invoked. Then, the connector can try to republish the event.

Example 8-5 shows how this functionality might be implemented.

Example 8-5 Holding an event in memory

```
BusinessObject eventOnHold;
pollForEvents(...) {
    ...
    if eventOnHold != null {
        event = eventOnHold;
        eventOnHold = null;
    } else {
        event = getNextUnprocessedEvent();
    }
    ...
    result = gotAppEvent( event );
    if (result == CWConnectorConstant.CONNECTOR_NOT_ACTIVE ) {
        eventOnHold = event;
        return CWConnectorConstant.SUCCEED;
    }
}
```

Note: If you pause the adapter while it is actively processing an event and then later terminate this adapter (or it terminates unexpectedly on its own), in-doubt events can result for these events that the connector (using the above logic) has copied to memory. Different adapters have different strategies for how to handle in-doubt events. However, the result of this logic can mean the creation of in-doubt events even though the adapter was seemingly terminated properly. These events are not lost.

When implementing the `pollForEvents()` response to the `CONNECTOR_NOT_ACTIVE` return status, keep in mind that the programming approaches discussed here assume that the adapter places an event in an in-progress state while it processes and sends the event to the integration broker. However, not all adapters are implemented this way. An adapter simply might receive an event from a source and then call the `gotAppEvent()` method to send it to the integration broker. If this adapter terminates in the time between when it receives

the event and when it calls the `gotAppEvent()` method, the event is lost. When such an adapter is restarted, it has no way of reprocessing the event.

8.4.8 Completing the processing of an event

The processing of an event is complete when:

1. The poll method has retrieved the application data for the event and created a business object that represents the event.
2. The poll method has sent the business object to the Adapter Framework.

Note: For hierarchical business objects, the event processing is complete when the poll method has retrieved the application data for the parent business object and all child business objects and sent the complete hierarchical business object to the Adapter Framework. The event notification mechanism must retrieve and send the entire hierarchical business object, not just the parent business object.

The poll method must ensure that the event status correctly reflects the completion of the event processing. Therefore, it must handle both of the following conditions:

- ▶ Handling successful event processing
- ▶ Handling unsuccessful event processing

Handling successful event processing

The processing of an event is successful when the tasks complete successfully. The following steps show how the poll method should finish processing a successful event:

1. Receives a “success” return code from the `gotAppEvent()` method that signifies the Adapter Framework’s successful delivery of the business object to the messaging system.
2. Copies the event to the archive store. For more information, see 8.4.9, “Archiving the event” on page 131.
3. Sets the status of the event in the archive store.
4. Deletes the event record from the event store. Until the event delivery is successful, the poll method should not remove the event from the event table.

Note: The order of the steps might be different for different implementations.

Handling unsuccessful event processing

If an error occurs in processing an event, the connector should update the event status to indicate that an error has occurred. Table 8-17 shows the possible event-status values, based on errors that can occur during event processing.

Table 8-17 Possible event status after errors in event processing

State of event delivery	Event status	Does polling terminate?
If an error occurs in processing an event	ERROR_PROCESSING_EVENT	No, retrieve the next event from the event store.
If the event delivery fails	ERROR_POSTING_EVENT	Yes.
If no subscriptions exist for the event	UNSUBSCRIBED	No, retrieve the next event from the event store.

For example, if there are no application entities matching the entity key, the event status should be updated to “error processing event.” If the event cannot be successfully delivered, its event status should be updated to “error posting event.” As discussed in 8.4.7, “Sending the business object to Adapter Framework” on page 125, the poll method should check the return code from the `getAppEvent()` method to ensure that any errors that are returned are handled appropriately.

In any case, the event should be left in the event store to be analyzed by a system administrator. When the poll method queries for events, it should exclude events with the error status so that these events are not picked up. After an event’s error condition has been resolved, the system administrator can manually reset the event status so that the event is picked up by the connector on the next poll.

8.4.9 Archiving the event

Archiving an event consists of moving the event record from the event store to an archive store. The Java connector library provides the `CWConnectorEventStore` class to represent an event store, which includes the archive store. Table 8-18 lists the method that the Java connector library provides to archive events.

Table 8-18 Method for archiving events

Java connector library class	Method
<code>CWConnectorEventStore</code>	<code>updateEventStatus()</code> , <code>archiveEvent()</code> , <code>deleteEvent()</code>

Note: For a general introduction to archiving, see 8.3.3, “Archiving events” on page 113.

To archive event records from this event store, the poll method:

1. Ensures that archiving is implemented by checking the value of the appropriate connector configuration property, such as `ArchiveProcessed`.
2. Copies the event record from the archive store to the event store with the `archiveEvent()` method.

To provide event archiving, you must implement the `archiveEvent()` method as part of the `CWConnectorEventStore` class. This method identifies the event record to copy by its event ID.

The `archiveEvents()` method should throw the `ArchiveFailedException` exception if the application is unable to archive the event because it is unable to access the event store. When the `pollForEvents()` method catches this exception, it can return the `APPRESPONSETIMEOUT` outcome status to indicate the lack of response from the application’s event store.

3. Updates the event status of the archive record with the `updateEventStatus()` method to reflect the reason for archiving the event.

Table 8-19 shows the likely event-status constants that the archive record will have.

Table 8-19 Event-status constants in an archive record

Event status	Description
SUCCESS	The event was detected, and the connector created a business object for the event and sent the business object to the Adapter Framework. For more information, see “Handling successful event processing” on page 130.
UNSUBSCRIBED	The event was detected, but there were no subscriptions for the event, so the event was not sent to the Adapter Framework and on to the integration broker. For more information, see 8.4.5, “Checking for subscriptions to the event” on page 121.
ERROR_PROCESSING_EVENT	The event was detected, but the connector encountered an error when trying to process the event. The error occurred either in the process of building a business object for the event or in sending the business object to Adapter Framework. For more information, see “Handling unsuccessful event processing” on page 131.

The `updateEventStatus()` method should throw the `StatusChangeFailedException` exception if the application is unable to change the event status because it is unable to access the event store. When the `pollForEvents()` method catches this exception, it can return the `APPRESPONSETIMEOUT` outcome status to indicate the lack of response from the application's event store.

4. Deletes the event record from the event store with the `deleteEvent()` method. You must implement the `deleteEvent()` method as part of the `CWConnectorEventStore` class. This method uses the event ID to identify the event record to delete. The `deleteEvents()` method should throw the `DeleteFailedException` exception if the application is unable to delete the event because it is unable to access the event store. When the `pollForEvents()` method catches this exception, it can return the `APPRESPONSETIMEOUT` outcome status to indicate the lack of response from the application's event store.

Example 8-6 contains a code fragment that archives an event:

Example 8-6 Archiving an event

```
// Archive the event if ArchiveProcessed is set to true.
if (arcProcessed.equalsIgnoreCase("true")) {
    // Archive the event in the application's archive store.
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store.
    evts.deleteEvent(evtObj.getEventID());
}
```

After archiving is complete, your poll method should set the appropriate return code:

- ▶ If the archiving takes place after an event is successfully delivered, the return code is “success”, indicated with the `SUCCESS` outcome-status constant.
- ▶ If archiving is due to some error condition (such as unsubscribed events or an error in processing the event), the poll method might need to return a “fail” status, indicated with the `FAIL` outcome-status constant.

8.4.10 Releasing event store resources

Often, the `pollForEvents()` method needs to allocate resources to access the event store. To prevent excessive memory usage by these resources, you can release them at the end of the poll method. Table 8-20 on page 134 lists the method that the Java connector library provides to release event-store resources.

Table 8-20 Method for releasing event-store resources

Java connector library class	Method
CWConnectorEventStore	cleanupResources()

For example, if the event store is implemented as event tables in a database, the `pollForEvents()` method might allocate SQL cursors to access these tables. You can implement a `cleanupResources()` method to free these SQL cursors. At the end of `pollForEvents()`, you can then call the `cleanupResources()` method to free the memory that these cursors use.

Note: The `CWConnectorEventStore` class does not provide a default implementation of the `cleanupResources()` method. To free event-store resources, you must override the `cleanupResources()` method with a version that releases the resources needed to access your event store.

8.5 Processing delete events

An application can support one of the following types of delete operations:

- ▶ Physical delete (data is physically deleted from the database).
- ▶ Logical delete (a status column in a database entity is set to an inactive or invalid status, but the data is not deleted from the database).

It might be tempting to implement delete event processing in a manner that is consistent with the application. For example, when an application entity is deleted, a connector poll method for an application that supports physical deletes might publish a business object with the Delete verb. A connector poll method for an application that supports logical deletes might publish a business object with the Update verb and the status value changed to inactive.

Problems can arise with this approach when a source application and a destination application support different delete models. Suppose that the source application supports logical delete and the destination application supports physical delete. Assume that an enterprise is synchronizing between the source and destination applications. If the source connector sends a change in status (in other words, a delete event) as a business object with the Update verb, the destination connector might be unable to determine that the business object actually represents a delete event.

Therefore, event publishing must be designed so that source connectors for both types of applications can publish delete events in such a way that destination connectors can handle the events appropriately. The Delete verb in an event notification business object should represent an event where data was deleted,

whether the delete operation was a physical or logical delete. This ensures that destination connectors will be correctly informed about a delete event.

8.5.1 Setting the verb in the event record

The event detection mechanism for both logical and physical delete connectors should set the verb in the event record to Delete:

- ▶ For a physical delete connector, this is the standard implementation.
- ▶ For a connector whose application supports logical deletes, the event detection mechanism must be designed to determine when update events actually represent deletion of data.

In other words, it must differentiate update events for modified entities from update events for logically deleted entities. For logically deleted entities, the event detection mechanism should set the verb in the event record to Delete even if the event in the application was an Update event that updated a status column.

8.5.2 Setting the verb in the business object

The poll method for both logical and physical delete connectors should generate a business object with the Delete verb:

- ▶ If the application supports logical deletes, the connector poll method retrieves the delete event from the event store, creates an empty business object, sets the key, sets the verb to Delete, and sends the business object to the Adapter Framework.

For hierarchical business objects, the connector should not send deleted children. The connector can constrain queries to not include entities with the status of inactive, or child business objects with a status of inactive can be removed in mapping.

- ▶ If the application supports physical deletes, the connector might not be able to retrieve the application data. In this case, the connector poll method retrieves the delete event from the event store, creates an empty business object, sets the key values, sets the values of other attributes to the special Ignore value (Cxlgnore), sets the verb in the business object to Delete, and sends the business object to the Adapter Framework.

8.5.3 Setting the verb during mapping

If you use InterChange Server, mapping between the application-specific business object and the generic business object should map the verb as Delete. This ensures that the correct information about an event is sent to the collaboration, which might perform special processing based on the verb.

Follow these recommendations for relationship tables:

- ▶ For delete events for a logical delete application, leave relationship entries in the relationship table.
- ▶ For delete events for a physical delete application, delete relationship entries from the relationship table.

8.6 Guaranteed event delivery

The guaranteed event delivery feature enables the Adapter Framework to guarantee that events are never sent twice between the connector's event store and the integration broker.

Important: This feature is available only for JMS-enabled connectors, that is, those connectors that use Java Messaging Service (JMS) to handle queues for their message transport. A JMS-enabled connector always has its `DeliveryTransport` connector property set to JMS. When the connector starts, it uses the JMS transport; all subsequent communication between the connector and the integration broker occurs through this transport. The JMS transport ensures that the messages are eventually delivered to their destination.

Without the use of the guaranteed event delivery feature, a small window of possible failure exists between the time that the connector publishes an event (when the connector calls the `gotAppEvent()` method within its `pollForEvents()` method) and the time it updates the event store by deleting the event record (or perhaps updating it with an “event posted” status). If a failure occurs in this window, the event has been sent but its event record remains in the event store with a “ready for poll” status. When the connector restarts, it finds this event record still in the event store and sends it, resulting in the event being sent twice.

You can provide the guaranteed event delivery feature to a JMS-enabled connector in one of the following ways:

- ▶ With the *container managed events* feature: If the connector uses a JMS event store (implemented as a JMS source queue), the Adapter Framework acts as a container and manages the JMS event store.

- With the *duplicate event elimination* feature: The Adapter Framework can use a JMS monitor queue to ensure that no duplicate events occur. This feature is usually used for a connector that uses a non-JMS event store (for example, implemented as a JDBC table, inbox, or flat files).

8.6.1 Container-managed events

If the JMS-enabled connector uses JMS queues to implement its event store, the Adapter Framework can act as a “container” and manage the JMS event store (the JMS source queue). One of the roles of JMS is to ensure that once a transactional queue session starts, the messages are cached there until a commit is issued. If a failure occurs or a rollback is issued, the messages are discarded. Therefore, in a single JMS transaction, the Adapter Framework can remove a message from a source queue and place it on the destination queue. This container-managed-events feature of guaranteed event delivery enables the Adapter Framework to guarantee that events are never sent twice between the JMS event store and the destination’s JMS queue.

Enabling the feature for connectors with JMS event stores

To enable the guaranteed event delivery feature for a JMS-enabled connector that has a JMS event store, set the connector configuration properties, as shown in Table 8-21.

Table 8-21 Connector properties for a connector with a JMS event store

Connector property	Value
DeliveryTransport	JMS
ContainerManagedEvents	JMS
PollQuantity	The number of events to processing in a single poll of the event store
SourceQueue	<p>Name of the JMS source queue (event store) that the Adapter Framework polls and from which it retrieves events for processing</p> <p>The source queue and other JMS queues should be part of the same queue manager. If the connector’s application generates events that are stored in a different queue manager, you must define a remote queue definition on the remote queue manager. WebSphere MQ can then transfer the events from the remote queue to the queue manager that the JMS-enabled connector uses for transmission to the integration broker.</p>

Note: A connector can use only one of these guaranteed event delivery features: container-managed events or duplicate event elimination. Therefore, you cannot set the ContainerManagedEvents property to JMS and the DuplicateEventElimination property to true.

In addition to configuring the connector, you must also configure the data handler that converts between the event in the JMS store and a business object. This data-handler information consists of the connector configuration properties, as summarized in Table 8-22.

Table 8-22 Data-handler properties for guaranteed event delivery

Data-handler property	Value	Required?
MimeType	The MIME type that the data handler handles. This MIME type identifies which data handler to call.	Yes
DHClass	The full name of the Java class that implements the data handler.	Yes
DataHandlerConfigMOName	The name of the top-level metaobject that associates MIME types and their data handlers.	Optional

Note: The data-handler configuration properties reside in the connector configuration file with the other connector configuration properties.

End users that configure a connector that has a JMS event store to use guaranteed event delivery must be instructed to set the connector properties as described in Table 8-21 on page 137 and Table 8-22. To set these connector configuration properties, use the Connector Configurator tool. Connector Configurator displays the connector properties in Table 8-21 on page 137 on its Standard Properties tab. It displays the connector properties in Table 8-22 on its Data Handler tab.

Note: Connector Configurator activates the fields on its Data Handler tab only when the DeliveryTransport connector configuration property is set to JMS and ContainerManagedEvents is set to JMS.

Effect on event polling

If a connector uses guaranteed event delivery by setting `ContainedManagedEvents` to `JMS`, it behaves slightly differently from a connector that does not use this feature. To provide container-managed events, the Adapter Framework takes the following steps to poll the event store:

1. Starts a JMS transaction.
2. Reads a JMS message from the event store. The event store is implemented as a JMS source queue. The JMS message contains an event record. The name of the JMS source queue is obtained from the `SourceQueue` connector configuration property.
3. Calls the appropriate data handler to convert the event to a business object. The Adapter Framework calls the data handler that has been configured with the properties in Table 8-22 on page 138.
4. When WebSphere Message Broker or WebSphere Application Server is the integration broker, converts the business object to a message based on the configured wire format (XML).
5. Sends the resulting message to the JMS destination queue.

If you use InterChange Server, the message sent to the JMS destination queue is the business object. If you use Message Broker, the message sent to the JMS destination queue is an XML message.

6. Commits the JMS transaction. When the JMS transaction commits, the message is written to the JMS destination queue and removed from the JMS source queue in the same transaction.
7. Repeats steps 1 through 6 in a loop. The `PollQuantity` connector property determines the number of repetitions in this loop.

Important: A connector that sets the `ContainerManagedEvents` property to `JMS`, does not call the `pollForEvents()` method to perform event polling. If the connector's base class includes a `pollForEvents()` method, this method is not invoked.

8.6.2 Duplicate event elimination

The Adapter Framework can use duplicate event elimination to ensure that duplicate events do not occur. This feature is usually enabled for JMS-enabled connectors that use a non-JMS solution to implement an event store (such as a JDBC event table, inbox, or flat files). This duplicate-event-elimination feature of guaranteed event delivery enables the Adapter Framework to guarantee that events are never sent twice between the event store and the destination's JMS queue.

Note: JMS-enabled connectors that use a JMS event store usually use the container-managed-events feature. However, they can use duplicate event elimination instead of container-managed events.

This section provides the following information about use of the guaranteed event delivery feature with a JMS-enabled connector that has a non-JMS event store:

- ▶ Enabling the feature for connectors with non-JMS event stores
- ▶ The effect on event polling

Enabling the feature for connectors with non-JMS event stores

To enable the guaranteed event delivery feature for a JMS-enabled connector that has a non-JMS event store, you must set the connector configuration properties as shown in Table 8-23.

Table 8-23 Connector properties for a connector with a non-JMS event store

Connector property	Value
DeliveryTransport	JMS
DuplicateEventElimination	true
MonitorQueue	Name of the JMS monitor queue, in which the Adapter Framework stores the ObjectEventId of processed business objects

Note: A connector can use only one of these guaranteed event delivery features: container-managed events or duplicate event elimination. Therefore, you cannot set the DuplicateEventElimination property to true and the ContainerManagedEvents property to JMS.

End users that configure a connector to use guaranteed event delivery must be instructed to set the connector properties as described in Table 8-23. To set these connector configuration properties, use the Connector Configurator tool. It displays these connector properties on its Standard Properties tab.

Effect on event polling

If a connector uses guaranteed event delivery by setting `DuplicateEventElimination` to `true`, it behaves slightly differently from a connector that does not use this feature. To provide the duplicate event elimination, the Adapter Framework uses a *JMS monitor queue* to track a business object. The name of the JMS monitor queue is obtained from the *MonitorQueue* connector configuration property.

After the Adapter Framework receives the business object from the application-specific component (through a call to the `getApp1Event()` method in the `pollForEvents()` method), it must determine if the current business object (received from the `getApp1Events()` method) represents a duplicate event. To make this determination, the Adapter Framework retrieves the business object from the JMS monitor queue and compares its `ObjectEventId` with the `ObjectEventId` of the current business object:

- ▶ If these two `ObjectEventIds` are the same, the current business object represents a duplicate event. In this case, the Adapter Framework ignores the event that the current business object represents. It does not send this event to the integration broker.
- ▶ If these `ObjectEventIds` are not the same, the business object does not represent a duplicate event. In this case, the Adapter Framework copies the current business object to the JMS monitor queue and then delivers it to the JMS delivery queue, all as part of the same JMS transaction. The name of the JMS delivery queue is obtained from the *DeliveryQueue* connector configuration property. Control returns to the connector's `pollForEvents()` method, after the call to the `getApp1Event()` method.

For a JMS-enabled connector to support duplicate event elimination, you must make sure that the connector's `pollForEvents()` method includes the following steps:

1. When you create a business object from an event record retrieved from the non-JMS event store, save the event record's unique event identifier as the business object's `ObjectEventId` attribute.

The application generates this event identifier to uniquely identify the event record in the event store. If the connector goes down after the event has been sent to the integration broker, but before this event record's status can be changed, this event record remains in the event store with an `In-Progress` status. When the connector comes back up, it should recover any `In-Progress` events. When the connector resumes polling, it generates a business object for the event record that still remains in the event store. However, because both the business object that was already sent and the new one have the same event record as their `ObjectEventIds`, the Adapter Framework can recognize the new business object as a duplicate and not send it to the

integration broker. A Java connector can use the `setDEEId()` method of the `CWConnectorBusObj` class to assign the event identifier to the `ObjectEventId` attribute, as follows:

```
busObj.setDEEId(event_id);
```

2. During connector recovery, make sure that you process In-Progress events before the connector begins polling for new events. Unless the connector changes any *In-Progress* events to *Ready-for-Poll* status when it starts up, the polling method does not pick up the event record for reprocessing.

Synchronous event processing (callback)

In many of today's custom and off-the-shelf applications, a requirement exists to provide an immediate, or synchronous, response to an internal event within the application. Although this response could be fulfilled through the existing event framework, event processing is asynchronous in nature and does not easily allow an adapter developer to correlate an event with this disparate business object request that can arrive later.

To overcome this issue, a new method was introduced in the Version 2.1 release of the WebSphere Business Integration Adapter Framework that allows for synchronous handling of requests from an application. This method, `executeCollaboration`, handles the sending of the request to the broker automatically and correlates the broker response for reply to the application, which simplifies the coding requirements for a custom adapter.

This chapter provides details about this new method.

9.1 Overview of the executeCollaboration method

The executeCollaboration method sends a synchronous business object request to the Adapter Framework that forwards it to a business process within the integration broker.

The syntax of this method is:

```
public void executeCollaboration(String busProcName,  
CWConnectorBusObj theBusObj,  
CWConnectorReturnStatusDescriptor rtnStatusDesc);
```

The first thing to notice is that the method has a void return parameter type. Thus, all success or failure information status information of the call is returned through the CWConnectorReturnStatusDescriptor. This parameter is optional. However, if you need to notify the back-end application of the success or failure of the call, this parameter must be specified.

9.1.1 Parameters

This method has the following parameters:

busProcName	Specifies the name of the business process to execute the business object request. If InterChange Server is your integration broker, the business-process name must be the name of an active collaboration on the server. When Interchange Server is the broker, there might be situations where different collaborations need to be executed based on the data supplied by the back-end application. If this is required, it is the responsibility of the custom adapter developer to implement some type of lookup mechanism to determine the name of the collaboration. If WebSphere Business Integration Message Broker, WebSphere Application Server, or WebSphere Business Integration Server Foundation is the broker environment, a value must be specified for this parameter. However, the value is not validated or used by the server. In these cases, the flow, process, or application that is executed is determined by the broker environment.
--------------------	--

theBusObj

The triggering event and the business object returned from the business process. It is important to note here that the inbound and outbound business objects are required to be the same. If this is not the case, as an adapter developer, you might want to consider making both the required inbound and outbound business objects children of a new parent business object. If this is not possible, the `executeCollaboration` method is not viable for your solution. Unlike standard asynchronous event processing, it is the responsibility of the adapter developer to convert between the application-specific data and the WebSphere Business Integration Adapter business object format. If you plan to use this method to handle synchronous events, you might want to move any application to business object conversion routines into a separate class which can be used by both the code that wrappers this function and the `BOHandler`.

rtnStatusDesc

The return-status descriptor that holds the success or failure response of the synchronous event request. This parameter is optional. However, it is the only mechanism for understanding the success or failure the synchronous event. Unlike standard adapter processing, the descriptor is not used to send information to the broker, so it is unnecessary to populate the descriptor through any `Set` methods prior to calling the `executeCollaboration` method.

9.2 Using the `executeCollaboration` method

Unlike asynchronous event processing, there is no support within the WebSphere Business Integration Adapter Framework to simplify the use of the `executeCollaboration` method. It is the responsibility of the adapter designer to consider the following:

- ▶ Accepting the request from the application.
- ▶ Mapping the application-specific data into a business object.
- ▶ Ensuring that the application data is sent to the correct business process.
- ▶ Mapping the returned business object into the application-specific data.
- ▶ Returning the data to the application.

In most adapters that support this capability, the `executeCollaboration` method is wrapped within a method, typically within the connector agent class, that is called through the connectivity code used to integrate with the application. This method is typically registered or identified to the connectivity code. When a

sequential event processing request is received, this method is called to undertake the interaction with the adapter. This action is typically known as *callback*, because the communication code must callback into the adapter to interact with the framework and ultimately the broker. Although it is possible to embed all functionality within the communication code, you would still need to call all WebSphere Business Integration Adapter Framework code through a callback to methods that were inherited by the custom connector agent class.

Note: We do not recommended that you call the `executeCollaboration` method from within an event processing mechanism, because this method is synchronous and blocks the event processing until a response is received.

What does it mean to develop a custom adapter? The following are two cases that use different technologies to call the `executeCollaboration` method:

- ▶ Case 1: Interacting with a back-end application with RMI
Within this scenario, the back-end application is capable of issuing RMI calls to an adapter that then issue the `executeCollaboration` method to process the synchronous event request.
- ▶ Case 2: Interacting with a back-end application with TCP/IP sockets
With any TCP/IP sockets transport, there are multiple ways of interacting with the back-end application. For simplicity, we assume that the adapter opens a separate server socket to handle synchronous event requests from the back-end application.

9.3 Single or multithreaded support for sequential event processing

When designing the sequential event processing mechanism for a custom adapter, you must understand the back-end application requirements. Will the application only make sequential event requests serially, or will the application require the custom adapter to process multiple sequential events concurrently? Fortunately, beyond potentially making the development of the communication code more complex, multithreaded support is available.

Returning to our previous two scenarios, using RMI as the transport mechanism provides multithreaded support automatically, because each call into the RMI stub runs on a thread that is created by the RMI. So, if you were to make simultaneous multiple calls to the sequential event processing, RMI would start multiple threads automatically to concurrently process the request. Within

scenario two, you would need to enhance your design to enable process multiple concurrent sequential event requests.

9.4 Adapter termination and sequential event processing

This section describes the different behaviors of adapter termination when you use sequential event processing.

9.4.1 Normal termination

When an adapter is shut down normally, the Adapter Framework attempts to call the termination method of the custom connector agent class. This method should undertake the necessary processing to terminate the sequential processing mechanism. We recommend that any currently awaiting requests be handled by returning a failure status to the back-end application, if at all possible.

Current, in-progress requests will most likely be lost or might need to be thrown away depending on the connectivity to the application. When an adapter is terminated, a sequential event request will likely be in one of three states:

- ▶ Awaiting processing by the sequential event processing mechanism
- ▶ Sent to the broker and awaiting a response
- ▶ Processing the broker response to send back to the application

The first point has been discussed previously, and ideally, we would prefer to process any currently available responses prior to returning from the terminate method of the custom adapter agent, which can be achieved with thoughtful adapter design. One consideration that adapter developers should take into account is the longevity of the broker flows or processes that might be executed. Ideally, these processes should be short-lived. If this cannot be guaranteed, the adapter developer must consider weighing the implications of delaying adapter termination and risk the user pressing Ctrl+C to terminate the adapter abnormally or terminating the adapter with outstanding requests.

9.4.2 Abnormal termination or no neat adapter shutdown situations

If a custom adapter with sequential event processing is abnormally terminated, the back-end application will be potentially left in state where it is awaiting responses from the adapter that it will never receive. Action might be required within the application to clean-up the application state.

In addition, depending on the broker, you can find messages on the Synchronous Response Queue. On adapter startup, these response messages are not processed, because the Adapter Framework cannot correlate the response message with any currently outstanding sequential event processing requests within the adapter.



Local versus remote deployment

This chapter discusses in more detail the differences when deploying adapters in the same environment as the broker versus a remote environment.

10.1 Overview of adapter deployment

There are number of considerations that affect the decision about where to deploy a specific adapter. The local installation is considered to be the most typical and easiest to maintain, particularly when you are developing and testing interfaces, because all the components in the business integration system are installed on a single computer. However, there can be valid reasons to deploy the adapter to a separate environment, for example:

- ▶ The Adapter Framework does not support the broker platform operating system.
- ▶ There are different brokers connected to the adapter.
- ▶ The application requires the adapter to be in the same environment.
- ▶ The adapter is deployed on the partner's environment.
- ▶ Performance-related issues (broker platform needs to be dedicated to the broker alone).

Figure 10-1 illustrates adapter deployment possibilities.

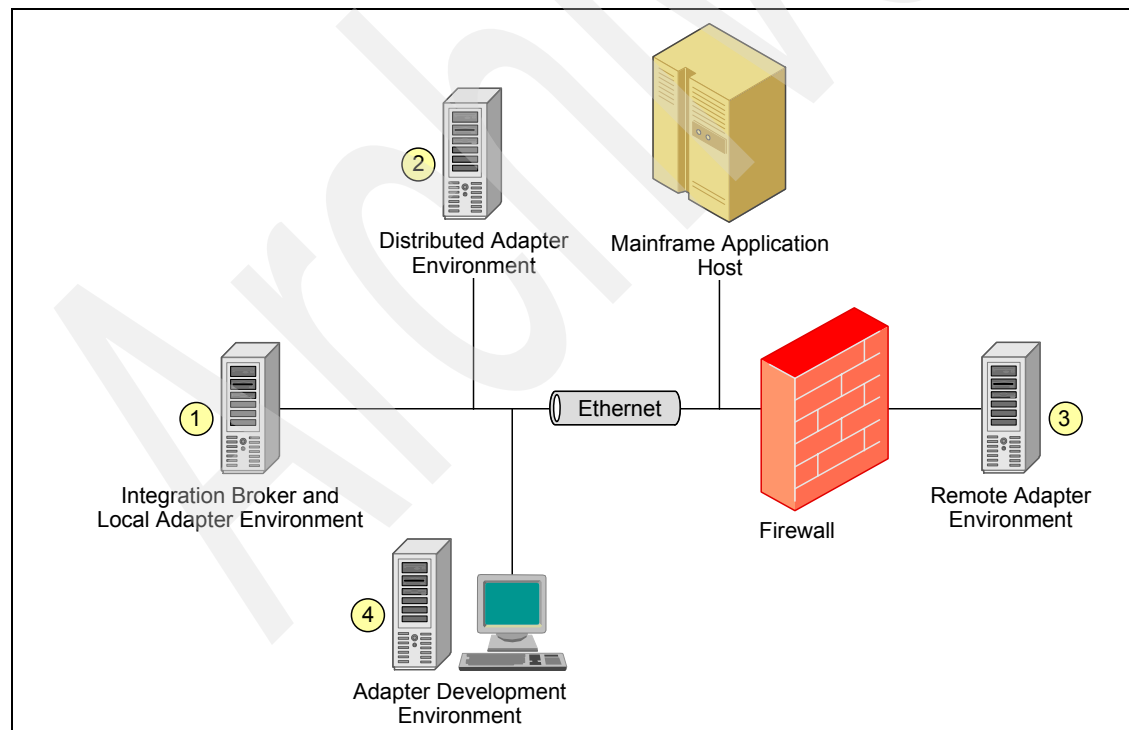


Figure 10-1 Adapter deployment possibilities

10.2 Local deployment

When an adapter is deployed locally, which means that it is installed on the same machine as the broker, there are certain issues that you need to consider, depending on the operating system. When deploying adapters locally, you should:

1. Obtain computers that satisfy the hardware requirements for each of the required environments.

For more information about hardware requirements, see the *Installation Guide for WebSphere Business Integration Adapters*, available at:

<http://www.ibm.com/software/integration/wbiadapters/library/infocenter/index.html>

2. If your integration broker is WebSphere MQ Integration Broker, WebSphere Business Integration Message Broker, or WebSphere Application Server, install the WebSphere MQ client as described in the WebSphere MQ documentation.
3. Do one of the following depending on which integration broker you are implementing:
 - If your integration broker is InterChange Server, install the InterChange Server and all its prerequisite software as described in the *System Installation Guide* for Microsoft® Windows® or UNIX®.
 - If your integration broker is one of the supported message brokers or WebSphere Application Server, install the Adapter Framework as described in *Installation Guide for WebSphere Business Integration Adapters*.

Tip: If your integration broker is WebSphere InterChange Server and if you are installing WebSphere Business Integration Adapters on the same computer on which InterChange Server is installed, do not install the Adapter Framework. The Adapter Framework is installed with WebSphere InterChange Server.

4. Read the “Installing and configuring the connector” chapter in the guide for each adapter that you plan to install in the environment, and determine any data handler requirements for the environment.
5. Install each data handler that is required for the environment as described in the *Installation Guide for WebSphere Business Integration Adapters*.
6. Install each adapter that is required for the environment as described in the *Installation Guide for WebSphere Business Integration Adapters*.

7. Perform any adapter-specific installation steps, such as installing the application client on the adapter host computer, as described in the guide for each adapter that is required in the environment.

10.2.1 Configuring startup in the Windows environment

A typical requirement for software that runs on a Windows platform is to make it a Windows service. WebSphere Business Integration Adapters can also be configured to run as Windows services, which makes them more manageable for starting, stopping, and checking the status of the adapter.

To configure an adapter to run as a Windows service:

1. Create a copy of the batch file that is used to start the connector and open the copy in a text editor, such as Notepad. Then, do the following:

- a. Use a find and replace function to replace the following string with the actual name of the connector (for example, JDBCConnector):

`%CONNAME%Connector`

- b. Use a find and replace function to replace the following string with the actual name of the InterChange Server instance with which the connector will communicate if InterChange Server is the broker:

`%SERVER%`

If you are using a broker other than InterChange Server, specify a placeholder value such as WMQI or WAS.

- c. Save and close the file.

Note: It is particularly important to use a copy of the batch file that starts a connector when configuring it as a service, because some connectors share a common batch file. If you make such edits to a batch file that is shared by multiple connectors, those connectors will no longer work.

2. Execute the following command from the command line to configure an adapter as a Windows service:

```
ProductDir\bin\cwservice -xi -mode=ServiceStartupType -tCONNECTOR  
-cConnectorBatchFile -sConnectorName -iInterChangeServer -tThreadMode
```

In this command:

<i>ProductDir</i>	Is the product directory.
<i>ServiceStartupType</i>	Is set to Auto if you want the service to start automatically or is set to Manual if you want to start the service manually.

<i>ConnectorBatchFile</i>	Is set to the path and name of the batch file that starts the connector (for example: C:\IBM\WebSphereAdapters\connectors\JDBC\start_service_JDBC.bat).
<i>ConnectorName</i>	Is set to the name of the connector (for example, JDBCCConnector).
<i>InterChangeServer</i>	Is the name of the InterChange Server instance with which the connectors communicate if InterChange Server is the broker. If you are using a broker other than InterChange Server, specify a placeholder value such as WMQI or WAS.
<i>ThreadMode</i>	Is set to MULTI_THREADED or SINGLE_THREADED.

The command creates a Windows service named CWInterchange *ConnectorName*, where *ConnectorName* is the value that you specified for the *ConnectorName* parameter when executing the command.

10.2.2 Configuring startup in a UNIX environment

In UNIX environments, you should create a startup script that contains the commands that are needed to initiate the adapter. This script should be included in the system startup routines so that the adapter gets started when the system starts.

10.3 Remote deployment

Even though the local installation might be the most desirable choice for the installation, sometimes installing the adapter to another platform might be required, as discussed in 10.1, “Overview of adapter deployment” on page 150. When installing an adapter to a remote platform, you should consider the following:

1. Make sure that the platform requirements are fulfilled, such as installing adapter locally and installing the MQClient (see 10.2, “Local deployment” on page 151). Make sure that the Java Messaging feature is installed with the MQClient installation.

2. You have to install the WebSphere Business Integration Adapter Framework regardless which broker you are using. The installation differs depending on the broker:
 - If your integration broker is WebSphere Application Server or one of the supported messaging brokers, do the following:
 - i. Install the Adapter Framework as described in the *Installation Guide for WebSphere Business Integration Adapters*. Read the “Installing and configuring the connector” chapter in the guide for each adapter that you plan to install in the environment and determine any data handler requirements for the environment.
 - ii. Install each data handler that is required for the environment and each adapter that is required, as described in the *Installation Guide for WebSphere Business Integration Adapters*.
 - If your integration broker is WebSphere InterChange Server, do the following:
 - i. Run the installer for WebSphere InterChange Server on the distributed computer to install the proper version of the Adapter Framework on it. When the InterChange Server Configuration Wizard opens, specify the same configuration values as those specified when installing InterChange Server on the broker host computer. For more information about installing WebSphere InterChange Server, see the *System Installation Guide for Windows or UNIX*. Note that you should not start this instance of InterChange Server. You are only installing it to install a compatible version of the Adapter Framework on the adapter host computer.
 - ii. Apply any patches that have been applied to the InterChange Server host computer to the distributed computer so that the broker and adapter environments are running at the same patch level.
 - iii. Install the adapter as described in the *Installation Guide for WebSphere Business Integration Adapters*. At the IBM WebSphere InterChange Server window, specify the name of the InterChange Server instance on the broker host computer, rather than the name of the InterChange Server instance that you installed on the adapter host computer (on which you had to install to install the Adapter Framework).
 - iv. Import the connector definition that was created in the repository directory into your development environment, as described in the *Implementation Guide for WebSphere InterChange Server*.
 - v. Configure the connector as described in the guide for the adapter.

- vi. Deploy the connector to the InterChange Server repository as described in the *Implementation Guide for WebSphere InterChange Server*.
3. Complete the following steps to configure the environment to communicate with the broker.

Note: This step is for the InterChange Server only, where you are using the IDL (IIOP) or MQ (MQ/IIOP) selected transports.

- a. Open the shared environment file in the ProductDir/bin directory in a text editor.
 - On Windows systems, the shared environment file is named CWSHaredEnv.bat.
 - On UNIX systems, the shared environment file is named adapterEnv.sh or CWSHaredEnv.sh, depending on the broker that you are using and the version of its release.
 - b. Set the value of the ORB_PORT property to the port over which the Object Request Broker installed on the broker computer is configured to communicate.
 - c. Set the value of the ORB_HOST property to the IP address of the computer on which the broker is installed.
4. Perform any adapter-specific installation steps, such as installing the application client on the adapter host computer, as described in the guide for each adapter required in the environment.

10.4 Setting up the communication between a remote agent and a broker

When you deploy an adapter across the network, there are special considerations that you need to take. In the case where there is a need for secure messaging between the agent and broker, you need to take some additional steps to the basic configuration of a remote setup (see 10.3, “Remote deployment” on page 153).

There are basically two transport protocol alternatives to use between the agent and broker in a distributed environment:

- ▶ Native WebSphere MQ
- ▶ HTTP/HTTPS

10.4.1 Native WebSphere MQ

The use of native WebSphere MQ means the use of asynchronous communication using messages and queues. The remote agent and the broker are sending business objects through WebSphere MQ messaging. The transport protocol between the remote agent and broker is handled through a WebSphere MQ channel, which is a layer on top of a standard transport protocol, such as TCP or NetBIOS. WebSphere MQ provides additional transactionality on top of the base protocol, which guarantees the transactionality of the message delivery.

You can also use secure channels between the agent and broker with which the messaging traffic can be encrypted using Secure Socket Layer (SSL). This method can be useful in situations where the agent and the broker are communicating over public networks.

This section describes an example of how you could set up the communication links between the remote agent WebSphere MQ queue manager and the broker environment queue manager.

Configuration steps for the remote agent connectivity

You must create bidirectional communication links between the remote site and the broker, which means that you have to create WebSphere MQ channel definitions to both directions. The assumptions here are that you are using TCP as the underlying transport protocol and that the default TCP/IP port (1414) is used by the both queue managers.

Note: WebSphere MQ uses port 1414 as a default. If for some reason you would like to use another port for your configuration, you have to complete some additional steps in your configuration.

Figure 10-2 illustrates the communication between the remote agent and the broker.

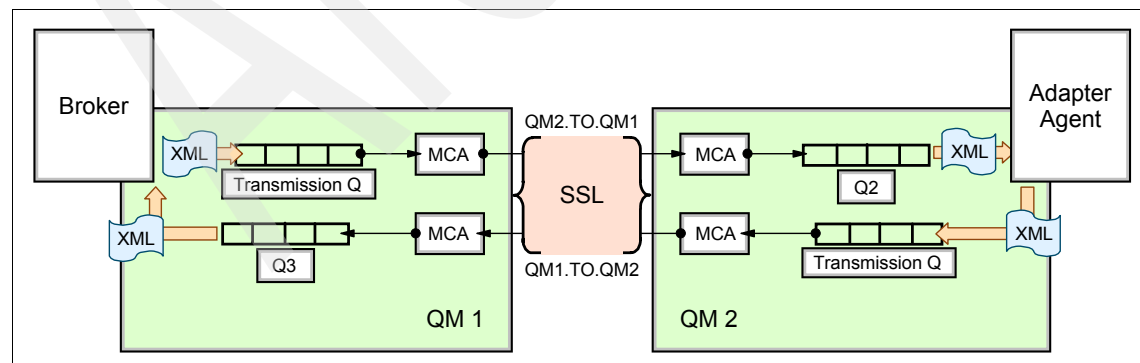


Figure 10-2 Remote agent deployment using native WebSphere MQ

The communication uses messages, which are delivered to the target destination using WebSphere MQ channels. If the target system is not alive or if the network connection is down, WebSphere MQ makes sure that the messages are sent when the connection is alive.

If the broker is WebSphere Business Integration Message Broker or WebSphere Application Server Enterprise, the messages are XML formatted messages. If the broker is InterChange Server, the messages are actually business objects.

Note: The naming of WebSphere MQ objects is not strictly specified anywhere. The names have certain limitations, such as length, but otherwise, they can be anything.

To establish a 2-way communication between two queue managers:

1. Define the transmission queue in QM1 called QM2.
2. Define the transmission queue in QM2 called QM1.
3. Define the sender channel in QM1 called QM1.TO.QM2.
4. Define the receiver channel in QM2 called QM1.TO.QM2.

Note: It is important that the sender and receiver channels have the same name. Otherwise, the channel will not work.

5. Define the sender channel in QM2 called QM2.TO.QM1.
6. Define the receiver channel in QM1 called QM2.TO.QM1.

Assume that there are two firewalls between the connection, as shown in Figure 10-3 on page 158.

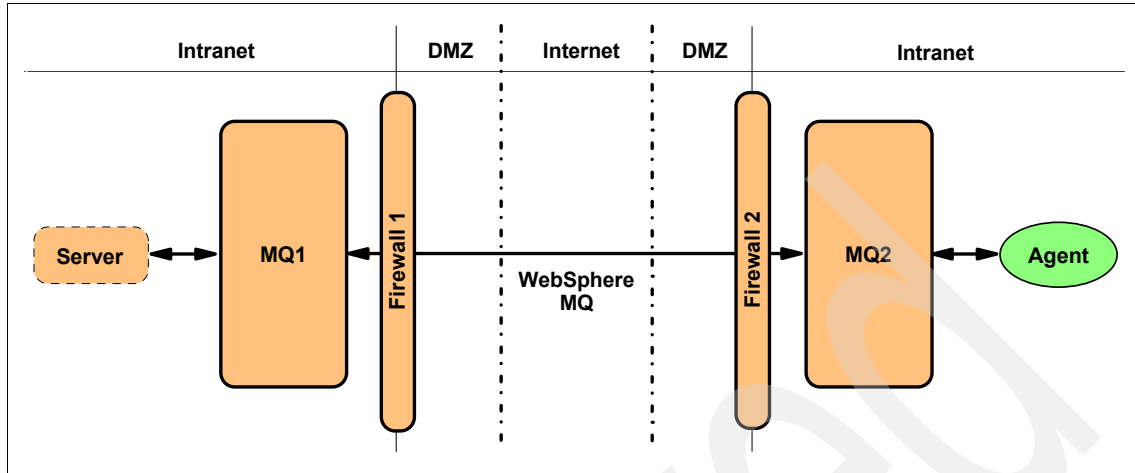


Figure 10-3 Native WebSphere MQ through firewalls

The sender channels need to be configured so that the target IP address is the receiving firewall, and the firewalls need to be configured to forward the traffic to the final destination, as shown in the following steps:

1. Define the target IP address of sender channel QM1.TO.QM2 as the IP address of firewall 2.
2. Define the target IP address of sender channel QM2.TO.QM1 as the IP address of firewall 1.
3. Configure firewall 1 to forward traffic on port 1414 to QM1 and configure firewall 2 to forward traffic on port 1414 to QM2.
4. Define the dead-letter queues for both queue managers.
5. Make sure that *fault* queue is local for both queue managers.

Note: In addition to these actual communication definitions, the normal queue definitions between the broker and remote agent queue managers have to be defined. Thus, the queues are used to deliver the actual business objects between components.

10.4.2 HTTP/HTTPS

Sometimes it is required to use standard HTTP/HTTPS protocol to exchange information between the remote agent and the broker (for example, if the firewall policies do not allow to open additional ports).

An additional component called WebSphere MQ Internet Pass-Thru (IPT) can be installed and configured to enable WebSphere MQ messages to be sent and received using HTTP/HTTPS.

The difference in configuration is that the queue managers are connected to the local instance of WebSphere MQ IPT, instead of the actual target systems, as shown in Figure 10-4.

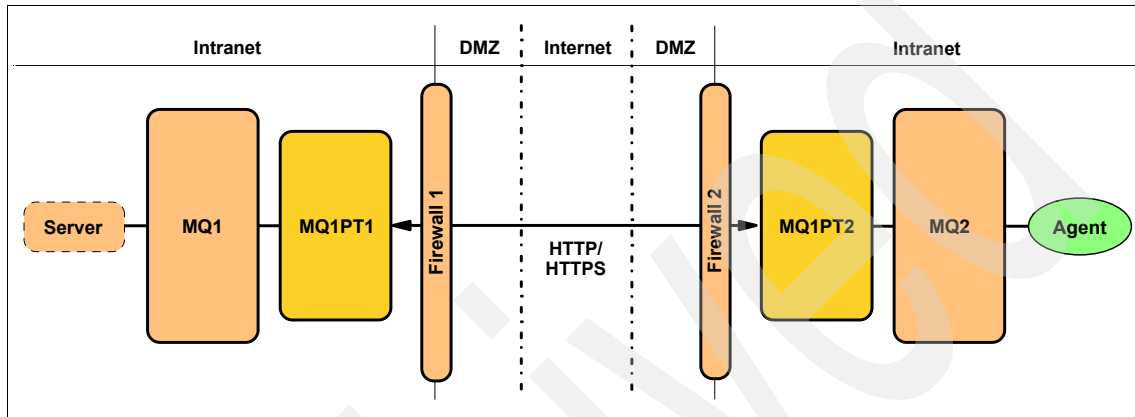


Figure 10-4 Remote deploy using WebSphere MQ Internet Pass-Thru

Firewall 1 needs to be configured to forward all traffic to listener port of MQIPT1 and vice versa.

Configuring routes for MQIPT1

To configure routes for MQIPT1:

► Route1

Set the following parameters:

- ListenerPort = Port on which MQIPT1 is listening for messages from queue manager MQ1
- Destination = Domain name or IP address of MQIPT2
- DestinationPort = Port on which MQIPT2 is listening
- HTTP = true
- HTTPS = true
- HTTPProxy = IP address of firewall 2 (or a proxy server if there is one in the DMZ)
- SSLClient = true
- SSLClientKeyRing = Path to the file that contains the MQIPT1 certificate
- SSLClientKeyRingPW = Path to the file that contains the password for the ClientKeyRing file

- SSLClientCAKeyRing = Path to the file that contains the trusted CA certificates
 - SSLClientCAKeyRingPW = Path to the file that contains the password for the CAKeyRing file
- Route2
- Set the following parameters:
- ListenerPort = Port on which MQIPT1 is listening for messages from MQIPT2
 - Destination = Domain name or IP address for queue manager MQ1
 - DestinationPort = Port on which MQ1 is listening
 - SSLServer = true
 - SSLServerKeyRing = Path to the file that contains the MQIPT1 certificate
 - SSLServerKeyRingPW = Path to the file that contains the password for the ServerKeyRing file
 - SSLServerCAKeyRing = Path to the file that contains the trusted CA certificates
 - SSLServerCAKeyRingPW = Path to the file that contains the password for the CAKeyRing file

Configuring routes for MQIPT2

To configure routes for MQIPT2:

- Route1
- Set the following parameters:
- ListenerPort = Port on which MQIPT2 is listening for MQIPT1
 - Destination = Domain name or IP address of queue manager MQ2
 - DestinationPort = Port on which MQ2 is listening
 - SSLServer = true
 - SSLServerKeyRing = Path to the file that has MQIPT2's certificate
 - SSLServerKeyRingPW = Path to the file that has the password for the ServerKeyRing file
 - SSLServerCAKeyRing = Path to the file that contains the trusted CA certificates
 - SSLServerCAKeyRingPW = Path to the file that contains the password for the CAKeyRing file
- Route2
- Set the following parameters:
- ListenerPort = Port on which MQIPT2 is listening for messages from MQ2
 - Destination = Domain name or IP address of MQIPT1
 - DestinationPort = Port on which MQIPT1 is listening
 - HTTP = true
 - HTTPS = true

- HTTPProxy= IP address of firewall 1 (or a proxy server if there is one in the DMZ)
- SSLClient = true
- SSLClientKeyRing = Path to the file that contains the MQIPT2 certificate
- SSLClientKeyRingPW = Path to the file that contains the password for the ClientKeyRing file
- SSLClientCAKeyRing = Path to the file that has trusted CA certificates
- SSLClientCAKeyRingPW = Path to the file that contains the password for the CAKeyRing file

10.4.3 Configuring the adapter startup in remote environments

The startup configuration of a remote site is almost the same as it is in local environments. See 10.2.1, “Configuring startup in the Windows environment” on page 152 and 10.2.2, “Configuring startup in a UNIX environment” on page 153 for information about how to set up the adapter startup in different environments.

Component deployment

This chapter describes how WebSphere Business Integration Adapters can be deployed in various configurations to support business integration solutions that involve different integration servers. In every case, the components that are used by the adapter at run-time are developed in a consistent manner through the WebSphere Business Integration System Manager perspective within the Eclipse Workbench, although their deployment to the integration server can vary. Also, the application-specific component of the adapter is unchanged in the different deployments. The Adapter Framework is configured to accommodate communications with the appropriate integration server.

11.1 Specifics about component deployment

This section describes the component deployment and provides specifics for each of the integration broker types.

11.1.1 WebSphere InterChange Server

The processes that executes within WebSphere InterChange Server to choreograph automated interactions across distributed systems are called *collaborations*. Collaborations interact with each adapter through the Connector Controller, which is a run-time service that mediates communication between collaborations and adapters.

Although all Connector Controllers are identical except for their configuration, WebSphere InterChange Server instantiates a separate Connector Controller for each adapter. The Connector Controller exchanges business objects with the adapter to support all the interaction patterns described previously. The Connector Controller also exchanges administrative messages with the adapter, such as the adapter status, and commands to pause, startup, or shutdown.

The communication transport between WebSphere InterChange Server and the adapter can be provided through Java Messaging Service (JMS) queues, IIOP, or a combination of the two. When used in combination, JMS is used for asynchronous event notification, and IIOP is used for WebSphere InterChange Server-initiated request processing and administrative messages.

Business object definitions are represented as XML schemas, but because both WebSphere InterChange Server and the adapters use business objects, they are simply serialized in communication. The adapter is configured through the WebSphere Business Integration System Manager perspective within the Eclipse Workbench, which deploys the adapter artifacts together with other integration artifacts to WebSphere InterChange Server. The adapter can be configured to receive all of its configuration information from WebSphere InterChange Server on startup or from a local repository of XML schemas.

Figure 11-1 on page 165 shows how the adapter is deployed with WebSphere InterChange Server.

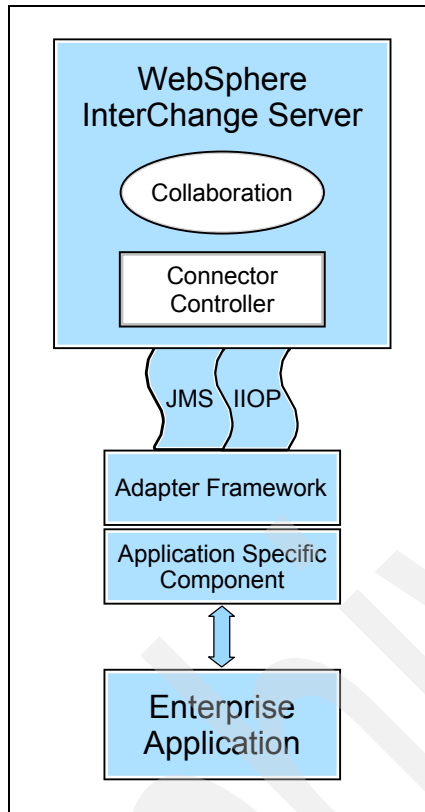


Figure 11-1 Deployment to WebSphere InterChange Server

11.1.2 WebSphere Business Integration Message Broker

Message transformation rules and routing logic are contained in components called *message flows* that execute within the WebSphere message brokers. These message flows include WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker. Message flows are invoked by messages that the message listener receives, and these flows put new messages onto queues. Message flows interact with adapters by exchanging XML messages over WebSphere MQ queues.

The Adapter Framework interacts with the queues through the JMS interface and translates between business objects in memory and the XML message representation of those objects. The business object definitions that are associated with an adapter are represented as XML schemas and are readily imported into the WebSphere Message Broker message repository for use in message flows.

The adapter is configured through the WebSphere Business Integration System Manager perspective within the Eclipse Workbench. The System Manager also provides basic monitoring and administration capabilities. The adapter receives all its configuration data from a local repository of XML schemas upon startup.

Figure 11-2 shows how the adapter is deployed with WebSphere InterChange Server.

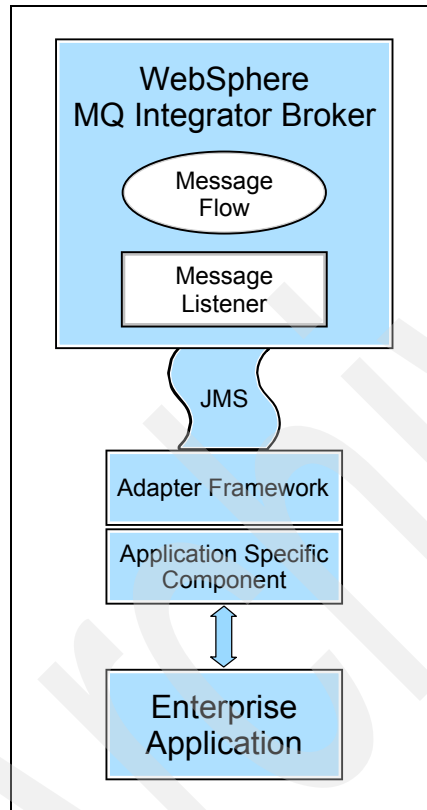


Figure 11-2 Deployment to Message Broker

11.1.3 WebSphere Application Server

WebSphere Application Server Enterprise Edition provides the infrastructure for e-business applications. The e-business application interacts with an adapter by exchanging XML messages over JMS queues. The communications are encapsulated in Session Enterprise JavaBeans™ (EJBs) and Message-Driven Beans (MDBs) to accommodate the various interactions.

The actual interface that is used to make the communication calls and handle message formatting is Web Services Invocation Framework (WSIF). At run-time, WSIF uses Web Services Description Language (WSDL) definitions that are obtained from the adapter. These WSDL definitions describe the adapter interface and configuration (that is, the services that are available).

The adapter is configured through the WebSphere Business Integration System Manager perspective in WebSphere Studio Application Developer Integration Edition, which is based on the Eclipse Workbench. The adapter definition and its associated business object definitions are represented as WSDL and XML schemas and are exported as a Service Project to the Business Integration perspective, where the run-time EJBs and MDBs are generated and deployed to WebSphere Application Server Enterprise.

The WebSphere Administrative Console enables monitoring and administration of the adapters. The WebSphere Business Integration System Manager perspective also provides basic monitoring and administration capabilities.

The Adapter Framework interacts with the message queues through the JMS interface and translates between business objects in memory and the XML message representation of those objects. The adapter receives all of its configuration information from a local repository of XML schemas upon startup.

Figure 11-3 on page 168 shows how the adapter is deployed with WebSphere InterChange Server.

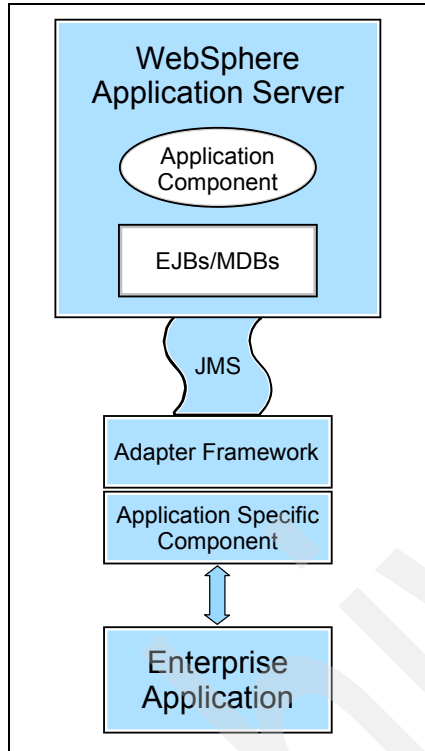


Figure 11-3 Deployment to WebSphere Application Server



Part 2

Developing our custom adapter

ARCHIVED



Setting up the development environment

This chapter describes the development environment that you need to create a custom adapter. It includes installation instructions as well as component information.

12.1 Installing the necessary components

The custom adapter that we built Java-based. For the development environment, you need the following:

- ▶ Integrated Development Environment (IDE)
In our scenario, we used WebSphere Studio Application Developer Integration Edition V5.0.1.
- ▶ Java Development Kit
In our scenario, we used IBM JDK 1.3.1. If you are developing a C or C++ adapter, then you need an appropriate C or C++ compiler.
- ▶ WebSphere MQ v5.3.0.2 with CSD05

Important: If you do not apply CSD05, the connectors will not initialize.

- ▶ WebSphere Business Integration Adapter Framework.
In our scenario, we used Version 2.4 in this implementation.
- ▶ WebSphere Business Integration Adapter Development Kit (optional).

12.1.1 Installing WebSphere Studio Application Developer Integration Edition V5.0.1

Install WebSphere Studio Application Developer Integration Edition V5.0.1 according to the product instructions.

12.1.2 Installing IBM JDK 1.3.1

Install IBM JDK 1.3.1 according to the product instructions.

12.1.3 Installing WebSphere MQ v5.3.0.2

Install WebSphere MQ v5.3.0.2 according to the product instructions. Install CSD05 according to the product instructions.

In our scenario, we chose a Typical install with the following installation path:

C:\Program Files\IBM\WebSphere MQ

12.1.4 Installing WebSphere Business Integration Adapter Framework V2.4

WebSphere Business Integration Adapter Framework is included with WebSphere InterChange Server.

Important: Do not install WebSphere Business Integration Adapter Framework on the same computer as WebSphere InterChange Server.

To install the Adapter Framework:

1. Navigate to the Microsoft Windows directory of the installation media and double-click setupwin32.exe to launch the WebSphere Business Integration Adapter Framework Installer, as shown in Figure 12-1.

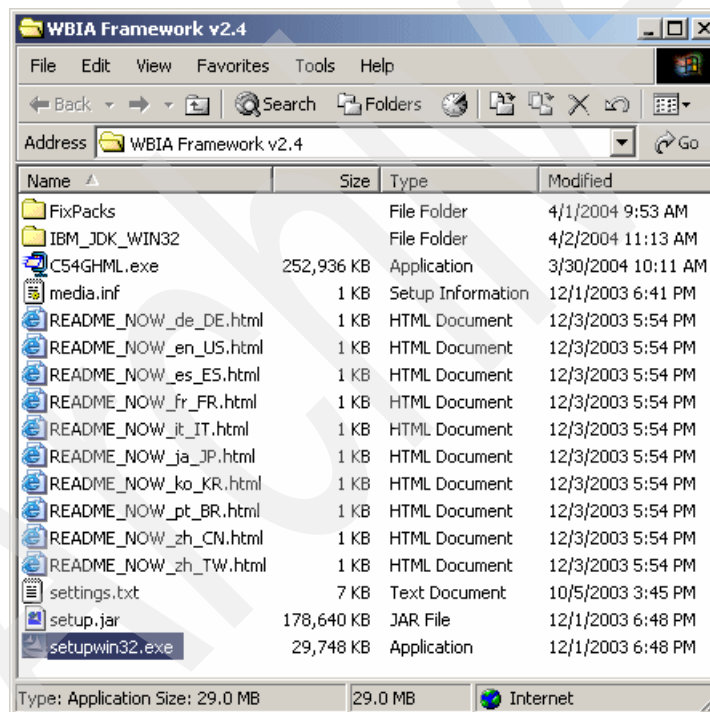


Figure 12-1 Installation for WebSphere Business Integration Adapter Framework

2. Choose the language for the product and click **OK** (Figure 12-2 on page 174).



Figure 12-2 Select your language

3. When you see the Welcome screen that is shown in Figure 12-3, click **Next** to continue.



Figure 12-3 Welcome screen

4. Accept the terms of the license agreement and click **Next** to continue, as shown in Figure 12-4.

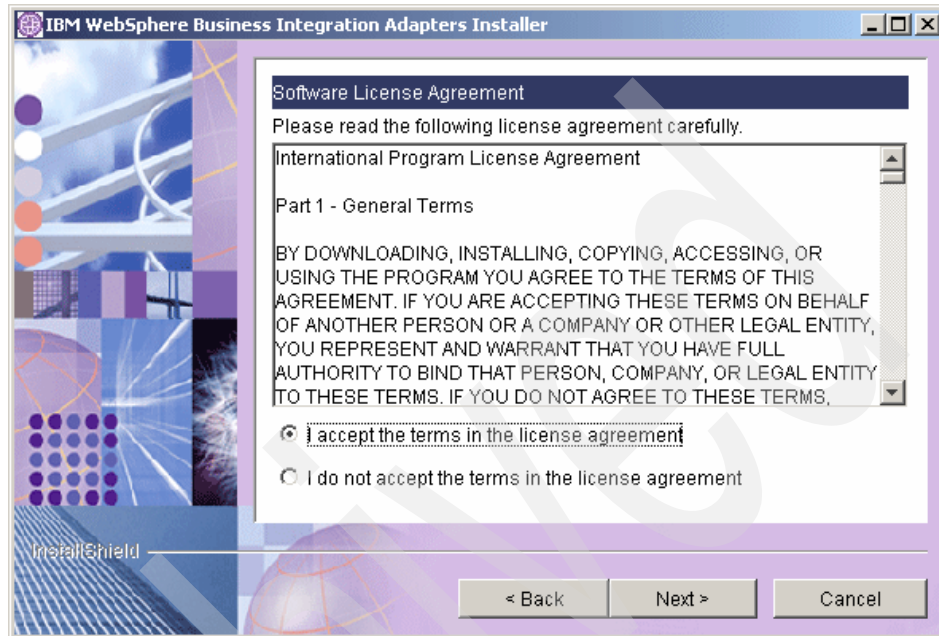


Figure 12-4 License agreement

5. Enter the directory location that you want for the product installation and click **Next** to continue, as shown in Figure 12-5. In our scenario, we used the default location. If you choose another directory, do not use spaces in your directory path.

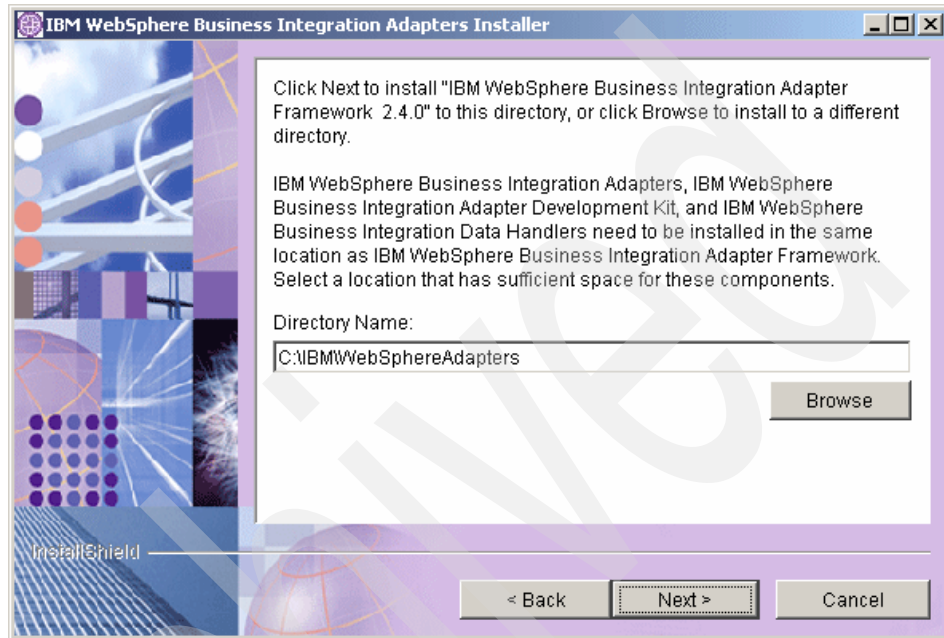


Figure 12-5 Install directory location

6. Review the summary screen (Figure 12-6) and click **Next** to continue.

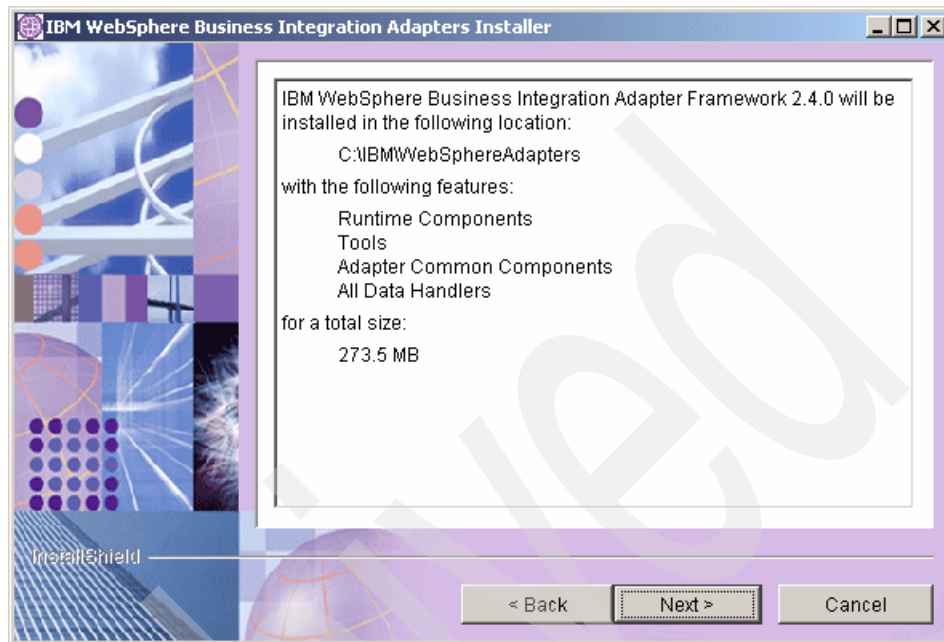


Figure 12-6 Summary screen

7. Enter the location of the WebSphere MQ Java directory and click **Next** to continue, as shown in Figure 12-7.

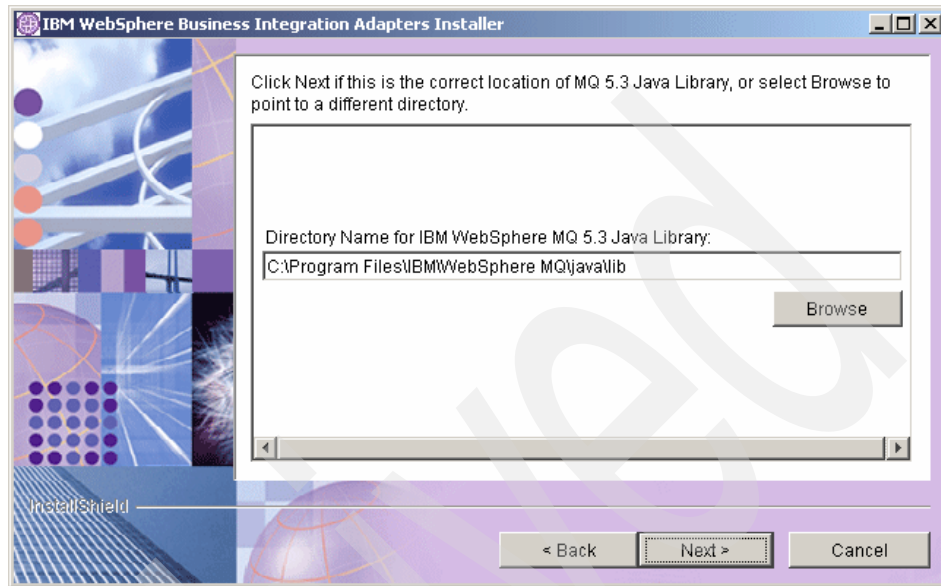


Figure 12-7 WebSphere MQ Java location

8. Choose whether to install into an existing WebSphere Studio Application Developers Integration Edition environment or to a separate WebSphere Studio Workbench. Click **Next** to continue.

We chose to install into our existing WebSphere Studio Application Developer Integration Edition environment, as shown in Figure 12-8.

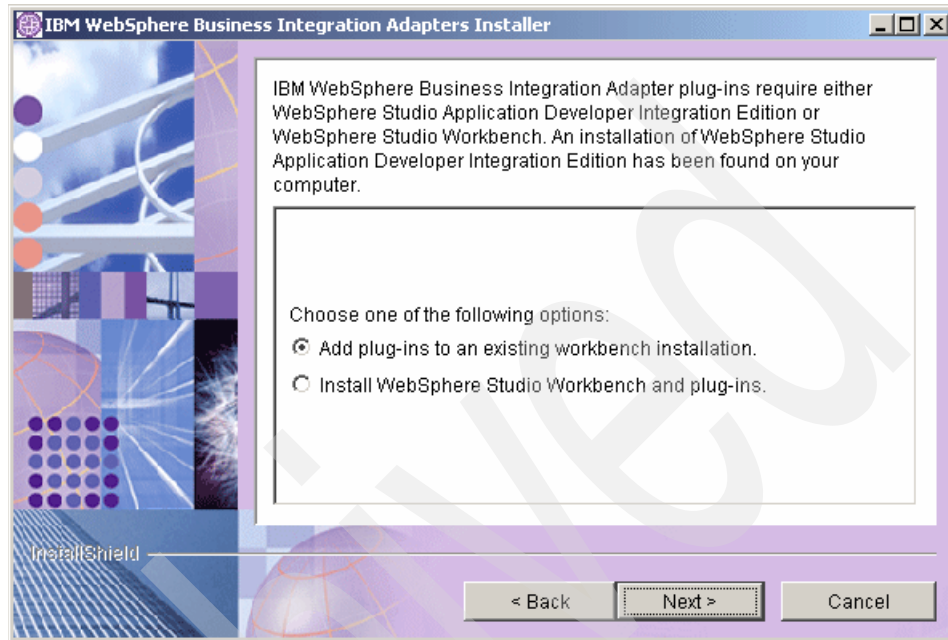


Figure 12-8 Choose WebSphere Studio preference

9. If you choose to install into an existing WebSphere Studio Application Developer Integration Edition environment, enter the location of the product (Figure 12-9). Click **Next** to continue.

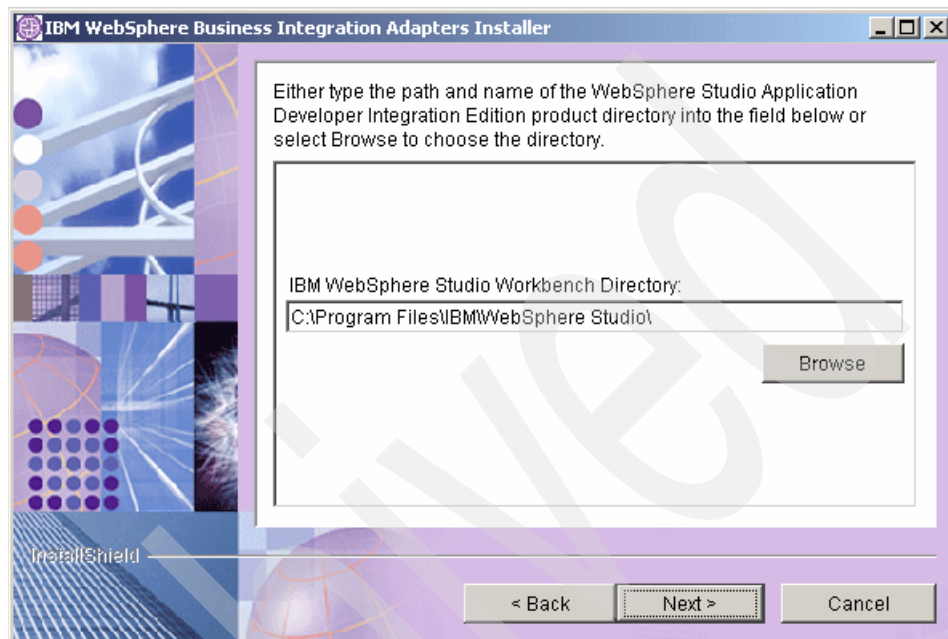


Figure 12-9 WebSphere Studio Application Developer Integration Edition location

10. Enter the Program Group you in which you want the Program icons to be included. We chose the default Program Group, as shown in Figure 12-10.

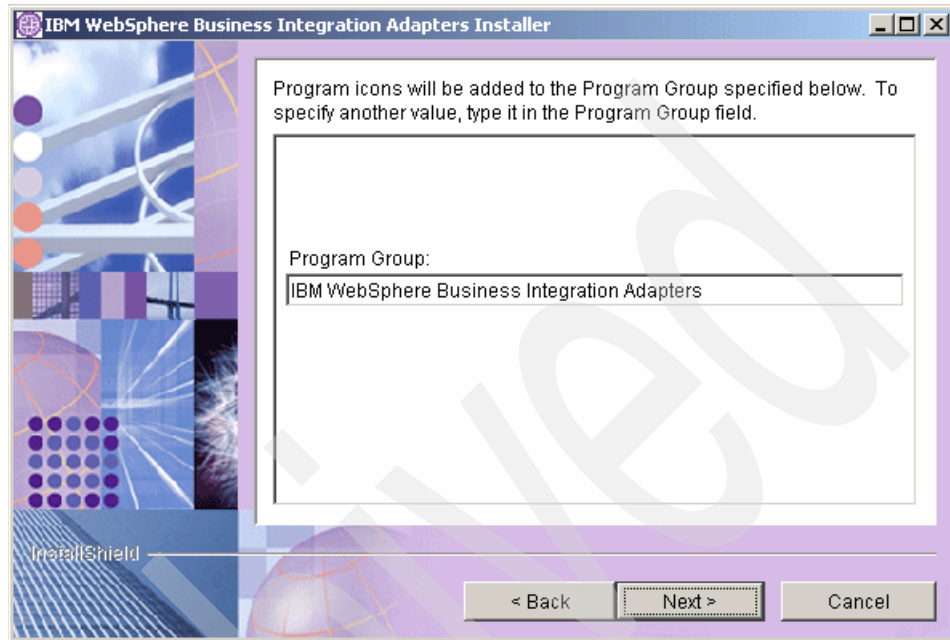


Figure 12-10 Enter Program Group

11. Click **Next** to continue.

Installing the product might take a few minutes, during which the installation screen displays, as shown in Figure 12-11.

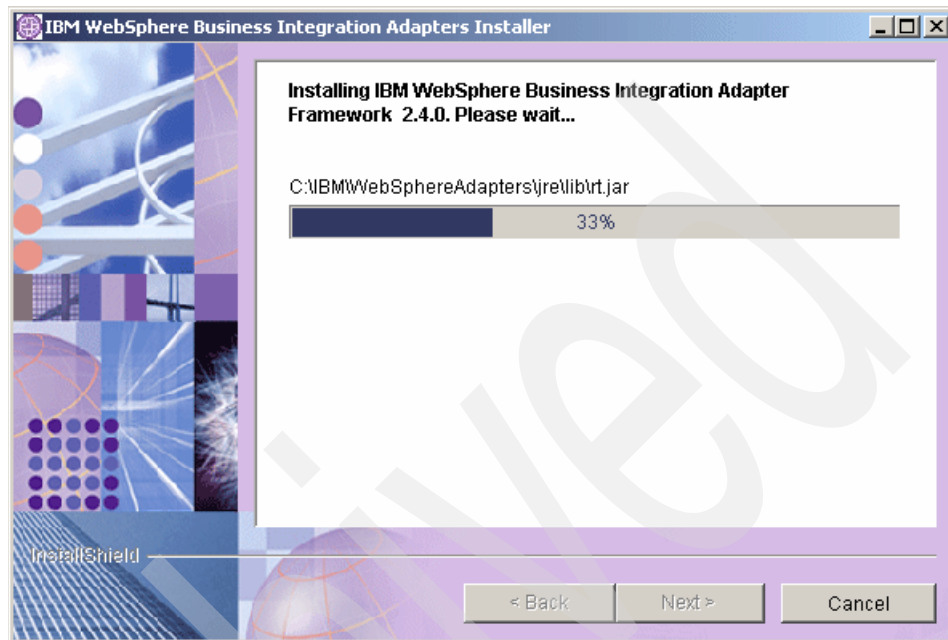


Figure 12-11 Install screen

12. Click **Finish** to complete the installation (Figure 12-12).

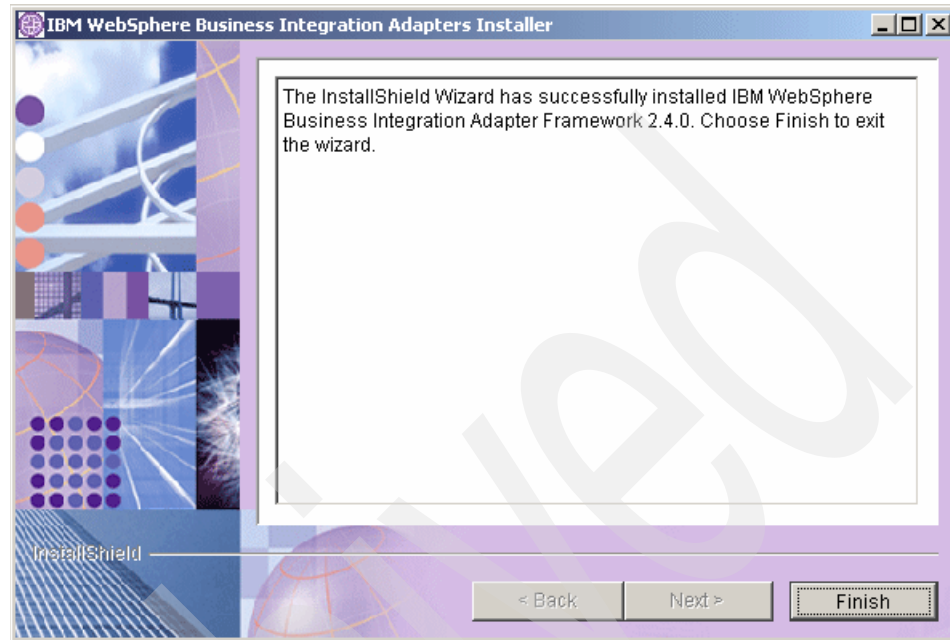


Figure 12-12 Installation complete

12.1.5 Installing WebSphere Business Integration Adapter Development Kit V2.4

Installing the WebSphere Business Integration Adapter Development Kit V2.4 is optional. The WebSphere Business Integration Adapter Development Kit provides samples to ease development.

To install the development kit:

1. Navigate to the Windows directory of the installation media and double-click setupwin32.exe to launch the WebSphere Business Integration Adapter Development Kit Installer, as shown in Figure 12-13.

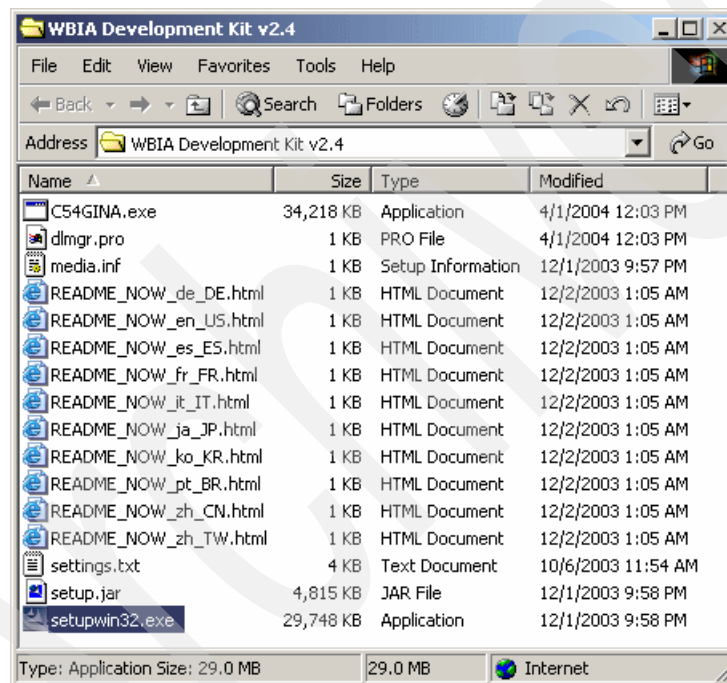


Figure 12-13 Installation for WebSphere Business Integration Adapter Development Kit

2. Choose the language for the product and click **OK** (Figure 12-14).



Figure 12-14 Select your language

3. When you see the Welcome screen, click **Next** to continue (Figure 12-15).



Figure 12-15 Welcome screen

4. Accept the terms of the license agreement and click **Next** to continue, as shown in Figure 12-16.

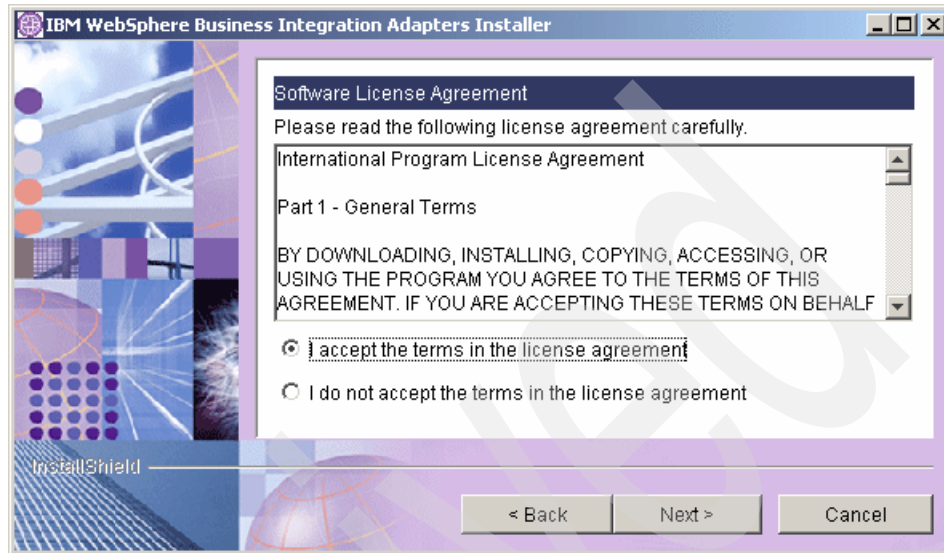


Figure 12-16 License agreement

5. Enter the directory location that you want for the product installation and click **Next** to continue. We used the default location, as shown in Figure 12-17.

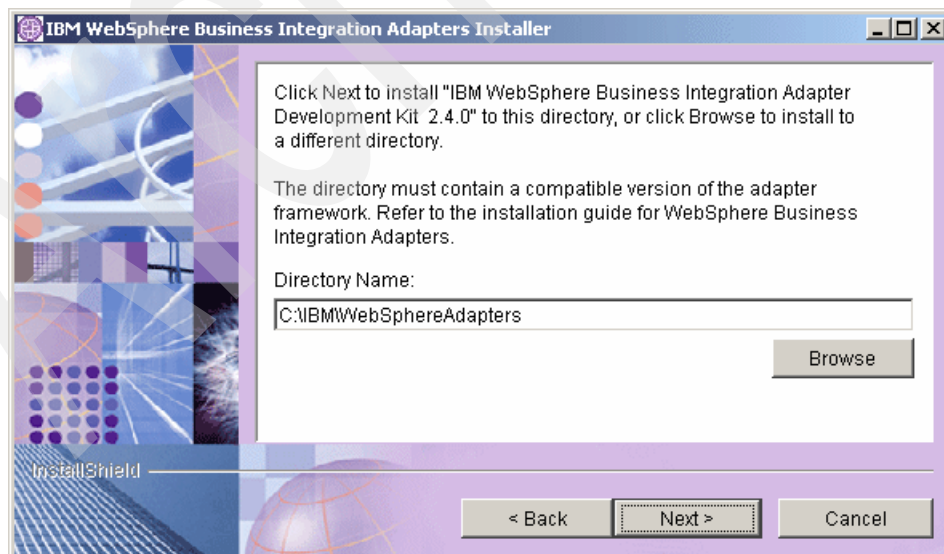


Figure 12-17 Install directory location

Note: The install location must contain an installation of a compatible version of WebSphere Business Integration Adapter Framework.

6. Review the summary screen and click **Next** to continue, as shown in Figure 12-18.

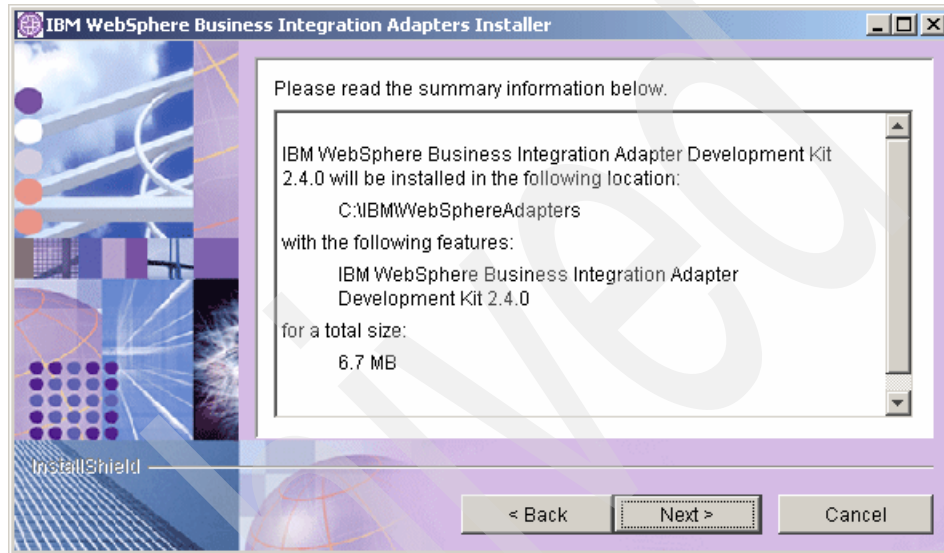


Figure 12-18 Summary screen

This installation might take a few minutes, during which the installation screen displays, as shown in Figure 12-19 on page 188.

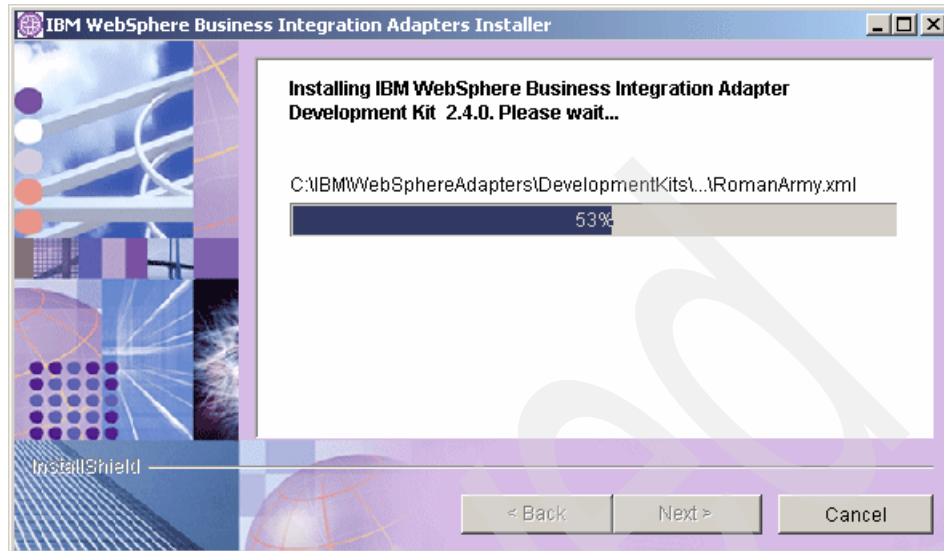


Figure 12-19 Install screen

7. Click **Finish** to exit the installation wizard (Figure 12-20).

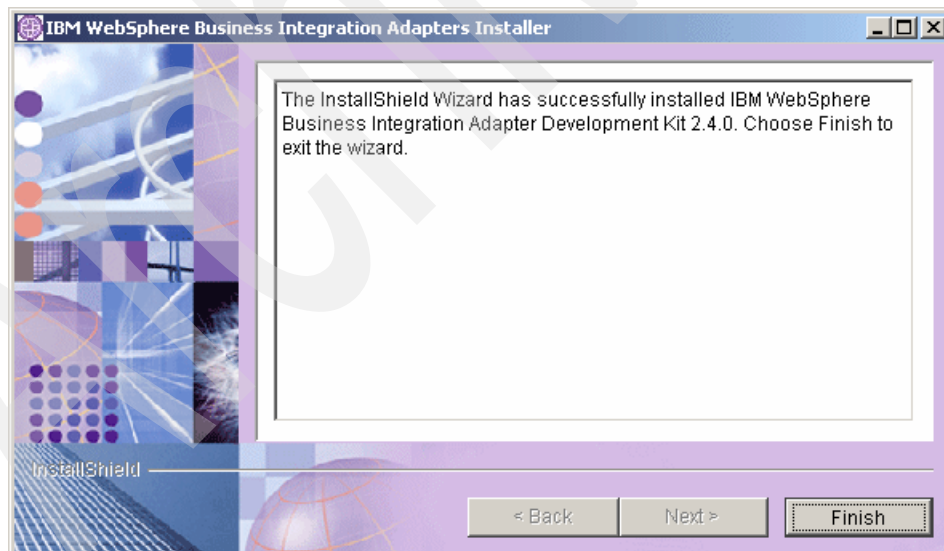


Figure 12-20 Exit the installation wizard

8. Click **Finish** to complete the installation (Figure 12-21 on page 189).

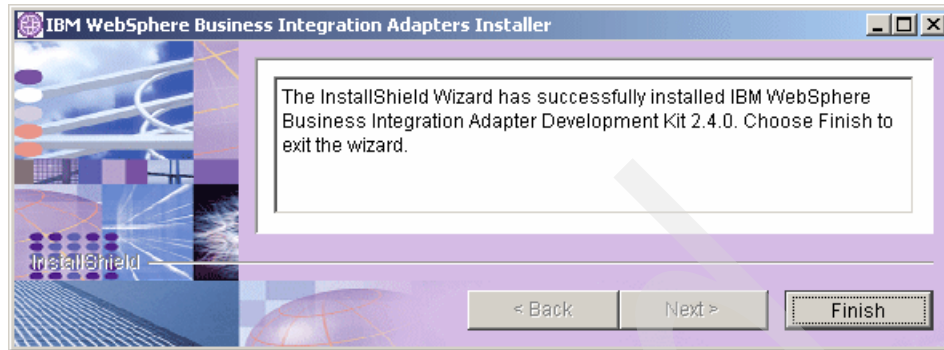


Figure 12-21 Installation complete

You are now ready to begin developing the adapter.

12.2 Developing the adapter

In the WebSphere Business Integration system, the connector is the *representative* of the application (that is, it performs tasks in support of the application). For example, the connector might poll the application for events such as changes to the application and, in support, retrieve that changed data from the application to send to the integration broker.

The connector might also perform tasks as a representative of the integration broker. For example, the connector could request modifications to application data on request from the integration broker. The two main components of a connector are:

- ▶ Adapter Framework
- ▶ Application-specific component

Figure 12-22 on page 190 shows, at a high-level, the components that make up a connector that represents the application or the integration broker.

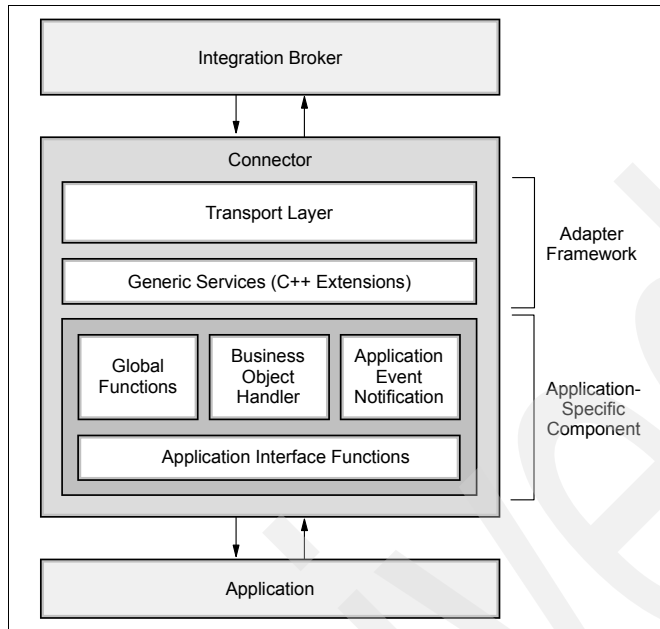


Figure 12-22 Connector components

12.2.1 Adapter Framework

The Adapter Framework is provided as a component of the WebSphere Business Integration Adapters. This component communicates with the integration broker and provides the following services:

- Connector Controller

Note: The Connector Controller is for the InterChange Server only.

- Transport layer
 - Handles the exchange of business objects between the connector and the integration broker.
 - For the InterChange Server, the transport layer handles the exchange of startup and administrative messages between the Connector Controller and the client Adapter Framework.
 - Keeps a list of all subscribed business objects

- Java connector library
 - Provides generic services to the application-specific component in the form of Java classes and methods.

As shown in Figure 12-23, if you are using the Message Broker or WebSphere Application Server as your integration broker, the Adapter Framework resides entirely on the adapter machine (that is, the Adapter Framework is contained entirely within the connector itself).

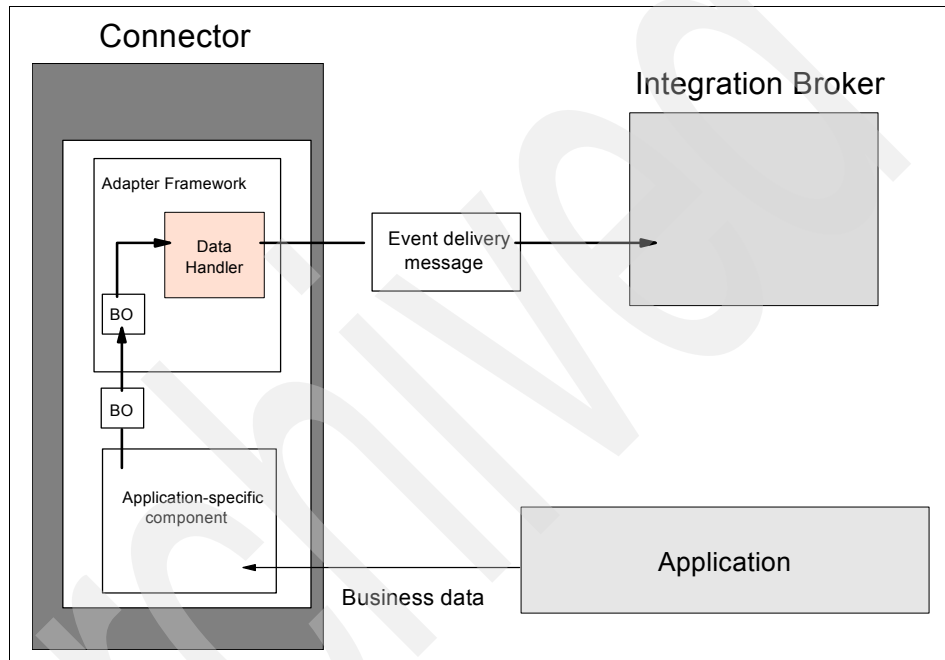


Figure 12-23 High-level architecture with a Message Broker

If you are using the InterChange Server (as shown in Figure 12-24 on page 192) as the integration broker, the Adapter Framework is distributed to take advantage of certain InterChange Server functions.

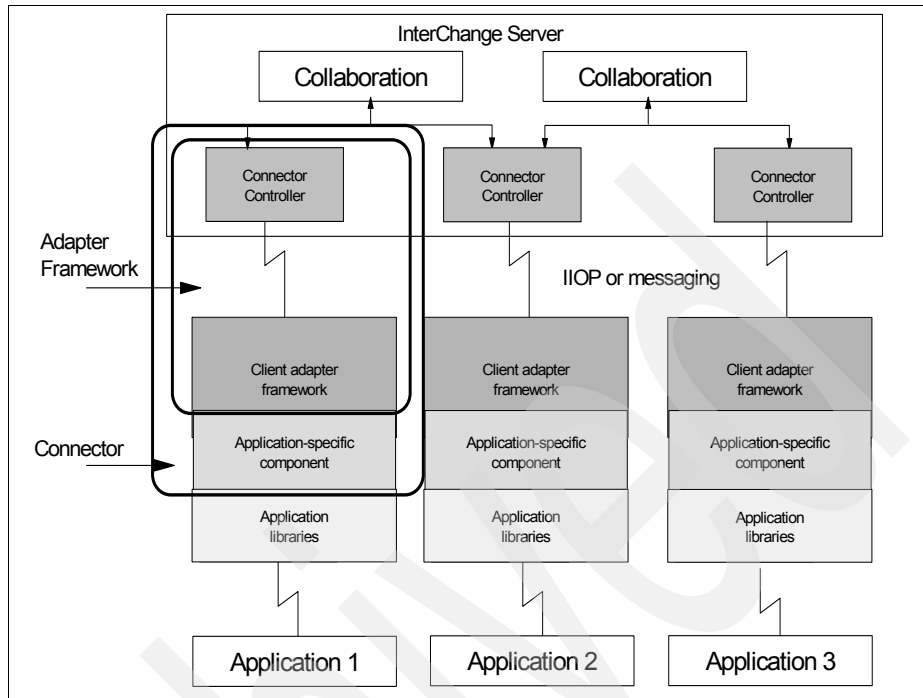


Figure 12-24 High-level architecture with the InterChange Server

The distributed Adapter Framework consists of two components:

- **Client Adapter Framework**

Part of the connector process on the client or connector machine, it supports the transport layer and the Java connector library components of the Adapter Framework.

- **Connector Controller**

Running within the InterChange Server itself, the Connector Controller manages communications between the Adapter Framework and collaborations. A Connector Controller is instantiated by the InterChange Server for each connector that has been defined in the InterChange Server repository. This component is internal to the InterChange Server and coding for the Connector Controller is not required.

We provide this information as background only. No coding is required for the Adapter Framework, regardless of your integration broker type.

12.2.2 Application-specific component

As the name implies, the application-specific component contains the code you write for actions that are application-specific. This code includes tasks such as initialization and setup methods, handling of business objects, event notification, and polling. The application-specific component is the part of a connector that you design and code, because it contains code that is tailored specifically to communicate with and represent your application to the integration broker.

The application-specific components (as shown in Figure 12-22 on page 190) include:

- ▶ A connector base class, which initializes and sets up the connector.
- ▶ A business object handler, which responds to request business objects that are initiated by the integration broker requests.
- ▶ An event notification mechanism, which detects and responds to application events.

Because our API is written in Java, we wrote the application-specific portion of our connector in Java, accessing the services of the Adapter Framework through the Java connector library.

12.2.3 Developing the application-specific component

Proceed with connector development following this series of steps for the application-specific component are:

1. Identify the application entities that the connector will make available to other applications through the integration broker.
2. Create an ODA for ease of application-specific business object creation.
3. Create the required application-specific business objects.

When using the InterChange Server as the integration broker, identify or create the generic business objects (GBOs) for use in the collaborations. We detail our GBOs as part of our InterChange Server-specific implementation.

4. Define a connector base class for the application-specific component to implement functions that initialize and terminate the connector.
5. Define a business object handler class and one or more business object handlers to handle requests.
6. Define the mechanism to detect application events.
7. Implement a mechanism to support event subscriptions and event polling.

Important: It is essential for each of these steps that you implement error handling and correct logging and error messages for all of the connector methods.

8. Build the connector.
9. Unit test the connector.
10. Configure the connector for use with the brokers.
11. Test and debug the connector with the broker.



Adapter design and environment

This chapter discusses design requirements and the environment for our test scenario. It identifies the type of processing that our adapter is required to perform and the business entities or *objects* that need to be accounted for to enable integration.

13.1 Understanding our scenario environment

When you begin adapter development, it is crucial to understand the design requirements for integration. In our scenario, the company is a rental management company that runs all of the operations for tenant maintenance through a call center. Management wants to scale back call center operations because they are proving to be costly.

Currently, the call center operators access each of the supporting applications separately. This environment 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.

In this environment, if the company's own tradespeople cannot carry out the work when maintenance requests are created, the call-center operators must contact the external contracting company and then update the details in the RedMaintenance application manually — a time-consuming and potentially inconsistent method for updating data. The call center operators must then contact the in-house maintenance teams and external contractors to update the progress of maintenance requests.

Figure 13-1 illustrates the current environment.

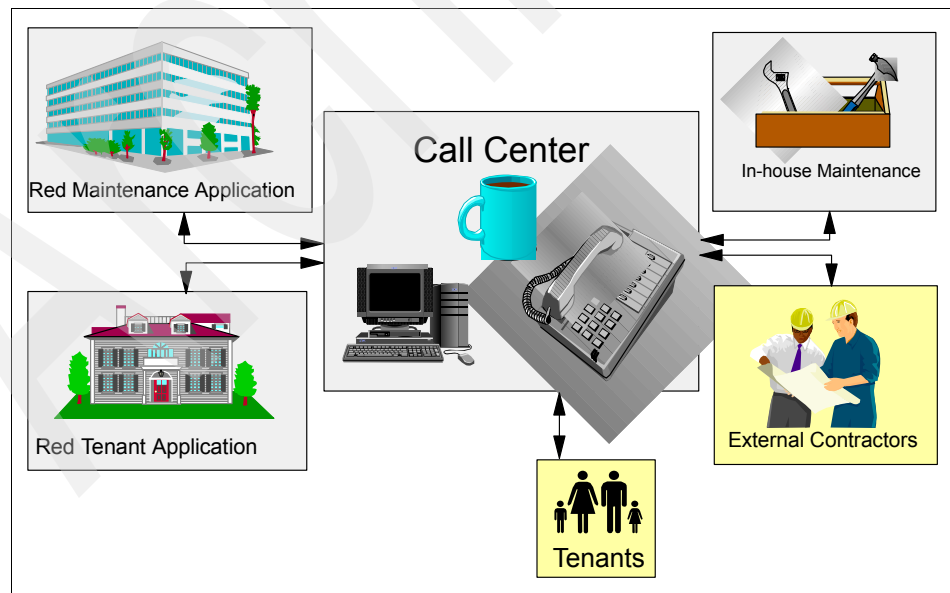


Figure 13-1 Current call center environment

13.1.1 The requirements

Management requires that the solution:

- ▶ Provide as much automation and synchronization as possible.
- ▶ Give the tenants the ability to query and request their own maintenance using the Internet.
- ▶ Is extensible enough so that, at some point in the future, tenants can use e-mail to request maintenance or to receive updates.

In addition, management also requests the following restrictions:

- ▶ The solution cannot alter existing applications.
- ▶ The solution must use an integration broker for any data modification, enhancement, or semantic mediation.

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.

In our scenario, the two applications will be integrated using a combination of adapters and the WebSphere integration brokers.

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

Because the back-end application has an exposed API, we use a custom adapter which was built for the back-end application. This custom 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 using the exposed API.

Figure 13-2 illustrates the complete integration scenario for the company in our scenario.

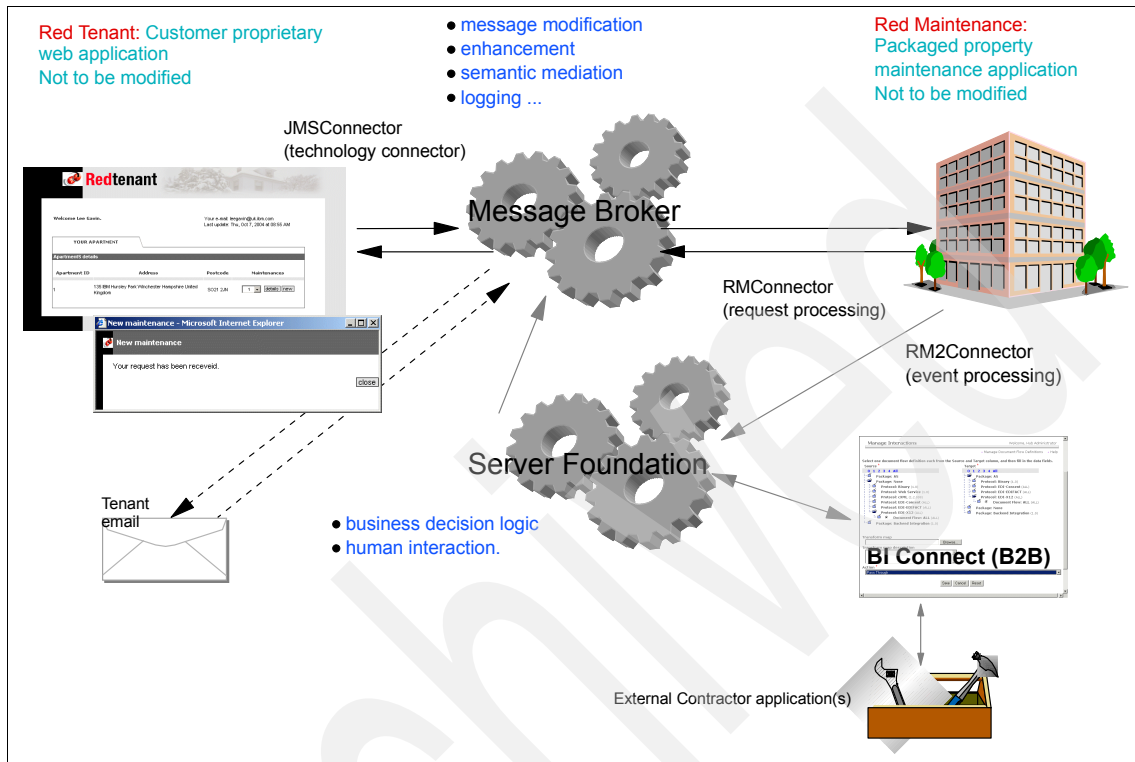


Figure 13-2 Complete integration solution

From this point, we begin the technical assessment for the feasibility of a connector development project.

13.2 Examining the application environment

As documented in Chapter 1, “Adapter development” on page 3, you can use a list of topics and questions to obtain an understanding of the aspects of an application that affect connector development. Some of the key points about our major application are as follows:

1. Operating system:
 - Supports Java 1.3.2 or above and UDB
 - Runs on a Windows platform

2. Programming language:
 - Application written in Java
3. Application execution architecture:
 - Multi-threaded application using an RMI communications mechanism for connectivity
4. Database Type:
 - IBM DB2® UDB
 - Relational
5. Distributed application:
 - Is the application distributed across multiple servers?
Potentially, it will in the future. Currently, it does not.
 - Is the application database distributed across multiple servers?
Potentially, it will in the future. Currently, it does not.

Early in the project planning phase, we need to determine what roles the connector will perform for the application:

- ▶ Request processing
Update application data at the request of an integration broker.
- ▶ Event notification
Detect application events and send notification of events to the integration broker.

To be **bidirectional**, the connector needs to support both event notification and request processing. Our application does.

13.3 Examining the API

Ideally, an application provides an API that includes all of the following features:

- ▶ Support for CRUD operations at the object level
- ▶ Encapsulation of all of the application business logic
- ▶ Support for delta and after-image operations

The recommended approach to connector development is to use the API that the application provides. The use of an API helps ensure that connector interactions with the application abide by application business logic. In particular, a high-level API is usually designed to include support for the business logic in the application. A low-level API might bypass application business logic.

As an example, a high-level API call to create a new record in a database table might evaluate the input data against a range of values, or it might update several associated tables as well as the specified table. Using SQL statements to write directly to the database, on the other hand, might bypass the data evaluation and related table updates performed by an API.

If no API is provided, the application might allow its clients to access its database directly using SQL statements. If you use SQL statements to update application data, work closely with someone who knows the application well so that you can be sure that your connector will not bypass application business logic. This aspect of the application has a major impact on connector design because it affects the amount of coding that the connector requires. The easiest application for connector development is one that interacts with its database through a high-level API. If the application provides a low-level API or has no API, the connector will probably require more coding.

The following questions to ask about the API can help you decide if it is a good candidate for an adapter:

1. Is there an existing mechanism that the connector can use to communicate with the application?

Yes

2. Does the API allow access for CRUD operations?

Yes:

- Create, Retrieve, and Update are possible.
- Delete.

There is no physical delete. A delete request would result in the status of cancelled or deleted only.

3. Does the API provide access to all attributes of a data entity?

No, some data entities are internal to the application.

4. Are there inconsistencies in the API implementation?

No

5. Is the navigation to the CRUD functions the same regardless of the entity?

Yes

6. Describe the transaction behavior of the API.

Each Create, Update, or Delete operation operates within its own transactional unit.

7. Does the API allow access to the application for event detection?

Yes

8. Is the API suited for metadata design? APIs that are forms-, table-, or object-based are good candidates.

Yes. All APIs are object-based.

9. Does the API enforce application business rules?

Yes. The API interacts at object level.

13.4 Designing application-specific business objects

Application-specific business objects (ASBOs) are the units of work that are triggered within the application, created and processed by the connector, and sent to the integration broker. A connector uses these business objects to export data from its application to other applications and to import data from other applications. The connector exposes all the information about an application entity that is necessary to allow other applications to share the data. After the connector makes the entity available to other applications, the integration broker can route the data to any number of other applications through their connectors.

Designing the relationship between the connector and its supported ASBO is one of the tasks of adapter development. ASBO design can generate requirements for connector programming logic that must be integrated into the connector development process. Therefore, business object and connector developers must work together to develop specifications for the connector and its business objects. The back-end application works with objects only, no direct database access is allowed. We need to ensure that our business object definitions meet the requirements for the application.

13.5 Conclusion

The back-end application is an excellent candidate for a metadata-driven adapter. By using a series of connectors and a broker hub for all of your connected applications, you are enabling more controlled automating and synchronizing of data. You can also enable expansion of the solution to include e-mail by simply adding an e-mail connector into the application, connector and broker network.



Object Discovery Agent

This chapter describes how to build the Object Discover Agent (ODA) for the custom adapter.

14.1 Developing the ODA

The ODA for this scenario is developed using WebSphere Studio.

14.1.1 Setting up the development environment

When you develop and test the ODA, specify the necessary *.jar files as follows:

1. Open the pop-up menu by clicking the right mouse button on your project.
2. Select **Properties** (Figure 14-1).

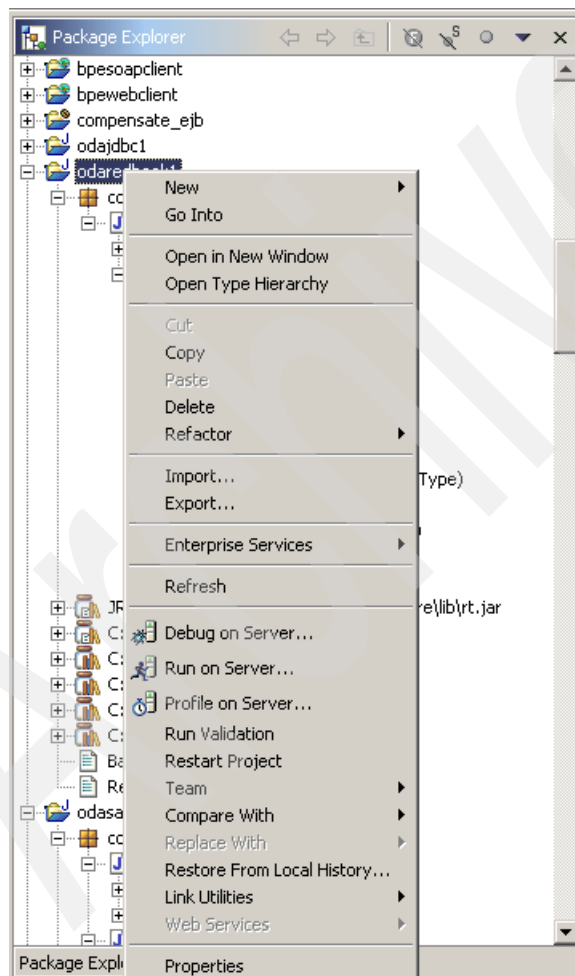


Figure 14-1 Project properties

The Java Build Path dialog box displays, as shown in Figure 14-2.

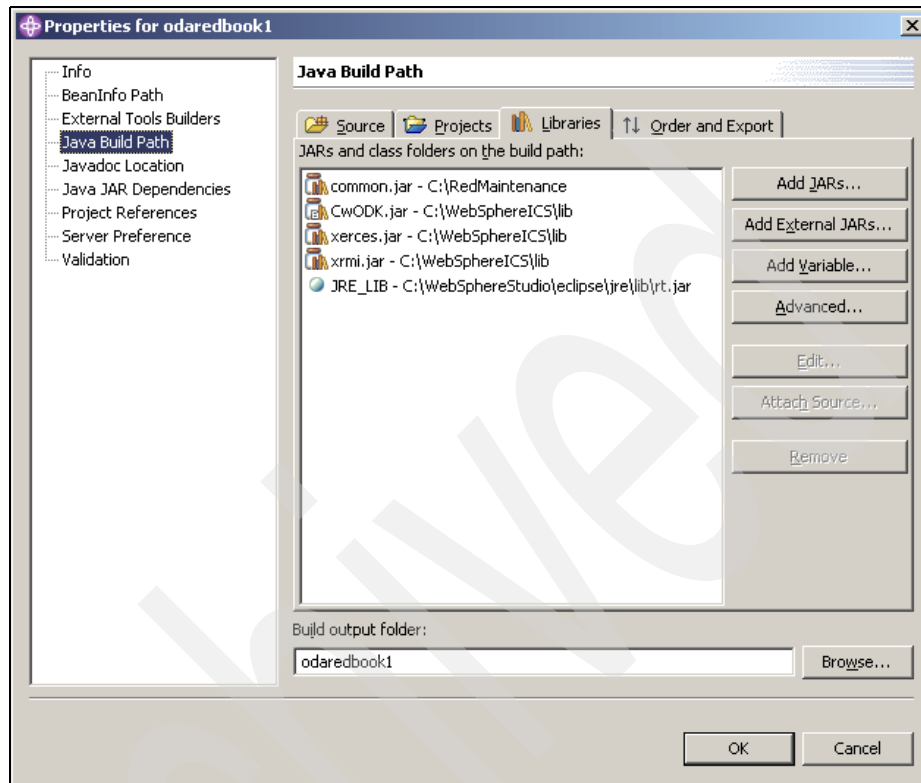


Figure 14-2 Java Build Path dialog box

3. Select the Libraries tab and click **Add External JARS**. Specify the following necessary .jar files in the \lib\ subdirectory:

- CwODK.jar
- xerces.jar
- xrmijar

You also need the common*.jar file for they back-end application that the ODA uses. This file is located at the IBM Redbooks Web site in the SG246345 folder of the Additional Materials (see “Locating the Web material” on page 887 for further information).

14.1.2 Naming conventions for the ODA

Naming conventions provide a way to make your ODA code easier to locate and identify. This list shows a suggested naming convention for an ODA:

ODA name	srcDataNameODA
ODA package and class name	com.ibm.oda.srcDataName.ODAName
ODA startup script	start_ODAName
ODA library file	ODAName.jar
ODA run-time directory	ODA\srcDataName

Give each ODA a name that uniquely identifies it within the WebSphere business integration system. By convention, an ODA name (ODAName) takes the following form:

srcDataNameODA

The unique string, *srcDataName*, is a unique string that identifies the source data or application that the ODA converts.

14.2 Implementing the ODA

This section discusses the steps in creating the ODA.

14.2.1 Extending the ODA base class

The `ODKAgentBase2` class, which is the ODA base class, includes methods for initialization, setup, and termination of the ODA. To implement your ODA, extend this ODA base class to create an ODA class.

To derive an ODA class, do the following:

1. Create the ODA class `RedMaintenanceAgent` that extends the `ODKAgentBase2` class.
2. Define your package name package, `com.ibm.itso.rm.oda.RedHouse`, in the ODA class file.
3. Define `import com.crossworlds.ODK.*;` in the ODA class file to access the methods of the ODK API. Also, define the other imports for your back-end API.
4. Implement the abstract methods of the `ODKAgentBase2` class for your ODA class as shown in Table 14-1 on page 207.

Table 14-1 Abstract ODKAgentBase2 methods

Abstract ODKAgentBase2 method	Description
getAgentProperties()	This method performs the following tasks: <ul style="list-style-type: none"> ► Define the configuration properties needed to initialize the ODA. ► Send the configuration properties in an array to the Business Object Wizard.
getMetaData()	Instantiate the AgentMetaData object that contains the ODA's metadata
init()	Initialize the ODA
terminate()	Perform cleanup

5. Implement the methods of the content-generation interface in the ODA class as in Table 14-2.

Table 14-2 Content-generation interface

Content-generation Interface	Methods
IGeneratesBoDefs	getContentProtocol() getTreeNodes() generateBoDefs() getBoDefs()

14.2.2 Obtaining the handle to the ODKUtility object

The ODKUtility object provides the ODA code with access to the following:

- Objects in the memory of the ODA run-time, such as configuration properties and business-object properties.
- Utility methods that provide tracing and display user-response dialog boxes.

Declare the handle to the ODKUtility object as global to the entire ODA class, so all methods within this class can access the utility methods. Define a member variable named `m_utility` in its ODA class and initialize it as follows:

```
ODKUtility m_utility = ODKUtility.getODKUtility();
```

14.2.3 Initializing the configuration-property array

The Business Object Wizard uses the configuration-property array, `getAgentProperties()`. It returns to initialize the Configure Agent dialog box. This dialog box displays all ODA configuration properties and allows users to enter or change their values. The configuration-property array is an array of

AgentProperty objects. The `getAgentProperties()` sends an array of AgentProperty objects that describe the ODA configuration properties to the Business Object Wizard.

To initialize the configuration-property array in `getAgentProperties()`:

1. Instantiate an AgentProperty object for a configuration property, initializing it with the appropriate property information.
2. Store the initialized AgentProperty object in the configuration-property array.
3. Return the initialized configuration-property array from the `getAgentProperties()` method.

We need to display the host name or IP address of the back-end application in the Business Object Wizard, in which users can specify it. Implement the `getAgentProperties()` method as shown in Example 14-1.

Example 14-1 GetAgentProperties() method

```
public AgentProperty[] getAgentProperties() throws
com.crossworlds.ODK.ODKException
{
    AgentProperty aphost = new AgentProperty("Host",
AgentProperty.TYPE_STRING, "Host name or IP address of RedMaintenance
application", true, false, ODKConstant.SINGLE_CARD, null, null);
    AgentProperty[] props = new AgentProperty[] {aphost};
    return props;
}
```

14.2.4 Initializing ODA metadata

After the Business Object Wizard calls the `getAgentProperties()` method, it calls the `getMetaData()` method to initialize the ODA metadata. The `getMetaData()` method is defined in the ODA base class, `ODKAgentBase2`, then inherited by the ODA class. It returns an `AgentMetaData` object, which contains the ODA metadata.

To initialize the ODA metadata, implement the `getMetaData()` method by doing the following:

1. Create an instance of the `AgentMetaData` class, passing in a reference to the ODA itself and an optional ODA version.
2. Return the initialized `AgentMetaData` object from the `getMetaData()` method.

Example 14-2 shows the `getMetaData()` method for our scenario.

Example 14-2 getMetaData() method

```
public AgentMetaData getMetaData()
{
    AgentMetaData amd = new AgentMetaData(this, "Red Maintenance
application ODA v. 0.0.1");
    return amd;
}
```

14.2.5 Initializing the ODA startup

After the Business Object Wizard calls the ODA `getMetaData()` method, it calls the `init()` method to begin initialization of the ODAstart. The `init()` method is part of the low-level ODA base class, `ODKAgentBase`. It is inherited by the ODA base class, `ODKAgentBase2`, then inherited in turn by the ODA class. This method performs initialization steps for the ODA.

Implement the `init()` method, which performs the following tasks:

1. Retrieving all configuration properties into a Java hash table object by the `getAllAgentProperties()` method, defined in the `ODKUtility` class.
2. Retrieving the Host element from the hash table by the `get()` method that is defined in the Java hash table class.
3. Getting the value that the user has specified for the configuration property Host by the `allValues` member variable, defined in the `AgentProperty` class.
4. Getting the RMI interface object for our back-end application with the Host value by the `Naming.lookup` method.
5. Clearing `m_generatedB0s` vector by the `clear` method defined in the Java vector class.

Example 14-3 shows the `init()` method.

Example 14-3 Init() method

```
public void init() throws com.crossworlds.ODK.ODKException
{
    Hashtable h = m_utility.getAllAgentProperties();
    AgentProperty prophost = (AgentProperty)h.get("Host");
    String host = prophost.allValues[0].toString();
    String name = "/" + host + "/rm";
    try {
        if (System.getSecurityManager() == null)
            System.setSecurityManager(new RMISecurityManager());
    }
```

```

        appI = (RMSEServerRMIInterface) Naming.lookup(name);
        System.out.println("Found RMI application");
    } catch (Exception e) {
        e.printStackTrace();
    }
    m_generatedBOs.clear();
}

```

14.2.6 Choosing the ODA content protocol

An ODA can generate a particular content type using either of the content protocols listed in Table 14-3.

Table 14-3 Content protocol

Content protocol	How to call the content-generation method	Implementation of content-generation method
On request	Business Object Wizard explicitly calls the content-generation method to initiate content generation	Method must generate content for the source nodes passed in its argument and return the appropriate content to the Business Object Wizard
Callback	Business Object Wizard never explicitly calls the content-generation method as the content generation is initiated by the ODA for this content protocol	Method throws an exception as it is never called directly. Actual generation of content is performed externally to the content-generation method, in a different method, class, or even process.

Example 14-4 shows the `getContentProtocol()` method.

Example 14-4 `getContentProtocol()` method

```

public long getContentProtocol(ContentType ct)
{
    //BO Designer must request the
    business objects
    if (ct == ContentType.BusinessObject) return
    ODKConstant.CONTENT_PROTOCOL_ONREQUEST;
    return -1;
}

```

14.2.7 Generating source nodes

The Business Object Wizard calls the `getTreeNodes()` method to discover the source nodes in the ODA data source and to create the source-node hierarchy. The Business Object Wizard displays this information in the Select Source dialog box. The `getTreeNodes()` method is part of the `IGeneratesBoDefs` interface, which the ODA class must implement to support generation of business object definitions.

The Business Object Wizard uses the tree-node array that `getTreeNodes()` returns to initialize the Select Source dialog box. This dialog box displays the source-node hierarchy, allowing users to move through the source nodes obtained from the data source and to select those for which the ODA generates business object definitions.

The back-end API does not support the hierarchy of the data source and returns only top-level data. Therefore, in our scenario, we implement only the top-level nodes and do not implement expandable child nodes to search from the top node in the Select Source dialog box.

The `getTreeNodes()` method:

1. Obtains the ODA implementation object of the back-end application by the `retrieveODA` method that is provided by the back-end application.
2. Gets each element object of the data source by the `getStructures` method that is provided by the back-end application.
3. Gets each component name of the element object by the `elementAt` method that is defined in the Java vector class.

Creates each tree node object by the `getTreeNodes()` method in the ODK API to show in the Select Source dialog box and to add it to Vector object by the `add` method in Java vector class.

4. Creates the tree-node array of each tree node object created in the previous steps by the `getTreeNodes()` method in the ODK API.
5. Returns the tree-node array.

Example 14-5 shows the `getTreeNodes()` method.

Example 14-5 `getTreeNodes()` method

```
public TreeNode[] getTreeNodes(String parentNodePath, String searchPattern)
throws ODKException
{
    //when no node path is
    //specified, return the top
    //node(s)
    Vector nodes = new Vector();
    //nodes are stored here
    //adjust the pattern, consider
    //just the first letter
    if (searchPattern == null || searchPattern.length() == 0) searchPattern = "";
    else searchPattern = new String(new char[] {searchPattern.charAt(0)});
    try {
        RMODAIImplementation rmODAI = (RMODAIImplementation)
appI.retrieveODA();
        odaStuff = rmODAI.getStructures();
        for (int i = 0; i < odaStuff.size(); i++) {
            RMODADefinition rmODAD = (RMODADefinition) odaStuff.elementAt(i);
            String compName = rmODAD.getComponentName();
            TreeNode tn =
nodes that satisfy the age range
                new TreeNode(compName, "Component",
                    true,
                    false, null,
ODKConstant.NODE_NATURE_NORMAL);
            nodes.add(tn);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    TreeNode[] tna = new TreeNode[nodes.size()];
    System.arraycopy(nodes.toArray(), 0, tna, 0, nodes.size());

    return tna;
}
```

14.2.8 Generating business object definitions

After users have selected the source nodes in the Select Nodes dialog box, the ODA is ready to generate content. The Business Object Wizard calls the `generateBoDefs()` content-generation method to generate business object definitions for the user-selected source nodes. The Business Object Wizard sends the list of source nodes to the ODA. The goal of the business object

definition generation process is to create a business object definition for each selected source node.

If, during the content-generation process, the ODA requires additional information, it can display the Business Object Properties dialog box and request values for business-object properties. Implement this to add a business object name prefix that is specified in the Business Object Property dialog box.

The `generateBoDefs()` method:

1. Creates the business-property array for the business object name prefix by the `AgentProperty` method.
2. Calls the `getBOSpecificProps()` method in the ODK utility class to display the business-object property for the business object name prefix.
3. Obtains a user-initialized value for the business-object property of the business object name prefix by the `getBOSpecificProperty` method in the ODK utility class.
4. Gets each node that is selected in Select Source dialog box by looping a String array that is passed as a parameter.
5. Gets each node name, adding the business object prefix if the user-specified prefix is in the Business Object Property dialog box. Creates a business object with the name, including the prefix, using the `BusObjDef` method in the ODK API.
6. Gets the back-end application data object that corresponds to the selected node name by the `elementAt` method in a Java class `Vector`.
7. Gets a vector object for each field definition of the back-end application data object by using the `elementAt` method in a Java class `Vector`.
8. Gets name, type, length, key, and required information of each field by using the `getFieldName`, `getFieldType`, `getFieldLength`, `isKey`, and `isRequired` methods that are provided by the back-end application.
9. Creates and sets the attribute object by the `BusObjAttr` and `setter` methods in the ODK API.
10. Inserts each attribute object into the business object to generate by the `insertAttribute` method in the ODK API.
11. Sets the application-specific information and supported verbs to the business object by the `setAppInfo` and `insertVerb` methods in the ODK API.
12. Populates the `m_generatedBOs` vector with the generated business object definitions.
13. Returns the content-metadata object that describes the generated content with the `generateBoDefs()` method.

Example 14-6 shows the generateBoDefs() method.

Example 14-6 generateBoDefs() method

```
public ContentMetaData generateBoDefs(String[] nodes) throws ODKException
{
    AgentProperty agtProps[] = new AgentProperty[1];
    agtProps[0] = new AgentProperty("Prefix", AgentProperty.TYPE_STRING,
        "Prefix that should be applied to each business object name",
        false, false, ODKConstant.SINGLE_CARD, null, null);
    m_utility.getBOSpecificProps(agtProps, "For all the Tables selected");
    AgentProperty propPrefix = m_utility.getBOSpecificProperty("Prefix");
    RMODADefinition rmODAD = null;
    for (int i = 0; i < nodes.length; i++)
    {
        String compName;
        if (propPrefix.allValues != null && propPrefix.allValues[0] != null)
            compName = propPrefix.allValues[0] + nodes[i];
        else
            compName = nodes[i];
        BusObjDef bo = new BusObjDef(compName);
        BusObjAttr attr;
        for (int j = 0; j < odaStuff.size(); j++) {
            rmODAD = (RMODADefinition) odaStuff.elementAt(j);
            if (compName.compareTo(rmODAD.getComponentName()) == 0) {
                break;
            }
        }
        Vector rmFDV = rmODAD.getFieldDefinitions();
        for (int j = 0; j < rmFDV.size(); j++) {
            RMFieldDefintions rmFD = (RMFieldDefintions) rmFDV.elementAt(j);
            String attrName = rmFD.getFieldName();
            String attrType = rmFD.getFieldType();
            if (attrType.compareTo(RMFieldDefinitionsConstants.TYPE_STRING) ==
0) {
                attr = new BusObjAttr(attrName, BusObjAttrType.STRING,
BusObjAttrType.AttrTypes[BusObjAttrType.STRING]);
                attr.setMaxLength(rmFD.getFieldLength());
            } elseif (attrType.compareTo(RMFieldDefinitionsConstants.TYPE_INT)
== 0)
                attr = new BusObjAttr(attrName, BusObjAttrType.INTEGER,
BusObjAttrType.AttrTypes[BusObjAttrType.INTEGER]);
            else if
(attrType.compareTo(RMFieldDefinitionsConstants.TYPE_BOOLEAN) == 0)
                attr = new BusObjAttr(attrName, BusObjAttrType.BOOLEAN,
BusObjAttrType.AttrTypes[BusObjAttrType.BOOLEAN]);
            else if
(attrType.compareTo(RMFieldDefinitionsConstants.TYPE_DOUBLE) == 0)
```



```

        attr = new BusObjAttr(attrName, BusObjAttrType.DOUBLE,
BusObjAttrType.AttrTypes[BusObjAttrType.DOUBLE]);
        else if
        (attrType.compareTo(RMFieldDefinitionsConstants.TYPE_FLOAT) == 0)
            attr = new BusObjAttr(attrName, BusObjAttrType.FLOAT,
BusObjAttrType.AttrTypes[BusObjAttrType.FLOAT]);
        else
            attr = new BusObjAttr(attrName, BusObjAttrType.STRING,
BusObjAttrType.AttrTypes[BusObjAttrType.STRING]);
            attr.setIsKey(rmFD.isKey());
            attr.setIsRequiredKey(rmFD.isRequired());
            attr.setAppText("attr="+attrName);
            bo.insertAttribute(attr);
        }
        bo.setAppInfo("obj="+compName);
        bo.insertVerb("Create", "verb=Create");
        bo.insertVerb("Update", "verb=Update");
        bo.insertVerb("Retrieve", "verb=Retrieve");
        bo.insertVerb("Delete", "verb=Delete");
        bo.insertVerb("RetrieveByContent", "verb=RetrieveByContent");

        m_generatedBOs.add(bo);
    }

    return new ContentMetaData(ContentType.BusinessObject, -1, 1);
}

```

14.2.9 Providing access to generated business object definitions

The `generateBoDefs()` method does not return the actual generated business object definitions. For the Business Object Wizard to be able to access the generated content, the ODA class must implement the content-retrieval method for business object definitions. The Business Object Wizard uses the information in the content-metadata object (which `generateBoDefs()` does return) to determine whether to call the appropriate content-retrieval method. If `generateBoDefs()` has successfully generated business objects, the Business Object Wizard calls the `getBoDefs()` method to retrieve the generated business object definitions.

For our ODA, the `generateBoDefs()` method populates the `m_generatedBOs` vector with its generated business object definitions. Therefore, the `getBoDefs()` method retrieves the specified number of business object definitions from this vector and copies them into its return array.

Example 14-7 shows the `getBoDefs()` method for the ODA in our scenario.

Example 14-7 `getBoDefs()` method

```
public BusObjDef[] getBoDefs(long index) throws ODKException
{
    BusObjDef[] bos = null;
    if (index == ODKConstant.GET_ALL_OBJECTS)//were all B0s requested?
    {
        bos = new BusObjDef[m_generatedB0s.size()];
        System.arraycopy(m_generatedB0s.toArray(), 0, bos, 0,
m_generatedB0s.size());
    } else bos = new BusObjDef[]
{(BusObjDef)m_generatedB0s.get((int)index)};
    return bos;
}
```

Note: The source for the ODA in our scenario can be found in the Additional Materials in the ODA .jar file.

14.3 Testing the ODA

After you have written the application-specific component for the connector, you must compile it into an executable format, its connector library.

To compile a Java connector:

1. Use a JDK 1.3.1 development environment.
2. Ensure that both of the following files are in the `\lib\` subdirectory of the product directory:
 - `crossworlds.jar`
 - `WBIA.jar`
3. Include `crossworlds.jar` in the project path. Also, include in the project path any application-specific .jar (Java archive) files that the application-specific component requires.
4. Compile the connector source (.java) files into class (.class) files with the Java compiler.
5. Create the Java connector's library file, which is a .jar file containing the compiled Java code.

After coding and compiling the ODA, you can test the ODA in the WebSphere Studio Development environment.

14.3.1 Setting up the test environment

To set up the test environment:

1. Select **Run** from the tool bar.
2. Select **Debug**, as shown in Figure 14-3.

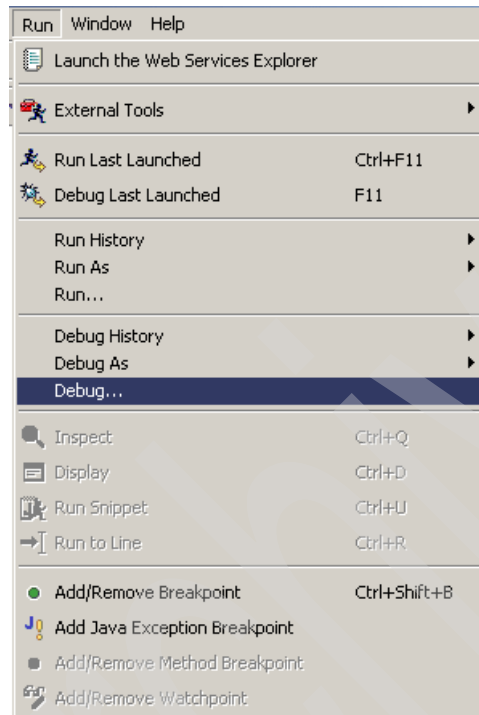


Figure 14-3 Select Debug menu

3. Select **Java Application** in the Launch Configurations window and then select **New** as shown in Figure 14-4.

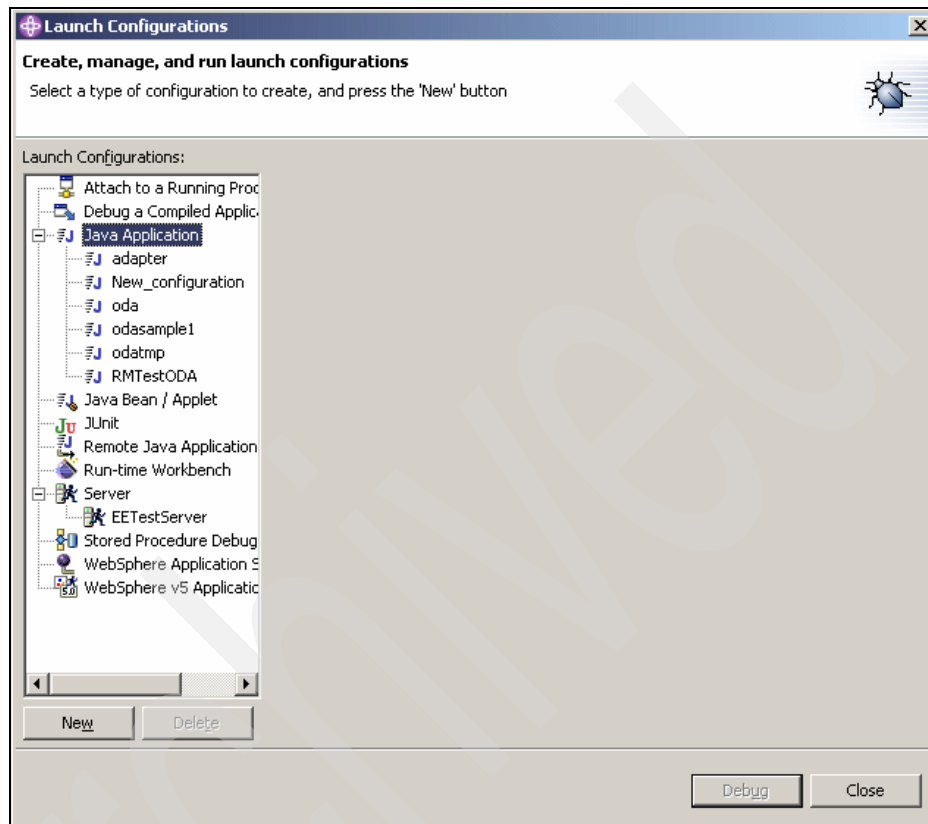


Figure 14-4 Launch Configurations window

4. In Name field of the Launch Configurations window, specify an appropriate name. In the Project field, specify your project. Enter `com.crossworlds.ODKInfrastructure.XRmiAgent` in the Main class field (Figure 14-5).

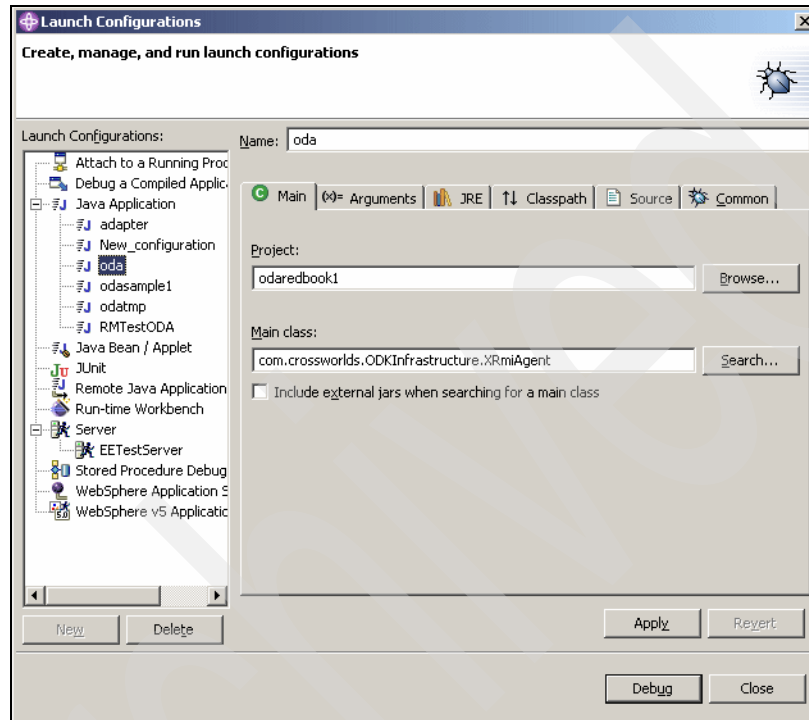


Figure 14-5 Main class

5. Enter the class name in the Program arguments field (Figure 14-6) as follows:

```
-lRedMaintenance -ccom.ibm.itso.rm.oda.RedHouse.RedMaintenanceAgent
```

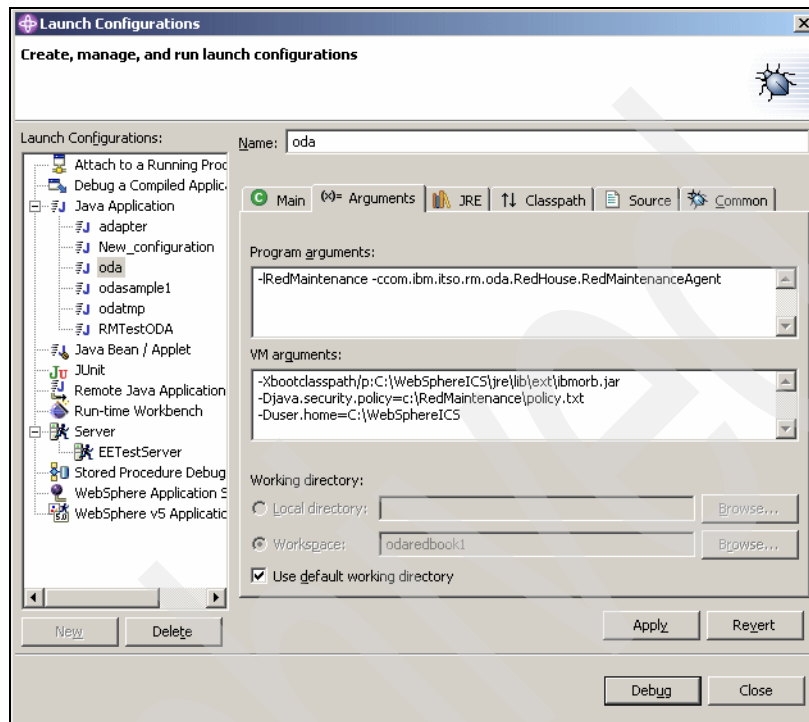


Figure 14-6 Arguments

6. Enter the following in the VM arguments field:

```
-Xbootclasspath/p:C:\WebSphereICS\jre\lib\ext\ibmorb.jar  
-Djava.security.policy=c:\RedMaintenance\policy.txt  
-Duser.home=C:\WebSphereICS
```

Note: -Djava.security.policy=c:\RedMaintenance\policy.txt is necessary only when you run the ODA from our scenario. C:\WebSphereICS\ might vary according to your Adapter Framework directory.

7. Select **Debug** to start.

The ODA is now running under WebSphere Studio. You can debug your ODA by creating business objects as shown in 14.5, “Generating business objects using the ODA” on page 225.

14.4 Deploying the ODA

When the ODA is error- free, you are ready to deploy it.

14.4.1 Exporting the ODA

When you are ready to compile the ODA, it is compiled and exported as a .jar file in WebSphere Studio Application Development.

To export the ODA:

1. Right-click the **ODA package** and select **Export** (Figure 14-7).

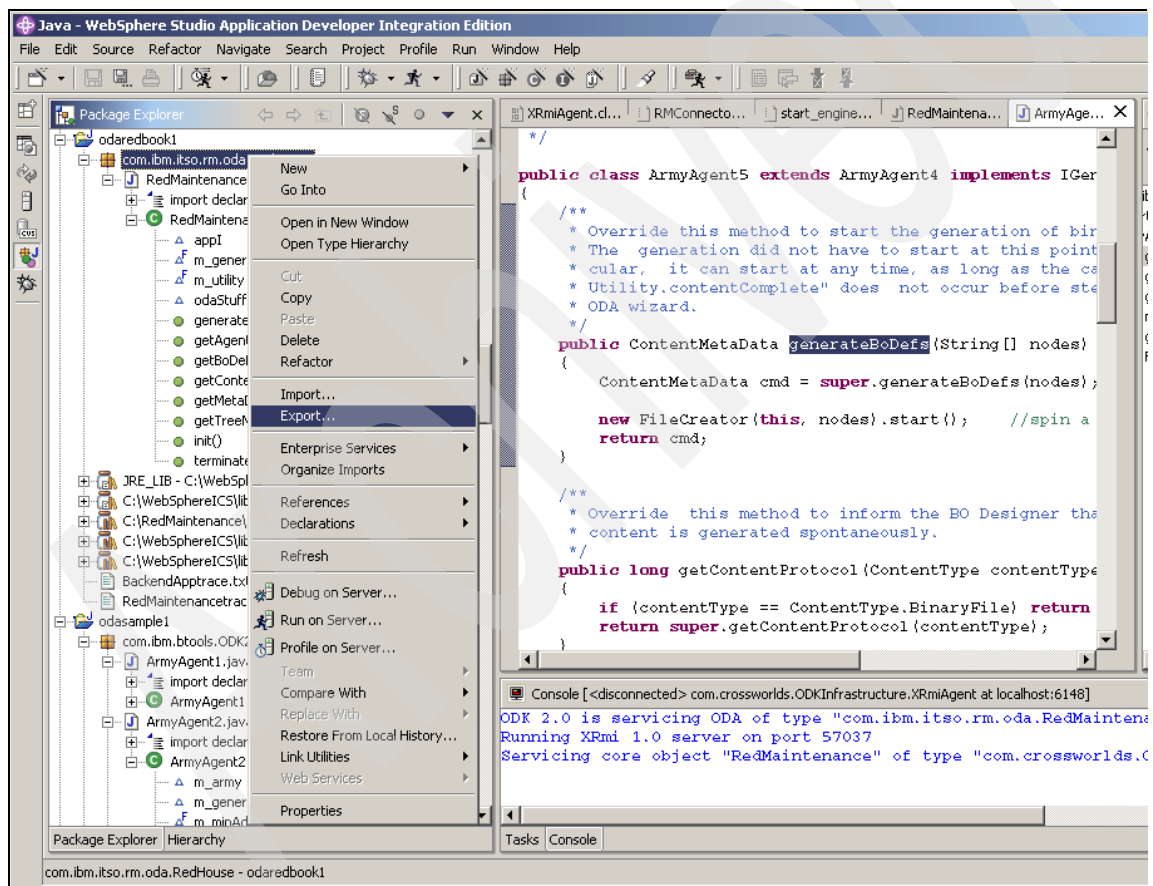


Figure 14-7 Export

2. Select **JAR file** and click **Next**, as shown in Figure 14-8.



Figure 14-8 Select JAR file

3. Select the following (as shown in Figure 14-9):
 - Export generated class files and resources
 - Compress the contents of the JAR file
4. In the Select the export destination field, enter in a directory where you want the .jar file to reside.
5. Click **Finish**.

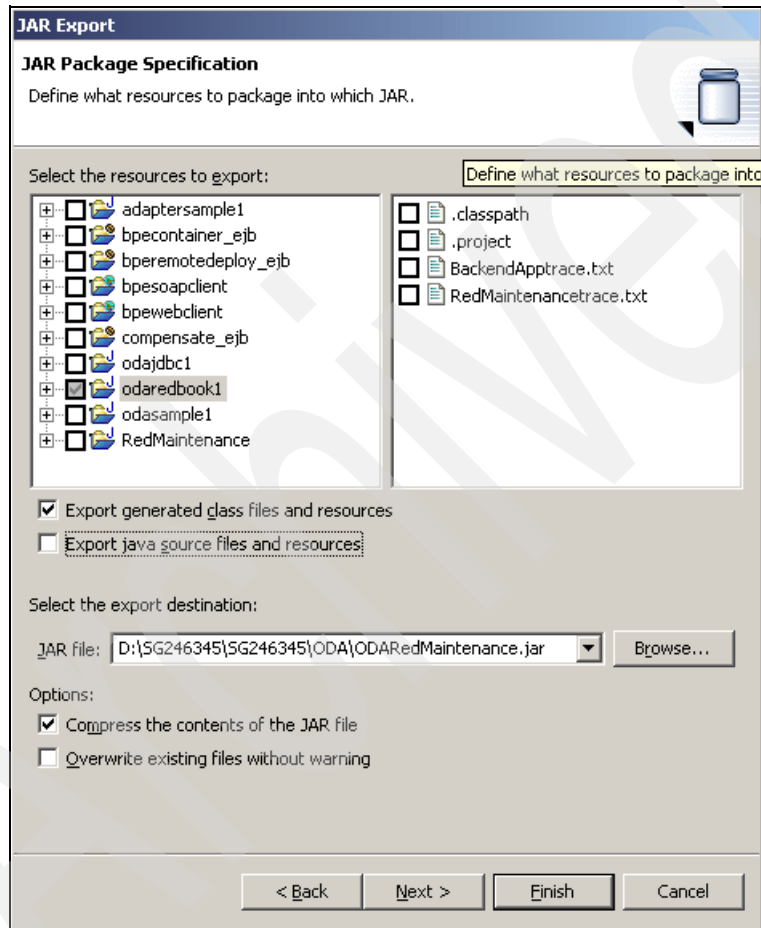


Figure 14-9 Select export destination

14.4.2 Creating the startup scripts

Example 14-8 includes a sample startup script for the ODA to run on Windows.

Note: If you are not using our sample for your adapter, change the following for your ODA:

```
RedMaintenance
ODARedMaintenance
com.ibm.itso.rm.oda.RedHouse.RedMaintenanceAgent
```

You can also remove the `.\common.jar` file and `-Djava.rmi.server.codebase=file:/common.jar` file, because they are necessary for only our adapter and our back-end application.

Example 14-8 Startup script

```
REM @echo off
setlocal
REM about to call the shared env file to set the ORB and JRE properties
call "%CROSSWORLDS%\bin\CWSharedEnv.bat

set AGENTNAME=RedMaintenance
set AGENT=.\ODARedMaintenance.jar;.\common.jar
set AGENTCLASS=com.ibm.itso.rm.oda.RedHouse.RedMaintenanceAgent

set
JCLASSES=%AGENT%;"%CROSSWORLDS%\lib\xrmi.jar;"%CROSSWORLDS%\lib\xerces.jar;"%
CROSSWORLDS%\lib\CwODK.jar

%CWJAVA% -Djava.rmi.server.codebase=file:/common.jar
-Djava.security.policy=policy.txt -Duser.home="%CROSSWORLDS%" -mx128m
-classpath %JCLASSES% com.crossworlds.ODKInfrastructure.XRmiAgent
-l%AGENTNAME% -c%AGENTCLASS%

endlocal
pause
```

To create a startup script:

1. Create a directory named RM under the directory `%CROSSWORLDS%\ODA`.
2. In the directory, create a script named `start_ODARedMaintenance.bat` (as shown Example 14-8).

3. Copy the following files that are provided by our back-end application into the same directory:
 - policy.txt
 - common.jar

Note: These files are included in the Additional Materials.

4. Create a shortcut whose target is start_ODARedMaintenance.bat.
5. Use the shortcut to start the ODA. If the ODA ends and there are errors, check any errors and restart the ODA.
6. Leave the resulting window open for the next steps in the process.

14.5 Generating business objects using the ODA

You are now ready to generate some application business objects using the ODA.

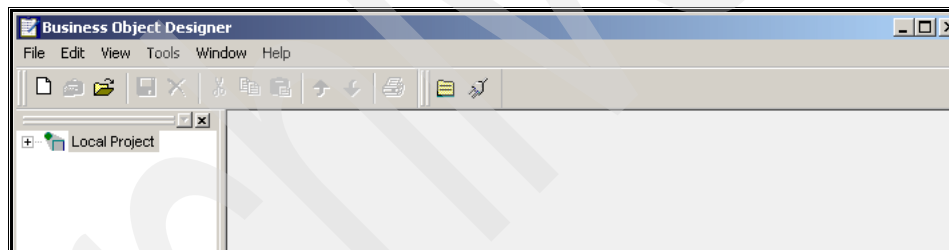


Figure 14-10 Business Object Designer

To generate business objects using the ODA:

1. Start the Business Object Designer. The shortcut to the Business Object Designer can vary depending on which broker you are using, but it is in the Development Tools.

2. Select **File** → **New Using ODA** (Figure 14-11) to open the first window of the Business Object Wizard.

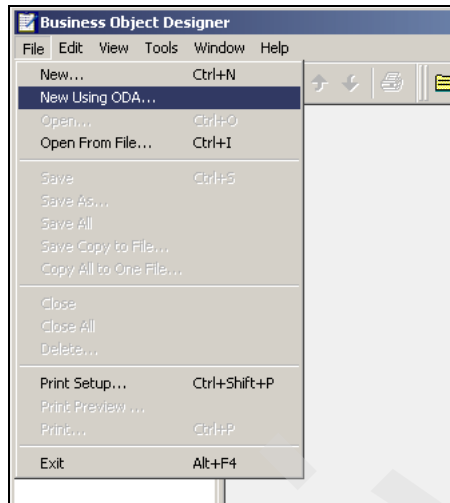


Figure 14-11 ODA menu

3. Click **Find Agents** to find your ODA. If you have a large network of ODAs to search through, it might be quicker to type in the name of the ODA that you expect to run. Remember, you are using the ORB to find the Agents.

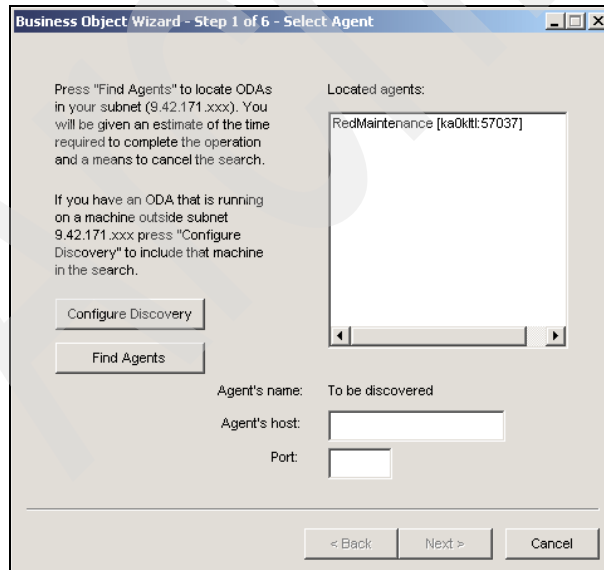


Figure 14-12 Select Agent

While the search is carried out, the message that is shown in Figure 14-13 displays.

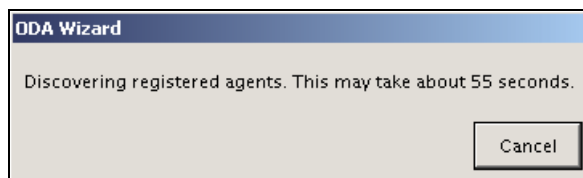


Figure 14-13 Discovering Agent

4. When the ODA is located, the window shown in Figure 14-14 displays. Select the agent in the Located agents field and click **Next**.

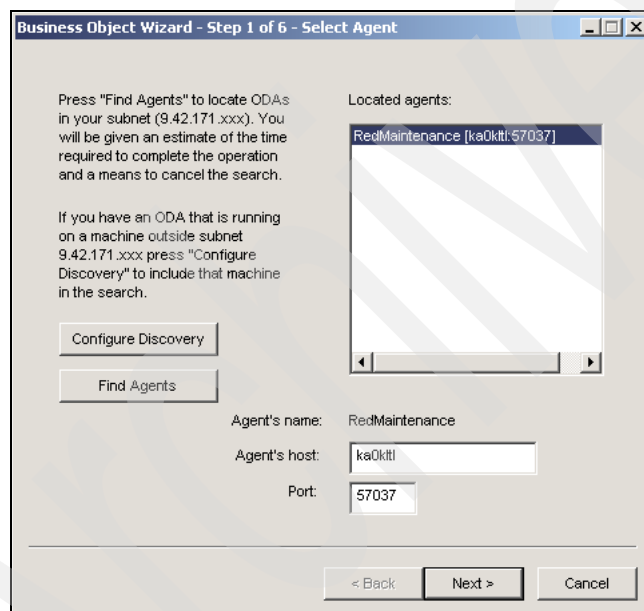


Figure 14-14 Located agent

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

5. Enter the required information in the Property and Value fields (Figure 14-15):
- Host name of the back-end application to which to connect.
 - A trace file.
 - Required trace level. In our scenario, we chose five for maximum trace information.
 - Message file that contains the messages for the ODA.

The dialog box is titled "Business Object Wizard - Step 2 of 6 - Configure Agent". It contains a "Profiles" section with a "Current profile:" dropdown menu and "Save", "New", and "Remove" buttons. Below this is a table with four columns: "Property", "Value", "Type", and "Description". The table contains four rows of data. At the bottom of the dialog are "< Back", "Next >", and "Cancel" buttons.

	Property	Value	Type	Description
1	Host	kaOkItI	String	Host name or IP address of RedMai
2	TraceFileName	RedMaintenancetrac	String	Name of the trace file
3	TraceLevel	5	Integer	Trace level for the agent
4	MessageFile	RedMaintenanceAge	String	Name of the error and message fil

Figure 14-15 Properties

6. Click **Next**.

7. The components that were discovered from the back-end application display (Figure 14-16). If you do not see these components, check your ODA settings and check that the back-end application is running correctly.

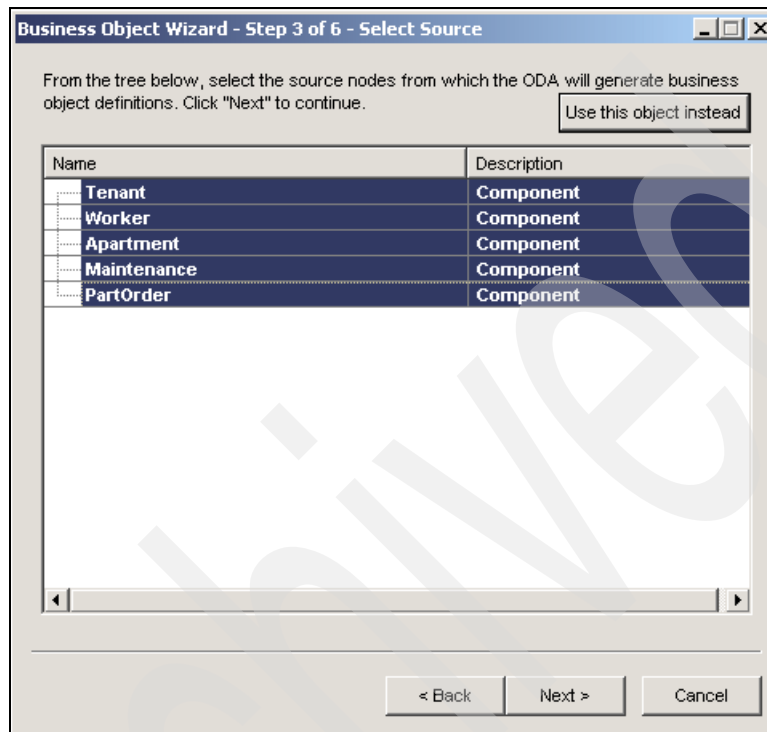


Figure 14-16 Select nodes

8. Select the nodes from which the ODA will generate business objects. For our scenario, we selected all of them.
9. Click **Next**.

10. You are asked for confirmation, as shown in Figure 14-17. Click **Next**.

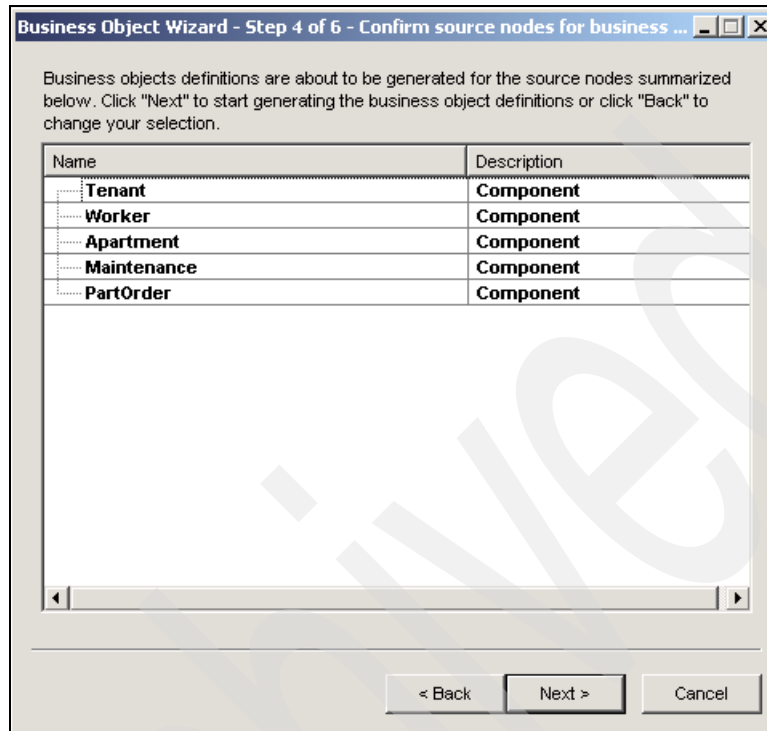


Figure 14-17 Confirmation for generation

11. Enter any additional properties that are needed for the business object generation (Figure 14-18 on page 231).
12. Enter a prefix for the business object names that are to be generated. We selected RM_ as a prefix.
13. Click **OK**.

The business object definition generation begins.

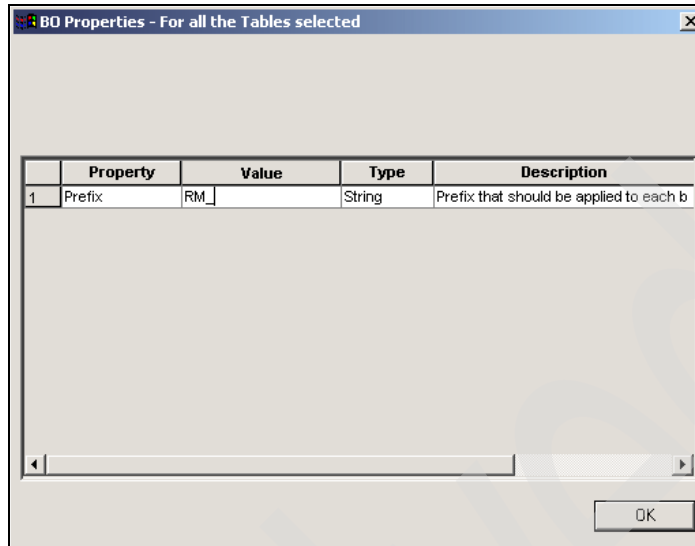


Figure 14-18 Business object Property

14. Save the business objects (Figure 14-19). Select **Shutdown ODA RedMaintenance** and click **Finish**.

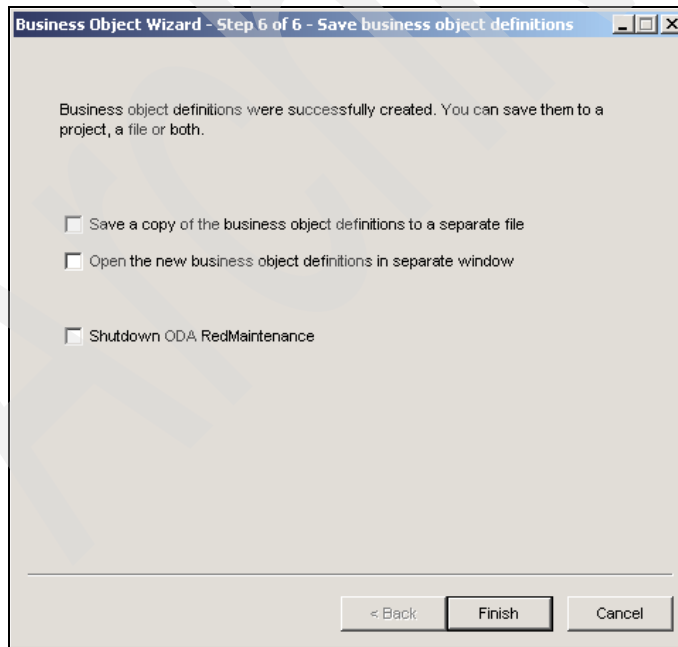


Figure 14-19 Confirmation

In the Business Object Designer, you see the newly created business object definitions (as shown in Figure 14-20). The asterisk next to each business object indicates that they are as yet unsaved.

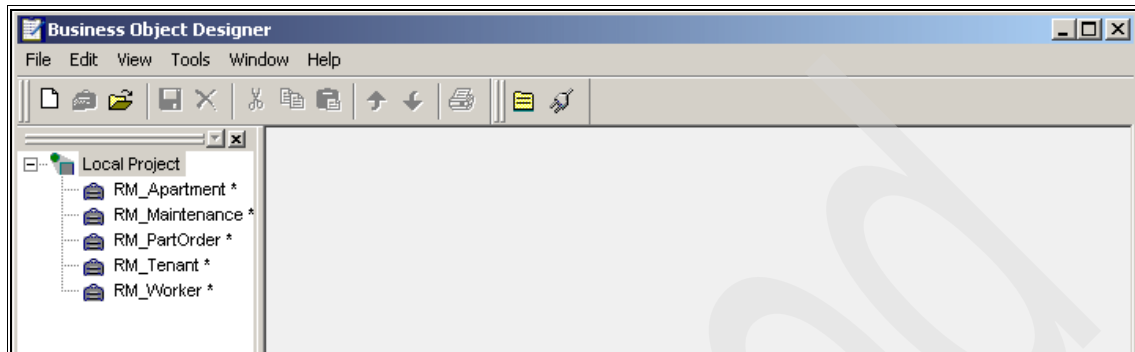


Figure 14-20 Generated Business Objects

You can also see from the activity window of the back-end application that the ODA has successfully contacted the application, as shown in Figure 14-21.



Figure 14-21 RetrieveODA

14.5.1 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. The ODA can only generate definitions based on what the application or database broadcasts to it. In still another case, it might incorrectly report column attributes of a database schema based on the way that it interacts with the database. Such is the case with the ODA. The back-end application has told the ODA about the different components that it contains, but it has not told you about the relationships that exist between these components or the dependencies between them.

Important: At this stage, you will need the assistance of someone who has a detailed knowledge of the application itself. Do not skip this step.

It is crucial that the business object design be verified against the application with which it is interacting.

This section provides information about the business objects that were generated by the ODA. It then discusses what changes need to be made to the business objects to allow our adapter to interact successfully with the API.

The business objects that are generated for the application objects are all correct in terms of the supported verbs and the application object names for the application-specific information, as shown in Figure 14-22.

General | Attributes

Business Object Level Application-specific information:

obi= Tenant

Supported Verbs:

	Name ▾	Application-specific information
1	Create	verb=Create
2	Delete	verb=Delete
3	Retrieve	verb=Retrieve
4	RetrieveByContent	verb=RetrieveByContent
5	Update	verb=Update
6		

Figure 14-22 General properties - Tenant

Figure 14-23 on page 234 shows the attributes that are generated by the ODA. The application-specific information is correct in that the attribute names of the application object have been generated correctly. You also see key fields and mandatory fields reported.

However, you need to check whether this information is 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 objects. This information is not accurate because the tenant application object is a parent object which contains apartment and possibly maintenance child objects. You will change this as you continue.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Name
3	3	ApartmentId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId
4	4	Email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Email
5	5	ObjectEventId	String							
6	6			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-23 Tenant business object

In Figure 14-24, 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.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentNumber
3	3	AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine1
4	4	AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine2
5	5	AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine3
6	6	AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine4
7	7	PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		20		attr=PostCode
8	8	ObjectEventId	String							
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-24 Apartment business object

Figure 14-25 on page 235 shows that the maintenance business object also has multiple keys flagged. This information, again, is not the case and must be corrected. You need to modify the generated business objects to accurately reflect the application view of the objects.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId
3	3	TenantId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=TenantId
4	4	ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=ProblemDescription
5	5	StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=StatusDescription
6	6	ExpectedCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ExpectedCompletion
7	7	ActualCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ActualCompletion
8	8	ObjectEventId	String							
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-25 Maintenance business object

The apartment business object is correct as discovered (Figure 14-26). So you can save this business object.

	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentNumber
3	3	AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine1
4	4	AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine2
5	5	AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine3
6	6	AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine4
7	7	PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		20		attr=PostCode
8	8	ObjectEventId	String							
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-26 Correct apartment business object

Note: When you save each of the business objects, save a copy of the business object to file. The schema files generated will be needed later, depending on which integration broker you use.

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. The application-specific information *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 shown in Figure 14-27, 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

Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId;chfk=RM_Tenant.Ap
3	Status	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=Status
4	TenantId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=TenantId;chfk=RM_Tenant.Id
5	ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=ProblemDescription
6	StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=StatusDescription
7	ExpectedCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ExpectedCompletion
8	ActualCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ActualCompletion
9	ObjectEventId	String							
10			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-27 Correct maintenance business object

When the adapter is performing a retrieve operation for all of the maintenance records for a particular tenant, these values are passed to the application for the search through the following steps:

1. Modify the maintenance business object to add the application-specific information to these attributes, as shown in Figure 14-27.
2. Modify the business object to reflect that the ApartmentId and TenantId are foreign keys.
3. Modify the business object to reflect that the dates in the object have a type of date.
4. Remove the key value flags.
5. Save the business object.

As mentioned earlier, the tenant business object is a parent object of one apartment and 0 - *n* maintenance objects. Reflect this by:

1. Highlighting the row for ObjectEventId by clicking the far left number.
2. Right-clicking and selecting **Insert Above**.
3. In the Type column, selecting the **RM_Apartment** type from the drop-down box.
4. Naming this Attribute RM_Apartment.
5. Setting the Cardinality to one (1).

Only one occurrence of this child object is allowed. Each tenant has only one apartment record.

6. Repeating these steps for the RM_Maintenance, setting the Cardinality to N.
For an apartment, there might be multiple maintenance records.
7. Set the Foreign Key flag on for the ApartmentId attribute.
8. Update the application-specific information to indicate that this attribute is part of a foreign key, with the foreign key attribute being RM_Apartment.Id.
9. Save this business object as shown in Figure 14-28.

General Attributes									
Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default	App Spec Info
1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Name
3	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentId; fkey=RM_Apartment.Id
4	Email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Email
5	田 RM_Apartment	RM_Apartment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
6	田 RM_Maintenance	RM_Maintenance	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
7	ObjectEventId	String							
8			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-28 Correct tenant business object

If you expand the child objects within the parent object, you can see the correct hierarchical structure of the tenant object with all of the keys and foreign keys set correctly, as shown in Figure 14-29 on page 238.

Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application
Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Name
ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentId
Email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Email
RM_Apartment	RM_Apartment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentNumber
AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine1
AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine2
AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine3
AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine4
PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		20		attr=PostCode
ObjectEventId	String							
RM_Maintenance	RM_Maintenance	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId
Status	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=Status
TenantId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=TenantId
ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=ProblemDescription
StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=StatusDescription
ExpectedCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ExpectedCompletion
ActualCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ActualCompletion
ObjectEventId	String							
ObjectEventId	String							
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 14-29 Tenant business object with the correct hierarchy



Initializing and terminating the adapter agent

This chapter outlines the initialization and termination processing that we need for successful start up and shutdown of our adapter. This includes establishing connectivity to the back-end application.

15.1 Extending the connector base class

The `CWConnectorAgent` class includes methods for startup, subscription checking, business object subscription delivery, and shutdown. To implement your own connector, extend this connector base class to create the connector class.

To derive the connector class:

1. Create the connector class, `RMAgent`, that extends the `CWConnectorAgent` class.
2. Define the package name package, `com.ibm.itso.RedHouse.RM.Adapter`, in the connector class file.
3. Define the following imports in the connector class file to access methods of the ADK API:
 - a. `Import com.crossworlds.cwconnectorapi.*;`
 - b. `Import com.crossworlds.cwconnectorapi.exceptions.*;`
 - c. `Import com.crossworlds.cwconnectorapi.CWConnectorAgent;`
4. Define the following imports in the connector class file to access methods of the back-end API:
 - a. `Import com.ibm.itso.rm.common.RMCallbackRMImplementation;`
 - b. `Import com.ibm.itso.rm.common.RMDataInterface;`
 - c. `Import com.ibm.itso.rm.common.RMRMIRegisterException;`
 - d. `Import com.ibm.itso.rm.common.RMServerRMIInterface;`
5. Implement the methods shown in Table 15-1 of the `CWConnectorAgent` class for the connector class.

Table 15-1 *CWConnectorAgent methods*

Method	Description
<code>agentInit()</code>	Initializes the adapters application-specific component
<code>getVersion()</code>	Returns the version of the connector
<code>getconnectorBOHandlerForBO()</code>	Sets up one or more business object handlers
<code>getEventStore</code>	Obtains the event store object for the adapter
<code>terminate()</code>	Performs cleanup tasks upon termination

15.2 Initializing the adapter agent

To begin initialization, the adapter framework calls the initialization method, `agentInit()`, in the connector base class, `CWConnectorAgent`. This method performs initialization steps for the connector's application-specific component. For the connector implementation, perform the following tasks in the `agentInit()` method:

- 1. Retrieve connector configuration properties.
- 2. Establish a connection to the application.
- 3. Check the connector version.

15.2.1 Retrieving connector configuration properties

The first thing your initialization method needs to do is retrieve and validate the connector configuration properties. Use the `getConfigProp()` method to obtain the value of a connector configuration property. When the value is obtained, validate that the following is true:

- ▶ If the property value is not set, your adapter can set a default value for optional properties. For required properties, your adapter raises a `PropertyNotSetException`.
- ▶ If the property value is set, verify that it contains a valid entry. If it does not, then either default the value for optional properties or raise an exception of type `PropertyNotSetException` for required properties.

We created the additional methods in Table 15-2 to:

- ▶ Make the `agentInit()` method easier to read.
- ▶ Simplify retrieval and validation of connector configuration properties in our connector.

Table 15-2 Additional methods

Method	Description
<code>italizeProperties()</code>	Initializes all properties, raising a <code>PropertyNotSetException</code> if any errors occur.
<code>getIntProperty(String propertyName)</code>	Generically retrieves an <code>int</code> property and validates that the value is a valid <code>int</code> .
<code>getStringProperty(String propertyName)</code>	Generically retrieves a string property and validates that the value is not blank or null.
<code>getStringProperty(String propertyName, String defaultValue)</code>	Generically retrieves a string property and validates that the value is not blank or null. If the property is not set, the default value will be returned instead of an exception.

Example 15-1 shows the initializeProperties() method.

Example 15-1 initializeProperties

```
private void initializeProperties() throws PropertyNotSetException
{
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Entering RMAgent.initializeProperties");

    // get configuration properties
    // To-Do
    // connectionURL
    // ApplicationUserID
    // ApplicationPassword
    // event info - hierarchical property?
    // event table bo name
    // archive table bo name
    // connector id (for event table)
    // ReconnectOnConnectionLoss - do not use for ICS implementations
    // - to be used to try to regain connection to the application if
    // connection is lost instead of terminating adapter
    // NumberOfReconnectAttempts
    // - number of retries to be attempted before terminating adapter

    maxConn = getIntProperty("MaxConnection");
    host = getStringProperty("ApplicationHost");
    registeredName = getStringProperty("ApplicationRegisteredName");

    try
    {
        // Note: If this property does not exist we default to false to allow
        the adapter to start.
        callbackPrimary = getStringProperty("CallbackPrimary");
    } catch (PropertyNotSetException e)
    {
        // default to false
        callbackPrimary = "false";
    }

    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "initializeProperties: Connector properties initialized
successfully");
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.initializeProperties");
}
```

Example 15-2 shows the `getIntProperty(String propertyName)` method.

Example 15-2 `getIntProperty(String propertyName)`

```
private int getIntProperty(String propertyName)
    throws PropertyNotSetException
{
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.getIntProperty");
    int iProperty = 0;
    String property = CWConnectorUtil.getConfigProp(propertyName);
    if (property == null || property.length() == 0)
    {
        RMLogger.logMsg(30600, CWConnectorUtil.XRD_ERROR, propertyName);
        PropertyNotSetException p = new PropertyNotSetException();
        p.setStatus(CWConnectorConstant.FAIL);
        throw p;
    }
    try
    {
        iProperty = (new Integer(property)).intValue();
    } catch (NumberFormatException ne)
    {
        RMLogger.logMsg(30600, CWConnectorUtil.XRD_ERROR, propertyName);
        RMLogger.trace(
            CWConnectorUtil.LEVEL5,
            "getIntProptery: Exception is : " + ne.toString());
        PropertyNotSetException p = new PropertyNotSetException();
        p.setStatus(CWConnectorConstant.FAIL);
        throw p;
    }

    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "getIntProptery: Retrieved property: "
        + propertyName
        + " = "
        + iProperty);
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMAgent.getIntProperty");

    return iProperty;
}
```

Example 15-3 shows the `getStringProperty(String propertyName)` method.

Example 15-3 getStringProperty(String propertyName)

```
private String getStringProperty(String propertyName)
    throws PropertyNotSetException
{
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.getStringProperty");
    String property = CWConnectorUtil.getConfigProp(propertyName);
    if (property == null || property.length() == 0)
    {
        RMLogger.logMsg(30601, CWConnectorUtil.XRD_ERROR, propertyName);
        PropertyNotSetException p = new PropertyNotSetException();
        p.setStatus(CWConnectorConstant.FAIL);
        throw p;
    }

    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "getStringProperty: Retrieved property: "
        + propertyName
        + " = "
        + property);
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.getStringProperty");

    return property;
}
```

Example 15-4 shows the `getStringProperty(String propertyName, String defaultValue)` method.

Example 15-4 `getStringProperty(String propertyName, String defaultValue)`

```
private String getStringProperty(String propertyName, String defaultValue)
{
    String property = null;

    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Entering RMAgent.getStringProperty(Default)");
    property = CWConnectorUtil.getConfigProp(propertyName);
    if (property == null || property.length() == 0)
    {
        property = defaultValue;
        RMLogger.trace(
            CWConnectorUtil.LEVEL5,
            "Defaulted property: " + propertyName + " = " + property);
    }

    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "Retrieved property: " + propertyName + " = " + property);
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.getStringProperty(Default)");

    return property;
}
```

15.2.2 Establishing a connection to the application

The main task of the `agentInit()` initialization method is to establish a connection to the application. It executes successfully if the connector succeeds in opening a connection. If the connector cannot open a connection, the initialization method must throw the `ConnectionFailureException` to indicate the cause of the failure. The connector might also need to log into the application. If this log on attempt fails, the initialization method must throw the `LogonFailedException` to indicate the cause of the failure.

Our connector implementation uses a connection pool, `RMConnectionPool` class, to use one or more connections to the application based on the value of a configuration property. We also created a method, `connectToRedMaintenance()`, to make the `agentInit()` method easier to read.

Example 15-5 shows the RMConnectionPool class.

Example 15-5 RMConnectionPool class

```
package com.ibm.itso.RedHouse.RM.Adapter;

import com.crossworlds.cwconnectorapi.*;
import com.crossworlds.cwconnectorapi.exceptions.*;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.Vector;

import com.ibm.itso.rm.common.RMServerRMIInterface;

public class RMConnectionPool
{
    private Vector inUseVector;
    private Vector freeVector;
    private int maxConn;
    private String host;
    private String registeredName;

    /**
     * Constructor for RMConnectionPool.
     * @param sHost
     * @param sRegName
     * @param iMaxConn
     */
    public RMConnectionPool(String sHost, String sRegName, int iMaxConn)
    {
        super();
        RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool");
        host = sHost;
        registeredName = getRegisteredName(sHost, sRegName);
        maxConn = iMaxConn;

        inUseVector = new Vector();
        freeVector = new Vector();
        RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool");
    }

    /**
     * Method getRegisteredName.
     * @param sHost
     * @param sRegName
     * @return String
     */
}
```



```

    */
private String getRegisteredName(String sHost, String sRegName)
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool.getRegisteredName");
    String name = "/" + sHost + "/" + sRegName;

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.getRegisteredName");

    return name;
}

/**
 * Method getConnection.
 * @return RMServerRMIInterface
 */
public synchronized RMServerRMIInterface getConnection() throws ConnectionFailureException
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool.getConnection");
    RMServerRMIInterface connection = null;

    // check if maximum connections are in use
    while(inUseVector.size() == maxConn)
    {
        try
        { // wait until a connection frees up
            wait();
        } catch (InterruptedException ie)
        {
            CWConnectorExceptionObject cwExObject = new CWConnectorExceptionObject();
            String logMessage = RMLogger.generateMsg(CWConnectorUtil.LEVEL1, 40100,
CWConnectorUtil.XRD_FATAL);

            RMLogger.logMsg(CWConnectorUtil.XRD_FATAL, logMessage);
            cwExObject.setMsg(logMessage);
            cwExObject.setStatus(CWConnectorConstant.FAIL);

            throw new ConnectionFailureException(cwExObject);
        }
    }
    // make one more check before obtaining a connection, in case another thread has
    // already obtained one
    if(inUseVector.size() < maxConn)
    {
        // check if a new connection needs to be obtained to fill the connection pool
        // to maximum capacity
        if(freeVector.isEmpty())
        {
            // obtain a new connection

```

```

        try
        {
            connection = (RMServerRMIInterface) Naming.lookup(registeredName);
        } catch (Exception me)
        {
            CWConnectorExceptionObject cwExObject = new CWConnectorExceptionObject();

            String logMessage = RMLogger.generateMsg(CWConnectorUtil.LEVEL1, 40100,
CWConnectorUtil.XRD_FATAL, me.toString());

            RMLogger.logMsg(CWConnectorUtil.XRD_FATAL, logMessage);
            cwExObject.setMsg(logMessage);
            cwExObject.setStatus(CWConnectorConstant.FAIL);

            throw new ConnectionFailureException(cwExObject);
        }

        inUseVector.addElement(connection);
        RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: Connection obtained",
CWConnectorUtil.XRD_TRACE);

        Integer count = new Integer(inUseVector.size());
        RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: Number of connections in
use: {" + count + "}", CWConnectorUtil.XRD_TRACE);
        RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.getConnection");
        return connection;
    } // if(freeVector.isEmpty())
    else
    {
        // obtain a connection from the connection pool
        RMServerRMIInterface connection1 =
(RMServerRMIInterface)freeVector.firstElement();
        inUseVector.addElement(connection1);
        if(!freeVector.removeElement(connection1))
        {
            // unable to remove connection from freeVector list
            RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: WARNING! Unable to
remove connection from freeVector list.", CWConnectorUtil.XRD_TRACE);
        }
        RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: Connection obtained",
CWConnectorUtil.XRD_TRACE);

        Integer count = new Integer(inUseVector.size());
        RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: Number of connections in
use: {" + count + "}", CWConnectorUtil.XRD_TRACE);
        RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.getConnection");
        return connection1;
    }
} // if(inUseVector.size() < maxConn)

```

```

        else
        {
            // no connection available!
            RMLogger.trace(CWConnectorUtil.LEVEL5, "getConnection: No connections available.",
CWConnectorUtil.XRD_TRACE);
            RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.getConnection");
            return null;
        }
    } // getConnection

/**
 * Method releaseConnection.
 * @param connection1
 */
public synchronized void releaseConnection(RMServerRMIInterface connection1)
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool.releaseConnection");
    // Add current connection to the freeVector connection pool
    freeVector.addElement(connection1);
    // Remove current connection from the inUseVector connection pool
    if(!inUseVector.removeElement(connection1))
    {
        // unable to remove connection from inUseVector list
        RMLogger.trace(CWConnectorUtil.LEVEL5, "releaseConnection: WARNING! Unable to remove
connection from inUseVector list.", CWConnectorUtil.XRD_TRACE);
    }
    // send a notify request to current threads that are waiting for a connection to free up
    notify();
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.releaseConnection");

    return;
} // releaseConnection

/**
 * Method closeConnection.
 * @param connection1
 */
public synchronized void closeConnection(RMServerRMIInterface connection1)
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool.closeConnection");
    // close current connection either because there is a problem with the
    // current connection or because the connector is shutting down.

    // don't add this connection to the free vector list
    if(!inUseVector.removeElement(connection1))
    {
        // unable to remove connection from inUseVector list
        RMLogger.trace(CWConnectorUtil.LEVEL5, "closeConnection: WARNING! Unable to
remove connection from inUseVector list.", CWConnectorUtil.XRD_TRACE);
    }
}

```

```

    }
    notify();
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.closeConnection");

    return;
}

/**
 * Method cleanup.
 */
public void cleanup()
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMConnectionPool.cleanup");
    // close free connections
    for(int i=0; i<freeVector.size(); i++)
    {
        RMServerRMIInterface connection1 = (RMServerRMIInterface)freeVector.elementAt(i);
        connection1 = null;
    }
    // close connections in use
    for(int i=0; i<inUseVector.size(); i++)
    {
        RMServerRMIInterface connection1 = (RMServerRMIInterface)inUseVector.elementAt(i);
        connection1 = null;
    }
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMConnectionPool.cleanup");
    return;
} // cleanup
}

```

Example 15-6 shows the `connectToRedMaintenance()` method.

Example 15-6 connectToRedMaintenance

```
/**
 * Method connectToRedMaintenance.
 */
private void connectToRedMaintenance() throws ConnectionFailureException
{
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Entering RMAgent.connectToRedMaintenance");

    // Initialize security manager
    if (System.getSecurityManager() == null)
        System.setSecurityManager(new RMISecurityManager());

    // establish connection to the application using connection pooling
    connPool = new RMConnectionPool(host, registeredName, maxConn);

    // test connection to ensure application is active
    RMServerRMIInterface connection = connPool.getConnection();
    if (connection == null)
    {
        CWConnectorExceptionObject cwExObject =
            new CWConnectorExceptionObject();

        String logMessage =
            RMLogger.generateMsg(
                CWConnectorUtil.LEVEL1,
                30300,
                CWConnectorUtil.XRD_FATAL);

        RMLogger.logMsg(CWConnectorUtil.XRD_FATAL, logMessage);
        cwExObject.setMsg(logMessage);
        cwExObject.setStatus(CWConnectorConstant.FAIL);

        throw new ConnectionFailureException(cwExObject);
    }
    // release connection used for test purposes
    connPool.releaseConnection(connection);

    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "connectToRedMaintenance: Connection pool initialized");
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting RMAgent.connectToRedMaintenance");
}
```

15.2.3 Checking the connector version

The `getVersion()` method returns the version of the adapter. It is called in both of the following contexts:

- ▶ The initialization method calls `getVersion()` to check the connector version.
- ▶ The adapter framework calls the `getVersion()` method when it needs to get a version for the adapter.

Example 15-7 shows the `getVersion()` method.

Example 15-7 getVersion

```
/**
 * @see AppSide_Connector.ConnectorInterface#getVersion()
 */
public String getVersion()
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMAgent.getVersion");

    // check application version if possible
    // To-Do

    // return connector version
    RMLogger.trace(
        CWConnectorUtil.LEVEL1,
        "getVersion: Connector version is " + connectorVersion);

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMAgent.getVersion");
    return connectorVersion;
}
```

15.2.4 Terminating the adapter agent

The Adapter Framework calls the `terminate()` method when the connector is shutting down. In your implementation of this method, it is good practice to free all the memory and log off from the application. You must implement this method for the adapter. Your implementation needs to perform the following tasks:

- ▶ Close connection(s) to the application
- ▶ Clean up any global variables

Important: Do not program your adapter agent to call the `terminate()` method directly to shut down the adapter. Doing so could cause the Adapter Framework to be unaware that the agent has terminated, potentially resulting in hanging threads and events waiting for a response that will never come.

Instead of calling the `terminate()` method directly, your adapter needs to return the appropriate status or exception to the Adapter Framework, resulting in the Adapter Framework shutting down the adapter agent correctly.

Example 15-8 shows the `terminate()` method.

Example 15-8 `terminate()`

```
/**
 * @see AppSide_Connector.ConnectorInterface#terminate()
 */
public int terminate()
{
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering RMAgent.terminate");
    int status = CWConnectorConstant.SUCCEED;

    // close connection(s) to the application
    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "terminate: Closing all connections");
    connPool.cleanup();
    RMLogger.trace(
        CWConnectorUtil.LEVEL5,
        "terminate: Connections closed.");

    // clean up any global variables
    // To-Do
    connPool = null;

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting RMAgent.terminate");
    return status;
}
```

Implementing a business object handler

This chapter explains how we implement a business object handler for our custom connector. It also provides details concerning how we use the Java API of the WebSphere Business Integration Adapter library in the development of our custom business object handler.

Creating a business object handler involves the following steps:

- ▶ Extending the Java business-object-handler base class.
- ▶ Implementing the `doVerbFor()` method.
- ▶ Performing the verb operation.

16.1 Extending the business-object-handler base class

In the Java connector library, the base class for a business object handler is named `CWConnectorBOHandler`. The `CWConnectorBOHandler` class provides methods for defining and accessing a business object handler. To implement your own business object handler, extend this business-object-handler base class to create your own business-object-handler class.

To derive a business-object-handler class for a Java connector, do the following:

1. Create a class that extends the `CWConnectorBOHandler` class, and name this class as follows:

connectorNameBOHandler.java

Let *connectorName* uniquely identify the application or technology with which the connector communicates. For example, to create a business object handler for the RedMaintenance application, create a business object handler class called `RMBOHandler`. If your connector design implements multiple business object handlers, include the name of the business objects in the name of the business object handler class.

2. Implement the `doVerbFor()` method to define the behavior of the business object handler.

The following example shows the basic structure of our class:

Example 16-1 Basic structure of the out custom business object

```
public class RMBOHandler extends CWConnectorBOHandler {  
    public int doVerbFor(CWConnectorBusObj bo)  
        throws ConnectionFailureException, VerbProcessingFailedException {  
    }  
}
```

16.2 Implementing the `doVerbFor()` method

The `doVerbFor()` method provides the functionality for the business object handler. When the Adapter Framework receives a request business object, it calls the `doVerbFor()` method for the appropriate business object handler to perform the action of this verb. For a Java connector, the `CWConnectorBOHandler` class defines the `doVerbFor()` method in which you define the verb processing.

However, the actual `doVerbFor()` method that the Adapter Framework invokes is the low-level version, which the `CWConnectorBOHandler` class inherits from the

BOHandlerBase class of the low-level Java connector library. This low-level doVerbFor() method calls the user-implemented doVerbFor() method. Therefore, as part of the business-object-handler class (an extension of CWConnectorBOHandler), you must provide an implementation of the doVerbFor() method.

The business object handler:

1. Receives business objects from the Adapter Framework.
2. Processes each business object based on the active verb.
3. Sends requests for operations to the application.
4. Returns status to the Adapter Framework.

Note: Java connectors must be thread-safe. For Java connectors, the Adapter Framework uses separate threads to call into the doVerbFor() method.

Important: If the business integration system uses InterChange Server and collaborations are coded to be multi-threaded, the Adapter Framework might call into the doVerbFor() method with multiple threads representing request processing.

16.2.1 Obtaining the active verb

To determine which actions to take, the doVerbFor() method must first retrieve the verb from the business object that it receives as an argument. This incoming business object is called the *request business object*. The verb that this business object contains is the *active verb*, which must be one of the verbs that the business object definition supports.

Obtaining the active verb from the request business object generally involves the following steps:

1. Verifying that the request business object is valid.

Before the connector calls the getVerb() method, it verifies that the incoming request business object is not null. The incoming business object is passed into the doVerbFor() method as a CWConnectorBusObj object.

2. Obtaining the active verb with the getVerb() method.

After the request business object is validated, you can use the getVerb() method that is inherited from the CWConnectorBusObj class to obtain the active verb from this business object.

3. Verifying that the active verb is valid.

When the connector has obtained the active verb, it verifies that this verb is neither null nor empty. If either the request business object or the active verb is invalid, the connector does not continue with verb processing. Instead, it takes the following steps:

- a. Logging an error message to the log destination to indicate the cause of the verb-processing error.
- b. Instantiating an *exception-detail object* to hold the exception information.
- c. Setting the status information within an exception-detail object:
 - Setting a message to indicate the cause of the verb-processing failure.
 - Setting the status to the FAIL outcome status, which the Adapter Framework includes in its response to the integration broker.
- d. Throwing a `VerbProcessingFailureException` exception.

The `doVerbFor()` uses the `VerbProcessingFailureException` to tell the Adapter Framework that a verb-processing error has occurred. This exception object contains the exception-detail object that was initialized in step 2 on page 257.

When the low-level `doVerbFor()` method catches this exception object, it copies the message and status from the exception-detail object into the return-status descriptor. The `doVerbFor()` method returns it to the Adapter Framework, which in turn returns it to the integration broker.

Example 16-2 on page 259 contains a fragment of the `doVerbFor()` method that obtains the active verb with the `doVerbFor()` method. This code uses the try and catch statements to ensure that the request business object and its active verb are not null. If either of these conditions exists, the code fragment throws the `VerbProcessingFailedException` exception, which the Adapter Framework catches.

16.2.2 Verifying the connection before processing the verb

When the `agentInit()` method in the connector class initializes the application-specific component, one of its most common tasks is to establish a connection to the application. The verb processing that `doVerbFor()` performs requires access to the application. Therefore, before the `doVerbFor()` method begins processing the verb, it verifies that the connector is still connected to the application. The way to perform this verification is application-specific.

A good design practice is to code the connector application-specific component to shut down whenever the connection to the application is lost. If the connection

has been lost, the connector does not continue with verb processing. Instead, it takes the following steps to notify the Adapter Framework of the lost connection:

1. Logs an error message to the log destination to indicate the cause of the error.

The connector logs a fatal error message so that e-mail notification is triggered if the LogAtInterchangeEnd connector configuration property is set to True.

2. Sets the exception-detail object with:
 - A message to indicate the cause of the connection failure.
 - The status of the APPRESPONSETIMEOUT outcome status, which the Adapter Framework includes in its response to the integration broker.
3. Throws a `ConnectionFailureException` exception, which the `doVerbFor()` method uses to tell the Adapter Framework that a verb-processing cannot continue because the connection to the application has been lost. This exception object contains the exception-detail object we initialized in step 2.

When the low-level `doVerbFor()` method catches this exception object, it copies the message and status from the exception-detail object into the return-status descriptor that it returns to the Adapter Framework. If you have not set the status in the `ConnectionFailureException` exception-detail object, the Adapter Framework sets the status to `APPRESPONSETIMEOUT`. The Adapter Framework includes this return-status descriptor as part of its response to the integration broker. The integration broker can check the return-status descriptor to determine that the application is not responding.

After it has sent the return-status descriptor, the Adapter Framework stops the process in which the connector runs. A system administrator must fix the problem with the application and restart the connector to continue processing events and business object requests.

Example 16-2 shows how to use the `doVerbFor()` method to obtain the verb and verify the connection.

Example 16-2 doVerbFor: obtaining the verb and verifying the connection

```
String method = "doVerbFor: ";
int status = CWConnectorConstant.FAIL;
RMLogger.trace(RMLogger.LEVEL4, method + "Entering into the method");
CWConnectorExceptionObject exObj = new CWConnectorExceptionObject();
// verify business object (bo) is not null
if (bo == null) {
    RMLogger.logMsg(50000, CWConnectorConstant.FAIL);
    exObj.setStatus(CWConnectorConstant.FAIL);
    exObj.setMsg(method + "Invalid Business Object passed in");
    throw new VerbProcessingFailedException(exObj);
}
```

```

    }
    //verify if connections from connection pool is lost
    RMSEServerRMIInterface connection = null;
    try {
        // verifying connection from connection pool if is active
        connection = connectionPool.getConnection();
    }
    catch (Exception e) {
        RMLogger.logMsg(50001, CWConnectorConstant.FAIL);
        exObj.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        exObj.setMsg(method + "Connection timed out");
        throw new VerbProcessingFailedException(exObj);
    }

    // verify if verb is valid
    if (verb == null || verb.intern() == "".intern()) {
        RMLogger.logMsg(50000, CWConnectorConstant.FAIL);
        exObj.setStatus(CWConnectorConstant.FAIL);
        exObj.setMsg(method + "Invalid Business Object passed in");
        throw new VerbProcessingFailedException(exObj);
    }
    // branch on valid action verb
    // create(bo, connection), retrieve(bo, connection),
    // update(bo, connection), delete(bo, connection).

```

16.2.3 Branching on the active verb

Verb processing ensures that the application performs the operation that is associated with the active verb. The action to take on the active verb depends on whether the `doVerbFor()` method has been designed as a *basic method* or a *metadata-driven* method.

Tip: You can obtain a list of a business object's supported verbs with the `getSupportedVerbs()` method of the `CWConnectorBusObj` class.

Attention: As part of the verb-branching logic, you must include a test for an invalid verb. If the request business object's active verb is not supported by the business object definition, the business object handler must take the appropriate recovery actions to indicate an error in verb processing.

Example 16-3 on page 261 shows a code fragment of the `dispatchOnVerb()` method that branches off the business object active verb's value. For each verb the business object supports, you must provide a branch in this code.

Example 16-3 Example code of branching on the active verb

```
// branch on valid action verb
// create(bo, connection);
// retrieve(bo, connection);
// update(bo, connection);
// delete(bo, connection);
if (verb.equals(CWConnectorConstant.VERB_CREATE)) {
    status = create(bo, connection);
}
else if (verb.equals(CWConnectorConstant.VERB_RETRIEVE)) {
    status = retrieve(bo, connection);
}
else if (verb.equals(CWConnectorConstant.VERB_UPDATE)) {
    status = update(bo, connection);
}
else if (verb.equals(CWConnectorConstant.VERB_DELETE)) {
    status = delete(bo, connection);
}
else {
    //unsupported verb
    RMLogger.trace(RMLogger.LEVEL5,
        "dispatchOnVerb: Unsupported verb");
    CWConnectorExceptionObject exObj = new CWConnectorExceptionObject();
    exObj.setStatus(CWConnectorConstant.FAIL);
    exObj.setMsg(method + "Unsupported verb:"+verb);
    throw new VerbProcessingFailedException(exObj);
}
```

The code fragment in Example 16-3 is modularized. It puts the actual processing of each supported verb into a separate verb method, such as `create()`, `update()`, and so on. Be sure that each verb method meets the following minimal guidelines:

- ▶ Defines a `CWConnectorBusObj` parameter, so that the verb method can receive the request business object, and possibly send this updated business object back to the calling method.
- ▶ Throws any verb-specific exceptions to notify the `doVerbFor()` method of any verb-processing errors it encountered.
- ▶ Returns an outcome status, which the `doVerbFor()` method can then return to the Adapter Framework.

This modular structure greatly simplifies the readability and maintainability of the `doVerbFor()` method.

Note: For metadata-driven verb processing, the application-specific information for the verb contains metadata, which provides processing instructions for the request business object when that particular verb is active. The `getVerbAppText()` method is provided by the `CWConnectorBusObj` class to obtain application-specific information for the verb of a business object.

Tip: The verb application-specific information can contain the name of the method to call to process the request business object for that particular verb. In this case, the `doVerbFor()` method does not need to branch off the value of the active verb because the processing information resides in the application-specific information.

16.3 Performing the verb operation

Most verb operations involve obtaining information from the request business object. This section provides information about the steps that our `doVerbFor()` method takes to process the request business object for each active verb.

Important: These steps assume that your connector is designed to be *metadata-driven*. That is, they describe how to extract application-specific information from the business object definition and attributes to obtain the location within the application associated with each attribute. If your connector is not metadata-driven, you do not need to perform any steps that extract application-specific information.

16.3.1 Accessing the business object

As part of the Java connector, the `doVerbFor()` method receives the request business object as an instance of the `CWConnectorBusObj` class. To begin verb processing, the `doVerbFor()` method needs information from the business object definition. The `CWConnectorBusObj` class provides access to the business object, its business object definition, and attributes. Therefore, a Java `doVerbFor()` method does not need to instantiate a separate object for the business object definition. It can obtain information about the business object definition directly from the `CWConnectorBusObj` object passed into `doVerbFor()`.

A business object handler typically uses the business object definition to get information about its attributes or to get the application-specific information from the business object definition, attribute, or verb.

Extracting business object application-specific information

Business objects for metadata-driven connectors are usually designed to have application-specific information about the application structure. For such connectors, the first step in a typical verb operation is to retrieve the application-specific information from the business object definition associated with the request business object. The Java connector library provides the following methods to retrieve application-specific information from the business object definition:

- ▶ The `getAppText()` method, with no arguments, returns the application-specific information as a Java string. It can also retrieve the value of a specified name-value pair within the business object level application-specific information.
- ▶ The `getBusObjASIShashtable()` method returns the application-specific information as a Java hash table of name-value pairs.

Accessing the attributes

The connector can use attribute methods in the `CWConnectorBusObj` class to obtain information about an attribute, such as its cardinality or maximum length. Methods that access attribute properties provide the ability to access an attribute in the following ways:

- ▶ Attribute name
You can identify the attribute by its `Name` property to obtain its attribute object.
- ▶ Integer index
To obtain the attribute index (its ordinal position), you can:
 - Obtain a count of all attributes in the business object definition with `getAttrCount()` and loop through them one at a time, passing each index value to one of its attribute-access methods
 - Obtain the index for a particular attribute by specifying its name to `getAttrIndex()`.

Note: Both the `getAttrCount()` and `getAttrIndex()` methods are defined in the `CWConnectorBusObj` class.

Extracting attribute application-specific information

Business objects for metadata-driven connectors are designed to have application-specific information provide information about the application structure. The next step is to extract the application-specific information from each attribute in the request business object. The Java connector library provides methods to retrieve application-specific information from each attribute:

- ▶ The `getAppText()` method returns the application-specific information as a Java string. It can also retrieve the value of a specified name-value pair within the attribute application-specific information.
- ▶ The `getAttrASIShashtable()` method returns the application-specific information as a Java Hash table of name-value pairs.

Tip: If business objects have been designed to have application-specific information provide information for a table-based application, the application-specific information for the attribute can contain the name of the table column associated with this attribute. After extracting the application-specific information from the business object definition, the next step is to determine what columns in the application table are associated with the attributes in the request business object.

A verb operation can call `getAppText()` and pass it the position or name of the attribute to obtain the name of the column within the database table to access. To obtain the application-specific information for each attribute, the verb operation must loop through all attributes in the business object definition. Therefore, it must determine the total number of attributes in the business object definition. The most common syntax for looping through the attributes is a for statement that uses the following limits on the loop index:

- ▶ Loop index initialized to zero
If the verb operation processes the first attribute containing the key, the loop index variable starts at zero. However, if the verb is Create and your application generates keys, the Create verb operation must not process attributes containing keys. In this case, the loop index variable starts at a value other than zero.
- ▶ Loop index increments until they reach the total number of attributes in the business object definition
The `getAttrCount()` method returns the total number of attributes in the business object. However, this total includes the `ObjectEventId` attribute. Because the `ObjectEventId` attribute is used by the IBM WebSphere business integration system and is not present in application tables, a verb operation does not need to process this attribute. Therefore, when looping through

business object attributes, loop from zero to one less than the total number of attributes (that is, `getAttrCount() - 1`).

- Loop index increments by one

This increment of the index obtains the next attribute.

Within the for loop, the Java connector can use the `getAppText()` method to obtain each attribute's application-specific information (Example 16-4).

Example 16-4 How to extract information from the business object attributes

```
int attrCount = bo.getAttrCount() - 1; //ignore objeventID
for (i = 0; i < attrCount; i++) {
    String columnName = bo.getAppText(i, ATTR_TAG, DELIMITER);
    .....
    .....
}
```

Determining whether to process an attribute

Up to this point, the verb processing has used the application-specific information to obtain the application location for each attribute of the request business object. With this location information, the verb operation can begin processing the attribute.

As the verb operation loops through the business object attributes, you might need to confirm that the operation processes only certain attributes. The Java connector library provides the following methods to determine whether an attribute must be processed:

- `isObjectType()`

An attribute is a simple attribute and not an attribute that represents a contained business object.

- `isIgnore()`, `isBlank()`

The value of the attribute is not the special value of Blank (a zero-length string) or Ignore (a null pointer).

Extracting attribute values from a business object

When the verb operation has confirmed that the attribute is ready for processing, it usually needs to extract the attribute value in the following ways:

- For a Create or Update verb, the verb operation needs the attribute value to send it to the application, where it can be added to the appropriate application entity. For an Update verb, the verb operation also needs the attribute value from any key attribute that holds search information. The application uses this search information to locate the entity to update.

- ▶ For a Retrieve, RetrieveByContent, or Exists verb, the verb operation needs the attribute value from any key attribute (Retrieve or Exists) or non-key attribute (RetrieveByContent) that holds search information. The application uses this search information to retrieve the entity.
- ▶ For a Delete verb, the verb operation needs the attribute value from any key attribute that holds search information. The application uses this search information to locate the entity to delete.

The CWConnectorBusObj class provides type-specific methods for obtaining attribute values. These methods remove the need to cast the attribute value to match its type. You can choose which type-specific method to use by checking the attribute's data type with the `getTypeName()` or `getTypeNum()` method.

Saving attribute values in a business object

When the application operation has completed successfully, the verb operation might need to save new attribute values that are retrieved from the application into the request business object in the following ways:

- ▶ For a Create verb, the verb operation needs to save the new key values if the application has generated them as part of its Create operation.
- ▶ For an Update verb, the verb operation needs to save all attribute values. This includes any generated key values if the application has been designed to create a new entity when it does not find the specified entity to update.
- ▶ For a Retrieve or RetrieveByContent, the verb operation needs to save the attribute value for any attributes retrieved.

The CWConnectorBusObj class provides the following ways to save attribute values:

- ▶ The `setAttrValues()` method saves values for all attributes in a business object. It accepts the attribute values in a Java Vector object.
- ▶ The following methods save values in a business object and are type-specific:
 - `setbooleanValue()`
 - `setBusObjValue()`
 - `setdoubleValue()`
 - `setfloatValue()`
 - `setintValue()`
 - `setLongTextValue()`
 - `setStringValue()`

These methods remove the need to cast the attribute value to match its type. We can choose which type-specific method to use by checking the attribute's data type with the `getTypeName()` or `getTypeNum()` method.

16.3.2 Implementing our verb operation

After the verb operation has obtained the information it needs from the request business object, it is ready to send the application-specific command so that the application performs the appropriate operation. The command must be appropriate for the verb of the request business object. For a table-based application, this command might be an SQL statement or a JDBC call.

Note: In our scenario, the back-end application comes with a connection object that provides to an external application methods to create, retrieve, delete, or update single application entities. No other APIs are provided from the back-end application, such as an API to get primary keys or foreign keys that are owned by an entity application.

Implementing the create operation

The following steps outline the implementation for the Create verb:

1. Create an instance of `RMDataInterface`, the unique application entity wrapper provided from the back-end application. Set up its attributes using the data from the business object.
2. Set foreign key attributes in any current-level business objects from the value of the parent-level primary key. (If you stay with a root-level business object, you do not do this step).
3. Invoke the API to create the corresponding application entity.
Because the API generates its own primary key, you get these key values for insertion in the current-level business object.
4. Recursively create the application entities corresponding to the first-level child business objects. Continue recursively creating all child business objects at all subsequent levels in the business object hierarchy.

Example 16-5 shows our create operation implementation.

Example 16-5 Implementation of the create operation

```
private int create(
    CWConnectorBusObj bo,
    RMServerRMIInterface connection)
    throws CWException {
    String method = "create: ";
    RMLogger.trace(
        RMLogger.LEVEL4,
        method
        + "Entering into the method for BusinessObject:"
        + bo.getName());
    int status = CWConnectorConstant.FAIL;

    RMDDataInterface rmData = convertB0toRMDData(bo);

    //check the component name provided as ASI attribute at B0 level;
    String compName = bo.getAppText();
    rmData.setComponentName(compName);

    //setting foreign keys from parent keys if the current bo is a child bo
    CWConnectorBusObj parentBusinessObject =
        bo.getParentBusinessObject();
    if (parentBusinessObject != null) {
        //If parent is not null, this is a child
        Vector parentKeyValue =
            getKeys(parentBusinessObject);
        Vector parentKeyName =
            getKeyFieldName(parentBusinessObject);

        Iterator iterValue = parentKeyValue.iterator();
        for (Iterator iter = parentKeyName.iterator();
            iter.hasNext();
        ) {
            String keyName = (String) iter.next();
            String keyValue = (String) iterValue.next();
            rmData.setAttr(keyName, keyValue);
        }
        RMLogger.trace(
            CWConnectorUtil.LEVEL5,
            method
            + "Parent-child relationship: creating child with "
            + parentKeyName
            + "="
            + parentKeyName);
    }
}
```

```

##### INVOCATION #####
RMDDataInterface newRMDData = null;
try {
    newRMDData = connection.createObject(rmData);
    RMLogger.trace(
        RMLogger.LEVEL5,
        method + "createObject invocation OK");
}
catch (RemoteException rEx) {
    logAndThrowException(
        CWConnectorConstant.class,
        50504,
        CWConnectorConstant.APPRESPONSETIMEOUT,
        bo.getName());
}

//filling bo's primary keys from resulting RMDDataInterface
Vector keyNames = getKeyFieldName(bo);
for (Iterator iter = keyNames.iterator();
    iter.hasNext();
    ) {
    String name = (String) iter.next();
    String keyName =
        newRMDData.getAttr(
            bo.getAppText(
                name,
                ATTR_TAG,
                DELIMITER));
    if (keyName != null)
        bo.setStringValue(name, keyName);
}
status = CWConnectorConstant.VALCHANGE;

RMLogger.trace(
    CWConnectorUtil.LEVEL4,
    method + "Exiting ");

return status;
}

```

Implementing the retrieve operation

The following steps outline implementing the Retrieve verb:

1. Create an instance of RMDDataInterface and then set up its attributes using the primary key from the business object definition.
2. Start the API to retrieve the corresponding application entity to the current business object.

3. Make a loop to seed each business object's simple attribute from the RMDDataInterface.
4. Make a loop to retrieve recursively all child business objects.
 - a. Prepare a new RMDDataInterface instance to request from the back-end application all application entities related to the parent entity.
 - b. Retrieve by value all child business objects using their foreign keys.
 - c. Set up the child business object's key attributes using the results.

Note: The cross-reference between parent and child application entities is represented in the business objects as application-specific information. Write, as an application-specific information value in the business object definition, the following identity:

```
fkey=business_object_name.attribute_name
```

In this example, *business_object_name* is the parent business object if the foreign key is in the child or child business object if the foreign key is in the parent.

Example 16-6 shows the retrieve operation implementation that retrieves all the child business objects.

Example 16-6 Implementation of the retrieve operation

```
private int retrieve(CWConnectorBusObj bo, RMServerRMIInterface connection)
    throws CWException {
    String method = "retrieve: ";
    RMLogger.trace(CWConnectorUtil.LEVEL4, method + "retrieve: Entering");
    int status = CWConnectorConstant.FAIL;
    try {
        RMDDataInterface rmData = new RMDDataImplementation();

        //getting the component name provided as ASI attribute at BO level;
        String compName = getBOAppText(bo);
        rmData.setComponentName(compName);

        //setting the keys of the current bo into the RMDDataInterface
        Vector keyName = getKeyFieldName(bo);
        Vector keyVal = getKeys(bo);
        Iterator valueIter = keyVal.iterator();
        for (Iterator iter = keyName.iterator(); iter.hasNext();) {
            String element = (String) iter.next();
            String elementValue = (String) valueIter.next();
            rmData.setAttr(element, elementValue);
        }
        RMLogger.trace(
```



```

        RMLogger.LEVEL5,
        method
            + "keys for the current BO: "
            + bo.getName()
            + " are: "
            + keyName
            + "="
            + keyVal);

//retrieving the parant BO
RMDataInterface respRMData = null;
respRMData = connection.retrieveObject(rmData);
if (respRMData == null)
    return CWConnectorConstant.BO_DOES_NOT_EXIST;

//seting the attributes of the parent BO
int attrCount = bo.getAttrCount() - 1;
//ignore objeventID
String incomingAttribute = "";
String columnName = null;
for (int i = 0; i < attrCount; i++) {
    if (!bo.isObjectType(i)) {
        columnName = bo.getAppText(i, ATTR_TAG, DELIMITER);
        bo.setStringValue(i, respRMData.getAttr(columnName));
        RMLogger.trace(
            RMLogger.LEVEL5,
            "Application data TO BO "
            + columnName
            + "="
            + respRMData.getAttr(columnName));
    }
}

//loop for each attribute of the parent bo to build up recursively
the hierarchy
//ignore objeventID
for (int i = 0; i < attrCount; i++) {
    if (bo.isObjectType(i)) {
        String childBusinessObjectName = bo.getTypeName(i);
        CWConnectorBusObj childDef =
            CWConnectorUtil.createBusObj(childBusinessObjectName);

        int childObjectCount = bo.getObjectCount(i);
        for (int j = 0; j < childObjectCount; j++) {
            CWConnectorBusObj currentChild =
                bo.getBusObjValue(i, j);

            currentChild.setVerb(CWConnectorConstant.VERB_RETRIEVE);
            dispatchOnVerb(currentChild, connection);
        }
    }
}

```

```

    }

    if (childObjectCount == 0) {
        //finding child BO
        Vector vetChildBo =
            findChildsByForeignKeys(
                connection,
                respRMDData,
                bo,
                childBusinessObjectName,
                CWConnectorConstant.VERB_RETRIEVE);
        for (int j = 0; j < vetChildBo.size(); j++) {
            CWConnectorBusObj currentChild =
                (CWConnectorBusObj) vetChildBo.elementAt(j);
            dispatchOnVerb(currentChild, connection);
            bo.setBusObjValue(i, currentChild, j);
        }
    }
}

}

}

}

catch (RemoteException e) {
    logAndThrowException(
        CWException.class,
        50202,
        CWConnectorConstant.BO_DOES_NOT_EXIST,
        bo.getName());
}

RMLogger.trace(RMLogger.LEVEL4, method + "Exiting");

return CWConnectorConstant.VALCHANGE;
}

```

Example 16-7 shows retrieving the child business object by foreign keys.

Example 16-7 Retrieving the child business object by their foreign keys

```
private Vector findChildsByForeignKeys(
    RMServerRMIInterface connection,
    RMDDataInterface rmData,
    CWConnectorBusObj bo,
    String childBoName,
    String verb)
    throws CWException {
    Vector childBOs = new Vector();
    try {

        //loop on the child BO definition to looking for
        //its foreign key that match with the parent bo.
        //Once found, we set into the reqRMDData the value of the
        //child object's foreign key and execute a
        //retrieve by value in order to set the primary key of the
        //all instances of the retrieved child business objects.
        RMDDataInterface reqRMDData = new RMDDataImplementation();
        CWConnectorBusObj currentChild =
            CWConnectorUtil.createBusObj(childBoName);
        reqRMDData.setComponentName(getBOAppText(currentChild));
        Vector foreignKeyAttr = new Vector();
        for (int i = 0; i < currentChild.getAttrCount(); i++) {
            //looking for foreign keys into the child object
            try {
                String fkey =
                    currentChild.getAppText(i, FKEY_TAG, DELIMITER);
                if (fkey != null && fkey.intern() != "").intern()) {
                    //check if the found fkey references the parent BO
                    int dotPos = fkey.indexOf(".");
                    String boFKeyOwner = fkey.substring(0, dotPos);
                    String fkeyAttrName = fkey.substring(dotPos + 1);
                    if (boFKeyOwner.intern() == bo.getName().intern()) {
                        foreignKeyAttr.addElement(fkeyAttrName);
                        //getting the value of the foreign key from the
                        rmData containing the retrieved parent bo.
                        //since rmData fields have ASI names we use the ASI
                        value held of child BO attribute
                        String parentKeyASIName =
                            bo.getAppText(
                                fkeyAttrName,
                                FKEY_TAG,
                                DELIMITER);
                        String childFKeyASIName =
                            currentChild.getAppText(i, ATTR_TAG,
                                DELIMITER);
```

```

        String fkeyValue =
rmData.getAttr(parentKeyASIName);
        if (fkeyValue != null
            && fkeyValue.intern() != "".intern())
            reqRMDData.setAttr(
                childFKeyName,
                rmData.getAttr(parentKeyASIName));
    }
}
}
catch (WrongASIFormatException wEx) {
}
}
//retrieving childs byValue
RMDDataInterface respRMDData =
    connection.retrieveObjectByValue(reqRMDData);
Vector vetRMDData = respRMDData.getRetrieveByValue();

//seeding the key attributes of the current child object
for (Iterator iter = vetRMDData.iterator(); iter.hasNext();) {
    CWConnectorBusObj childBOInstance =
        CWConnectorUtil.createBusObj(childBoName);
    RMDDataInterface element = (RMDDataInterface) iter.next();
    for (Iterator iterator = foreignKeyAttr.iterator();
        iterator.hasNext();
    ) {
        String fKeyName = (String) iterator.next();
        String fkeyASIName =
            childBOInstance.getAppText(
                fKeyName,
                FKEY_TAG,
                DELIMITER);
        childBOInstance.setStringValue(
            fKeyName,
            element.getAttr(fkeyASIName));
    }
    childBOInstance.setVerb(verb);
    childBOs.addElement(childBOInstance);
}

//retrieving the foreign keys on the parente bo and
//the primary key ond the child bo.
//Once found, setting the child primary key field with
//the values from parent bo (these values are in the rmData)
CWConnectorBusObj childBOInstance =
    CWConnectorUtil.createBusObj(childBoName);
boolean found = false;
for (int i = 0; i < bo.getAttrCount(); i++) {
    try {

```

```

        String fkey = bo.getAppText(i, FKEY_TAG, DELIMETER);
        if (fkey != null && fkey.intern() != "".intern()) {
            int dotPos = fkey.indexOf(".");
            String boFkeyOwner = fkey.substring(0, dotPos);
            String fkeyAttrName = fkey.substring(dotPos + 1);
            if (boFkeyOwner.intern()
                == currentChild.getName().intern()) {
                //this child has a foreign key...
                found = true;
                String fkeyValue =
                    rmData.getAttr(
                        bo.getAppText(i, ATTR_TAG, DELIMETER));
                childBOInstance.setStringValue(
                    fkeyAttrName,
                    fkeyValue);
            }
        }
    }
    catch (WrongASIFormatException ex) {
    }
}
if (found) {
    childBOInstance.setVerb(verb);
    childBOs.addElement(childBOInstance);
}

}
catch (RemoteException e) {
    logAndThrowException(
        CWException.class,
        50200,
        CWConnectorConstant.BO_DOES_NOT_EXIST,
        bo.getName());
}

return childBOs;
}

```

Implementing event notification

This chapter describes how we implement the event notification mechanism in our custom adapter. Defining an event store for our custom adapter involves:

- ▶ Extending the Java event-store class
- ▶ Implementing the `fetchEvents()` method
- ▶ Implementing the `deleteEvent()` method
- ▶ Implementing the `setEventStoreStatus()` method
- ▶ Implementing the `archiveEvent()` method
- ▶ Implementing the `recoverInProgressEvents()` method

17.1 Extending the event store class

To define an event store, a connector developer must derive an event-store class from the `CWConnectorEventStore` class and must implement some of its methods for the event store. An *event store* is an application's mechanism for persistently storing events. For our adapter, we created the `RMEventStore` class.

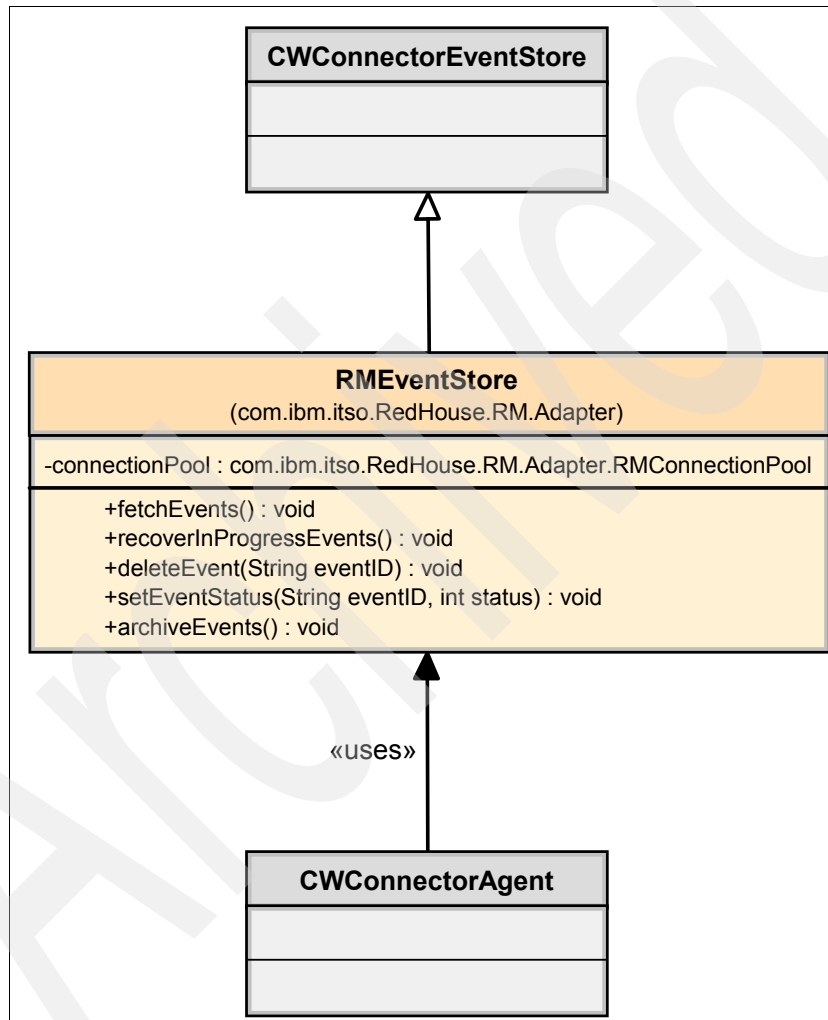


Figure 17-1 The class diagram for the `RMEventStore`

The `CWConnectorEventStore` class is a base class to provide a Java connector with the ability to access an event store. The application stores event records in the event store. The connector retrieves events from the event store and processes them for transferal to the integration broker.

Important: All Java connectors must extend this class to access the application's event store. To access the application's event store through the Java `CWConnectorEventStore` class, implement the following abstract methods in their derived event-store class:

- ▶ `deleteEvent()`
- ▶ `fetchEvents()`
- ▶ `setEventStatus()`

17.1.1 Implementing methods to in the `CWConnectorEventStore`

Table 17-1 summarizes the methods in the `CWConnectorEventStore` class.

Table 17-1 Member methods of the `CWConnectorEventStore` class

Member method	Description
<code>CWConnectorEventStore()</code>	Creates an event-store object.
<code>archiveEvent()</code>	Archives the event in the application's archive store with the appropriate status.
<code>cleanupResources()</code>	Releases resources that the poll method has used to access the event store.
<code>deleteEvent()</code>	Deletes the event from the application's event store.
<code>fetchEvents()</code>	Retrieves a specified number of Ready-for-Poll events from the application's event store.
<code>getB0()</code>	Builds a business object based on the information for an event from the event store.
<code>getNextEvent()</code>	Retrieves the next event object from the <code>eventsToProcess</code> vector.
<code>recoverInProgressEvents()</code>	Recovers any in-progress events in the event store.
<code>resubmitArchivedEvents()</code>	Copies the archived event from the application's archive store to the event store and changes the event status to <code>READY_FOR_POLL</code> .

Member method	Description
setEventStatus()	Sets the status of an event in the event store.
setEventsToProcess()	Sets the eventsToProcess vector with specified events.
setTerminate()	Sets the internal terminate-connector flag to true.
updateEventStatus()	Updates the event status both in the event store and in the event.

All Java connectors must extend this class to access the event store. To access the application's event store through the Java CWConnectorEventStore class, implement the following abstract methods in their derived event-store class:

- ▶ deleteEvent()
- ▶ fetchEvents()
- ▶ setEventStoreStatus()

To access an archive store, implement the archiveEvent() method.

To provide the ability to recover in-progress events, implement the recoverInProgressEvents() method.

To provide the ability to resubmit archived events for subsequent polls of the event store, implement the resubmitArchivedEvents() method.

Example 17-1 shows the skeleton for the RMEventStore class of our custom adapter. See Figure 17-1 on page 278 for the class diagram.

Example 17-1 The extended RMEventStore class

```

public class RMEventStore extends CWConnectorEventStore {
    public void fetchEvents() throws StatusChangeFailedException {
    }
    public void recoverInProgressEvents() throws StatusChangeFailedException {
    }
    public void deleteEvent(String eventID) throws DeleteFailedException {
    }
    public void setEventStatus(String eventID, int status)
        throws InvalidStatusChangeException {
    }
    public void archiveEvents() throws StatusChangeFailedException {
    }
}

```

Note: In the RMAAdapter, we do not provide the ability to resubmit archived events. As a result, we do not implement the resubmitArchiveEvents() method.

17.2 Implementing the fetchEvents() method

The `fetchEvents()` method retrieves a specified number of ready-for-poll events from the event store. This method searches the event store for event records with the `READY_FOR_POLL` status and puts them in the `eventsToProcess` vector.

The number of events that the `fetchEvents()` method retrieves is determined by the value of the `PollQuantity` connector configuration property. For each retrieved event, the method must create a `CWConnectorEvent` event object and put this event object into a Java vector. The method then calls the `setEventsToProcess()` method to save this event vector into the `eventsToProcess` vector, which is a member of the `CWConnectorEventStore` object.

The `fetchEvents()` method determines the order in which event objects are stored in the `eventsToProcess` vector.

Important: The `fetchEvents()` method is an abstract method. Therefore, the event-store class must implement this method to provide the ability to fetch `READY_FOR_POLL` events from the event store.

Example 17-2 shows the `fetchEvents()` method implementation for the `RMEventStore`.

Example 17-2 The `fetchEvents()` method

```
public void fetchEvents() throws StatusChangeFailedException {
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering fetchEvents");
    // find out how many events to process per poll (n)
    int pollQuantity =
        Integer.parseInt(CWConnectorUtil.getConfigProp("PollQuantity"));
    // retrieve (n) number of ready for poll events from the event table
    Vector fetchedEvents =
        fetchEvents(
            pollQuantity,
            CWConnectorEventStatusConstants.READY_FOR_POLL);
    // pass the vector of event objects to setEventsToProcess()
    setEventsToProcess(fetchedEvents);
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting fetchEvents");
}
```

The `fetchEvents()` method uses the private `fetchEvents(pollQuantity, status)` method, as shown in Example 17-3.

Example 17-3 The private `fetchEvents(pollQuantity, status)` method

```
private Vector fetchEvents(int pollQuantity, int status)
    throws StatusChangeFailedException {
    RMLogger.trace(RMLogger.LEVEL4, "Entering fetchEvents(int, int)");

    Vector fetchedEvents = new Vector();

    // obtain connection from connection pool
    RMServerRMIInterface connection;
    try {
        connection = (RMServerRMIInterface) connectionPool.getConnection();
    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to obtain connection to the application");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new StatusChangeFailedException(c);
    }

    // retrieve <pollQuantity> number of <status> events from the event table
    try {
        // for each event retrived...

        String eventId = "0";
        boolean otherEvents = true;
        int eventRetrieved = 0;

        while (otherEvents && eventRetrieved < pollQuantity) {
            //get the RMDDataImplementation object
            RMDDataImplementation inpDataImpl =
                getRMDDataImplementationInstance(eventId, status);
            //...retrieve the object
            try {
                RMDDataImplementation outDataImpl =
                    (RMDDataImplementation) connection.retrieveObject(
                        inpDataImpl);
                // ...create a CWConnectorEvent object...
                CWConnectorEvent aEvent =
                    getEventFromRMDDataImplementation(outDataImpl);
                //...save the key for the next event to retrieve...
                eventId = aEvent.getEventID();
                // ... and add it to the vector of fetched events
                fetchedEvents.addElement(aEvent);
                eventRetrieved++;
            } catch (RemoteException e) { //no other events to retrieve
                otherEvents = false;
            }
        }
    }
}
```

```

    }
}

RMLogger.trace(
    CWConnectorUtil.LEVEL5,
    "Event vector contains " + fetchedEvents.size() + " events");

// sort the vector of event objects
sortEvents(fetchedEvents);

// free up connection from connection pool
connectionPool.releaseConnection(connection);

} catch (AttributeNullValueException e) {
    CWConnectorExceptionObject c = new CWConnectorExceptionObject();
    c.setMsg("Found null value while retrieving events");
    c.setStatus(CWConnectorConstant.FAIL);
    throw new StatusChangeFailedException(c);
}

RMLogger.trace(RMLogger.LEVEL4, "Exiting fetchEvents(int, int)");
return fetchedEvents;
}

```

17.3 Implementing the deleteEvents() method

The `deleteEvent()` method deletes the event from the event store. This method is used mainly during archiving. It deletes the event from the event store after this event has been successfully moved to the application's archive store.

Important: The `deleteEvent()` method is an abstract method. Therefore, the event-store class must implement this method to provide the ability to delete an event from the event store.

Example 17-4 shows the deleteEvents() method implementation for the RMEventStore.

Example 17-4 The deleteEvents() method.

```
public void deleteEvent(String eventID) throws DeleteFailedException {
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering deleteEvent");

    // obtain connection from connection pool
    RMServerRMIInterface connection;
    try {
        connection = (RMServerRMIInterface) connectionPool.getConnection();
    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to obtain connection to the application");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new DeleteFailedException(c);
    }

    try {
        // delete event from event table
        RMDataImplementation aDataImpl =
            getRMDataImplementationInstance(eventID);
        connection.deleteObject(aDataImpl);

        // free up connection from connection pool
        connectionPool.releaseConnection(connection);

    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to delete event from the event table");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new DeleteFailedException(c);
    }

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting deleteEvent");
}
```

17.4 Implementing the setEventStatus() method

The setEventStatus() method sets the status of an event in the event store. It:

- ▶ Checks that the status value is valid and throws the InvalidStatusChangeException exception if it is not.
- ▶ Changes the status of the event that is identified by eventID in the event store.

Example 17-5 shows the setEventStatus() method implementation for the RMEventStore.

Example 17-5 The setEventStatus() method

```
public void setEventStatus(String eventID, int status) throws
InvalidStatusChangeException {

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering setEventStatus");
    // obtain connection from connection pool
    RMServerRMIInterface connection;
    try {
        connection = (RMServerRMIInterface) connectionPool.getConnection();
    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to obtain connection to the application");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new InvalidStatusChangeException(c);
    }

    try {
        // set status of event in event table to the status provided
        RMDataImplementation aDataImpl =
            getRMDataImplementationInstance(eventID, status);
        connection.updateObject(aDataImpl);

        // free up connection from connection pool
        connectionPool.releaseConnection(connection);

    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to delete event from the event table");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new InvalidStatusChangeException(c);
    }

    RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting setEventStatus");
}
```

17.5 Implementing the archiveEvents() method

The `archiveEvent()` method archives the specified event in the archive store with the appropriate status. This method is usually called from the `poll` method, `pollForEvents()`, to archive processed or unsuccessful events to the event archive store.

Important: The `archiveEvent()` method is not an abstract method. It is a synchronized method. However, the event-store class must implement this method to provide the ability to archive an event to the archive store.

This method returns an integer that indicates the outcome status of the archive operation. Compare this integer value with the following outcome-status constants to determine the status:

- ▶ `CWConnectorConstant.SUCCEED`
The archiving of the event succeeded.
- ▶ `CWConnectorConstant.FAIL`
The archiving of the event failed.

Example 17-6 shows the `archiveEvents()` method implementation for the `RMEventStore`.

Example 17-6 The archiveEvents() method

```
public synchronized int archiveEvent(String eventID)
    throws ArchiveFailedException, InvalidStatusChangeException {
    RMLogger.trace(CWConnectorUtil.LEVEL4, "Entering archiveEvent");

    // obtain connection from connection pool
    RMServerRMIIInterface connection;
    try {
        connection = (RMServerRMIIInterface) connectionPool.getConnection();
    } catch (Exception e) {
        CWConnectorExceptionObject c = new CWConnectorExceptionObject();
        c.setMsg("Unable to obtain connection to the application");
        c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
        throw new ArchiveFailedException(c);
    }

    try {
        //get the RMDataImplementation instance for the event
        RMDataImplementation inpDataImpl =
            getRMDataImplementationInstance(eventID);

        //...retrieve the event record from event table
```



```

RMDataImplementation outDataImpl =
    (RMDataImplementation) connection.retrieveObject(inpDataImpl);

// archive event record into archive table
connection.archiveObject(outDataImpl);

// delete event record from current event table
connection.deleteObject(inpDataImpl);

// free up connection from connection pool
connectionPool.releaseConnection(connection);

} catch (Exception e) {
    CWConnectorExceptionObject c = new CWConnectorExceptionObject();
    c.setMsg("Unable to archive the event");
    c.setStatus(CWConnectorConstant.APPRESPONSETIMEOUT);
    throw new ArchiveFailedException(c);
}

RMLogger.trace(CWConnectorUtil.LEVEL4, "Exiting archiveEvent");
return super.archiveEvent(eventID);
}

```

17.6 Implementing the recoverInProgress() method

The `recoverInProgressEvents()` method recovers any in-progress events in the event store by checking the event store for any events that currently have the `IN_PROGRESS` status. An event might remain in the event store with an event status of `IN_PROGRESS` if the connector was unexpectedly shut down.

Note: The `CWConnectorEventStore` class does not provide a default implementation for the `recoverInProgressEvents()` method. Therefore, the event-store class must implement this method to provide the ability to recover in-progress events at connector startup.

One possible way to implement the `recoverInProgressEvents()` method is to base its actions on the `InDoubtEvents` connector configuration property. If such events exist, the method can take one of the actions that are listed in Table 17-2 on page 288, based on the value of this property.

Table 17-2 Possible action for the `recoverInProgressEvents()` method

Value of <code>InDoubtEvents</code>	Action for <code>recoverInProgressEvents()</code>
Reprocess	Changes all events with the <code>IN_PROGRESS</code> status to <code>READY_FOR_POLL</code> status so that they are sent to the Adapter Framework in subsequent poll calls.
FailOnStartup	Logs a fatal error and returns a <code>FAIL</code> status to <code>agentInit()</code> . The <code>agentInit()</code> method, throws the <code>InProgressEventRecoveryFailedException</code> exception. This action also sends an automatic e-mail if <code>LogAtInterchangeEnd</code> is set to <code>True</code> .
LogError	Logs a fatal error but does not return a <code>FAIL</code> outcome status to <code>agentInit()</code> .
Ignore	Ignore the in-progress events.

Note: For `recoverInProgressEvents()` to work as described, the `InDoubtEvents` connector configuration property must be defined. If `InDoubtEvents` is not defined, `recoverInProgressEvents()` throws the `AttributeNullException` exception.

The `recoverInProgressEvents()` method returns an integer that indicates the outcome status of the recovery operation. Compare this integer value with the following outcome-status constants to determine the status:

- ▶ `CWConnectorConstant.SUCCEED`
The recovery of in-progress events succeeded.
- ▶ `CWConnectorConstant.FAIL`
The recovery of in-progress events failed.

The `recoverInProgressEvents()` method is usually called as part of the connector initialization process from within the `agentInit()` method. The `agentInit()` checks for the status from `recoverInProgressEvents()` and catches any exceptions as well. The `agentInit()` method throws an exception in either of the following cases:

- ▶ If `recoverInProgressEvents()` returns a `FAIL` outcome status.
- ▶ If `recoverInProgressEvents()` catches an exception.

For our adapter, you can change all events with the IN_PROGRESS status to the READY_FOR_POLL status, as shown in Example 17-7.

Example 17-7 The recoverInProgressEvents() method

```
public int recoverInProgressEvents()
    throws
        StatusChangeFailedException,
        InvalidStatusChangeException,
        AttributeNullValueException {
    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Entering recoverInProgressEvents");

    // fetch in-progress events
    Vector fetchedEvents =
        fetchEvents(
            Integer.MAX_VALUE,
            CWConnectorEventStatusConstants.IN_PROGRESS);

    // for each in progress event...
    for (int i = 0; i < fetchedEvents.size(); i++) {
        // ...set the status to ready for poll
        RMDDataImplementation event =
            (RMDDataImplementation) fetchedEvents.elementAt(i);
        setEventStatus(
            event.getAttr("EventId"),
            CWConnectorEventStatusConstants.READY_FOR_POLL);
    }

    RMLogger.trace(
        CWConnectorUtil.LEVEL4,
        "Exiting recoverInProgressEvents");
    return super.recoverInProgressEvents();
}
```

Polling for events

This chapter shows how to implement the polling mechanism for events, polling the event store for new events to be processed.

18.1 Using the pollForEvents() method

The CWConnectorAgent class is the default implementation for the pollForEvents() method. You can override this method to provide your own implementation.

Figure 18-1 shows a sequence diagram of the basic logic for the pollForEvents() method.

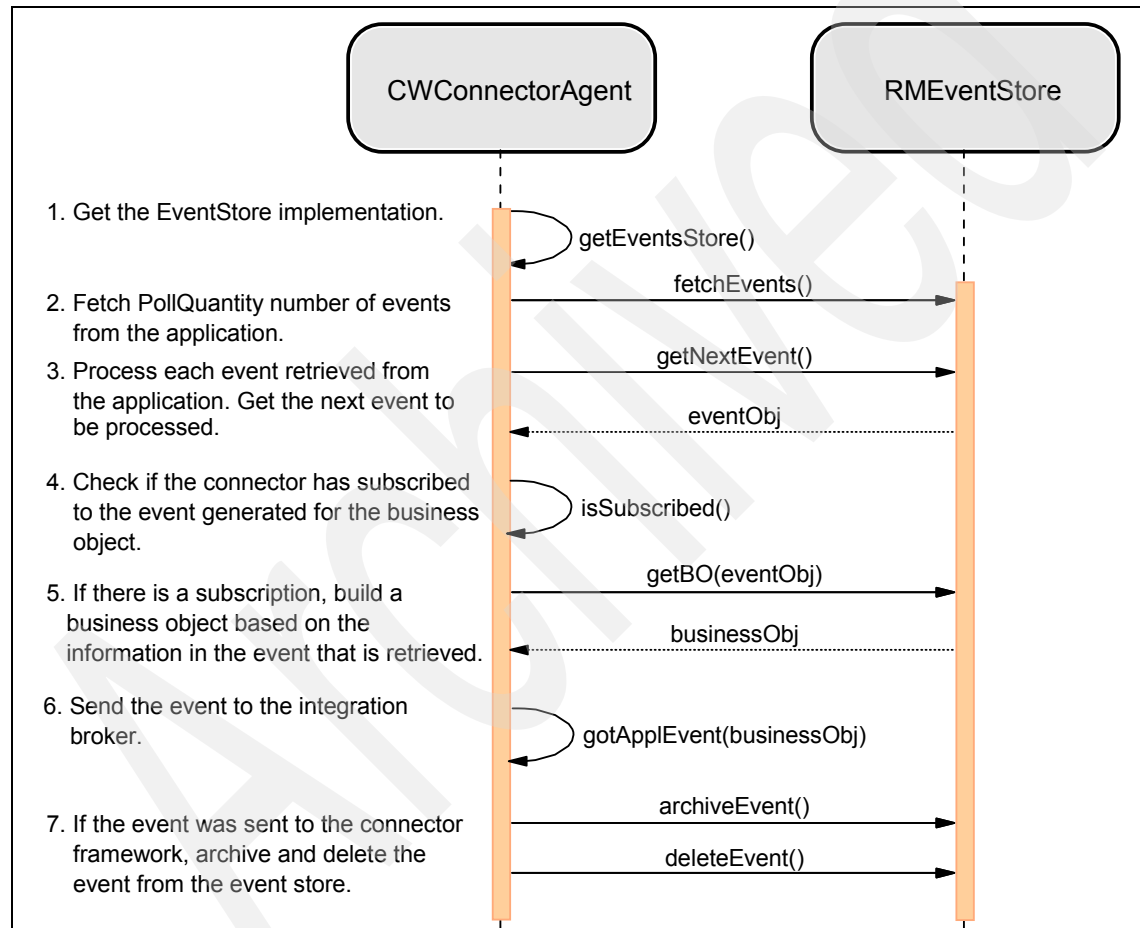


Figure 18-1 A sequence diagram for the pollForEvents() method

Example 18-1 shows the implementation of basic logic for the `pollForEvents()` method.

Note: For the adapter, use the default implementation because it meets the needs of the application.

Example 18-1 Default implementation for the `pollForEvents()` method

```
/** * Default implementation of pollForEvents. */
public int pollForEvents() {
    CWConnectorUtil.traceWrite(
        CWConnectorLogAndTrace.LEVEL5,
        "Entering pollForEvents.");

    // Get the EventStoreFactory implementation name from the
    // getEventStore() method.
    CWConnectorEventStore evts = getEventStore();
    if (evts == null) {
        CWConnectorUtil.generateAndLogMsg(
            10533,
            CWConnectorLogAndTrace.XRD_ERROR,
            0,
            0);
        return CWConnectorConstant.APPRESPONSETIMEOUT;
    }
    try {
        // Fetch PollQuantity number of events from the application.
        try {
            evts.fetchEvents();
        } catch (StatusChangeFailedException e) {
            CWConnectorUtil.generateAndLogMsg(
                10533,
                CWConnectorLogAndTrace.XRD_ERROR,
                0,
                0);
            CWConnectorUtil.logMsg(e.getMessage());
            e.printStackTrace();
            return CWConnectorConstant.APPRESPONSETIMEOUT;
        }
        // Get the property values for PollQuantity and ArchiveProcessed.
        int pollQuantity;
        String poll = CWConnectorUtil.getConfigProp("PollQuantity");
        try {
            if (poll == null || poll.equals(""))
                pollQuantity = 1;
            else
                pollQuantity = Integer.parseInt(poll);
        } catch (NumberFormatException e) {
            CWConnectorUtil.generateAndLogMsg(
```

```

        10544,
        CWConnectorLogAndTrace.XRD_ERROR,
        0);
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.FAIL;
}
String arcProcessed = CWConnectorUtil.getConfigProp("ArchiveProcessed");
// In case the ArchiveProcessed property is not set, use true
// as default.
if (arcProcessed == null || arcProcessed.equals(""))
    arcProcessed = CWConnectorAttrType.TRUESTRING;
CWConnectorEvent evtObj;
CWConnectorBusObj bo = null;
try {
    for (int i = 0; i < pollQuantity; i++) {
        // Process each event retrieved from the application.
        // Get the next event to be processed.
        evtObj = evts.getNextEvent();
        // A null return indicates that there were no events with
        // READY_FOR_POLL status. Return SUCCESS.
        if (evtObj == null) {
            CWConnectorUtil.generateAndLogMsg(
                10534,
                CWConnectorLogAndTrace.XRD_INFO,
                0,
                0);
            return CWConnectorConstant.SUCCEED;
        }
        // Check if the connector has subscribed to the event
        // generated for the business object.
        boolean isSub =
            isSubscribed(evtObj.getBusObjName(), evtObj.getVerb());
        if (isSub) {
            // Retrieve the complete CWConnectorBusObj corresponding
            // to the object using the getBO method in
            // CWConnectorEventStore. This method sets the verb on a
            // temporary business object to RetrieveByContent
            // and retrieves the corresponding data information to be
            // filled in the business object from the application.
            try {
                bo = evts.getBO(evtObj);
                // Terminate flag will be set in the event store when
                // the doVerbFor method returns APPRESPONSETIMEOUT in
                // getBO.
                if (evts.getTerminate())
                    return CWConnectorConstant.APPRESPONSETIMEOUT;
            } catch (AttributeNotFoundException e) {
                CWConnectorUtil.generateAndLogMsg(

```



```

10536,
CWConnectorLogAndTrace.XRD_ERROR,
0,
2,
"getB0",
"AttributeNotFoundException");
CWConnectorUtil.logMsg(e.getMessage());
e.printStackTrace();
// Update the event status to ERROR_PROCESSING_EVENT
evts.updateEventStatus(
    evtObj,
    CWConnectorEventStatusConstants
        .ERROR_PROCESSING_EVENT);
if (arcProcessed
    .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
    // Archive the event in the application's archive store
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store
    evts.deleteEvent(evtObj.getEventID());
}
continue;
} catch (SpecNameNotFoundException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "getB0",
        "SpecNameNotFoundException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    // Update the event status to ERROR_PROCESSING_EVENT
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants
            .ERROR_PROCESSING_EVENT);
    if (arcProcessed
        .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's archive store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
} catch (InvalidVerbException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,

```

```

        2,
        "getB0",
        "InvalidVerbException");
CWConnectorUtil.logMsg(e.getMessage());
e.printStackTrace();
// Update the event status to ERROR_PROCESSING_EVENT
evts.updateEventStatus(
    evtObj,
    CWConnectorEventStatusConstants
        .ERROR_PROCESSING_EVENT);
if (arcProcessed
    .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
    // Archive the event in the application's archive store
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store
    evts.deleteEvent(evtObj.getEventID());
}
continue;
} catch (WrongAttributeException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "getB0",
        "WrongAttributeException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    // Update the event status to ERROR_PROCESSING_EVENT
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants
            .ERROR_PROCESSING_EVENT);
    if (arcProcessed
        .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's archive store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
} catch (AttributeValueException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "getB0",
        "AttributeValueException");

```

```

CWConnectorUtil.logMsg(e.getMessage());
e.printStackTrace();
// Update the event status to ERROR_PROCESSING_EVENT
evts.updateEventStatus(
    evtObj,
    CWConnectorEventStatusConstants
        .ERROR_PROCESSING_EVENT);
if (arcProcessed
    .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
    // Archive the event in the application's archive store
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store
    evts.deleteEvent(evtObj.getEventID());
}
continue;

} catch (AttributeNullValueException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "getB0",
        "AttributeNullValueException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    // Update the event status to ERROR_PROCESSING_EVENT
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants
            .ERROR_PROCESSING_EVENT);
    if (arcProcessed
        .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's archive store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
}

// Log a fatal error in case the object is not found.
if (evtObj.getStatus()
    == CWConnectorEventStatusConstants
        .ERROR_OBJECT_NOT_FOUND) {
    CWConnectorUtil.generateAndLogMsg(
        10543,
        CWConnectorLogAndTrace.XRD_FATAL,
        0,
        0);
}

```

```

// Update the event status to ERROR_OBJECT_NOT_FOUND
evts.updateEventStatus(
    evtObj,
    CWConnectorEventStatusConstants
        .ERROR_OBJECT_NOT_FOUND);
if (arcProcessed
    .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
    // Archive the event in the application's archive store
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store
    evts.deleteEvent(evtObj.getEventID());
}
continue;
}
// In case the business object is null, the retrieve call
// returned an error.
if (bo == null) {
    CWConnectorUtil.generateAndLogMsg(
        10335,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        0);
    // Update the event status to ERROR_PROCESSING_EVENT
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants
            .ERROR_PROCESSING_EVENT);
    if (arcProcessed
        .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's archive store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
}
// Set the processing verb on the business object.
try {
    bo.setVerb(evtObj.getVerb());
} catch (InvalidVerbException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "setVerb",
        "InvalidVerbException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
}

```

```

// Update the event status to ERROR_PROCESSING_EVENT
evts.updateEventStatus(
    evtObj,
    CWConnectorEventStatusConstants
        .ERROR_PROCESSING_EVENT);
if (arcProcessed
    .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
    // Archive the event in the application's archive store
    evts.archiveEvent(evtObj.getEventID());
    // Delete the event from the event store
    evts.deleteEvent(evtObj.getEventID());
}
continue;
}
// Check again for subscription.
if (isSubscribed(bo.getName(), bo.getVerb())) {
    // Send the event to integration broker.
    int stat = gotAppEvent(bo);
    if (stat == CWConnectorConstant.CONNECTOR_NOT_ACTIVE) {
        CWConnectorUtil.generateAndTraceMsg(
            CWConnectorLogAndTrace.LEVEL3,
            10551,
            CWConnectorLogAndTrace.XRD_INFO,
            0,
            0);
        evts.updateEventStatus(
            evtObj,
            CWConnectorEventStatusConstants.READY_FOR_ROLL);
        // No need to archive the event, as the status is reset
        // to READY_FOR_POLL. It is as if this event never
        //reached the
        // connector for processing.
        return CWConnectorConstant.SUCCEED;
    }
    if (stat
        == CWConnectorConstant.NO_SUBSCRIPTION_FOUND) {
        CWConnectorUtil.generateAndLogMsg(
            10552,
            CWConnectorLogAndTrace.XRD_ERROR,
            0,
            0);
        // Update the event status to UNSUBSCRIBED.
        evts.updateEventStatus(
            evtObj,
            CWConnectorEventStatusConstants.UNSUBSCRIBED);
        if (arcProcessed
            .equalsIgnoreCase(
                CWConnectorAttrType.TRUESTRING)) {
            // Archive the event in the application's archive

```

```

        //store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
}
if (stat == CWConnectorConstant.SUCCEED) {
    // Update the event status to SUCCESS.
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants.SUCCESS);
    if (arcProcessed
        .equalsIgnoreCase(
            CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's archive
        //store
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
} else // gotApplEvent returned FAIL
{
    CWConnectorUtil.generateAndLogMsg(
        10532,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        0);
    // Update the event status to ERROR_POSTING_EVENT.
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants
            .ERROR_POSTING_EVENT);
    // Archive the event if ArchiveProcessed is set
    // to true.
    if (arcProcessed
        .equalsIgnoreCase(
            CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's
        // archive store.
        evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store.
        evts.deleteEvent(evtObj.getEventID());
    }
    return CWConnectorConstant.FAIL;
}
} else // Event unsubscribed.
{

```

```

        CWConnectorUtil.generateAndLogMsg(
            10552,
            CWConnectorLogAndTrace.XRD_ERROR,
            0,
            0);
        // Update the event status to UNSUBSCRIBED.
        evts.updateEventStatus(
            evtObj,
            CWConnectorEventStatusConstants.UNSUBSCRIBED);
        // Archive the event if ArchiveProcessed is set
        // to true.
        if (arcProcessed
            .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
            // Archive the event in the application's
            // archive store.
            evts.archiveEvent(evtObj.getEventID());
            // Delete the event from the event store.
            evts.deleteEvent(evtObj.getEventID());
        }
        continue;
    }
} else {
    CWConnectorUtil.generateAndLogMsg(
        10552,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        0);
    // Update the event status to UNSUBSCRIBED.
    evts.updateEventStatus(
        evtObj,
        CWConnectorEventStatusConstants.UNSUBSCRIBED);
    // Archive the event if ArchiveProcessed is set
    // to true.
    if (arcProcessed
        .equalsIgnoreCase(CWConnectorAttrType.TRUESTRING)) {
        // Archive the event in the application's
        // archive store. evts.archiveEvent(evtObj.getEventID());
        // Delete the event from the event store.
        evts.deleteEvent(evtObj.getEventID());
    }
    continue;
}
} //For loop
}

} catch (StatusChangeFailedException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,

```

```

        2,
        "updateEventStatus",
        "StatusChangeFailedException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.APPRESPONSETIMEOUT;
} catch (InvalidStatusChangeException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "updateEventStatus",
        "InvalidStatusChangeException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.APPRESPONSETIMEOUT;
} catch (ArchiveFailedException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "archiveEvent",
        "ArchiveFailedException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.APPRESPONSETIMEOUT;
} catch (DeleteFailedException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,
        "deleteEvent",
        "DeleteFailedException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.APPRESPONSETIMEOUT;
} catch (AttributeNullValueException e) {
    CWConnectorUtil.generateAndLogMsg(
        10536,
        CWConnectorLogAndTrace.XRD_ERROR,
        0,
        2,

```



```
        "get method in event store",
        "AttributeNullValueException");
    CWConnectorUtil.logMsg(e.getMessage());
    e.printStackTrace();
    return CWConnectorConstant.FAIL;
}
} finally {
    evts.cleanupResources();
}
return CWConnectorConstant.SUCCEED;
}
```



Part 3

Setting up applications for our scenario



Overview of our scenario and applications

This chapter provides some background information about our scenario. It outlines the current, fictitious environment and the integration project that our scenario addresses.

19.1 Understanding our scenario environment

When you begin adapter development, it is crucial to understand the environment and applications in that environment. In our scenario, the company is a rental property management company that runs all of the operations for tenant maintenance through a call center. Management wants to scale back call center operations because they are proving to be costly.

Currently, the call center operators access each of the supporting applications separately. This environment 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.

In this environment, if the company's own tradespeople cannot carry out the work when maintenance requests are created, the call-center operators must contact the external contracting company and then update the details in the RedMaintenance application manually — a time-consuming and potentially inconsistent method for updating data. The call center operators must then contact the in-house maintenance teams and external contractors to update the progress of maintenance requests.

Figure 19-1 illustrates the current environment.

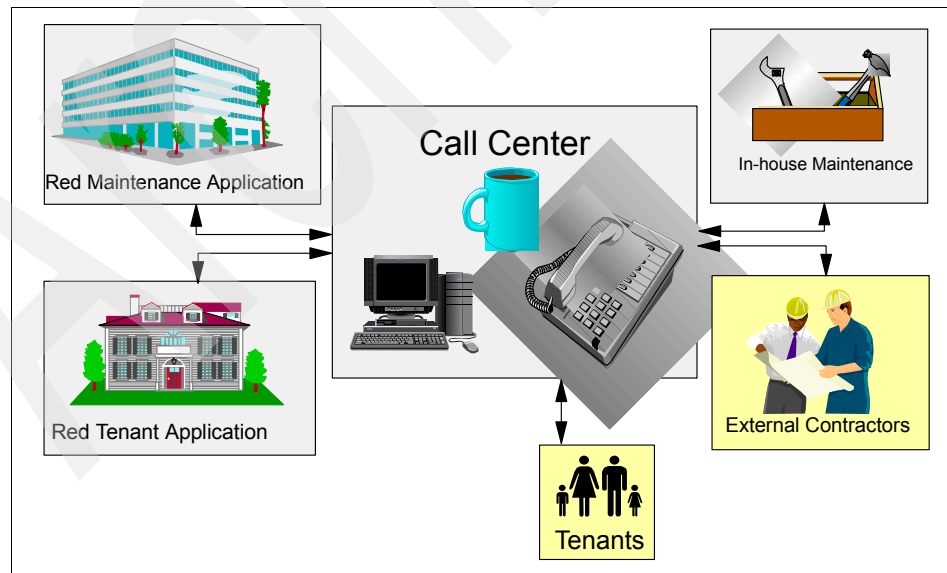


Figure 19-1 Current call center environment

19.2 The requirements

Management requires that the solution:

- ▶ Provide as much automation and synchronization as possible.
- ▶ Give the tenants the ability to query and request their own maintenance using the Internet.
- ▶ Is extensible enough so that, at some point in the future, tenants can use e-mail to request maintenance or to receive updates.

In addition, management also requests the following restrictions:

- ▶ The solution cannot alter existing applications.
- ▶ The solution must use an integration broker for any data modification, enhancement, or semantic mediation.

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

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.

In our scenario, the two applications will be integrated using a combination of adapters and the WebSphere integration brokers.

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

Because the back-end application has an exposed API, we use a custom adapter which was built for the back-end application. This custom 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 using the exposed API.

Figure 19-2 illustrates the complete integration scenario for the company in our scenario.

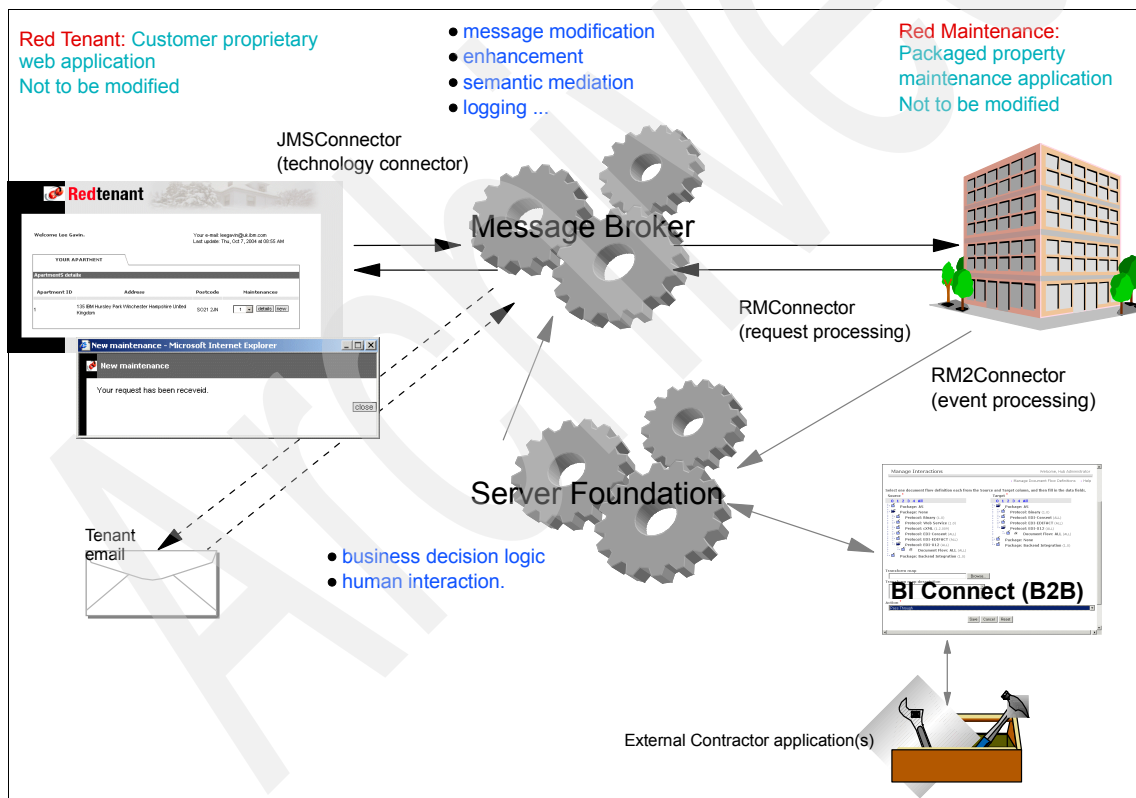


Figure 19-2 Complete integration solution

Installing and configuring the scenario infrastructure

This chapter discusses the installation and configuration of the front-end and back-end applications that we use in our scenario.

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

20.1 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

20.2 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 enters a valid tenant ID to retrieve the apartment details and maintenance list. Figure 20-1 shows the login page.

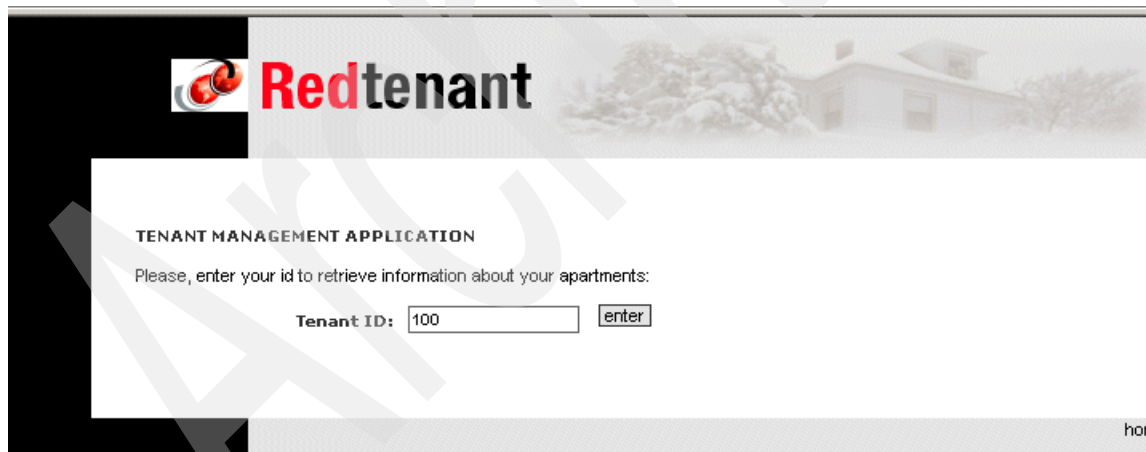


Figure 20-1 The RedTenant login page

If an error occurs, the application returns the login page with an error message as shown in Figure 20-2 on page 313.

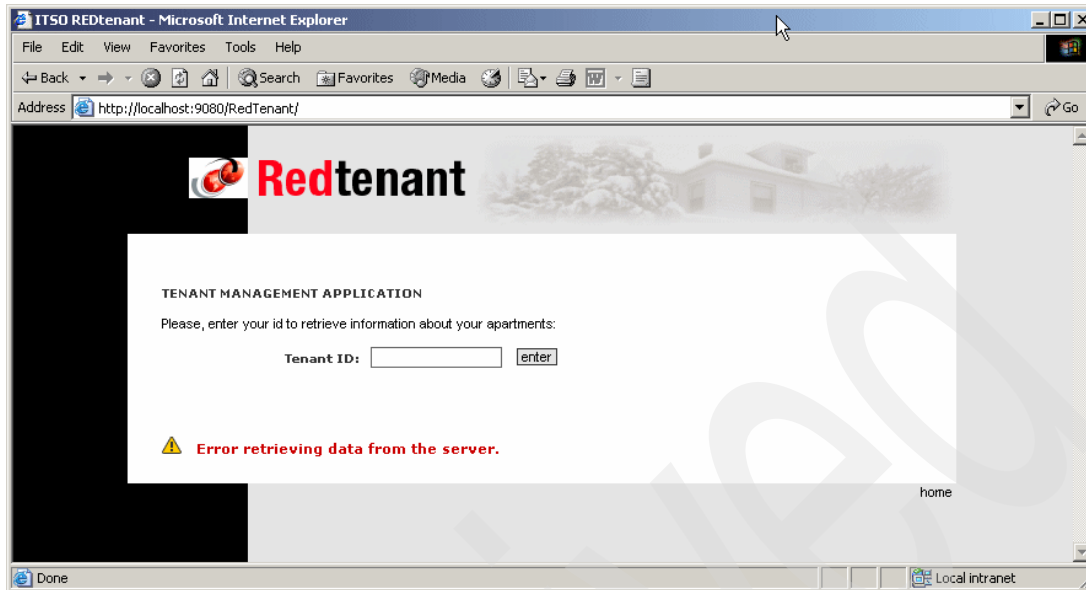


Figure 20-2 The RedTenant login page with an error message

2. If a valid tenant ID exists in the database, the application displays a page, as shown in the Figure 20-3, that lists the tenant's apartments. For each apartment, it shows the maintenance list.

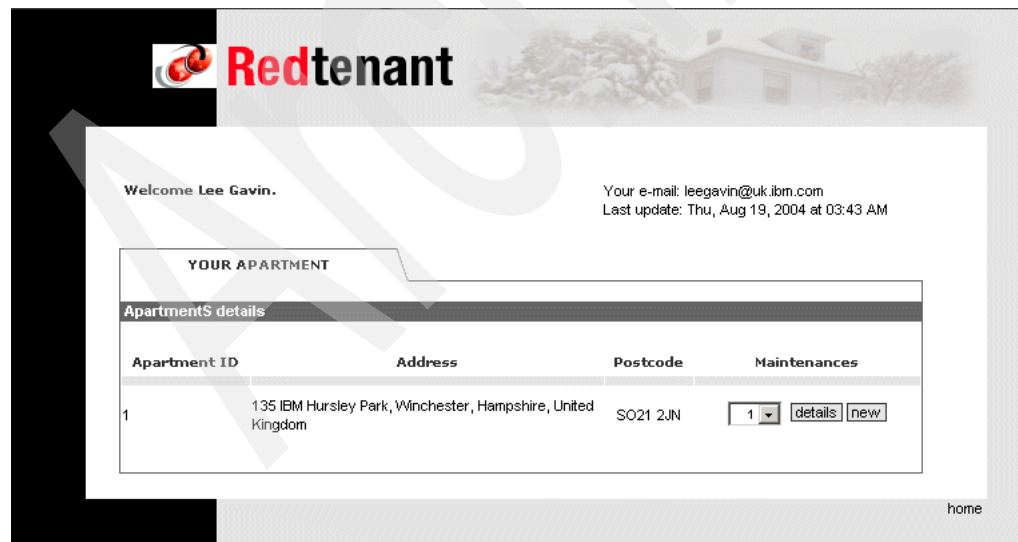


Figure 20-3 Maintenance information window

3. To see the maintenance details, the tenant selects the maintenance instances from the combination box. Selecting **details** displays the requested information. Figure 20-4 shows the information window for completed maintenance. Figure 20-5 shows the information window for an active maintenance request.

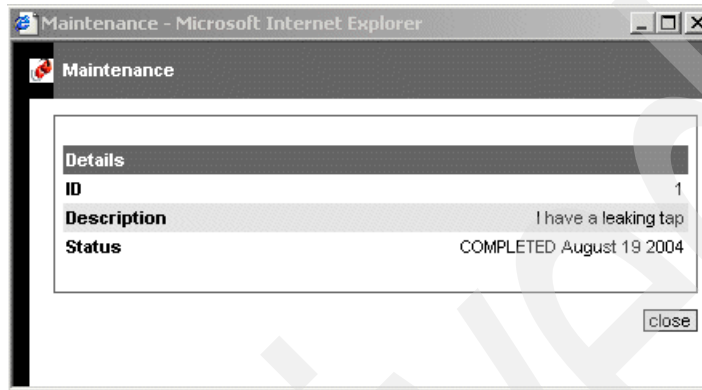


Figure 20-4 Completed maintenance information window

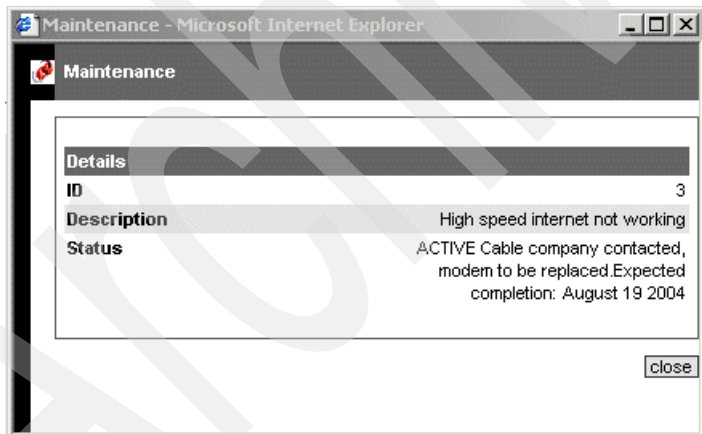


Figure 20-5 Active maintenance information window

4. Finally, the tenant can create a new maintenance request for a given apartment by selecting **new**, as shown in Figure 20-3 on page 313. A window appears with a text area for the tenant to enter a description of the request (Figure 20-6). To send the request, the tenant selects **create**.

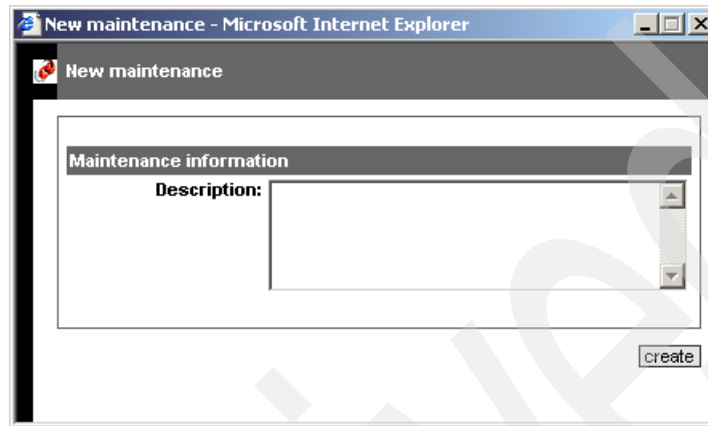


Figure 20-6 New maintenance request window

20.3 Installing the RedTenant application

The RedTenant application is a Web application (level 1.3) that is compliant with J2EE and that is based on the Apache Struts framework (V1.1). This application accesses the back-end system through a business integration system and sends XML messages to JMS message queues.

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 and then install WebSphere Application Server without the Embedded Messaging Server option.

We use WebSphere Studio Application Developer V5.0.1 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: You can find the RedTenantEAR.ear file 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.

20.4 Preparing the environment

To prepare for the RedTenant application installation, you need to:

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

20.4.1 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:

1. Select **Servers** → **Application Servers** in the navigation pane.
2. Click **New**.
3. Provide the application server name, for example RedTenantServer, and click **Next**.
4. The next window, as shown in Figure 20-7 on page 317, 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**.

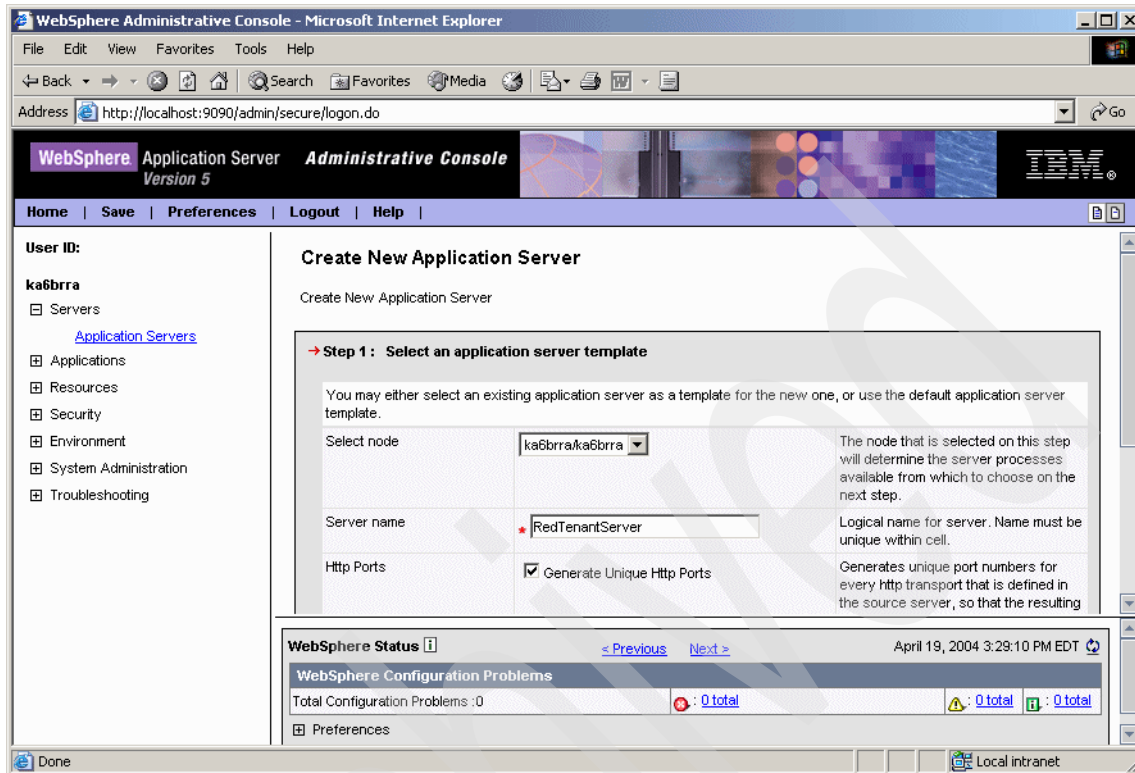


Figure 20-7 Creating the RedTenant application server

20.4.2 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:

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 20-8 on page 318.

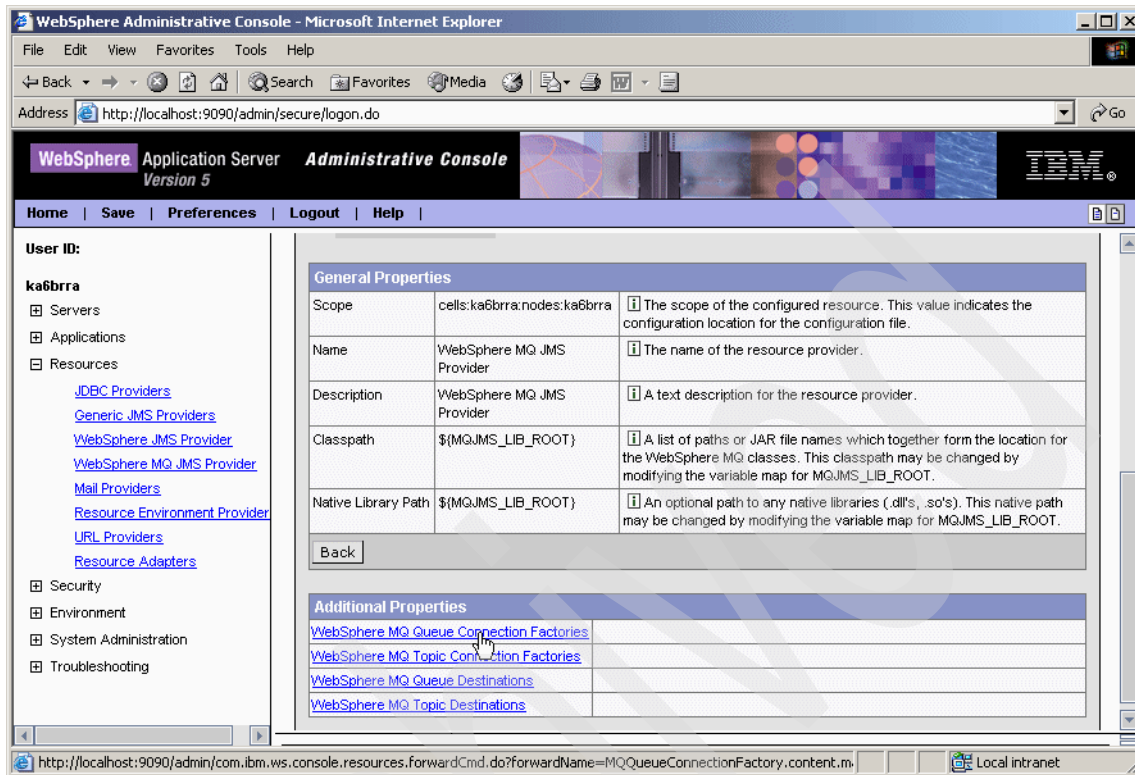


Figure 20-8 Configuring resources for the WebSphere MQ JMS Provider

- Click **New** and provide the information that is specified in Table 20-1.

Table 20-1 Queue connection factory properties for RedTenantCF

Property	Value	Description
Name	RedTenantCF	The queue connection factory name
JNDI Name	jms/RedTenantCF	The JNDI name for the resource
Queue Manager	REDTENANT	The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Figure 20-9 shows the New Connection Factory table.

Configuration		
General Properties		
Scope	* cells:ka6brra:nodes:ka6brra	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* RedTenantCF	The required display name for the resource.
JNDI Name	* jms/RedTenantCF	The JNDI name for the resource.
Description	RedTenant Connection Factory	An optional description for the resource.
Category		An optional category string which can be used to classify or group the resource.
Component-managed Authentication Alias	(none)	References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias	(none)	References authentication data for container-managed signon to the resource.
Mapping-Configuration Alias	DefaultPrincipalMapping	Select a suitable JAAS login configuration from the security-JAAS configuration panel to map the user identity and credentials to a resource principal and credentials that is required to open a connection to the back-end server.
Queue Manager	REDTENANT	The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Figure 20-9 Creating a connection factory

4. Click **OK**.

The WebSphere MQ Queue Connection Factories table shows the information that is shown in Figure 20-10.

[WebSphere MQ JMS Provider >](#)

WebSphere MQ Queue Connection Factories

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere MQ Queue Connection Factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider.

Total: 1

Filter

Preferences

New Delete

<input type="checkbox"/>	Name	JNDI Name	Description	Category
<input type="checkbox"/>	RedTenantCF	jms/RedTenantCF	RedTenant Connection Factory	

Figure 20-10 The WebSphere MQ Queue Connection Factories.

To create the first queue destination:

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 Figure 20-8 on page 318).
3. Click **New** and provide the information that is specified in Table 20-2.

Table 20-2 Queue destination properties for RTReq1Q

Property	Value	Description
Name	RTReq1Q	The queue destination name
JNDI Name	jms/RTReq1Q	The JNDI name for the resource.
Base Queue Name	RTREQ1Q	The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	REDTENANT	The name of the WebSphere MQ queue manager to which messages are sent.

Figure 20-11 shows the New Queue Destination table.

Configuration		
General Properties		
Scope	* cells:ka6brra.nodes:ka6brra	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* RTReq1Q	The required display name for the resource.
JNDI Name	* jms/RTReq1Q	The JNDI name for the resource.
Description	RedTenant - Tenant information request	An optional description for the resource.
Category		An optional category string which can be used to classify or group the resource.
Persistence	APPLICATION DEFINED	Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Priority	APPLICATION DEFINED	Whether the message priority for this destination is defined by the application or the Specified priority property.
Specified Priority	0	If the Priority property is set to Specified, type here the message priority for this queue, in the range 0 through 9.
Expiry	APPLICATION DEFINED	Whether the expiry timeout for this queue is defined by the application or the Specified expiry property, or messages on the queue never expire (have an unlimited expiry timeout).
Specified Expiry	0 milliseconds	If the Expiry timeout property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire.
Base Queue Name	* RTREQ1Q	The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	REDTENANT	The name of the WebSphere MQ queue manager to which messages are sent.

Figure 20-11 Creating a queue destination

- Click **OK**.
- Complete the configuration of the other three queue destinations by providing the information that is specified in Table 20-3, Table 20-4 on page 322, and Table 20-5 on page 322.

Table 20-3 Queue destination properties for RTRes1Q

Property	Value	Description
Name	RTRes1Q	The queue destination name
JNDI Name	jms/RTRes1Q	The JNDI name for the resource.
Base Queue Name	RTRES1Q	The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	REDTENANT	The name of the WebSphere MQ queue manager to which messages are sent.

Table 20-4 Queue destination properties for RTReq2Q

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

Table 20-5 Queue destination properties for RTRes2Q

Property	Value	Description
Name	RTRes2Q	The queue destination name
JNDI Name	jms/RTRes2Q	The JNDI name for the resource.
Base Queue Name	RTRES2Q	The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	REDTENANT	The name of the WebSphere MQ queue manager to which messages are sent.

- After you have completed the queue destination entries, the WebSphere MQ Queue Destinations table shows the information that is shown in Figure 20-12.

[WebSphere MQ JMS Provider](#) >

WebSphere MQ Queue Destinations

Queue destinations provided for point-to-point messaging by the WebSphere MQ JMS provider. Use WebSphere MQ Queue Destination administrative objects to manage queue destinations for the WebSphere MQ JMS provider. [?](#)

Total: 4

☐ Filter

☐ Preferences

<input type="checkbox"/>	Name ▾	JNDI Name ▾	Description ▾	Category ▾
<input type="checkbox"/>	RTReq1Q	jms/RTReq1Q	RedTenant - Tenant information request	
<input type="checkbox"/>	RTReq2Q	jms/RTReq2Q	RedTenant - Maintenance creation request	
<input type="checkbox"/>	RTRes1Q	jms/RTRes1Q	RedTenant - Tenant information response	
<input type="checkbox"/>	RTRes2Q	jms/RTRes2Q	RedTenant - Maintenance creation response	

Figure 20-12 The WebSphere MQ Queue Destinations

20.4.3 Deploying the application

We have created the JMS resources that the RedTenant application requires. To deploy the application using the Administrative Console:

- Select **Applications** → **Install New Application** in the navigation pane (see Figure 20-13 on page 324).

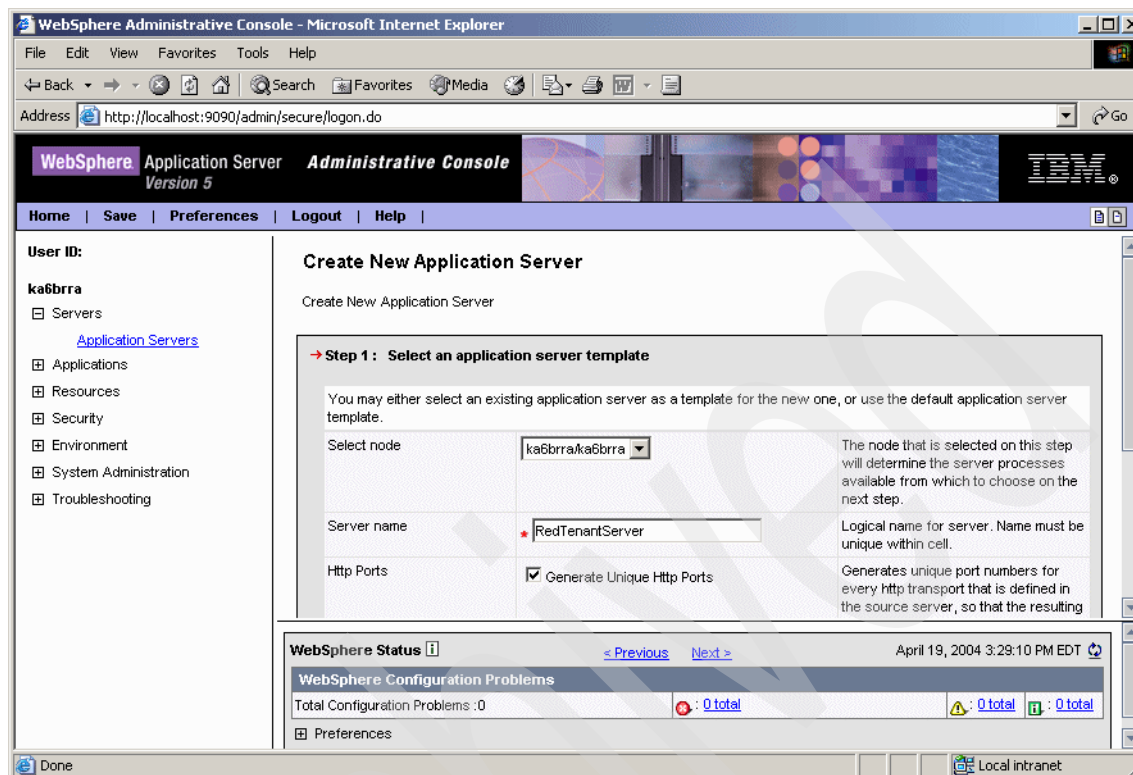


Figure 20-13 The first Preparing for the application installation page

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 20-14 on page 325 and click **Apply**. Click **Next**.

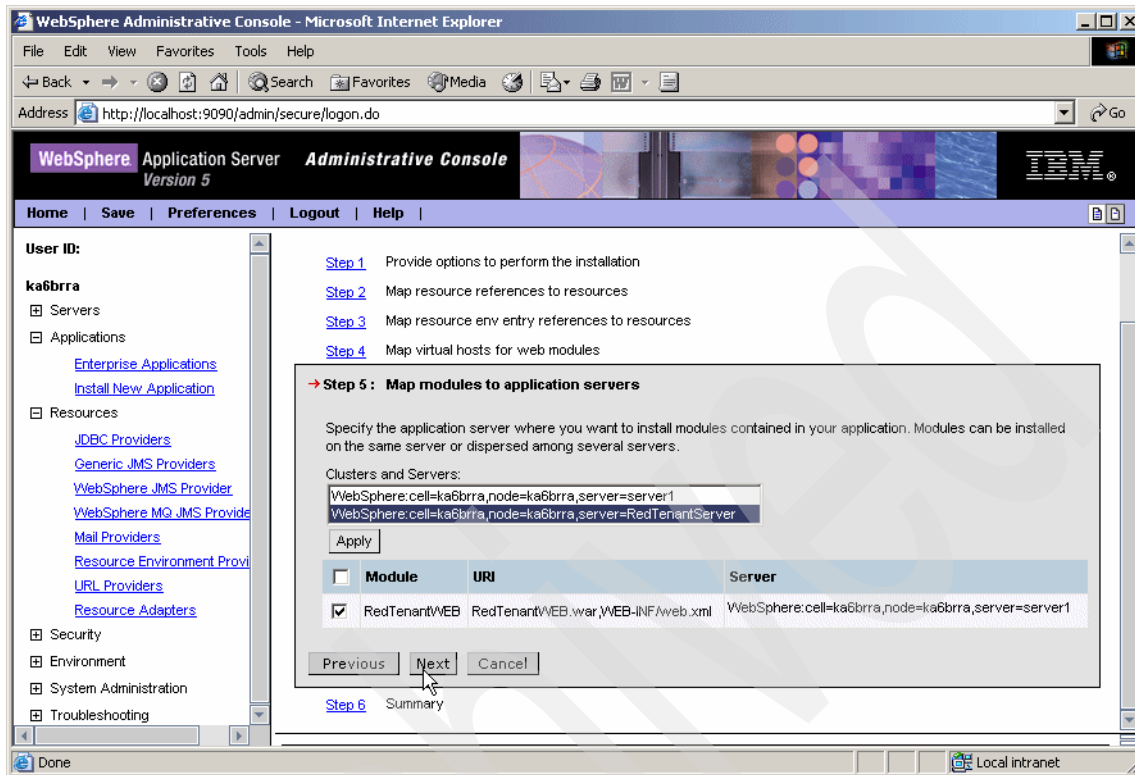


Figure 20-14 Mapping the RedTenantWEB module to the RedTenantServer

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.

20.4.4 A quick test

Start with the login screen that is shown in Figure 20-15.

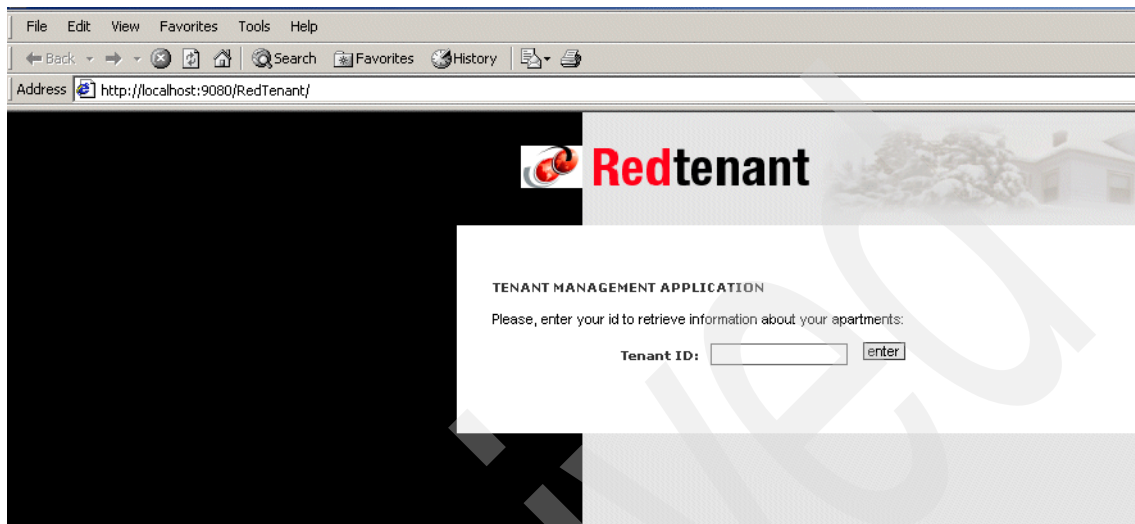


Figure 20-15 Login screen

If you use the setup materials for the database for the back-end application, discussed in 20.5, “Create the back-end application database” on page 328, you 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.

To test the application:

1. Enter the tenant ID into the front-end application, as shown in Figure 20-16.

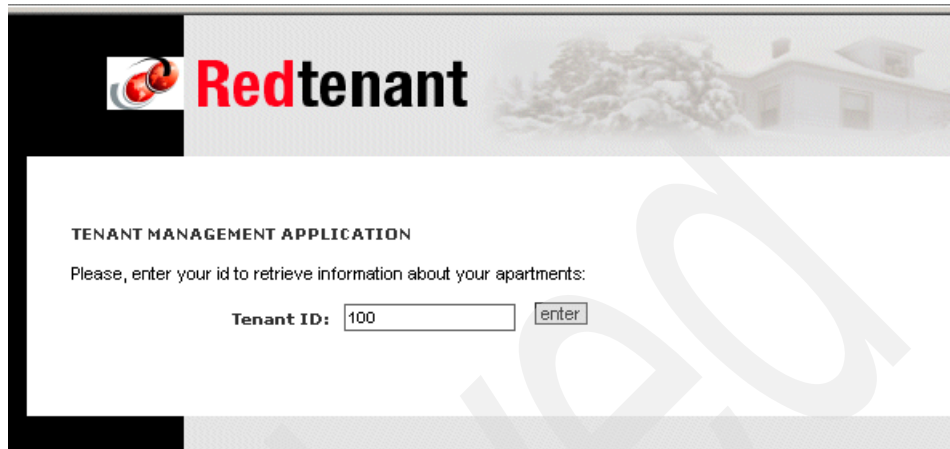
The screenshot shows the Redtenant web application. At the top, there is a header with the Redtenant logo (a red circle with a white 'R' and a red dot) and the text 'Redtenant' in a bold, sans-serif font. Below the header, the main content area has a title 'TENANT MANAGEMENT APPLICATION' and a prompt 'Please, enter your id to retrieve information about your apartments:'. There is a text input field labeled 'Tenant ID:' containing the value '100', and a small 'enter' button to its right. The background of the application has a faint image of a house and trees.

Figure 20-16 Enter valid tenant ID

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.

2. Use `rhfutl`, which is provided in the Additional Materials for this book. (See Appendix B, “Additional material” on page 887 for information about how to obtain the materials for this book.)
3. 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.
4. Select **read message**. Do not browse because we do not want to retain this message.
5. Look at the message data in XML format to see that a message has been created and is ready for sending (Figure 20-17 on page 328).

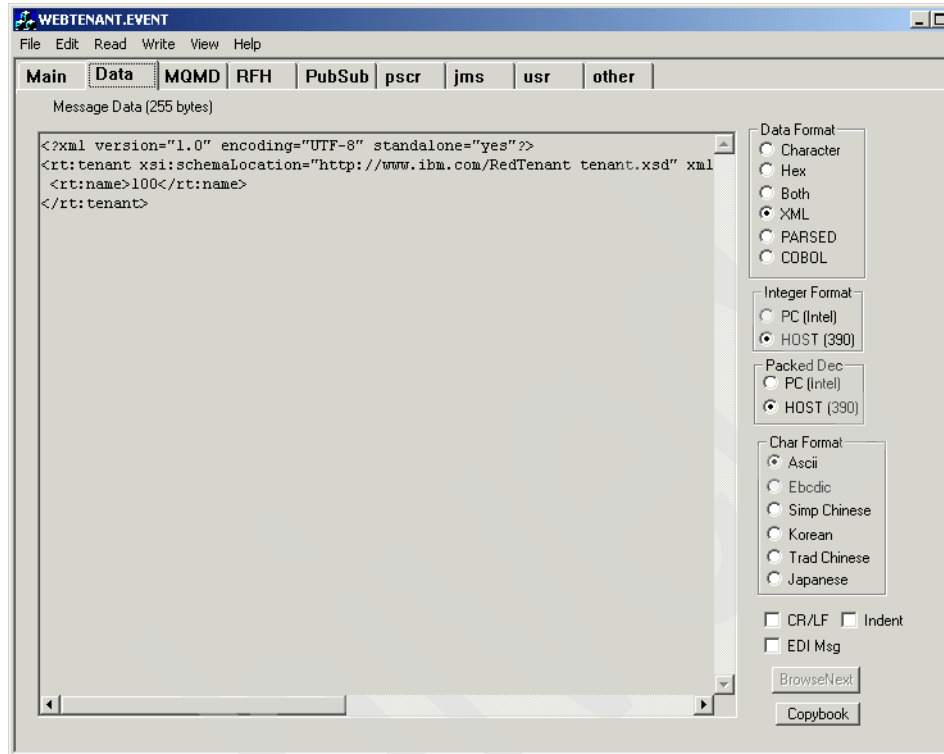


Figure 20-17 Message data on queue from the front-end

20.5 Create the back-end application database

This section deploys the back-end database and application.

First, 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:

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

```
CreateSG246345DB DB2 db2admin its04you
```

In this command, DB2 is the DB2 instance name, db2admin is the DB2 administrator user ID, and its04you is the DB2 administrator password.

- After 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 import the file as a script into the DB2 Command Center and run it from there (Figure 20-18).

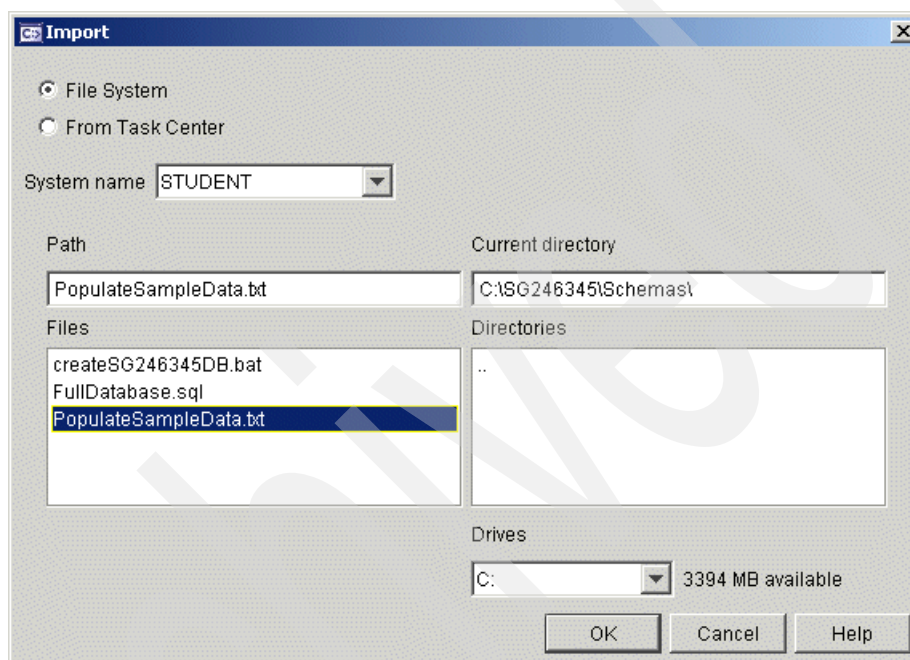


Figure 20-18 Import script

- Using your favorite DB2 tool, check that the database exists with the correct tables and the sample data. We use the DB2 Command Center as shown in Figure 20-19.

Interactive | Script | Results | Access Plan

Perform required changes and then click the commit update button.

TENANTID	NAME	APARTME...	EMAILID	STATUS	CREATION...	CHANGE...
100	Lee Gavin	1	leegavin@uk.ibm.com	A	Aug 25, 200...	Aug 25, 20
110	ITSO Resident	2	resident@us.ibm.com	A	Aug 25, 200...	Aug 25, 20
120	Redbook Reader	3	redbookreader@us.ibm.com	A	Aug 25, 200...	Aug 25, 20

Figure 20-19 Some sample data

20.6 Create the back-end application

Our back-end application is a Java application that uses RMI. To create the back-end application:

1. Ensure that the application configuration properties are correct.
2. In the RedMaintenance folder of the SG246345 directory, open the config file with a text editor. Example 20-1 shows the contents of the config file.

Example 20-1 The config file 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. Start **rmiregistry**.

Note: The **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 **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.

6. Open a command window and type:
`rmiregistry`
Leave this window open in the background.
7. Create a shortcut on the desktop for starting the application engine, as shown in Figure 20-20 on page 331. Ensure that the start directory is RedMaintenance.

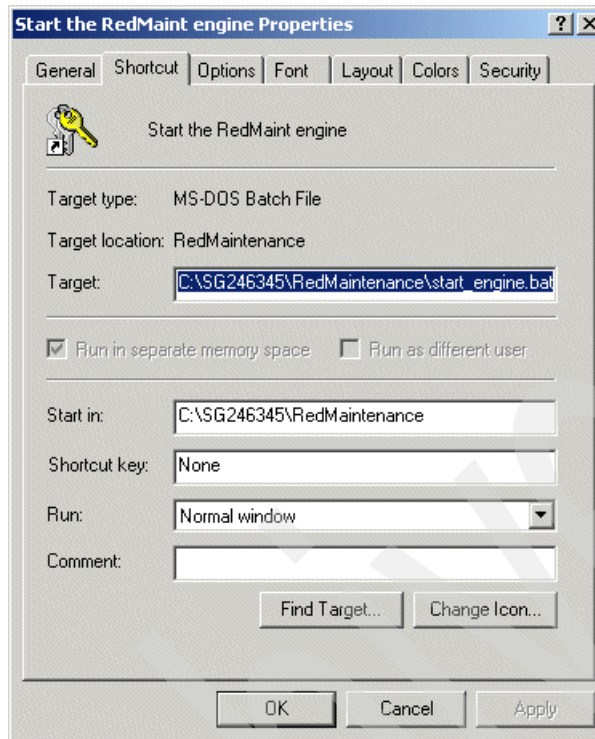


Figure 20-20 RedMaintenance shortcut

8. Using this shortcut, start the application.
9. You will see in the command window the startup parameters that we use for our application. Example 20-2 shows these parameters. Leave this window open.

Example 20-2 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:

```

10. You will also see a GUI window, shown in Figure 20-21, that is used for tracing application activity and for problem determination.



Figure 20-21 GUI log

The basic application components are now in place. We can now integrate these components and complete the initial testing of the adapter environments.



Part 4

Configuring and testing adapters and business objects



Creating the business objects and connector

This chapter explains how to create the business objects that are required for the front-end application and JMS connector in our scenario. These business objects transport data between the front-end application and the Message Broker. For our scenario, we installed the JMS Adapter and the XML DataHandler and the XML ODA that are used in this chapter.

Note: We have included repos files of the components in this step for you to use as an alternative to creating all of the business objects manually. You can find these files in the Additional Materials in the ReposJarFiles folder. See Appendix B, “Additional material” on page 887 for information about how to obtain the materials for this book.

21.1 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 using the schema definitions for these messages.

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

21.1.1 Creating business objects with the ODA

To create business objects:

1. Start the ODA and 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 shown in Figure 21-1.

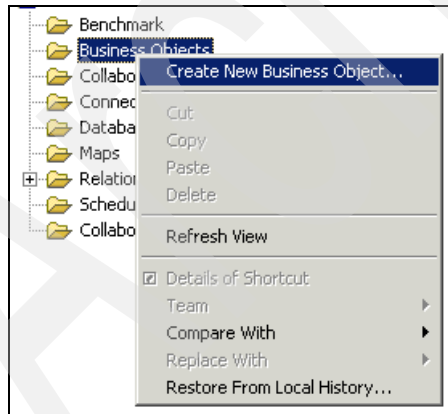


Figure 21-1 New business object

6. Close the new business object window and select **File** → **New Using ODA**.

If your ODA has started successfully, it appears in Located agents (Figure 21-2). If you do not see the XMLODA, check the command window for the ODA to find the error and then restart the ODA.

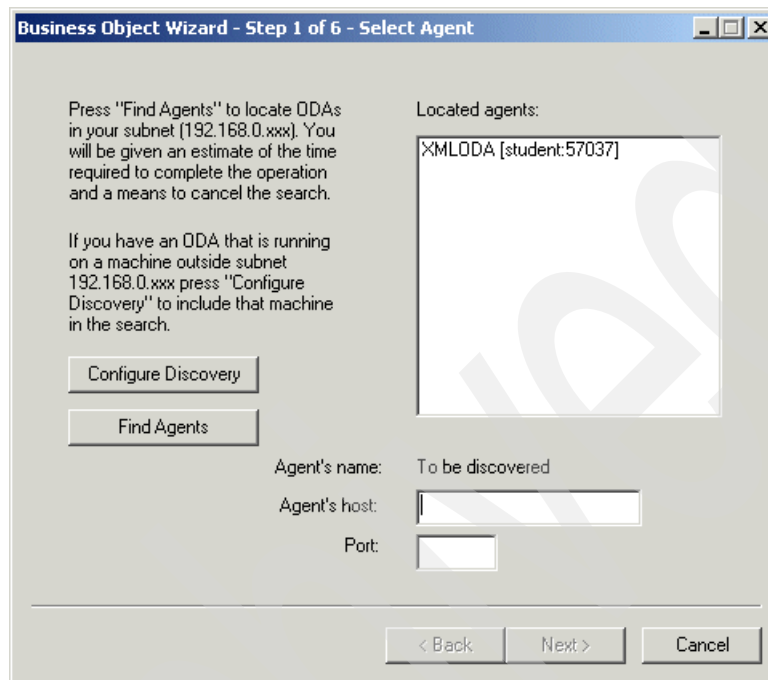


Figure 21-2 ODA window

7. Highlight the ODA and click **Next**, as shown in Figure 21-3.

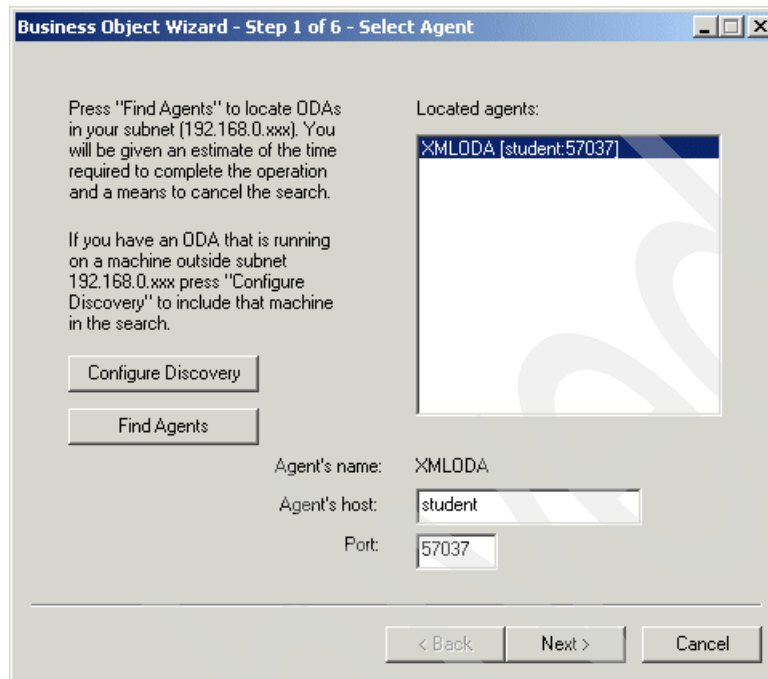


Figure 21-3 Found ODA

8. Enter the ODA properties as shown in Figure 21-4. The schema definition for the RedTenant messages is in the RedTenant folder in the Additional Materials.
9. Enter a BOPrefix of Web.
10. Click Next.

Business Object Wizard - Step 2 of 6 - Configure Agent

Profiles:

Current profile:

Save New Remove

	Property	Value	Type	Description
1	FileName	C:\Additional Materials\RedTenant\tenant.xsd	String	name of DTD or Schema file with full path
2	Root		String	Name of Root Element
3	TopLevel		String	Value to be used for Top Level Object Name
4	BOSelection	false	String	Selective Business Object Definition generation
5	BOPrefix	Web	String	Prefix that should be applied to each business object
6	DoctypeorSchemaLocation	true	String	Generate a DOCTYPE or schemaLocation
7	TypeSubstitution	false	String	XML Instance documents use xsi:types
8	TraceFileName	XMLODAtrace.txt	String	Name of the trace file
9	TraceLevel	5	Integer	Trace level for the agent
10	MessageFile	XMLODAAgent.txt	String	Name of the error and message file, relative to the trace file

< Back Next > Cancel

Figure 21-4 Properties

11. The ODA finds a top-level object for this XML schema. Highlight this top-level object, as shown in Figure 21-5, and click **Next**.

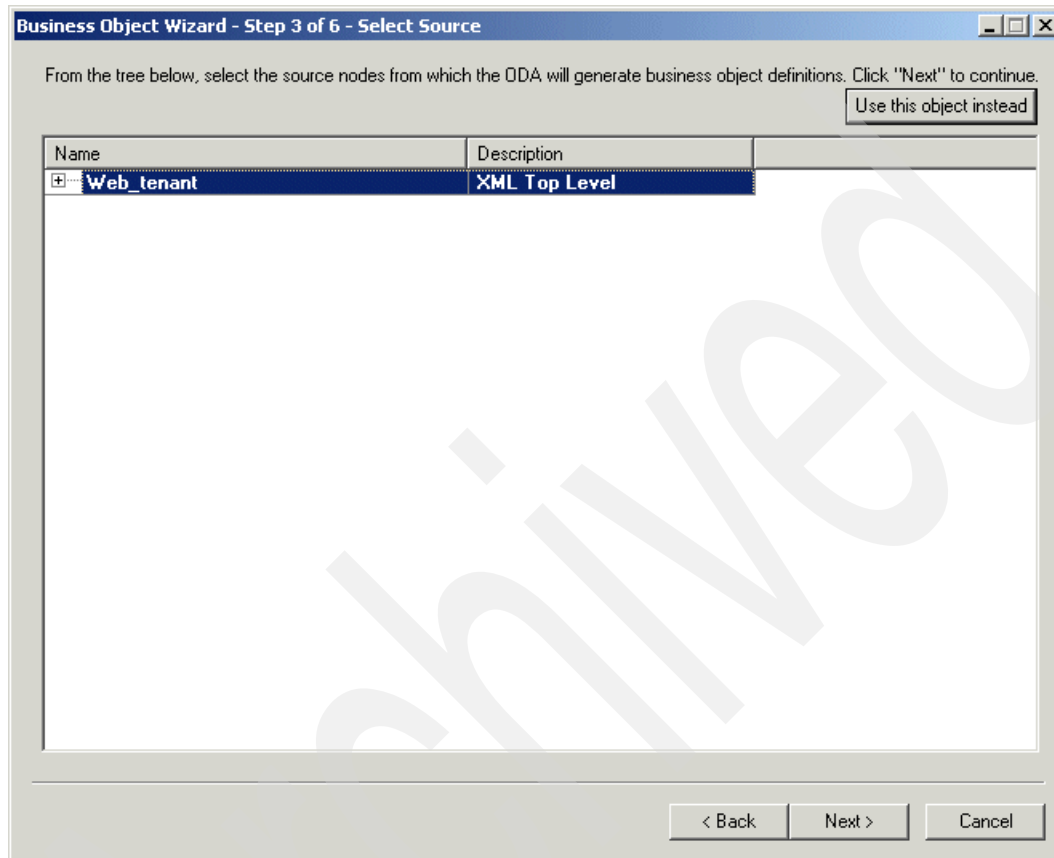


Figure 21-5 Select Source

12. Select the top-level object again to confirm, as shown in Figure 21-6, and click **Next**.

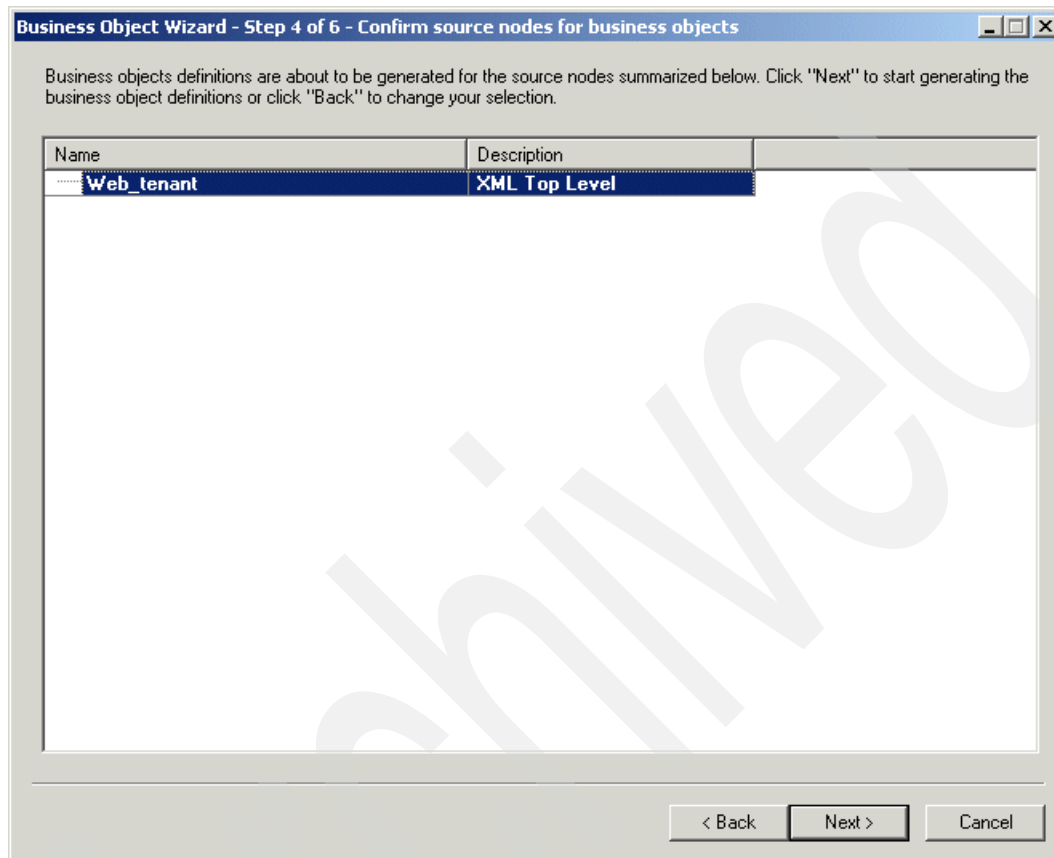


Figure 21-6 Confirm source

13. Click **OK** (Figure 21-7) to confirm that you will take all of the usual, supported verb (the default setting).



Figure 21-7 Select verbs

14. Save the new business objects to your ICL, as shown in Figure 21-8. Select to open the new business objects and to shut down the ODA after completion.
15. Click **Finish**.

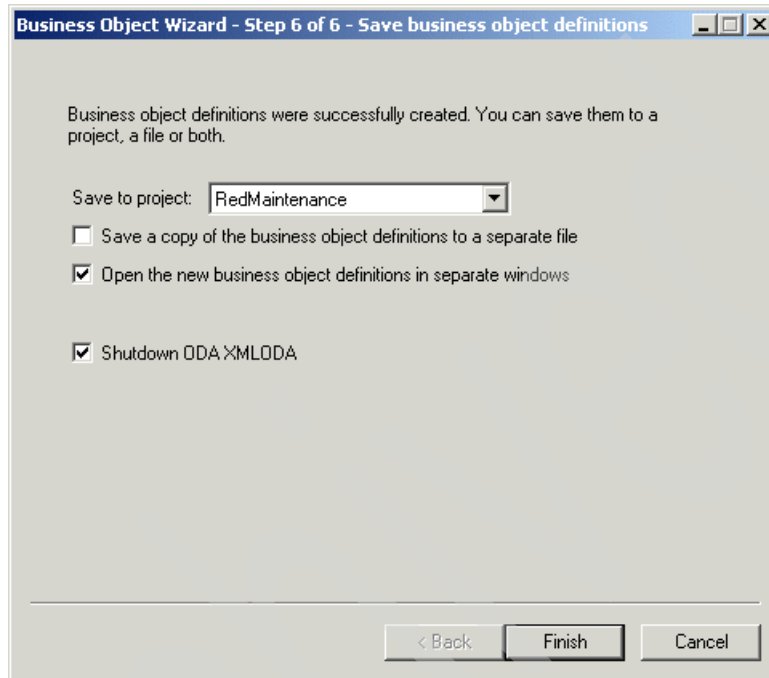


Figure 21-8 Finalize ODA

21.1.2 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. You can now do this for the 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 that the XML ODA generates has the following format:

BOPrefix_TopLevel_XMLelement

It also adds a suffix to the business object name of the child objects, as shown in Figure 21-9.

```
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 21-9 Business object log

These names result in some very long ESQL statements when you use the Message Broker for your flows. So, you will save each of them with a shortened name for ease of use later. Because most of these objects are also child objects, you will need to modify the parents to ensure that they are pointing to the correct children.

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

Also, as shown in Figure 21-9, the ODA has not set the key attribute on the main business object correctly. You will have to resolve this issue.

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

To resolve the key attribute for the main business object:

1. Open the Web_tenant business object.
2. As shown in Figure 21-10 on page 345, all of the child business objects are incorporated into the top-level business object. You must work from the bottom up to modify and save the business objects.
3. Eliminate the error from stopping the validation of the top-level business object. This is the key field, as shown in the error message in Figure 21-9.
4. Select the XMLDeclaration and ROOT business objects as the key attributes, and save the Web_tenant business object.

Pos	Name	Type	Key	Foreign Key	Required	Cardinality
1	XMLDeclaration	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2	ROOT	Web_tenant_P0690378762_tenant	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1
2.1	schemaLocation	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.2	id	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.3	name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2.4	email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.5	apartments	Web_tenant_apartments_P0690378762	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1
2.5.1	apartment	Web_tenant_apartment_P0690378762	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n
2.5.1.1	id	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
2.5.1.2	address	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.5.1.3	number	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.5.1.4	postcode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2.5.1.5	maintenances	Web_tenant_maintenances_P0690378762	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1
2.5.1.6	maintenanc	Web_tenant_maintenance_P0690378762	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n
2.5.1.7	ObjectEventId	String				
2.5.1.8	ObjectEventId	String				
2.5.2	ObjectEventId	String				
2.6	ObjectEventId	String				
3	ObjectEventId	String				

Figure 21-10 Generated business object

- Open the Web_tenant_maintenance_P0690378762 business object. This business object is correct as defined.
- Select **File** → **Save As**.
- Remove the number suffix so that the business object is named Web_tenant_maintenance, as shown in Figure 21-11.

New Business Object

Business Object Name:

Application Specific Information

OK Cancel

Figure 21-11 New business object name

8. Click **OK**.
9. Save this new business object again, but this time save a copy to file. The directory to which you need to save the business object is the repository directory. The repository directory is where the connector gets its run-time business object definitions.

The directory for our repository is C:\IBM\WebSphereAdapters\repository. The business object is saved with its name and an XSD schema file type, as shown in Figure 21-12.

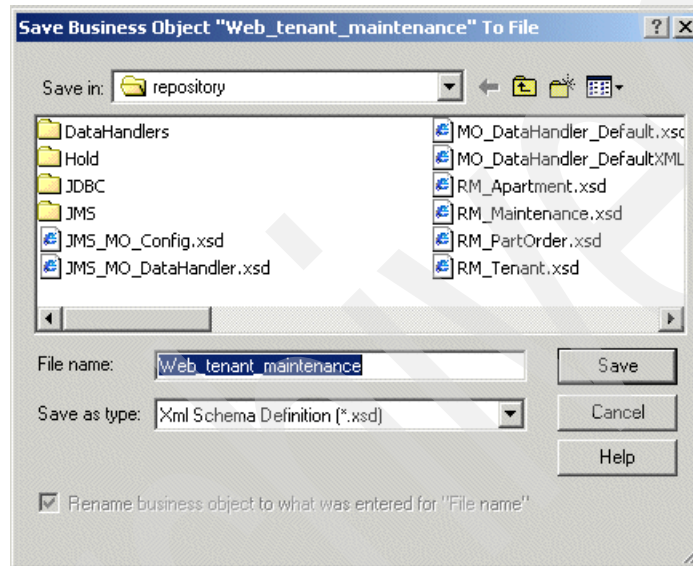


Figure 21-12 Save to file

10. Open the Web_tenant_maintenances_P0690378762 business object.
11. This business object 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 21-13 on page 347.

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 **ICL** and select **Refresh View**. Return to the Business Object Designer and proceed.

	Pos	Name	Type	Ke
1	1	maintenance	Web_tenant_maintenance_P0690378762	
2	2	ObjectEventId	Float	
3	3		Double	
			String	
			Date	
			LongText	
			Web_tenant	
			Web_tenant_apartment_P0690378762	
			Web_tenant_apartments_P0690378762	
			Web_tenant_maintenance	
			Web_tenant_maintenance_P0690378762	
			Web_tenant_maintenances_P0690378762	
			Web_tenant_P0690378762_tenant	

Figure 21-13 Change business object type in maintenances

12. Save this business object to your ICL with a name of Web_tenant_maintenances.
13. Also save this business object to a repository file.
14. Open the Web_tenant_apartment_P0690378762 object and change the child object for maintenances using the new Web_tenant_maintenances, as shown in Figure 21-14.

	Pos	Name	Type	h
1	1	id	String	
2	2	address	String	
3	3	number	String	
4	4	postcode	String	
5	5	maintenances	Web_tenant_maintenances_P0690378762	
6	6	ObjectEventId	Double	
7	7		String	
			Date	
			LongText	
			Web_tenant	
			Web_tenant_apartment_P0690378762	
			Web_tenant_apartments_P0690378762	
			Web_tenant_maintenance	
			Web_tenant_maintenance_P0690378762	
			Web_tenant_maintenances	
			Web_tenant_maintenances_P0690378762	
			Web_tenant_P0690378762_tenant	

Figure 21-14 Change business object type in apartment

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

16. Open Web_tenant_apartments_P0690378762.
17. Change the child business object for apartment to Web_tenant_apartment, as shown in Figure 21-15.

	Pos	Name	Type	Key
1	1	田 apartment	Web_tenant_apartment_P0690378762	<input checked="" type="checkbox"/>
2	2	ObjectEventId	String	
3	3		Date	
			LongText	
			Web_tenant	
			Web_tenant_apartment	
			Web_tenant_apartment_P0690378762	
			Web_tenant_apartments_P0690378762	
			Web_tenant_maintenance	
			Web_tenant_maintenance_P0690378762	
			Web_tenant_maintenances	
			Web_tenant_maintenances_P0690378762	
			Web_tenant_P0690378762_tenant	

Figure 21-15 Change business object in apartments

18. Save this business object to the ICL and to the repository as Web_tenant_apartments.
19. Open the Web_tenant_P0690378762_tenant business object.
20. Change the child business object for apartments to Web_tenant_apartments, as shown in Figure 21-16 on page 349.

	Pos	Name	Type	Key
1	1	schemaLocation	String	
2	2	id	String	
3	3	name	String	
4	4	email	String	
5	5	apartments	Web_tenant_apartments_P0690378762	
6	6	ObjectEventId	Date	
7	7		LongText	
			Web_tenant	
			Web_tenant_apartment	
			Web_tenant_apartment_P0690378762	
			Web_tenant_apartments	
			Web_tenant_apartments_P0690378762	
			Web_tenant_maintenance	
			Web_tenant_maintenance_P0690378762	
			Web_tenant_maintenances	
			Web_tenant_maintenances_P0690378762	
			Web_tenant_P0690378762_tenant	

Figure 21-16 Change business object in Web_tenant_tenant

21. The attribute schemaLocation is of the type xs:schemalocation. Set a default value and make it a required value to ensure that the default is picked up, if required.

22. Insert the correct values as shown in Figure 21-17.

Important: Note that there is a space between the words RedTenant and tenant.xsd in the schema location.

General		Attributes					
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality
1	1	schemaLocation	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
		Default Value		Application Specific Information			
		"http://www.ibm.com/RedTenant tenant.xsd"		attr_name=schemaLocation;type=xsschemalocation			

Figure 21-17 Modify schema location

23. Save this business object to the ICL and the repository as Web_tenant_tenant.
24. Modify the top level object, Web_tenant by first opening the Web_tenant business object.
25. Change the child object that forms the ROOT element of the business object to reflect the new Web_tenant_tenant business object, as shown in Figure 21-18.

	Pos	Name	Type	Key	Foreign Key	Required
1	1	XMLDeclaration	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	2	ROOT	Web_tenant_tenant	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	3	ObjectEventId	String			
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 21-18 Change business object

26. Click the General tab, which shows the schema namespace information for the business object and the supported verbs.

GeneralAttributes

Business Object Level Application-specific information:

target_ns=http://www.ibm.com/RedTenant;elem_fd=unqualified;attr_fd=unqualified

Supported Verbs:

	Name	Application-specific information
1	Create	
2	Delete	
3	Retrieve	
4	Update	
5		

Figure 21-19 Supported verbs

27. 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:
 - 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.

28. Check that these two conditions are met, as shown in Figure 21-20.

General Attributes										
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application
1	1	XMLDeclaration	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	xml version="1.0"	type=pi
2	2	ROOT	Web_tenant_tenant	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			elem_name=tenant;elem
3	3	ObjectEventId	String							
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 21-20 Top-level business object

29. The business object attribute named XMLDeclaration represents the XML declaration in the prolog. This attribute has a type of `pi` (which indicates XML processing instructions). You set a default value that might be used.

Give the attribute the value:

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

The XML data handler generates the following XML from this value:

```
<?xml version="1.0"?>
```

30. Expand all of the child objects within the business object, as shown in Figure 21-21. Check that all of your changes have been made and saved correctly.

General		Attributes							
	Pos	Name	Type	Key	Foreign	Required	Cardinality	Maximum Length	Default Value
1	1	XMLDeclaration	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	xml version="1.0"
2	2	ROOT	Web_tenant_tenant	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
2.1	2.1	schemaLocation	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	"http://www.ibm.com/RedTenant/tenant.xsd"
2.2	2.2	id	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.3	2.3	name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	
2.4	2.4	email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.5	2.5	apartments	Web_tenant_apartments	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
2.5.1	2.5.1	apartment	Web_tenant_apartment	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N		
2.5.1.1	2.5.1.1	id	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	
2.5.1.2	2.5.1.2	address	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.5.1.3	2.5.1.3	number	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.5.1.4	2.5.1.4	postcode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.5.1.5	2.5.1.5	maintenances	Web_tenant_maintenances	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		
2.5.1.5.1	2.5.1.5.1	maintenance	Web_tenant_maintenance	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N		
2.5.1.5.1.1	2.5.1.5.1.1	id	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	
2.5.1.5.1.2	2.5.1.5.1.2	status	String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	
2.5.1.5.1.3	2.5.1.5.1.3	description	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
2.5.1.5.1.4	2.5.1.5.1.4	ObjectEventId	String						
2.5.1.5.1.5	2.5.1.5.1.5	ObjectEventId	String						
2.5.1.5.1.6	2.5.1.5.1.6	ObjectEventId	String						
2.5.1.5.1.7	2.5.1.5.1.7	ObjectEventId	String						
2.6	2.6	ObjectEventId	String						
3	3	ObjectEventId	String						
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	

Figure 21-21 Expanded business object

31. 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.

Business Object Wizard - Step 2 of 6 - Configure Agent

Profiles

Current profile: []

[Save] [New] [Remove]

	Property	Value	Type	Description
1	FileName	C:\Additional Materials\RedMaintenance\maintenance.xsd	String	name of DTD or Schema file
2	Root		String	Name of Root Element
3	TopLevel		String	Value to be used for Top Level
4	BOSelection	false	String	Selective Business Object Discovery
5	BOPrefix	Web	String	Prefix that should be applied
6	DoctypeorSchemaLocation	true	String	Generate a DOCTYPE or schema
7	TypeSubstitution	false	String	XML Instance documents use
8	TraceFileName	XMLODAtrace.txt	String	Name of the trace file
9	TraceLevel	5	Integer	Trace level for the agent
10	MessageFile	XMLODAAgent.txt	String	Name of the error and message

[< Back] [Next >] [Cancel]

Figure 21-22 ODA settings for maintenance

Repeat steps 1 on page 344 through 31 on page 352 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 shut down each time it has finished its discovery.

Continue with the next step (Figure 21-23 on page 354).

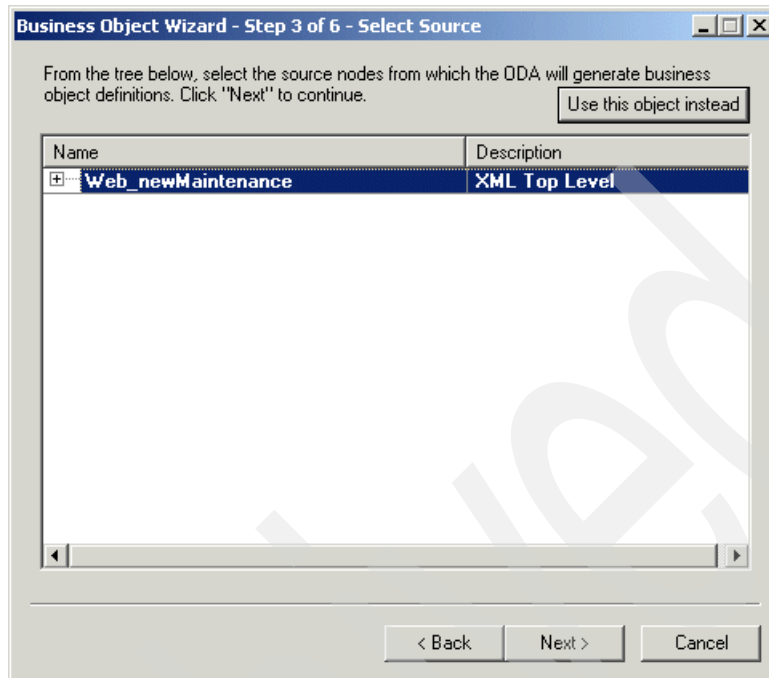


Figure 21-23 Select the source

32. After creating the business objects, add a key to the Web_newMaintenance business object and save it.
33. Open the Web_newMaintenance__N2116280397_newMaintenance business object.
34. Save this object to the project and to file as Web_newMaintenance_newMaintenance (Figure 21-24).

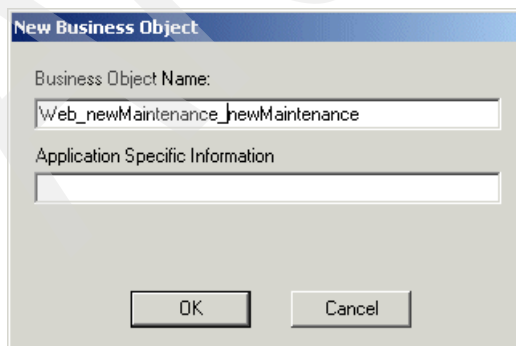


Figure 21-24 Save object

35. Open the Web_newMaintenance business object.
36. Change the ROOT element to use the Web_NewMaintenance_newMaintenance business object.
37. 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.

21.2 JMSConnector

Because the front-end application is messaging aware and uses XML data, a JMS connector is used to transport the business objects between the broker and the front-end application. You now need to create a JMS connector and to configure it. You will import the connector and its metaobjects from repository export files. You will create a connector from the beginning for the custom adapter in later chapters in this book.

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

21.2.1 Creating a JMS Connector

To create a JMS Connector:

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** (Figure 21-25 on page 356).

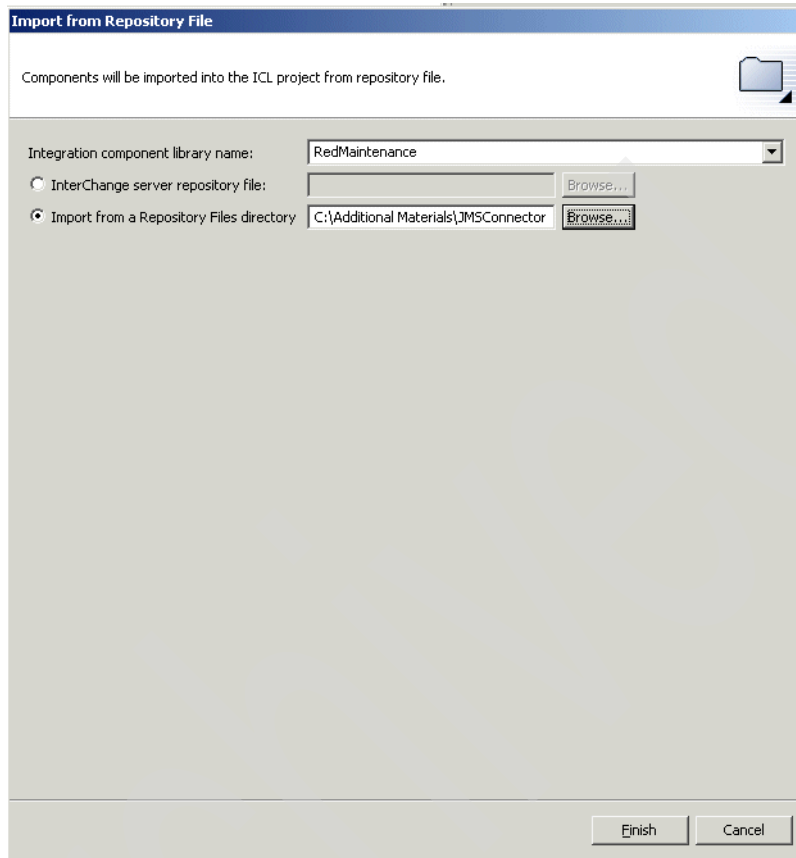


Figure 21-25 Import repository file

5. Navigate to the Additional Materials JMSConnector folder within the Repository Files Directory.
6. Click **Finish** to create the 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 (Figure 21-26 on page 357).

All connectors have standard properties. Some of these properties differ slightly, depending on the integration broker that you are using.

Standard Properties Connector-Specific Properties Supported Business Objects Trace/Log			
	Property ▾	Value	Update Method
1	AdminInQueue	WEBTENANT.ADMININ	component restart
2	AdminOutQueue	WEBTENANT.ADMINOUT	component restart
3	AgentTraceLevel	5	dynamic
4	ApplicationName	JMSConnector	component restart
5	BrokerType	WMQI	connector restart
6	CharacterEncoding	ascii7	component restart
7	ContainerManagedEvents		component restart
8	DeliveryQueue	WEBTENANT.DELIVERY	component restart
9	DeliveryTransport	JMS	component restart
10	DuplicateEventElimination	false	component restart
11	FaultQueue	WEBTENANT.FAULT	component restart
12	jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory	component restart
13	jms.MessageBrokerName	REDBROKER	component restart
14	jms.NumConcurrentRequests	10	component restart
15	jms.Password	*****	component restart
16	jms.UserName		component restart
17	Locale	en_US	component restart
18	MessageFileName	JMSConnector.txt	component restart
19	PollEndTime	HH:MM	component restart
20	PollFrequency	5000	dynamic
21	PollStartTime	HH:MM	component restart
22	RepositoryDirectory	C:\IBM\WebSphere\Adapters\repository	Agent restart
23	RequestQueue	WEBTENANT.REQUEST	component restart
24	ResponseQueue	WEBTENANT.RESPONSE	component restart
25	RestartRetryCount	3	dynamic
26	RestartRetryInterval	1	dynamic
27	RFH2MessageDomain	mrm	component restart
28	SynchronousRequestQueue	WEBTENANT.SYNCHRONOUSREQUEST	component restart
29	SynchronousRequestTimeout	0	component restart
30	SynchronousResponseQueue	WEBTENANT.SYNCHRONOUSRESPONSE	component restart
31	WireFormat	CwXML	agent restart
32	XMLNamespaceFormat	long	agent restart

Figure 21-26 Standard properties

Some of the properties are specific to using a connector with the Message Broker and you need to check some 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 depend on the transport type. For JMS transport type, all of these queues are required.
- The JMS factory classname must be supplied and correct.
- The JMS Message Broker Name is the name of the 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, which is 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 shown in Figure 21-27.

Standard Properties					Connector-Specific Properties					Supported Business Objects					Trace/Log File				
	Property	Value			Encrypt	Update Method													
1	DataHandlerConfigMO	JMS_MO_DataHandler			<input type="checkbox"/>	agent restart													
2	LogFileName	STDOUT			<input type="checkbox"/>	agent restart													
3	DataHandlerMimeType	text/xml			<input type="checkbox"/>	agent restart													
4	DataHandlerClassName	com.crossworlds.DataHandlers.text.xml			<input type="checkbox"/>	agent restart													
5	ConfigurationMetaObjec	JMS_MO_Config			<input type="checkbox"/>	agent restart													
6	TraceFileName	STDOUT			<input type="checkbox"/>	agent restart													
7	InDoubtEvents	Reprocess			<input type="checkbox"/>	agent restart													
8	ErrorQueue	queue://REDBROKER/WEBTENANT.ERROR			<input type="checkbox"/>	agent restart													
9	InProgressQueue	queue://REDBROKER/WEBTENANT.INPROGRESS			<input type="checkbox"/>	agent restart													
10	InputQueue	queue://REDBROKER/WEBTENANT.EVENT			<input type="checkbox"/>	agent restart													
11	UnsubscribedQueue	queue://REDBROKER/WEBTENANT.UNSUBSCRIBED			<input type="checkbox"/>	agent restart													
12	ReplyToQueue	queue://REDBROKER/WEBTENANT.RESULT			<input type="checkbox"/>	agent restart													
13	MessageResponseRes	queue://REDBROKER/WEBTENANT.RESULT			<input type="checkbox"/>	agent restart													
14	CTX_InitialContextFactor	com.sun.jndi.fscontext.ReifFSContextFactory			<input type="checkbox"/>	agent restart													
15	CTX_ProviderURL	file:/C:/temp			<input type="checkbox"/>	agent restart													
16	QueueConnectionFactory	MyQCF			<input type="checkbox"/>	agent restart													
17	DefaultVerb	Retrieve			<input type="checkbox"/>	agent restart													
18	UseDefaults	true			<input type="checkbox"/>	agent restart													

Figure 21-27 Connector-specific properties

- Figure 21-27 shows the configuration properties that are specific to this type of connector. You will investigate these properties as we progress.
- Normally, you would save this configuration to file before continuing.

21.3 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 scenario).

A connector instantiates a data handler based on the MIME type of an input file or the MIME type that is 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 that references 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 that are defined in data-handler metaobjects.

21.3.1 Creating metaobjects

To create a metaobject:

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**.

This creates a metaobject business object definition.

8. Navigate back to the System Manager and refresh the view.
9. Double-click the new JMSConnector to open the Connector Configuration properties (Figure 21-28).

Standard Properties Connector-Specific Properties Supported Business Objects		
	Property	Value
1	DataHandlerConfigMO	JMS_MO_DataHandler
2	LogFileName	STDOUT
3	DataHandlerMimeType	text/xml
4	DataHandlerClassName	com.crossworlds.DataHandlers.text.xml
5	ConfigurationMetaObject	JMS_MO_Config
6	TraceFileName	STDOUT
7	InDoubtEvents	Reprocess
8	ErrorQueue	queue://REDBROKER/WEBTENANT.ERROR
9	InProgressQueue	queue://REDBROKER/WEBTENANT.INPROGRESS
10	InputQueue	queue://REDBROKER/WEBTENANT.EVENT
11	UnsubscribedQueue	queue://REDBROKER/WEBTENANT.UNSUBSCRIBED
12	ReplyToQueue	queue://REDBROKER/WEBTENANT.RESULT
13	MessageResponseResultProperty	queue://REDBROKER/WEBTENANT.RESULT
14	CTX_InitialContextFactory	com.sun.jndi.fscontext.RefFSContextFactory
15	CTX_ProviderURL	file:/C:/temp
16	QueueConnectionFactoryName	MyQCF
17	DefaultVerb	Retrieve
18	UseDefaults	true

Figure 21-28 Connector properties

The names of the metaobjects appear in the Connector-specific properties:

- DataHandlerConfigMO

The metaobject that is passed to the data handler to provide configuration information and is called JMS_MO_DataHandler.

- ConfigurationMetaObject

The metaobject that contains the processing rules for the connector and is called JMS_MO_Config.

10. From the System Manager, open JMS_MO_DataHandler (Figure 21-29).

General		Attributes							
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value
1	1	text_xml	MO_DataHandler_DefaultXMLConfig	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
1.1	1.1	UseNewLine	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	false
1.2	1.2	EntityResolver	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
1.3	1.3	DTDPath	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
1.4	1.4	NameHandlerClass	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	com.crossworlds.DataHandlers.xml.TopElementNameHandler
1.5	1.5	Parser	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
1.6	1.6	Validation	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	false
1.7	1.7	ClassName	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	com.crossworlds.DataHandlers.text.xml
1.8	1.8	BOPrefix	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	Web
1.9	1.9	InitialBufferSize	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	2097152
1.10	1.10	DefaultEscapeBehavior	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	true
1.11	1.11	IgnoreUndefinedElements	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	false
1.12	1.12	IgnoreUndefinedAttributes	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	true
1.13	1.13	DummyKey	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	1
1.14	1.14	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
2	2	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			

Figure 21-29 Data handler metaobject

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

11. Open the JMS_MO_Config business object (Figure 21-30).

	Pos	Name	Type	Key	Foreign Key	Required	Maximum Length	Application Specific Information
1	1	Web_tenant_Retrieve	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	255	InputFormat=MGSTR;OutputFormat=MGSTR;OutputDestination=queue://REDTENANT/RTRES1Q
2	2	Web_newMaintenance_Create	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	255	InputFormat=MGSTR;OutputFormat=MGSTR;OutputDestination=queue://REDTENANT/RTRES2Q
3	3	Default	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	255	OutputDestination=queue://REDTENANT/RTRES1Q
4	4	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Figure 21-30 JMS_MO_Config business object

12. 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 indicates the following:
`InputFormat=MQSTR;OutputFormat=MQSTR;OutputDestination=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. The destination of that message is a queue named RTRES1Q at queue manager REDTENANT, which is our application's response message queue for retrieving queries.

There are many options available, but these alone suit our purposes.
 - The Web_newMaintenance_Create indicates the application response queue for the create requests.
13. Save these business objects to the repository directory and to file, and then close them.

21.4 Queue connection factory and queue objects

Because you are using queue based messaging, you must perform the following additional configuration tasks:

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 connects.
4. Ensure that each of the queues listed in Figure 21-31 on page 363 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.

Standard Properties Connector-Specific Properties Supported Business Objects		
	Property ▾	Value
1	AdminInQueue	WEBTENANT.ADMININ
2	AdminOutQueue	WEBTENANT.ADMINOUT
3	AgentTraceLevel	5
4	ApplicationName	JMSConnector
5	BrokerType	WMQI
6	CharacterEncoding	ascii7
7	ContainerManagedEvents	
8	DeliveryQueue	WEBTENANT.DELIVERY
9	DeliveryTransport	JMS
10	DuplicateEventElimination	false
11	FaultQueue	WEBTENANT.FAULT
12	jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory
13	jms.MessageBrokerName	REDBROKER
14	jms.NumConcurrentRequests	10
15	jms.Password	*****
16	jms.UserName	
17	Locale	en_US
18	MessageFileName	JMSConnector.txt
19	PollEndTime	HH:MM
20	PollFrequency	5000
21	PollStartTime	HH:MM
22	RepositoryDirectory	C:\IBM\WebSphereAdapters\repository
23	RequestQueue	WEBTENANT.REQUEST
24	ResponseQueue	WEBTENANT.RESPONSE
25	RestartRetryCount	3
26	RestartRetryInterval	1
27	RFH2MessageDomain	mrn
28	SynchronousRequestQueue	WEBTENANT.SYNCHRONOUSREQUEST
29	SynchronousRequestTimeout	0
30	SynchronousResponseQueue	WEBTENANT.SYNCHRONOUSRESPONSE
31	WireFormat	CwXML
32	XMLNameSpaceFormat	long

Figure 21-31 Standard properties

5. Click **Connector-Specific Properties**, as shown in Figure 21-32 on page 364.

Standard Properties		Connector-Specific Properties	Supported Business Objects	Trace/Log File
	Property	Value	Encrypt	Update Method
1	DataHandlerConfigMO	JMS_MO_DataHandler	<input type="checkbox"/>	agent restart
2	LogFileName	STDOUT	<input type="checkbox"/>	agent restart
3	DataHandlerMimeType	text/xml	<input type="checkbox"/>	agent restart
4	DataHandlerClassName	com.crossworlds.DataHandlers.text.xml	<input type="checkbox"/>	agent restart
5	ConfigurationMetaObjec	JMS_MO_Config	<input type="checkbox"/>	agent restart
6	TraceFileName	STDOUT	<input type="checkbox"/>	agent restart
7	InDoubtEvents	Reprocess	<input type="checkbox"/>	agent restart
8	ErrorQueue	queue://REDBROKER/WEBTENANT.ERROR	<input type="checkbox"/>	agent restart
9	InProgressQueue	queue://REDBROKER/WEBTENANT.INPROGRESS	<input type="checkbox"/>	agent restart
10	InputQueue	queue://REDBROKER/WEBTENANT.EVENT	<input type="checkbox"/>	agent restart
11	UnsubscribedQueue	queue://REDBROKER/WEBTENANT.UNSUBSCRIBED	<input type="checkbox"/>	agent restart
12	ReplyToQueue	queue://REDBROKER/WEBTENANT.RESULT	<input type="checkbox"/>	agent restart
13	MessageResponseRes	queue://REDBROKER/WEBTENANT.RESULT	<input type="checkbox"/>	agent restart
14	CTX_InitialContextFactor	com.sun.jndi.fscontext.RefFSContextFactory	<input type="checkbox"/>	agent restart
15	CTX_ProviderURL	file:/C:/temp	<input type="checkbox"/>	agent restart
16	QueueConnectionFactory	MyQCF	<input type="checkbox"/>	agent restart
17	DefaultVerb	Retrieve	<input type="checkbox"/>	agent restart
18	UseDefaults	true	<input type="checkbox"/>	agent restart

Figure 21-32 Queue objects and qcf

6. Because you are using JMS as the transport, ensure that JMS-related properties are correct and that the configuration is in place:
 - QueueConnectionFactoryName is the JMS queue connection factory object that is 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 that is established by the CTX_InitialContextFactory and CTX_ProviderURL properties.
 - CTX_InitialContextFactory is the name of the factory class that is used to establish an initial JNDI context.
 - CTX_ProviderURL is a fully-qualified URL that identifies the JNDI context where the connection factor is located. This value is passed to the context factor.
7. Update the JMS configuration by opening c:\WMQ\Java\bin\jmsAdmin.config and setting the following properties:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
PROVIDER_URL=file:/c:/temp
SECURITY_AUTHENTICATION=none
```

8. Create a file named MyJNDI.txt that contain the information that is shown in Example 21-1.

Example 21-1 JNDI

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

In this example:

- student is the host name of your broker machine
- 1421 is the listener port for the REDBROKER queue manager
- REDBROKER is the queue manager for your broker
- CHANNEL1 is a svrconn channel.

9. Change the directory to C:\WMQ\Java\bin.
10. Bind objects to JNDI names by running the following command:

```
jmsAdmin.bat < MyJNDI.txt
```

This now matches our configuration properties.

11. Ensure that all the queues that are 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.

21.5 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 contains the correct information to allow the Broker to identify and parse the message. The Message Set name is the one from our connector configuration and the Message is the name of our business object.

To add supported business objects:

1. Click the Supported Business Objects tab (Figure 21-33).

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.

2. Check that all of the message sets are WebTenant. Modify those that are not.

Standard Properties		Connector-Specific Properties		Supported Business	
	Business Object Name			Message Set ID	
1	JMS_MO_Config			WebTenant	
2	JMS_MO_DataHandler			WebTenant	
3	MO_DataHandler_Default			WebTenant	
4	MO_DataHandler_DefaultXMLConfig			WebTenant	
5	Web_tenant			WebTenant	
6	Web_tenant_maintenance			WebTenant	
7	Web_newMaintenance			WebTenant	
8					

From this queue the connector takes the raw application data and sends it to the DeliveryQueue (WEBTENANT.DELIVERY) as a business object that is 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.

To start the connector:

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 by selecting **Start** → **Programs** → **IBM WebSphere Business Integration Adapters** → **Adapters** → **Connectors** → **JMSConnector**.
6. When the connector has completed its initialization, you see a message that indicates that it is ready to accept polling requests.
7. Start the WebSphere Application Server that runs the front-end application from the First Steps menu of WebSphere.
8. When the server is initialized and opened for e-business, open a Web browser for the RedTenant application.

21.7 Unit Testing

To perform the first test:

1. Enter an ID for the application logon. Use 100, as shown in Figure 21-34 on page 368, because it is one that exists and is also one for which we have created some sample messages for testing.

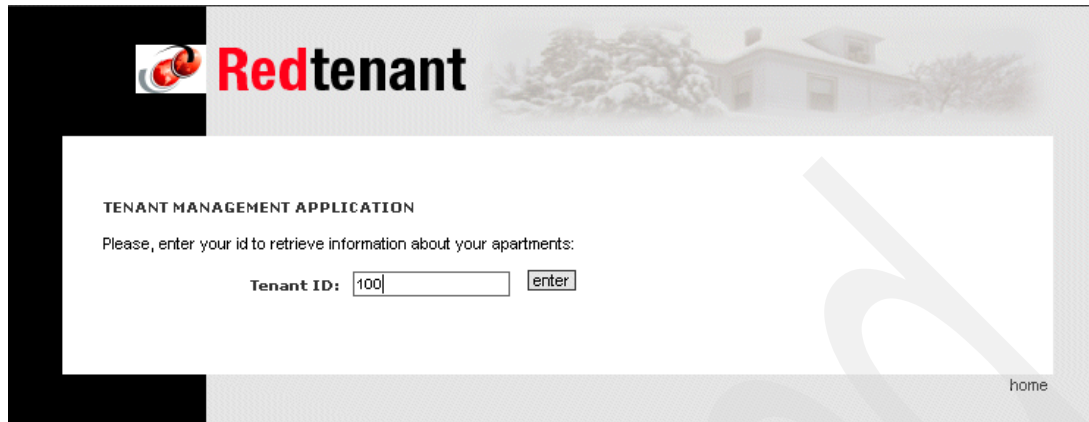


Figure 21-34 RedTenant home page

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

2. Start an instance of **rfhutl1**, which is included in the Additional Materials.
3. Select the **WEBTENANT.EVENT** queue on the **REDBROKER** queue manager (Figure 21-35).

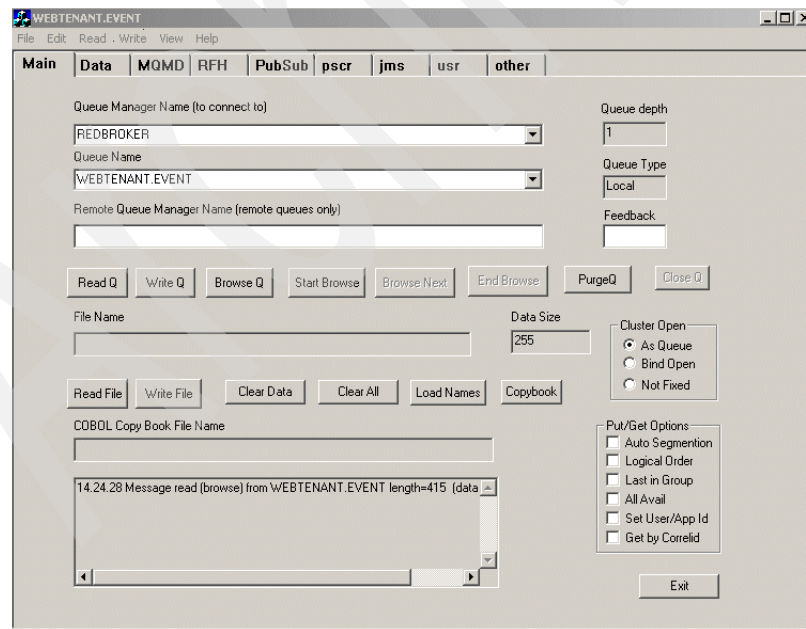


Figure 21-35 rfhutl1

4. Browse the message on the queue using Browse Q, not Read Q.
5. Switch to the Data tab and select the XML Data Format of the message, as shown in Figure 21-37 on page 370. This data is the raw data from the front-end application.

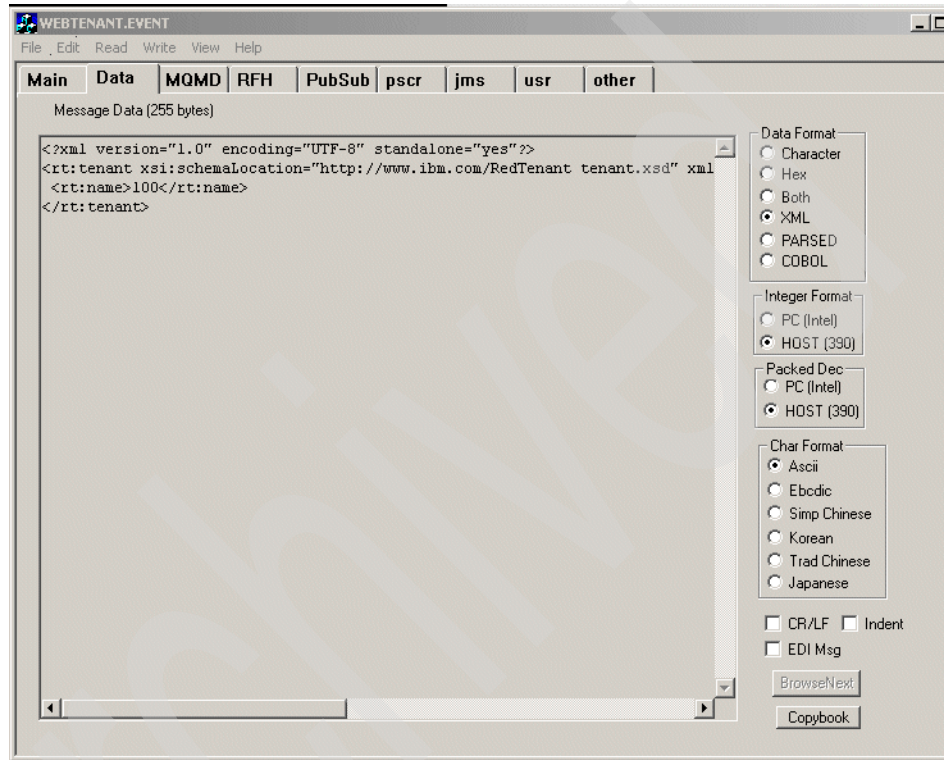


Figure 21-36 Event data

6. Switch to the RFH tab and look at the RFH header information, as shown in Figure 21-37.

The screenshot shows the 'WEBTENANT.EVENT' application window with the 'RFH' tab selected. The window contains several configuration sections:

- RFH V2 Fixed Data:**
 - Length: 160
 - RFH Encoding: Integer (PC, Host, Unix) with PC selected.
 - Pack Dec: PC, Host with Host selected.
 - Data Format: MQSTR
 - Code Page: 1208
 - Flags: 0
 - CCSID: 1208
- Message Domain:** jms_text
- Msg Set:** (empty text box)
- Msg Type:** (empty text box)
- Output Format:** (empty text box)
- V2 Folders:** mod, jms, usr, PubSub, pscr, other (mod and jms are checked).

V1 Fixed Data:

- Length: 0
- Encoding: Integer (PC, Host, Unix) with PC selected.
- Pack Dec: PC, Host with PC selected.
- Data Format: (empty text box)
- Code Page: 0
- Flags: 0

Appl Group: (empty text box)

Format Name: (empty text box)

V1 Contents: mod, PubSub, Resp (all are unchecked).

RFH Type: None, Version 1, Version 2, Both (Version 2 is selected).

Figure 21-37 RFH2 header

7. Go to the connector DOS window and enter p to begin polling for an event.
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.

8. Go back to **rfhuti1**, and get the message from the **WEBTENANT.DELIVERY** queue (Figure 21-38). There is no need to browse it, because you do not currently need this message.

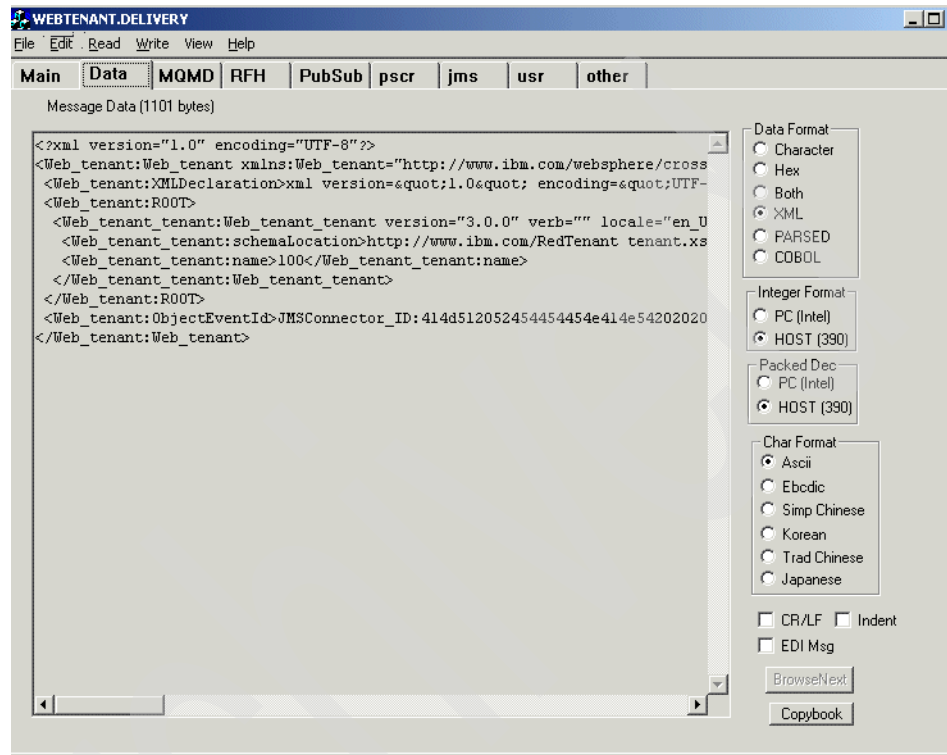


Figure 21-38 Message delivered to the broker

9. Select the **Data** tab to see the transformed data.
10. Select the **RFH** tab, as shown in Figure 21-39 on page 372, and notice that the information that you provided with the connector configuration and metaobjects has caused the RHF2 header to be populated correctly for the Broker to interpret as follows:
- Message Domain = mrm
 - Message Set = WebTenant
 - Message Type = Web_tenant
 - Message Format = CwXML
 - Data Format = MQSTR

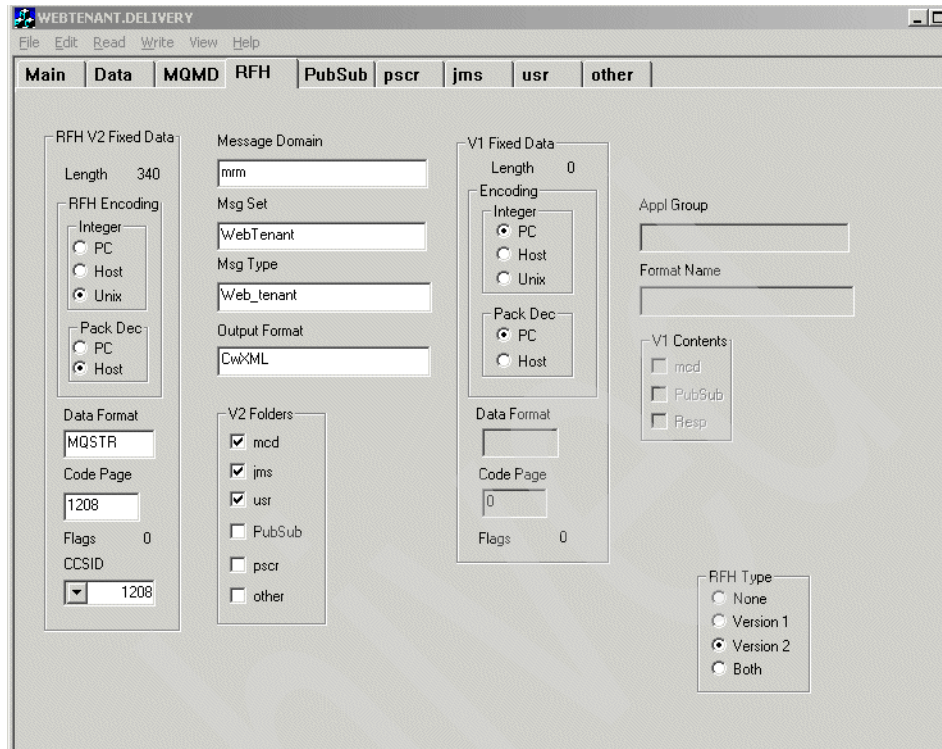


Figure 21-39 RFH2 header

The outbound request looks good, so you can try one further test that sends a response to the connector.

In the Additional Materials folder is folder called RedTenantTestMessages. You will use one of these messages to simulate a message coming from the Broker back to the connector to check that the configuration is sound in both directions.

To perform the second test:

1. Open an instance of **rfhuti1**.
2. Select the WEBTENANT.REQUEST queue at queue manager REDBROKER.
3. Using Read File, locate the test message named Web_tenant_Request, which indicates that a message was sent to the connector that requests an action from the application.

4. Select the Data tab to see the message contents. It is similar to that shown in Figure 21-40.

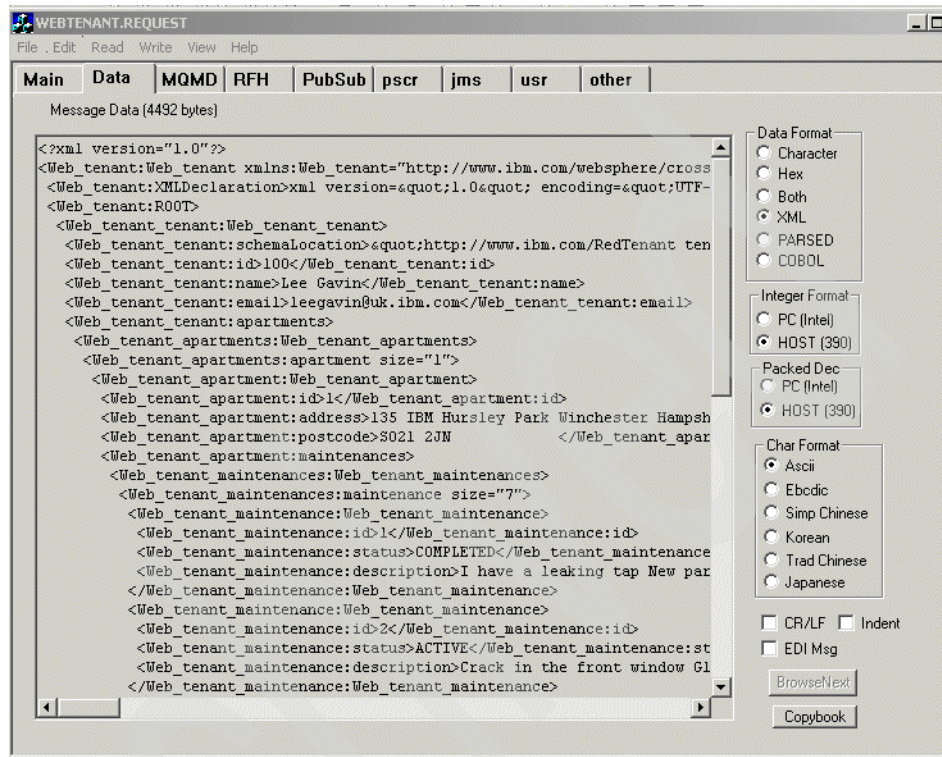


Figure 21-40 Request data

5. Select the RFH tab to see that the RFH2 header information is in place (Figure 21-41).

The screenshot shows the 'WEBTENANT.RESULT' application window with the 'RFH' tab selected. The window contains several configuration panels:

- RFH V2 Fixed Data:**
 - Length: 436
 - RFH Encoding: Integer (PC, Host, Unix) - PC is selected.
 - Pack Dec: PC, Host - Host is selected.
 - Data Format: MQSTR
 - Code Page: 437
 - Flags: 0
 - CCSID: 1208
- Message Domain:** mmm
- Msg Set:** WebTenant
- Msg Type:** Web_tenant
- Output Format:** CwXML
- V2 Folders:** mcd, jms, usr (all checked); PubSub, pscr, other (unchecked).
- V1 Fixed Data:**
 - Length: 0
 - Encoding: Integer (PC, Host, Unix) - PC is selected.
 - Pack Dec: PC, Host - Host is selected.
 - Data Format: (empty)
 - Code Page: 0
 - Flags: 0
- Appl Group:** (empty)
- Format Name:** (empty)
- V1 Contents:** mod, PubSub, Resp (all unchecked).
- RFH Type:** None, Version 1, Version 2, Both - Version 2 is selected.

Figure 21-41 RFH2 header

6. Write the Web_tenant_Request to the queue.
The connector picks up this message and transforms it from business object format to application format.
7. Using `rfhut i1`, browse the contents of the message on the RTRES1Q queue on queue manager REDTENANT.

8. Select the Data tab to see the application representation of the business object. It is similar to that shown in Figure 21-42.

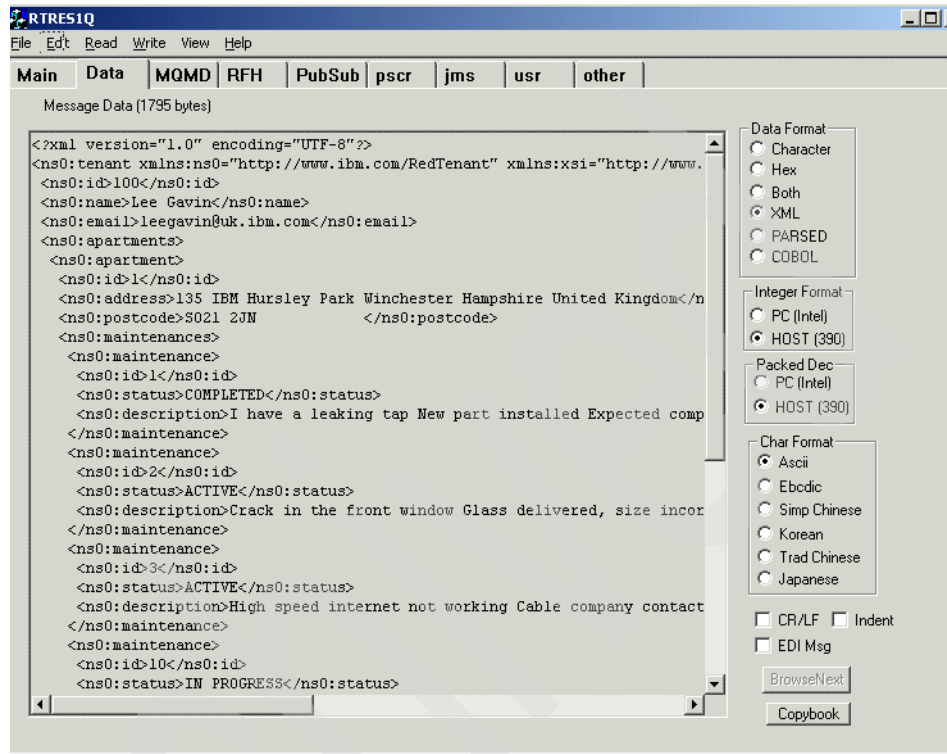


Figure 21-42 Application data

9. Go back to the browser and resubmit the query. This request is not an actual request and reply because you have already loaded the queue. However, you will see whether the application can handle the message data.



Figure 21-43 Tenant and apartment details

10. Select one of the maintenance records to confirm that the data you expect to see is there. It is e similar to that shown in Figure 21-44. Verify this data by checking in the back-end application database.

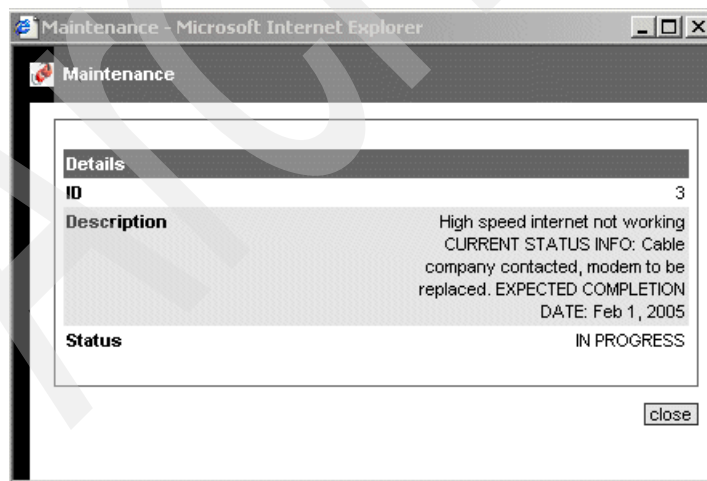


Figure 21-44 Maintenance detail

After you have the JMSConnector and business objects for the front-end application in place and working as expected, you can create all of the components that are required for a custom back-end application connector.

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

Archived



Object Discovery Agent

To create the business objects, you deploy an Object Discovery Agent (ODA), that is written for the custom adapter. You use this ODA to assist in building the required business objects for the back-end application.

22.1 Setting up the custom ODA

The custom ODA consists of several components:

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

Move these files into their correct place and create the start mechanism by following these steps:

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 the following folder:

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.

7. For the ODA to be able to communicate with the back-end application, the application must be running. If it is not, execute the **rmiregistry** command from a command prompt. Leave this window minimized.
8. Start the RedMaintenance Engine.
9. Using the shortcut that you created for the ODA, start the RedMaintenance ODA.

22.2 Generating business objects using the ODA

To generate business objects with the ODA:

1. Navigate to the System Manager.
2. From your RedMaintenance ICL, right-click **Business Objects** and select **Create New Business Object** to start the Business Object Designer.
3. Close the new business object window.
4. Select **File** → **New Using ODA**, as shown in Figure 22-1.

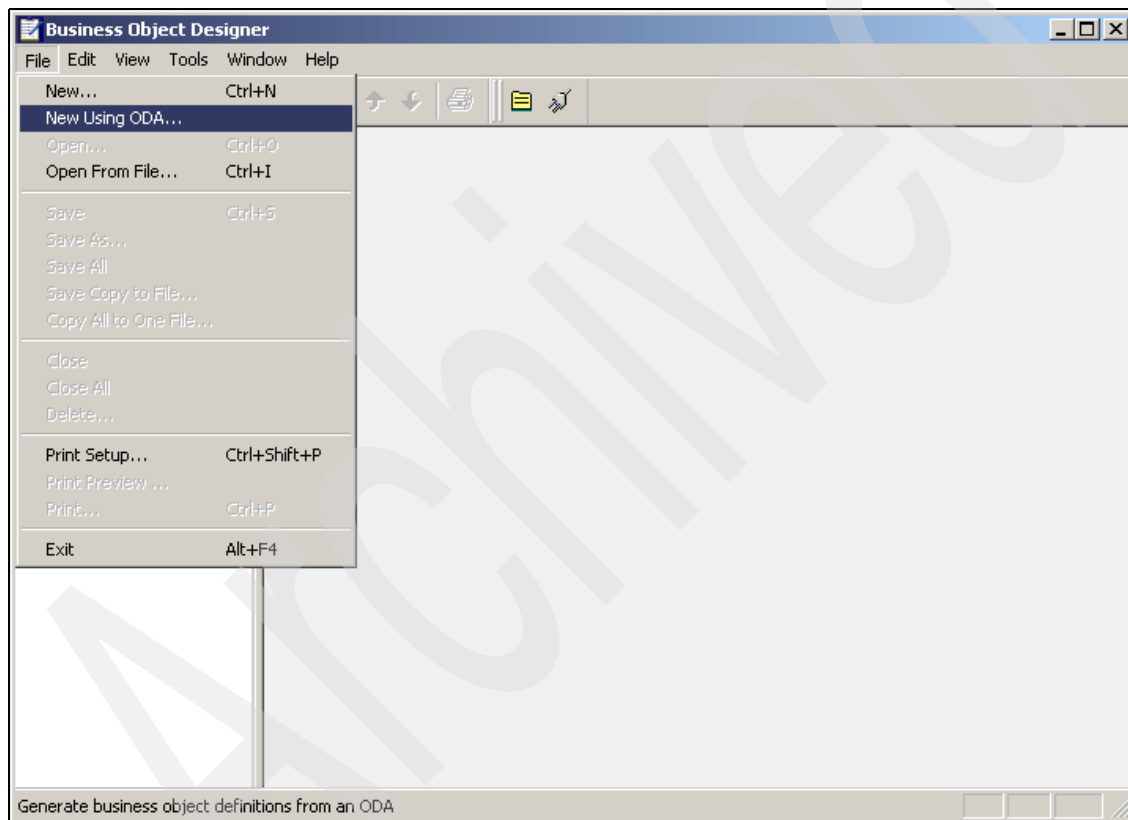


Figure 22-1 ODA menu

The first window of the Business Object Wizard displays, as shown in Figure 22-2.

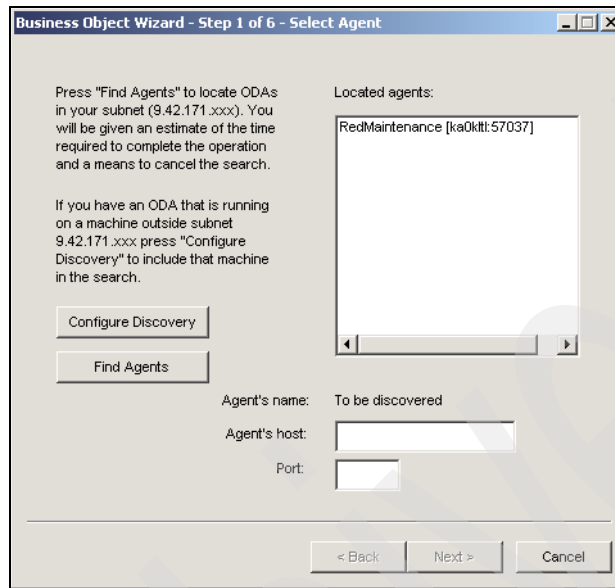


Figure 22-2 Select Agent

5. Click **Find Agent** to find the ODA.

Do not worry if you see the name of the XML ODA in the list. It is just remembering this name from 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 that you expect to be running. Remember that this ODA is using the ORB to find the agents that are running.

While the ODA Wizard is search for the registered agents, it displays a message (Figure 22-3).

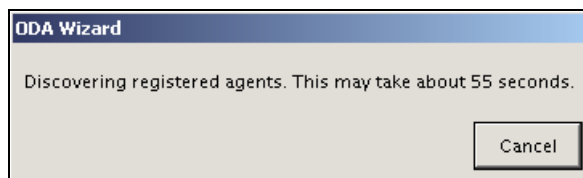


Figure 22-3 Discovering agent

- When the ODA is located, the wizard displays its information, as shown in Figure 22-4.

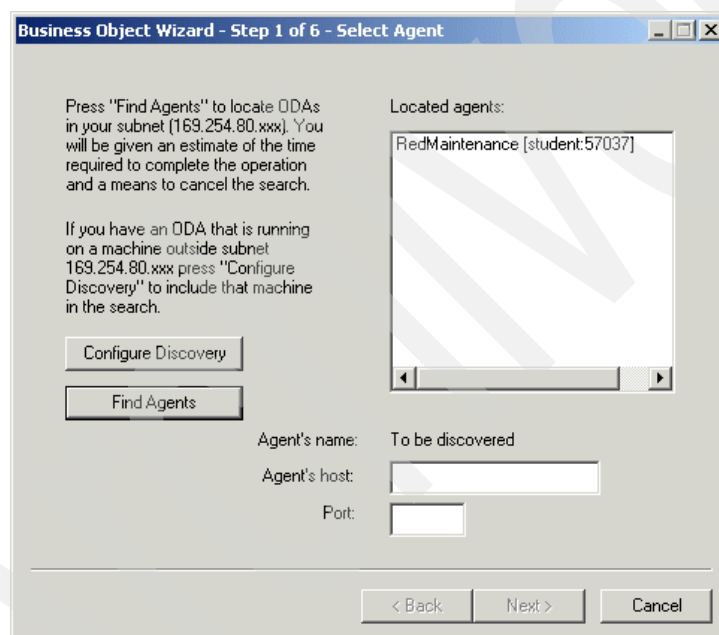
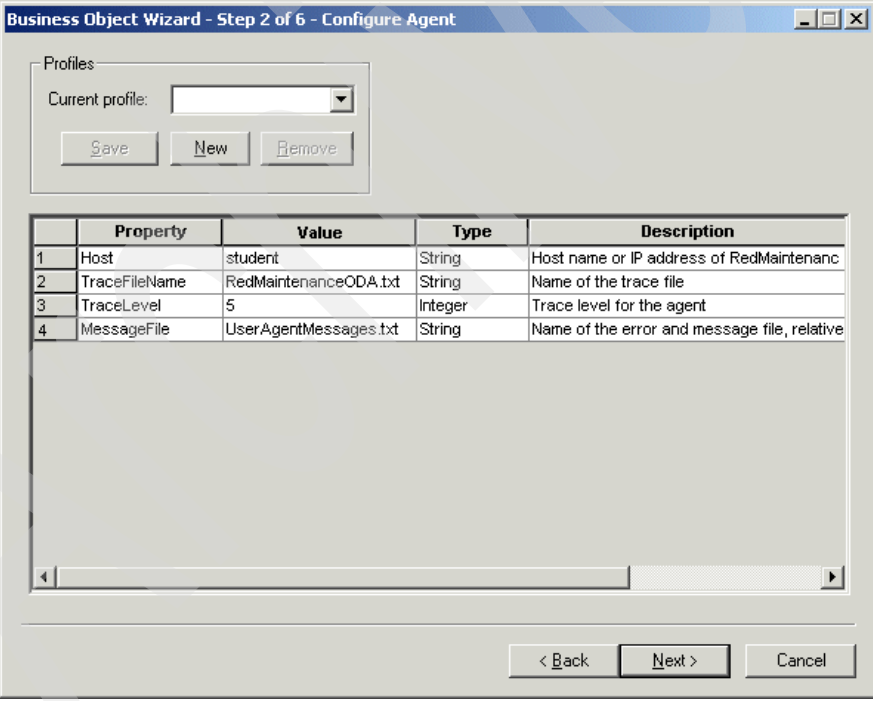


Figure 22-4 Located agent

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.

7. Select this agent and click **Next**.
8. In Figure 22-5, enter the required property values as follows:
 - 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.



The screenshot shows a window titled "Business Object Wizard - Step 2 of 6 - Configure Agent". It features a "Profiles" section with a "Current profile:" dropdown menu and "Save", "New", and "Remove" buttons. Below this is a table with four columns: "Property", "Value", "Type", and "Description". The table contains four rows of configuration data. At the bottom of the window are "< Back", "Next >", and "Cancel" buttons.

	Property	Value	Type	Description
1	Host	student	String	Host name or IP address of RedMaintenanc
2	TraceFileName	RedMaintenanceODA.txt	String	Name of the trace file
3	TraceLevel	5	Integer	Trace level for the agent
4	MessageFile	UserAgentMessages.txt	String	Name of the error and message file, relative

Figure 22-5 Properties

9. Click **Next**.

10. The components that have been discovered from the back-end application are displayed, as shown in Figure 22-6.

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

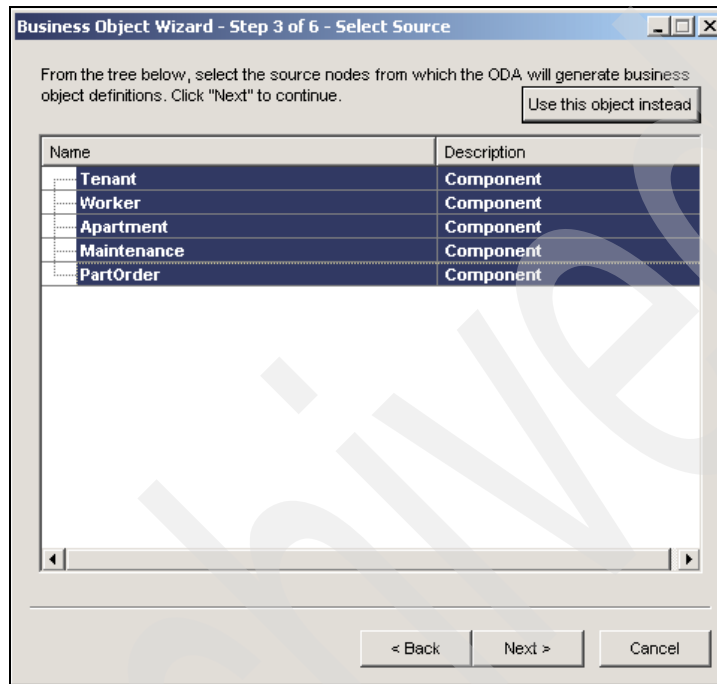


Figure 22-6 Select nodes

11. Select the nodes from which the ODA will generate business objects. In our testing, we select all of them.

12. Click **Next**.

13. You are asked for confirmation before you click **Next** again (Figure 22-7).

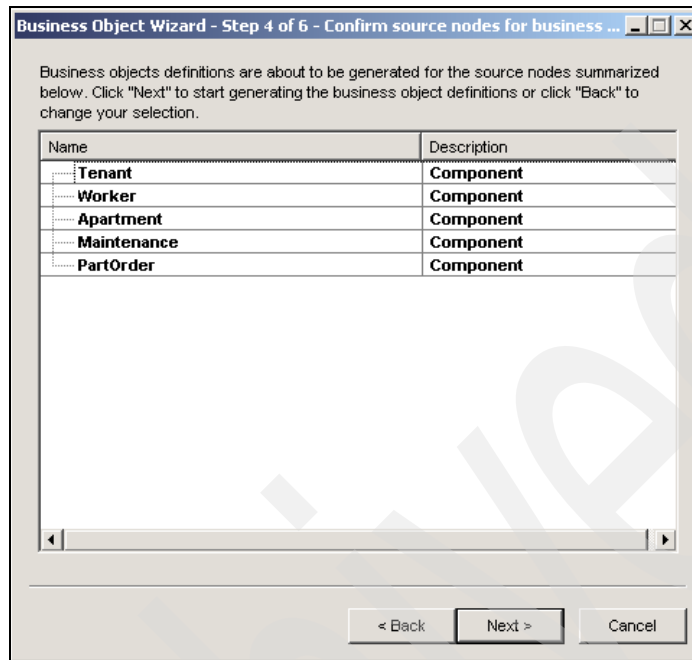


Figure 22-7 Confirmation for generation

14. Enter any additional properties that are needed for the business object generation, as shown in Figure 22-8.
15. Enter a prefix for the business object names that are to be generated. We select RM_. Click **OK**. The business object definition generation begins.

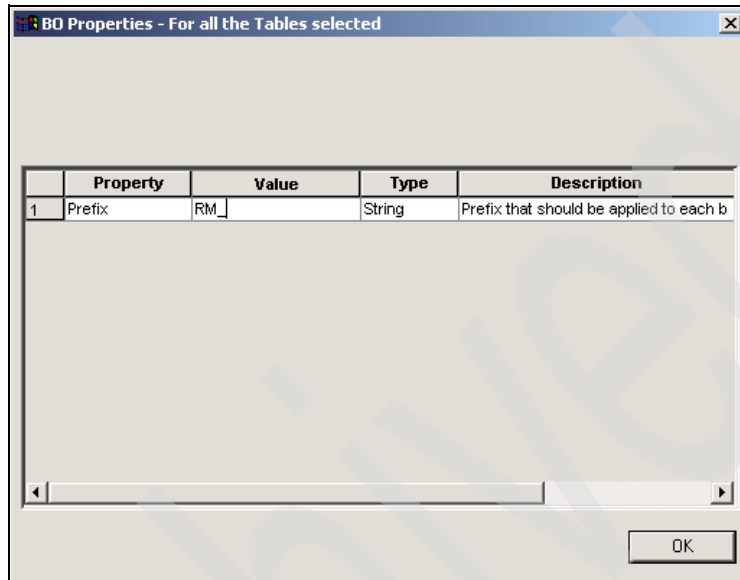


Figure 22-8 Business object property

16. Save the business objects.

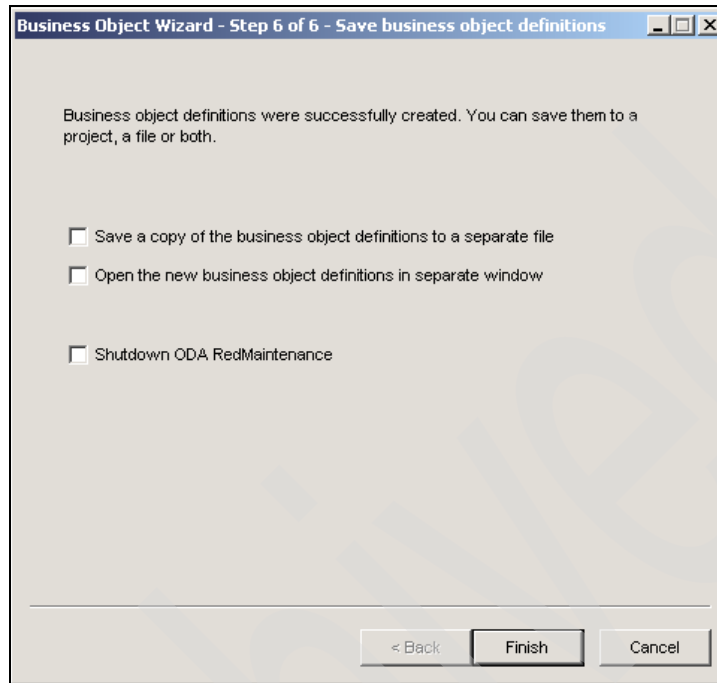


Figure 22-9 Confirmation

17. Select **Shutdown ODA RedMaintenance**. We no longer need the ODA.

18. Click **Finish**.

The Business Object Designer (Figure 22-10) lists the newly created business object definitions. The asterisk (*) beside each object indicates that they are unsaved.

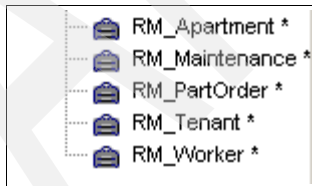


Figure 22-10 Generated business objects

You can also see from the GUI activity window of the back-end application that the ODA has successfully contacted the application (Figure 22-11 on page 389).

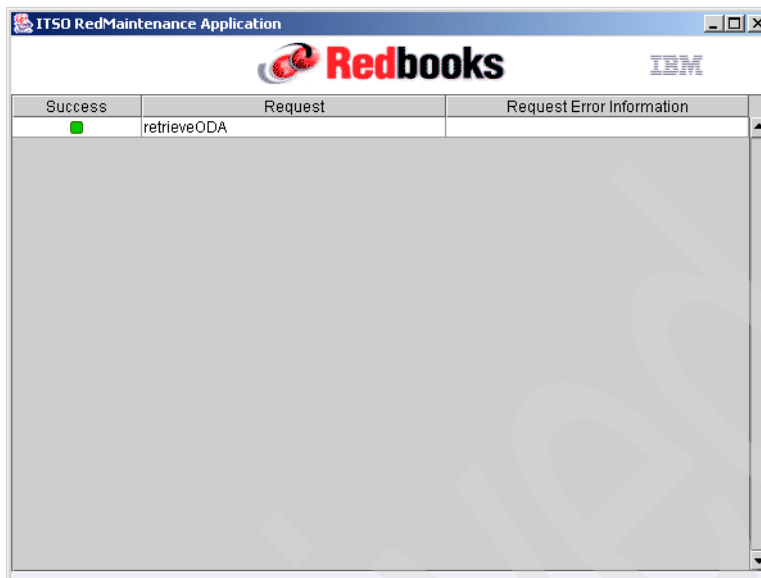


Figure 22-11 RetrieveODA

22.2.1 Completing the business objects

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

Important: In the real world, it is here that you would require the assistance of someone who has a detailed knowledge of the application itself. Do not skip 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 the ODA generated. Then, detail the amendments that you need to make to the business objects to allow the adapter to interact successfully with the API.

General		Attributes
Business Object Level Application-specific information:		
obj= Tenant		
Supported Verbs:		
	Name	Application-specific information
1	Create	verb=Create
2	Delete	verb=Delete
3	Retrieve	verb=Retrieve
4	RetrieveByContent	verb=RetrieveByContent
5	Update	verb=Update
6		

Figure 22-12 General properties: Tenant

The business objects that are generated for the application objects are all correct in terms of the supported verbs and the application object names, as shown in Figure 22-12 for the Tenant business object, for the application-specific information.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Name
3	3	ApartmentId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId
4	4	EMail	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=EMail
5	5	ObjectEventId	String							
6	6			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-13 Tenant business object

Figure 22-13 on page 390 lists the attributes that the ODA generates.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required Attribute	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentNumber
3	3	AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine1
4	4	AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine2
5	5	AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine3
6	6	AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine4
7	7	PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		20		attr=PostCode
8	8	ObjectEventId	String							
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-14 Apartment business object

The application-specific information is correct in that the attribute names of the application object have been generated correctly. Key fields and mandatory fields are reported. However, you need to check whether this information is 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 information is not accurate, because the tenant application object is a parent object which contains apartment and possibly maintenance child objects. The application-specific information about the EMail attribute is also incorrect. You will update all of these as you continue.

Figure 22-14 on page 391 shows 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.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId
3	3	TenantId	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=TenantId
4	4	ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=ProblemDescription
5	5	StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=StatusDescription
6	6	ExpectedCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ExpectedCompletion
7	7	ActualCompletion	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=ActualCompletion
8	8	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-15 Maintenance business object

Figure 22-15 shows the following errors that need correcting:

- ▶ The maintenance business object also has multiple keys flagged, which is not the case.
- ▶ 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.
- ▶ The ODA has interpreted the dates as string data when in fact they are dates.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Card	Maximum Length	Default	App Spec Info
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentNumber
3	3	AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine1
4	4	AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine2
5	5	AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine3
6	6	AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=AddressLine4
7	7	PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		20		attr=PostCode
8	8	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
9	9			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-16 Correct apartment business object

To modify the generated business objects to reflect accurately the application view of the objects, complete these steps:

1. The apartment business object in Figure 22-16 on page 392 is correct as discovered. Save this business object.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	2	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=ApartmentId; chfk=RM_Tenant.ApartmentId
3	3	Status	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		attr=Status
4	4	TenantId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				attr=TenantId; chfk=RM_Tenant.Id
5	5	ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=ProblemDescription
6	6	StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		500		attr=StatusDescription
7	7	ExpectedCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ExpectedCompletion
8	8	ActualCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				attr=ActualCompletion
9	9	ObjectEventId	String							
10	10			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-17 Correct maintenance 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.

2. Modify the maintenance business object to reflect that it contains foreign key values and column values that are contained in other business objects.
3. The application-specific information chfk reflects that the value for the attribute of this child object is to be obtained from an attribute contained in the parent object.

Figure 22-17 on page 393 shows the following values:

- 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 are passed to the application for the search.

General Attributes									
Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default	App Spec Info
1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				attr=Id
2	Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=Name
3	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				attr=ApartmentId;key=RM_Apartment.Id
4	EEmail	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		100		attr=EEmail
5	田 RM_Apartment	RM_Apartment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
6	田 RM_Maintenance	RM_Maintenance	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
7	ObjectEventId	String							
8			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22-18 Correct tenant business object

4. Modify the maintenance business object to add the application-specific information to these attributes as shown in Figure 22-17 on page 393.

General		Attributes						
	Pos.	Name	Type	Key	Foreign Key	Required	Cardinality	Application Specific Information
1	1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		attr=Id
2	2	Name	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=Name
3	3	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		attr=ApartmentId;fk=RM_Apartment.Id
4	4	Email	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=Email
5	5	RM_Apartment	RM_Apartment	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
5.1	5.1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		attr=Id
5.2	5.2	ApartmentNumber	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=ApartmentNumber
5.3	5.3	AddressLine1	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=AddressLine1
5.4	5.4	AddressLine2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=AddressLine2
5.5	5.5	AddressLine3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=AddressLine3
5.6	5.6	AddressLine4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=AddressLine4
5.7	5.7	PostCode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=PostCode
5.8	5.8	ObjectEventId	String					
6	6	RM_Maintenance	RM_Maintenance	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N	
6.1	6.1	Id	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		attr=Id
6.2	6.2	ApartmentId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		attr=ApartmentId;chfk=RM_Tenant.ApartmentId
6.3	6.3	Status	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=Status
6.4	6.4	TenantId	Integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		attr=TenantId;chfk=RM_Tenant.Id
6.5	6.5	ProblemDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=ProblemDescription
6.6	6.6	StatusDescription	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=StatusDescription
6.7	6.7	ExpectedCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=ExpectedCompletion
6.8	6.8	ActualCompletion	Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		attr=ActualCompletion
6.9	6.9	ObjectEventId	String					
7	7	ObjectEventId	String					
8	8			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Figure 22-19 Tenant business object

5. Modify the business object to reflect that the ApartmentId and TenantId are foreign keys.
6. Remove the key value flags.
7. Change the type for the date fields from String to Date.
8. Inset an attribute of Status with a type of String above the TenantId and application-specific information of attr=Status.
9. Save the business object.
10. Re-open the RM_Tenant business object.

The tenant business object is a parent object of one apartment and 0 - n maintenance objects. We need to reflect this.

- 11.Highlight the row for ObjectEventId by clicking the far left number.
- 12.Right-click and select **Insert Above** (Figure 22-18 on page 394).
- 13.In the Type column select the RM_Apartment type.
- 14.Name this Attribute, RM_Apartment.
- 15.Set the Cardinality to 1, meaning only one occurrence of this child object is allowed. Each tenant has only one apartment record.
- 16.Repeat this for the RM_Maintenance, setting the Cardinality to N, meaning for an apartment, there can be multiple maintenance records.
- 17.Set the Foreign Key flag on for the ApartmentId attribute.
- 18.Update the application-specific information to indicate that this attribute is part of a foreign key, with the foreign key attribute being RM_Apartment.Id.
- 19.Change the application-specific information about the EMail attribute from EMail to Email (with a lowercase letter “m”). All the application-specific information is case-sensitive.
- 20.Save this business object.

If you now expand the child objects within the parent object, you see the correct hierarchical structure of the 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.

The business objects are now ready for use.

Packaging 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.

This chapter discusses the connector naming conventions that make our connector files easier to locate and identify and explains how to define the connector. It also tells you how to create the WebSphere MQ objects, prepare the connector environment, create and execute the startup script for the connector, and create a Windows shortcut to use with the connector in order for the IBM WebSphere Business Integration System to use the connector.

Note: All of the files relating to the RM custom adapter can be found in the Additional Materials in the RM folder.

23.1 Connector naming conventions

Naming conventions provide a way to make our connector files easier to locate and identify. Table 23-1 summarizes the naming conventions that we used for connector files. Many of these file names are based on the connector name, which uniquely identifies the file within the WebSphere Business Integration System. The name, *connName*, can identify the application or technology with which the connector communicates. We named our adapter *RMAdapter*, and using the usual naming convention, we name our run-time connector *RMConnector*.

Table 23-1 Our naming conventions

Connector file	Name
Connector definition	<i>RMConnector</i>
Connector directory	<i>ProductDir</i> /connectors/ <i>RM</i>
Connector configuration file	<i>ProductDir</i> \connectors\RM\RMConnector.cfg
Connector class	<i>RM</i> Agent.java
Connector library	<ul style="list-style-type: none">▶ Java jar file: <i>connDir</i>\RMAdapter.jar▶ Java package: com.ibm.itso.RedHouse.RM.Adapter.RMAgent
Connector startup script	<ul style="list-style-type: none">▶ Windows platforms: <i>connDir</i>\start_RMConnector.bat▶ UNIX-based platforms: <i>connDir</i>/connector_manager_RMConnector.sh where <i>connDir</i> is the name of the connector directory, as defined above.

23.2 Defining the connector

For the custom adapter, you must create a connector definition. A connector obtains its configuration values at startup. This section defines 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 scenario with the Message Broker, the properties that we configure will form part of the local configuration file.

23.2.1 Defining the connector with the Connector Configurator

To define the connector, you create a *connector definition*. This connector definition includes the following information to define the connector in the repository. For initial deployment of a customer adapter, you create a connector definition with a set of standard properties — the properties that are common for most connectors. You also create the WebSphere MQ objects that are required for the connector. You then conduct some basic tests to ensure that the connector starts correctly before testing some of the business objects that you created previously. A tool called *Connector Configurator* collects this information and stores it in the repository.

Note: Not every connector uses all the 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.

Note: If your integration broker is the InterChange Server, the connector and its configuration must be deployed to the repository. The repository is a database with which InterChange Server communicates 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.

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 because the connector reads the business object definitions from the local file system. For basic testing at this stage, there is no deployment involved as such. You can save the configuration in a file and test immediately.

To define the connector with the Connector Configurator:

1. Create the directory which will hold the completed connector configuration (as shown in Table 23-1 on page 398). For our scenario, that directory is C:\IBM\WebSphereAdapters\connectors\RM.
2. From the System Manager, start the Connector Configurator by selecting **Tools** → **Connector Configurator**.
3. Enter the properties for the new connector as shown in Figure 23-1.

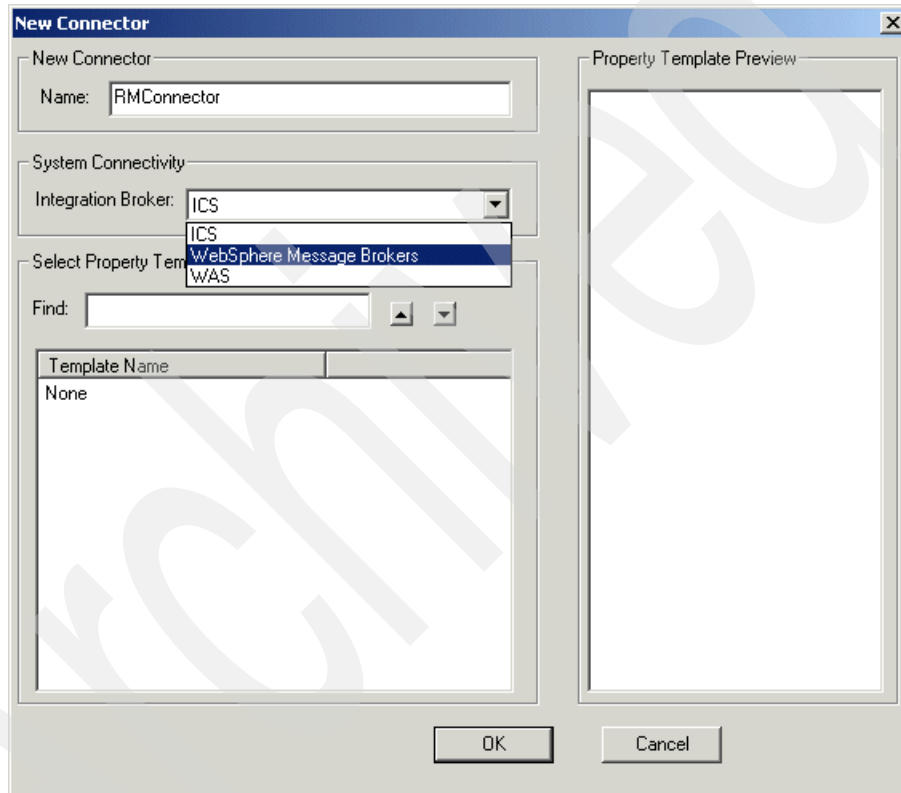


Figure 23-1 New connector

4. Name the connector. We use RMConnector as the name.
5. Select the broker, WebSphere Message Broker.
6. Select **None** for the template name, because you are creating a new custom connector and, as such, there is no template.
7. Click **OK**.

23.2.2 .Defining the connector configuration properties

The connector definition also contains the connector configuration properties. To tailor the properties accordingly:

1. Modify the Standard Properties as shown in Figure 23-2.

Standard Properties Connector-Specific Properties Supported Business Objects Trace/Log Fi			
	Property ▾	Value	
1	AdminInQueue	REDMAINT.ADMININQUEUE	
2	AdminOutQueue	REDMAINT.ADMINOUTQUEUE	
3	AgentTraceLevel	5	
4	ApplicationName	RMConnector	
5	BrokerType	VVMQI	
6	CharacterEncoding	ascii7	
7	ContainerManagedEvents		
8	DeliveryQueue	REDMAINT.DELIVERYQUEUE	
9	DeliveryTransport	JMS	
10	DuplicateEventElimination	false	
11	FaultQueue	REDMAINT.FAULTQUEUE	
12	jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory	
13	jms.MessageBrokerName	REDBROKER	
14	jms.NumConcurrentRequests	10	
15	jms.Password	*****	
16	jms.UserName		
17	Locale	en_US	
18	MessageFileName	c:\IBM\WebSphere\Adapters\connectors\messages\RMConnectorMessa	
19	PollEndTime	HH:MM	
20	PollFrequency	key	
21	PollStartTime	HH:MM	
22	RepositoryDirectory	c:\IBM\WebSphere\Adapters\repository	
23	RequestQueue	REDMAINT.REQUESTQUEUE	
24	ResponseQueue	REDMAINT.RESPONSEQUEUE	
25	RestartRetryCount	0	
26	RestartRetryInterval	1	
27	RFH2MessageDomain	mrm	
28	SynchronousRequestQueue	REDMAINT.SYNCHRONOUSREQUESTQUEUE	
29	SynchronousRequestTimeout	0	
30	SynchronousResponseQueue	REDMAINT.SYNCHRONOUSRESPONSEQUEUE	
31	WireFormat	CwXML	
32	XMLNamespaceFormat	long	

Figure 23-2 Customized standard properties

2. Table 23-2 lists the properties, which can differ depending on your broker choice.

Table 23-2 Properties with broker dependencies

Property	Notes
BrokerType	Select ICS / WMQI or WAS .
DeliveryTransport	JMS is the transport for WMQI and WAS, however it is a good idea to use JMS for ICS also.
RepositoryDirectory	For ICS, set this to remote. For WebSphere MQ Integrator and WebSphere Application Server, set this to C:\IBM\WebSphereAdapters\Repository.
WireFormat	For ICS, set this to CwBO. For WebSphere MQ Integrator and WebSphere Application Server, set this to CwXML.
XMLNamespaceFormat	Not used for ICS.

Important: Table 23-2 is not a comprehensive list of all of the properties that might differ between broker types. For more information, see the Standard Configuration Properties section in any of the delivered Adapter Guides.

3. Ensure that the queue names are 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 in Figure 23-3 and Table 23-3 on page 403. It is a good idea to use queue 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 RedMaintenance application and its connector.

REDMAINT.ADMININQUEUE	0	0	4
REDMAINT.ADMINOUTQUEUE	0	0	0
REDMAINT.DELIVERYQUEUE	12	0	0
REDMAINT.FAULTQUEUE	0	0	0
REDMAINT.REQUESTQUEUE	0	1	10
REDMAINT.RESPONSEQUEUE	0	0	0
REDMAINT.SYNCHRONOUSREQUESTQUEUE	0	0	0
REDMAINT.SYNCHRONOUSRESPONSEQUEUE	0	0	0

Figure 23-3 RMConnector queues

Table 23-3 Queues and usage

Queue Name	Usage
REDMAINT.ADMININQUEUE	Used by the broker to send administrative messages to the connector.
REDMAINT.ADMINOUTQUEUE	Used by the connector to send messages to the broker.
REDMAINT.DELIVERYQUEUE	Used by the connector to send business objects to the broker (JMS only).
REDMAINT.FAULTQUEUE	Used by the connector to forward messages if a fault occurs.
REDMAINT.REQUESTQUEUE	Used by broker to send business objects to the connector.
REDMAINT.RESPONSEQUEUE	Used for delivery of response messages from the Adapter Framework to the broker (ICS and JMS only).
REDMAINT.SYNCHRONOUSREQUESTQUEUE	Used for request messages from the Adapter Framework to the broker which require a synchronous response (JMS only).
REDMAINT.SYNCHRONOUS.RESPONSEQUEUE	Used for synchronous response messages from the broker to the Adapter Framework (JMS only).

4. There are other queues that are available which are used for JMS when specifying `ContainerManagedEvents`. This provides for guaranteed delivery over JMS. We will not be using these queues because they would override the `pollForEvents` that we want to use as part of our connector functionality.

You now enter any application-specific properties for the connector (Figure 23-4). These are the properties that uniquely define the information that is required to communicate with the application. If you are not the adapter developer, ask the developer which properties have been included in the adapter and the values that you need to specify for each.

Standard Properties			
Connector-Specific Properties			
Support			
	Property	Value	Encrypt
1	MaxConnection	1	<input type="checkbox"/>
2	CallbackBusObj		<input type="checkbox"/>
3	CallbackCollaboration		<input type="checkbox"/>
4	CallbackVerb		<input type="checkbox"/>
5	ApplicationHost	student	<input type="checkbox"/>
6	ApplicationRegisteredName	rm	<input type="checkbox"/>
7	CallbackPrimary	false	<input type="checkbox"/>

Figure 23-4 Application-specific properties

Note: Because this connector was not created from a template, the application-specific properties do not yet exist. Right-click **Properties** to add a new property.

5. Create the application-specific properties as shown in Figure 23-4. The properties that we require are:
 - The name of the host machine of the application and the RMI registered name of the application.
 - The maximum number of connections.
 - The Callback properties that relate to an ICS broker implementation only.
6. 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.
7. Save this connector definition to your project.
8. Save this connector definition to file. The file name is RMConnector.cfg in the directory that you previously created for the connector.

Note: As a standard property, there is also the MessageFileName property. Here, we write only the name of our message resource file. The agent will try to find it in the productDir\connector\message directory.

9. Copy the message resource file, `RMConnectorMessages.txt`, into this directory. It is in the Additional Materials in the RM folder.

23.3 Creating the WebSphere MQ objects

Using your favorite queue tool, you can create the required queues that are shown in Table 23-3 on page 403.

23.4 Preparing the connector's directory

The connector directory contains the run-time files for the connector. To prepare the connector directory:

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`, which is the same as the `ApplicationName` property in the connector configuration.

2. Move the 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`.)

23.5 Creating the startup script

A connector requires a startup script for the system administrator to start the connector process. The startup script that you use depends on the operating system on which you are developing your connector. Figure 23-5 shows the steps to start a connector on a Windows system.

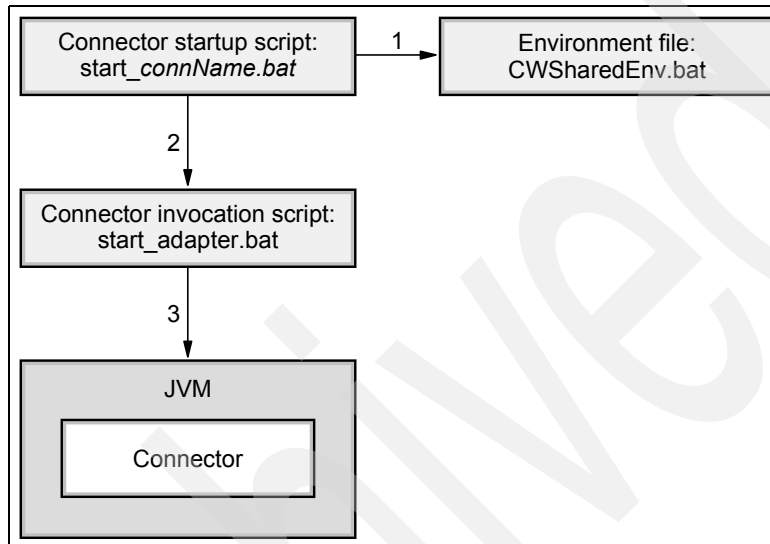


Figure 23-5 Starting a connector on a Windows System

23.5.1 Startup script for our connector

We have tailored a standard startup script for use with our custom adapter. Example 23-1 shows this startup script. Copy this script into the RMConnector directory. It is in the Additional Materials in the RM folder.

Example 23-1 RMConnector startup script

```
REM @echo off
call "%CROSSWORLDS%" \bin\CWConnEnv

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
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\xm14j.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%; %AG
ENT%

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%\%CONNAME%.jar;
AppEndWrapper
-1%CONNPACKAGENAME% -n%CONNAME%Connector -s%SERVER% %3 %4 %5

endlocal
pause
```

Note the following about our startup 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 23-4 specify.

Table 23-4 Environment variables in the connector startup script

Variable name	Value
ExtDirs	None
JCLASSES	Specify any application-specific jar files. Jar files are separated with a semicolon (;). See below
JVMArgs	None
LibPath	None

- Here we have our own environment variable of RED_MAINT for the library files for the back-end adapter API. Ensure that this variable is concatenated with the JCLASSES.
- We also specify the language-specific startup parameters required for a Java connector to specify connector-specific classes (CONNPACKAGENAME):

```
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 run-time (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 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 shown in Example 23-2.

Example 23-2 Adapter start

```
%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%\%CONNAME%.jar;
AppEndWrapper -l%CONNPACKAGENAME% -n%CONNAME%Connector -s%SERVER% %3 %4 %5
```

Note: 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 `RMConnector` directory with all of the other required files.

When we start the connector, we pass the following parameters:

- The name of connector definition: `-n (RM)`
- The name of the broker instance: `-s (WMQI_WAS)`
- The location of the local connector configuration file: `-c C:\IBM\WebSphereAdapters\connectors\RM\RMConnector.cfg`

23.6 Creating the Windows shortcut

The easiest way to start a connector running in a Windows environment is to have it running as a service. If you are running an ICS broker, the installer has a component that 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 also use a shortcut in the Start Programs folder.

To get started quickly and enable a small test:

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 target has our start parameters as follows:

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

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.

3. Before testing the start of the connector, check that 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

Also check in the messages folder of the connectors directory for the RMConnectorMessages.txt file.



Unit testing the connector

Before you deploy and build artifacts for the broker, you need to check that the adapter works correctly with our infrastructure. This chapter discusses how to test the interaction between the connector and the business objects as well as the interaction between the connector and the application.

24.1 Start the connector

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

The connector first parses 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 as follows:

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

After you have the connector started successfully, you run a quick unit test to verify that the connector can communicate with the application using your configuration.

24.1.1 Troubleshooting start errors

There are many things that can go wrong on start up. A log or trace file can be useful in such situations. A few common issues are:

1. Configuration file error as shown in Example 24-1.

Example 24-1 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.]
```

This error indicates that a configuration file of this name does not exist in the stated location. Check that the local configuration file has been saved to the correct directory with the correct name.

2. API error as shown in Example 24-2.

Example 24-2 API not found

```
[System: Server] [Thread: wbia_main (#1692490151)] [Msg:
java.lang.NoClassDefFoundError: com/ibm/its/rm/common/RMRMRegisterException
    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(AgentBusiness
ObjectManager.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)
at AppSide_Connector.AppEnd.<init>(AppEnd.java:117)
at AppSide_Connector.AppEnd.main(AppEnd.java:833)
at AppEndWrapper.main(AppEndWrapper.java:28)]
```

This error indicates that the piece that tells us about our API and how to communicate with the application is missing. Check that the common.jar file exists in the connector directory.

3. Queue Manager error as shown in Example 24-3.

Example 24-3 No connection to the Queue Manager

```
Can NOT create JMS connection to queue manager -- REDBROKER
[Type: Error] [MsgID: 9052] [Msg: 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'.]
```

This message indicates that 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. If the queue manager name is correct but it is not running, start the queue manager.

4. Invalid Queue error as shown in Example 24-4.

Example 24-4 No Queue connection

```
[Msg: Caught JMS error. Reason: javax.jms.InvalidDestinationException:
MQJMS2008: failed to open MQ queue REDMAINT.ADMININQUEUE.]
[System: ConnectorAgent] [SS: RMConnector] [Thread: wbia_main (#1692489475)]
[Type: Error] [MsgID: 9062] [Msg: JMS Exception Error code: MQJMS2008.
Exception message: com.ibm.mq.MQException: MQJE001: Completion Code 2, Reason
2085.]
[System: ConnectorAgent] [SS: RMConnector] [Thread: wbia_main (#1692489475)]
[Type: Error] [MsgID: 9052] [Msg: Unable to get a MQ series Queue Manager or
Queue Connection. Reason: failed to create sessionMQJMS2008.]
```

This error indicates that 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 exist and that they have the correct names.

Tip: If you are familiar with WebSphere MQ, the 2085 reason code is 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.

24.2 Testing the interaction between the connector and business objects

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

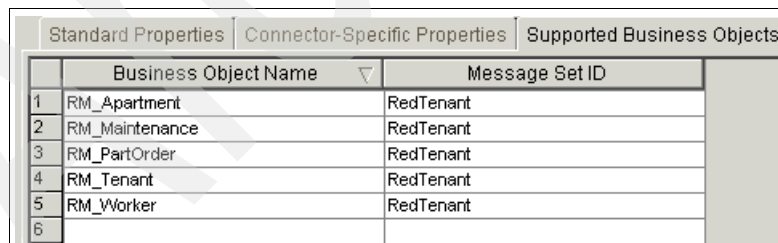
After you have verified that the interaction is working correctly, you will use the test data to actually send a request to the back-end application to verify that you get the anticipated response.

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

24.2.1 Adding supported business objects

To enable the Test Connector to send or receive business objects, add support for the business objects to the connector definition by following these steps:

1. Navigate to the System Manager.
2. Double-click **RMConnector** to open the Connector Configurator.
3. Select the Supported Business Objects tab, as shown in Figure 24-1.



	Business Object Name ▾	Message Set ID
1	RM_Apartment	RedTenant
2	RM_Maintenance	RedTenant
3	RM_PartOrder	RedTenant
4	RM_Tenant	RedTenant
5	RM_Worker	RedTenant
6		

Figure 24-1 Supported business objects

4. Add each of the RM_ business objects.
5. Add the message set identifier RedTenant to each business object.

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

At this point in the process, you 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 you determined, at Adapter Framework installation time, that you are using the Message Broker as your integration broker.

As 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 you start using the broker.

7. Stop the connector and restart it to check your business object support.
8. Errors similar to Example 24-5 might indicate that your business objects have not been saved to file in the correct directory. In this case, 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 24-5 Business object not found

[MSG: 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.

24.2.2 Starting the Test Connector

To start the Test Connector:

1. Start the Visual Test Connector by selecting **Start → Programs → IBM WebSphere Business Integration Adapters → Tools → Visual Test Connector**.
2. Create a profile that allows you to emulate the RMConnector by selecting **File → Create / Select Profile**, as shown in Figure 24-2.

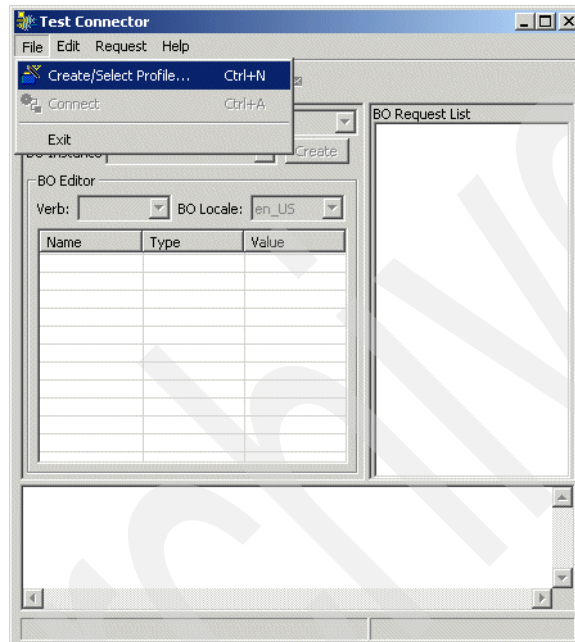


Figure 24-2 Select new profile

3. Select **File → New Profile**.
4. Enter the profile details as shown in Figure 24-3 on page 417.
 - Connector configuration file is the path to the saved configuration file for the RMConnector.
 - Connector name is RMConnector. This name is case-sensitive and must match exactly the ApplicationName connector property.
 - Broker type is WMQI.

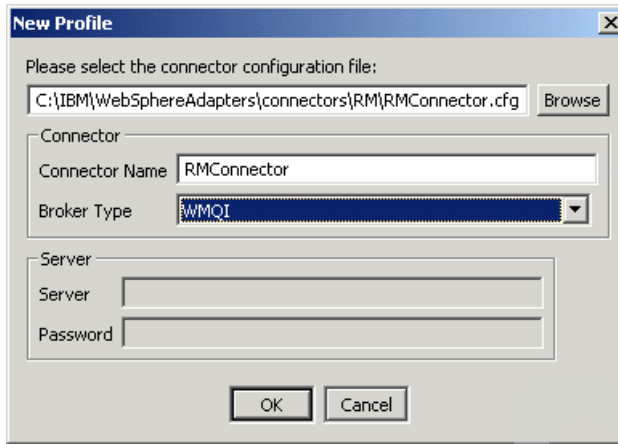


Figure 24-3 New profile

5. Click **OK**. The Connector Profile window displays, as shown in Figure 24-4.

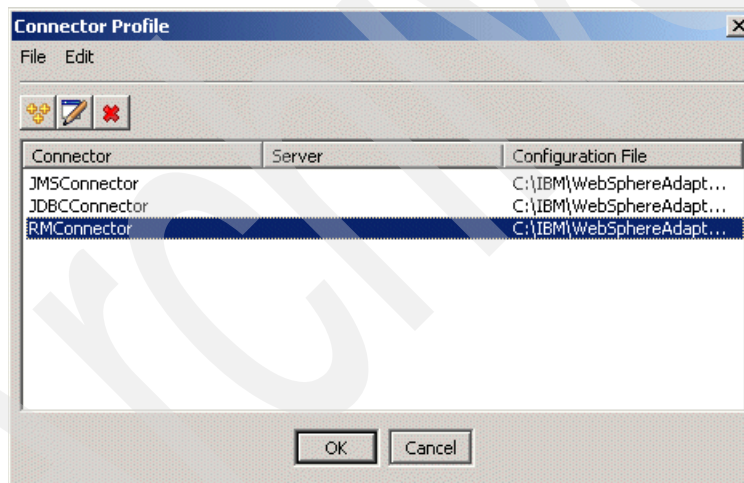


Figure 24-4 Select the new profile

6. Highlight the new profile for the RMConnector and click **OK**.
7. From the front screen, select **File** → **Connect**, as shown in Figure 24-5 on page 418.

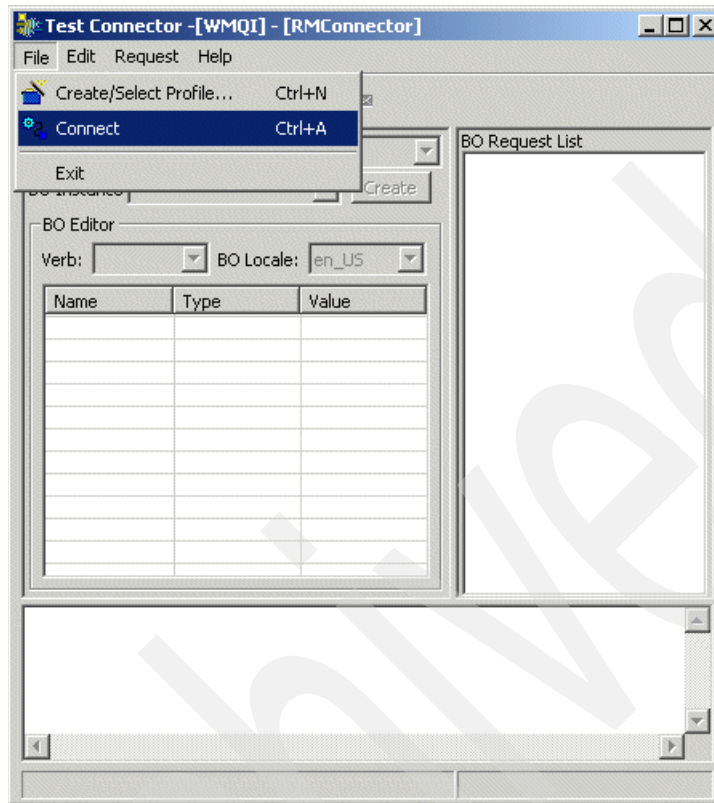


Figure 24-5 Connect

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

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

When the connector has initialized, you are able to check for your list of supported business objects and to prepare the test data.

24.2.3 Preparing test data for request processing (RM_Tenant.Retrieve)

To prepare the test data:

1. Select **RM_Tenant** from the Business Object Type drop-down list, as shown in Figure 24-6.

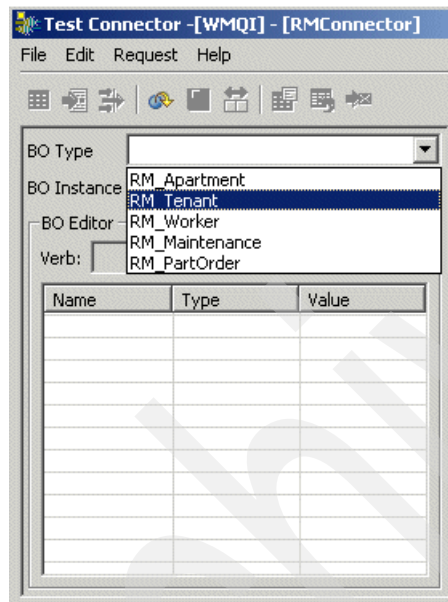


Figure 24-6 Select a business object

2. Click **Create**.
3. Give this new instance a name, as shown in Figure 24-7. The name is not important.

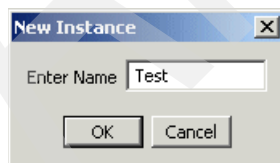


Figure 24-7 Name business object instance

4. Click **OK**.

You are presented with a template business object to fill with data for testing.

- Using the drop-down list, select **Retrieve**, as shown in Figure 24-9 on page 421. Retrieving data is the first test.

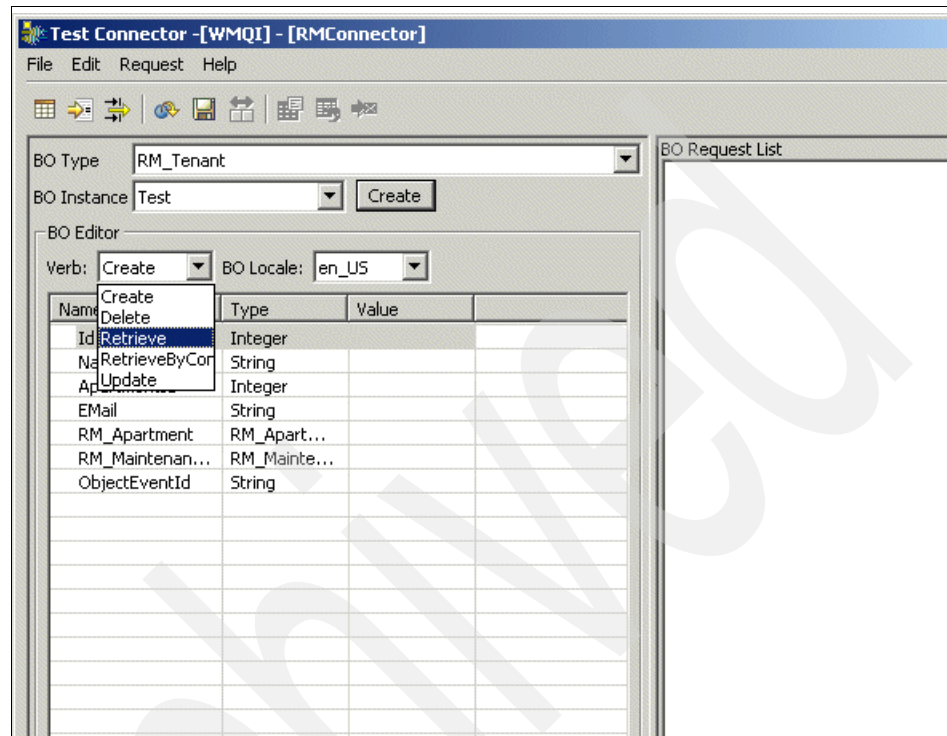


Figure 24-8 Set verb

6. Enter a tenant Id. If you are using our sample data, 100 is a tenant Id that we created for testing.

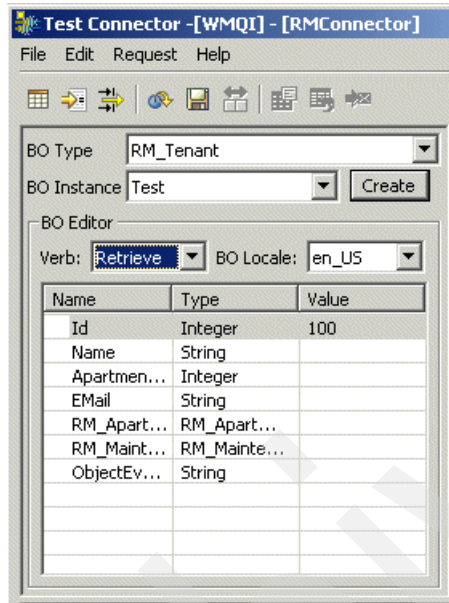


Figure 24-9 Enter tenant Id

Tip: There are several ways that you can test from this point forward. We have chosen to use a method using `rfhuti1`. When you have set up your test data, you have a reusable test scenario. You will send this business object to the connector. However, because of the JMS messaging configuration, the message will not really go to the queue that you need for sending in a request. The message will go to the `DELIVERYQUEUE`. Using `rfhuti1` allows you to capture the message, complete with all of the correct MQMD and JMS information, and replay it to the correct location.

With this testing method, you can perform initial tests for the connector without another feeder connector, a broker, or a connected application.

7. Select **Request** → **Send**, as shown in Figure 24-10, to send the business object.

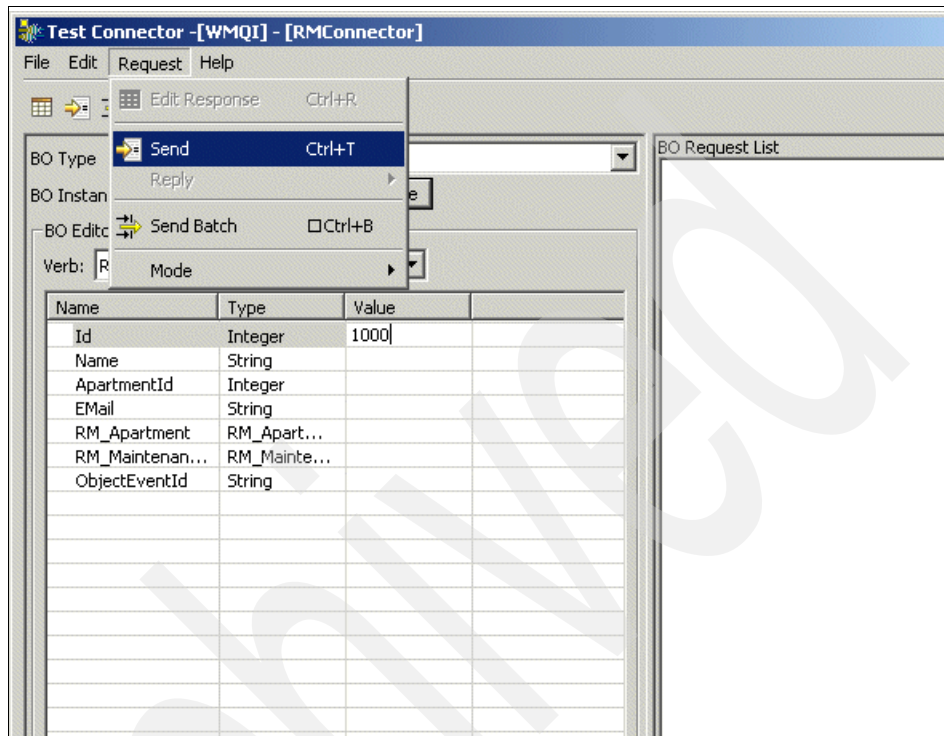


Figure 24-10 Sending the request

8. Start an instance of **rfhutil**, which is included in the Additional Materials.
9. If you are using our sample configuration, enter the queue manager **REDBROKER** and the queue **REDMAINT.DELIVERYQUEUE**.

10. Select **Read Q** to read the message from the queue (Figure 24-11). Do not use browse because you do not need to retain this message for future use.

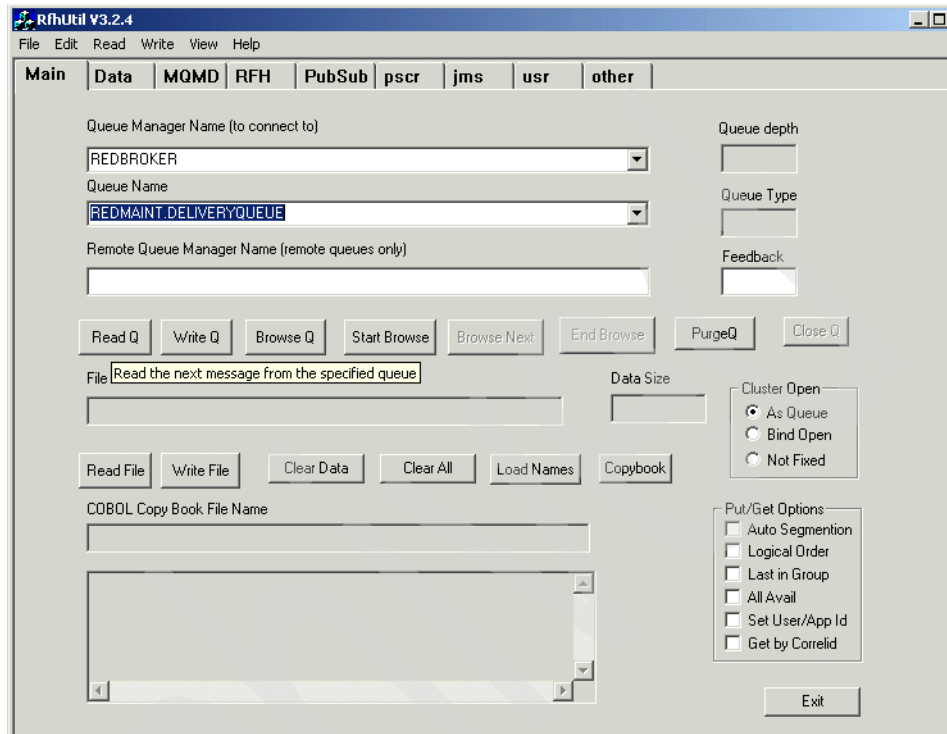


Figure 24-11 Select correct queue

11. Select the Data tab (see Figure 24-12), and select **XML Data Format**.

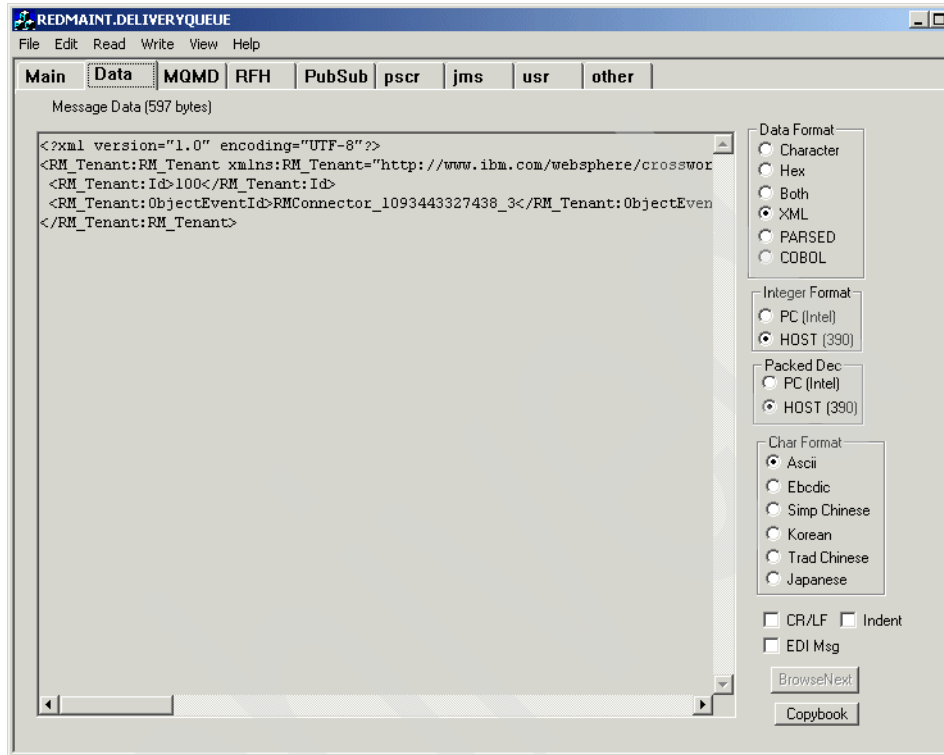


Figure 24-12 Business object message data

12. The business object is ready to be sent. Scroll to the right and verify that the verb is Retrieve.

13. Select the MQMD tab (see Figure 24-13) to see the correctly formatted message descriptor.

The screenshot shows the 'REDMAINT.DELIVERYQUEUE' application window with the 'MQMD' tab selected. The window contains various input fields and controls for configuring a message descriptor. The 'MQ Message Format' is set to 'MQHRF2', 'User Id' is 'db2admin', 'Code Page' is '819', and 'Backout Count' is '0'. The 'Put Date/Time' is '2004/08/19 11:25:10.29' and 'Expiry' is '-1'. The 'MsgType' is set to a dropdown menu. The 'Id Display' section has radio buttons for 'Ascii', 'Ebcidic', and 'Hex', with 'Ascii' selected. The 'Int Fmt' section has radio buttons for 'PC', 'Host', and 'Unix', with 'Unix' selected. The 'PD Fmt' section has radio buttons for 'PC' and 'Host', with 'Host' selected. The 'Offset' is '0' and 'Seq No' is '1'. The 'Group' section has radio buttons for 'No', 'Yes', and 'Last', with 'No' selected. The 'Segment' section has checkboxes for 'Yes' and 'Last', both of which are unchecked. The 'Report Opt' section has checkboxes for 'Except', 'Expire', 'COA', and 'COD', each with radio buttons for 'No' and 'Yes', and checkboxes for 'Data' and 'Full'. The 'PAN', 'NAN', and 'Pass Correl' checkboxes are also unchecked. The 'Persistent Msg' section has radio buttons for 'No', 'Yes', and 'As Queue', with 'Yes' selected. The 'Reset Ids' and 'Copy Msg Id to Correl Id' buttons are present. The 'Accounting Token' field contains a long alphanumeric string.

REDMAINT.DELIVERYQUEUE

File Edit Read Write View Help

Main Data **MQMD** RFH PubSub pscr jms usr other

MQ Message Format User Id Code Page Backout Count

MQHRF2 db2admin 819 0

Put Date/Time Expiry MsgType

2004/08/19 11:25:10.29 -1

Message ID

AMQ REDBROKER WL#A

Correl ID

Group Id

Application Identity

Application Origin Application Type

Put Application Name

ereAdapters\jre\bin\java.exe

Reply To Queue Manager

REDBROKER

Reply To Queue

Accounting Token

16010515000000A837D66532621F2A07E53B2BE9030000000000000000

Id Display

☒ Ascii ☐ Ebcidic ☐ Hex

Int Fmt

☐ PC ☐ Host ☒ Unix

PD Fmt

☐ PC ☒ Host

Offset

0

Seq No

1

Group

☒ No ☐ Yes ☐ Last

Segment

☐ Yes ☐ Last

Report Opt

Except ☒ No ☐ Yes ☐ Data ☐ Full

Expire ☒ No ☐ Yes ☐ Data ☐ Full

COA ☒ No ☐ Yes ☐ Data ☐ Full

COD ☒ No ☐ Yes ☐ Data ☐ Full

☐ PAN ☐ NAN ☐ Pass Correl

Persistent Msg

☐ No ☒ Yes ☐ As Queue

Reset Ids

Copy Msg Id to Correl Id

Figure 24-13 Message descriptor

14. Select the RFH tab (see Figure 24-14) to see the correct RFH header that shows the correct information for the JMS.

The screenshot shows the 'REDMAINT.DELIVERYQUEUE' application window with the 'RFH' tab selected. The window has a menu bar (File, Edit, Read, Write, View, Help) and a tab bar (Main, Data, MQMD, RFH, PubSub, pscr, jms, usr, other). The RFH tab contains several configuration sections:

- RFH V2 Fixed Data:**
 - Length: 340
 - RFH Encoding: Integer (PC, Host, Unix) - Unix is selected.
 - Pack Dec: PC, Host - Host is selected.
 - Data Format: MQSTR
 - Code Page: 1208
 - Flags: 0
 - CCSID: 1208
- Message Domain:** mrm
- Msg Set:** RedTenant
- Msg Type:** RM_Tenant
- Output Format:** CwXML
- V2 Folders:** mod, jms, usr (checked); PubSub, pscr, other (unchecked).
- V1 Fixed Data:**
 - Length: 0
 - Encoding: Integer (PC, Host, Unix) - PC is selected.
 - Pack Dec: PC, Host - PC is selected.
 - Data Format: (empty)
 - Code Page: 0
 - Flags: 0
- Appl Group:** (empty)
- Format Name:** (empty)
- V1 Contents:** mod, PubSub, Resp (all unchecked).
- RFH Type:** None, Version 1, Version 2 (selected), Both.

Figure 24-14 RFH header

15. On the RFH header tab, note the following:

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

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

16. Save this message by selecting **Write File**, as shown in Figure 24-15.

The great thing about using **rfhutil** is that when you save the message, you are also saving the message header information for later use.

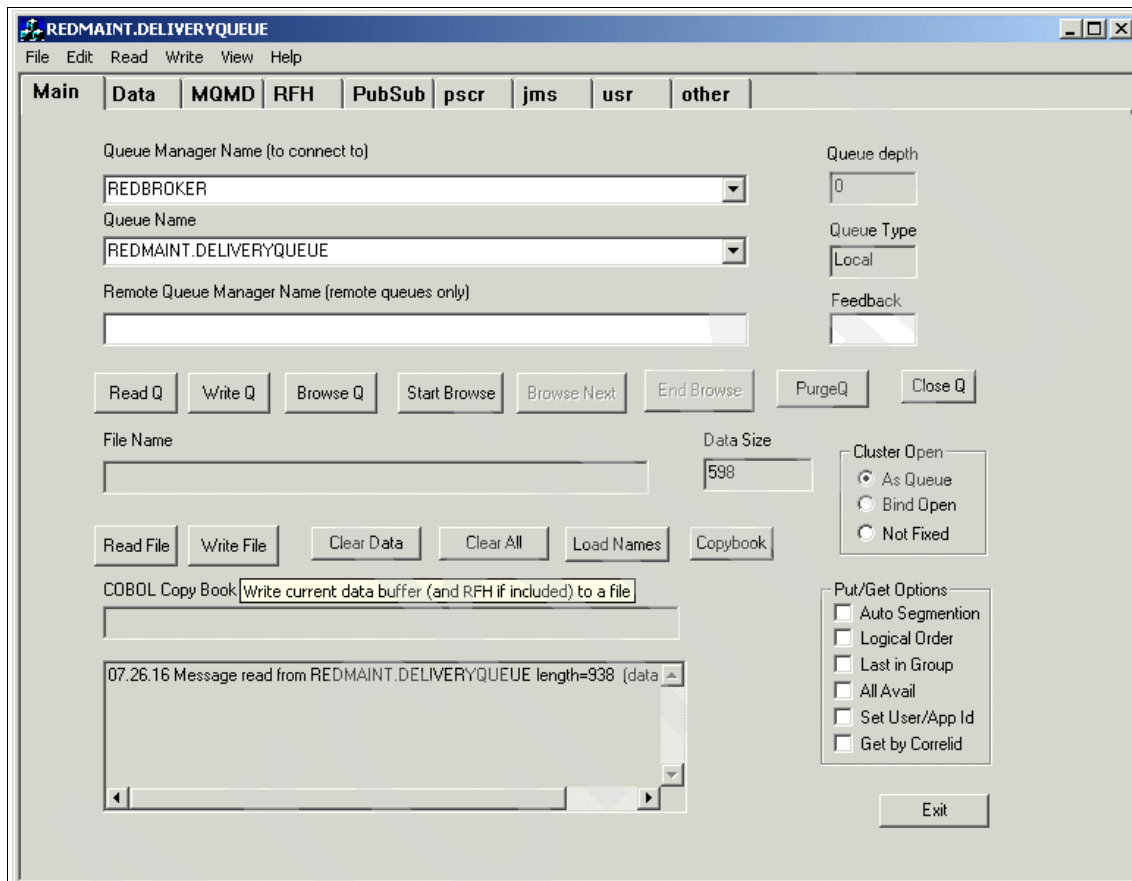


Figure 24-15 Save message

Note: A word of caution — you cannot edit this file with a text editor because it will mess up the headers.

You now have a message, complete with the header information that is required by the connector, that you can send to emulate an inbound request to the connector.

24.2.4 Sending test data to the Test Connector (RM_Tenant.Retrieve)

To send test data to the connector:

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 you created previously.
4. Select **Write Q** (Figure 24-16) to put a message into the queue.

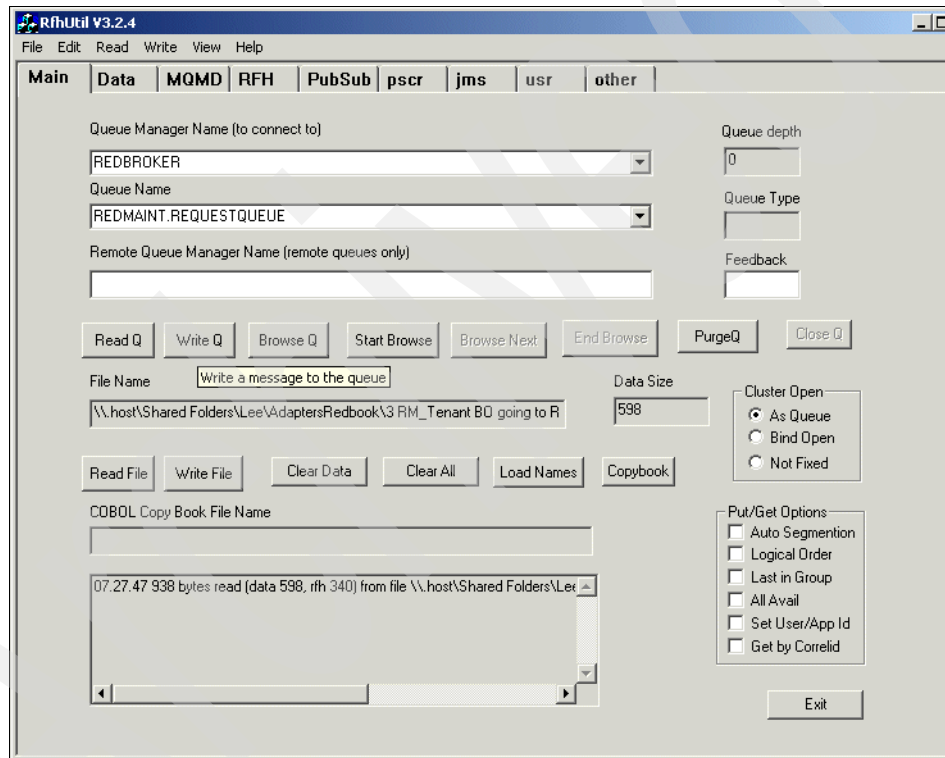


Figure 24-16 Write Q

5. Go back to the Test Connector DOS window.

You will see the arrival of the RM_Tenant.Retrieve business object as shown in Example 24-6. You might have to scroll back quite a bit.

Example 24-6 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
    ApartmentId = CxIgnore
    EMail = CxIgnore
<StartChild>
    RM_Apartment = CxBlank
<StartBO:RM_Apartment>
<EndBO:RM_Apartment>
<EndChild>
<StartChild>
    RM_Maintenance = CxBlank
<StartBO:RM_Maintenance>
<EndBO:RM_Maintenance>
<EndChild>
    ObjectEventId = RMConnector_1093461239688_2

<EndBO:RM_Tenant>
```

6. Go to the Test Connector GUI screen. You will see the arrival of the business object in the Business Object Request List on the right, as shown in Figure 24-17.



Figure 24-17 Business object request list

24.2.5 Sending a response to the request (RM_Tenant.Retrieve)

To send a test response:

1. Double-click the business object to open it.
2. Fill in some details for the RM_Tenant object.
3. Right-click the **RM_Apartment** object and select **Add Instance**, as shown in Figure 24-18.

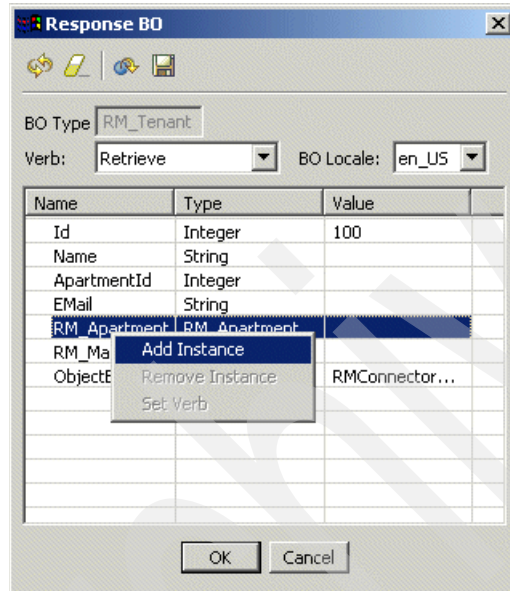


Figure 24-18 Response business object

4. Add some apartment data.
5. Right click the **RM_Maintenance** object and select **Add Instance**.
6. Add some maintenance data (see Figure 24-19 on page 431).

Response BO

BO Type:

Verb: BO Locale:

Name	Type	Value
Id	Integer	100
Name	String	Me
ApartmentId	Integer	1
EMail	String	ME@my.com
<input checked="" type="checkbox"/> RM_Apartment	RM_Apartment	
Id	Integer	1
ApartmentNumber	Integer	100
AddressLine1	String	MyHouse
AddressLine2	String	
AddressLine3	String	
AddressLine4	String	
PostCode	String	5021 2JN
ObjectEventId	String	
<input checked="" type="checkbox"/> RM_Maintenance[n]	RM_Maintenance	
<input checked="" type="checkbox"/> [0]	RM_Maintenance	
Id	Integer	1
ApartmentId	Integer	1
Status	String	A
TenantId	Integer	100
ProblemDescription	String	blah blah blah
StatusDescription	String	
ExpectedCompletion	String	
ActualCompletion	String	
ObjectEventId	String	
ObjectEventId	String	RMConnector...

OK Cancel

Figure 24-19 Completed response business object

7. When you have completed the response data, click **OK**.

8. Select **Request** → **Reply** → **Success**, as shown in Figure 24-20.

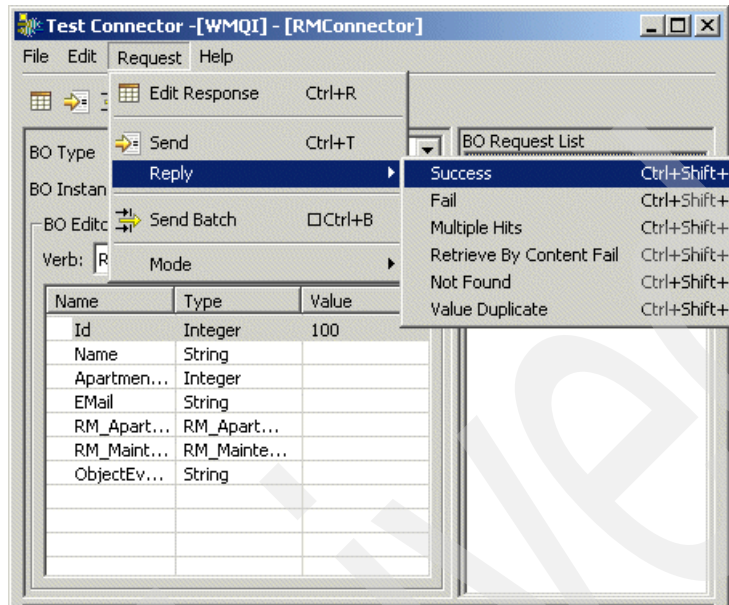


Figure 24-20 Send response

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

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

Although the connector has processed the response message successfully, the request message did not contain the information that the connector needs to know where to send the response Business Object Message Object (bomo). In other words, there is no reply to queue destination.

10. Go back to **rftut11** for the REQUESTQUEUE.

11. Select the MQMD tab. Enter the Reply to Queue Manager as REDBROKER and Reply to Queue as REDMAINT.RESPONSEQUEUE, as shown in Figure 24-21. This queue delivers the response message to the Brokers.

The screenshot shows the RfhUtil V3.2.4 application window with the MQMD tab selected. The interface includes a menu bar (File, Edit, Read, Write, View, Help) and a tabbed interface with tabs: Main, Data, MQMD, RFH, PubSub, pscr, jms, usr, and other. The MQMD tab contains the following fields and controls:

- MQ Message Format: []
- User Id: []
- Code Page: 437
- Backout Count: 0
- Put Date/Time: []
- Expiry: 0
- MsgType: []
- Id Display: ☒ Ascii, ☐ Ebdic, ☐ Hex
- Int Fmt: ☒ PC, ☐ Host, ☐ Unix
- PD Fmt: ☒ PC, ☐ Host
- Message ID: []
- Correl ID: []
- Group Id: []
- Offset: []
- Seq No: 0
- Group: ☐ No, ☐ Yes, ☐ Last
- Segment: ☐ Yes, ☐ Last
- Application Identity: []
- Application Origin: []
- Application Type: []
- Put Application Name: []
- Reply To Queue Manager: REDBROKER
- Reply To Queue: REDMAINT.RESPONSEQUEUE
- Accounting Token: []
- Report Opt:
Except: ☐ No, ☐ Yes, ☐ Data, ☐ Full
Expire: ☐ No, ☐ Yes, ☐ Data, ☐ Full
COA: ☐ No, ☐ Yes, ☐ Data, ☐ Full
COD: ☐ No, ☐ Yes, ☐ Data, ☐ Full
☐ PAN, ☐ NAN, ☐ Pass Correl
- Persistent Msg: ☒ No, ☐ Yes, ☐ As Queue
- Reset Ids: []
- Copy Msg Id to Correl Id: []

Figure 24-21 Enter reply to queue

12. Send the message again.

13. Complete a new response business object as before (see Figure 24-22).

The dialog box is titled "Response BO". It has a toolbar with icons for undo, redo, and save. Below the toolbar, there are two dropdown menus: "BO Type" set to "RM_Tenant" and "Verb" set to "Retrieve". To the right of the "Verb" dropdown is a "BO Locale" dropdown set to "en_US".

Name	Type	Value
Id	Integer	100
Name	String	Me Again
ApartmentId	Integer	1
Email	String	MeAgain@my.com
<input checked="" type="checkbox"/> RM_Apartment	RM_Apartment	
Id	Integer	1
Apartmen...	Integer	1
AddressLi...	String	House
AddressLi...	String	Street
AddressLi...	String	
AddressLi...	String	
PostCode	String	99999
ObjectEv...	String	
<input checked="" type="checkbox"/> RM_Mainten...	RM_Maintenance	
<input checked="" type="checkbox"/> [0]	RM_Maintenance	
Id	Integer	1
Apart...	Integer	1
Status	String	A
TenantId	Integer	100
Proble...	String	Blah Blah Blah Blah
Status...	String	
Expect...	String	
Actual...	String	
Object...	String	
ObjectEventId	String	RMConnector_1093461239...

At the bottom of the dialog box are "OK" and "Cancel" buttons.

Figure 24-22 New response

14. Send a reply of Success.

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

```
[Mesg: :sending the bomo -- CxCommon.Messaging.BusObjMsgObject@5999b71e to
the replyToQueue -- queue://REDBROKER/REDMAINT.RESPONSEQUEUE]
```

16. Go back to the instance of `rfhut11` that is looking at the RESPONSEQUEUE.
17. Select **Read Q**.
18. Select the Data tab and select **XML** Data Format.

At this point, you are expecting the message data to be a correct XML schema representation of our business object, similar to that shown in Figure 24-23.

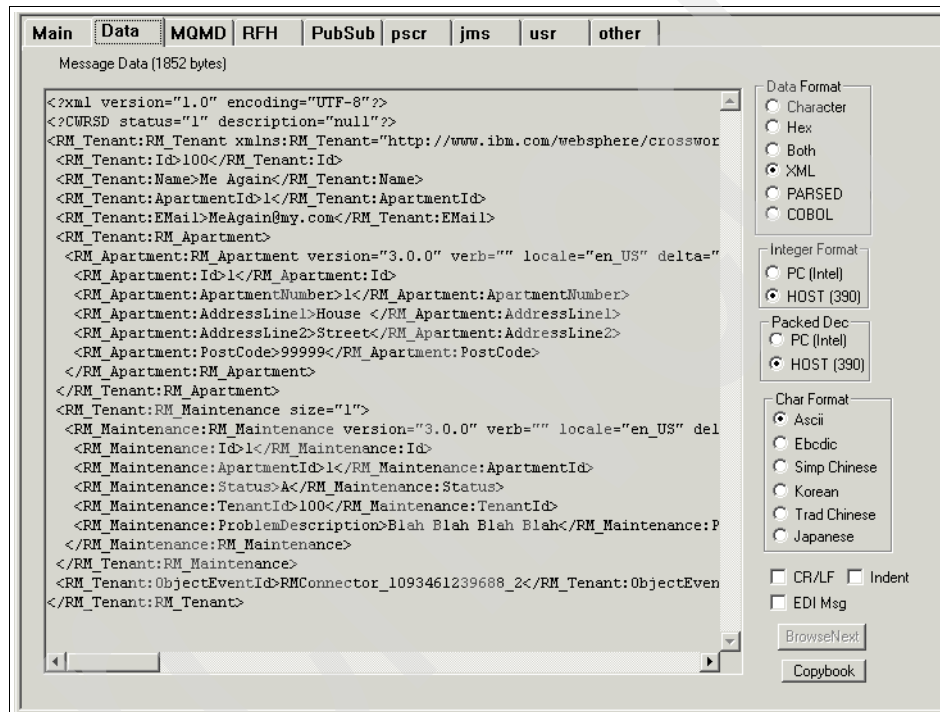
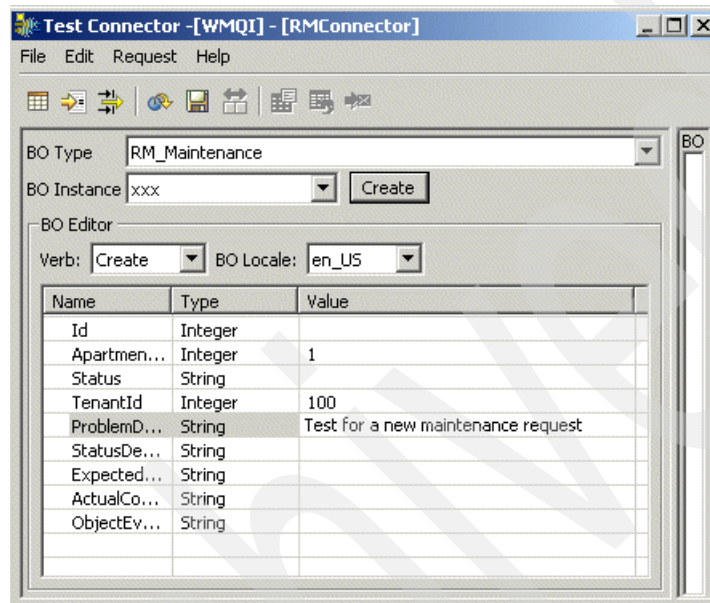


Figure 24-23 Response message

24.2.6 Preparing test data for processing (RM_Maintenance.Create)

To prepare the test data for processing:

1. Using the method described previously, create a test message to use with an RM_Maintenance, as shown in Figure 24-24.



Test Connector - [WMQI] - [RMConnector]

File Edit Request Help

BO Type: RM_Maintenance

BO Instance: xxx

BO Editor

Verb: Create BO Locale: en_US

Name	Type	Value
Id	Integer	
Apartmen...	Integer	1
Status	String	
TenantId	Integer	100
ProblemD...	String	Test for a new maintenance request
StatusDe...	String	
Expected...	String	
ActualCo...	String	
ObjectEv...	String	

Figure 24-24 Test business object for create

2. Send this business object.
3. Save the message on the DELIVERYQUEUE to a file for future use.

24.2.7 Sending test data (RM_Maintenance.Create)

To send the test data:

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 24-25).

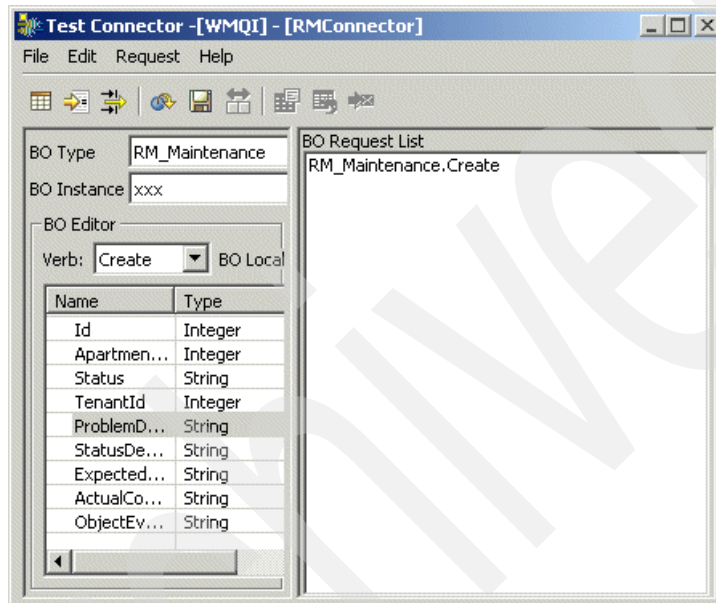


Figure 24-25 Request list

3. Open the request business object.
4. Enter data that emulates a response from the application (see Figure 24-26 on page 438). 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.

Response B0

BO Type:

Verb: BO Locale:

Name	Type	Value
Id	Integer	4
ApartmentId	Integer	1
Status	String	A
TenantId	Integer	100
ProblemDesc...	String	Test for a ne...
StatusDescri...	String	
ExpectedCo...	String	
ActualCompl...	String	
ObjectEventId	String	RMConnector...

OK Cancel

Figure 24-26 Response to be sent

5. Send this response as a SUCCESS reply.
6. Check the Test Connector log for the response message that is to be sent, as shown in Example 24-7.

Example 24-7 Test Connector log

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

7. Go to `rfhutil` and check the contents of the message on the `RESPONSEQUEUE` (see Figure 24-27).

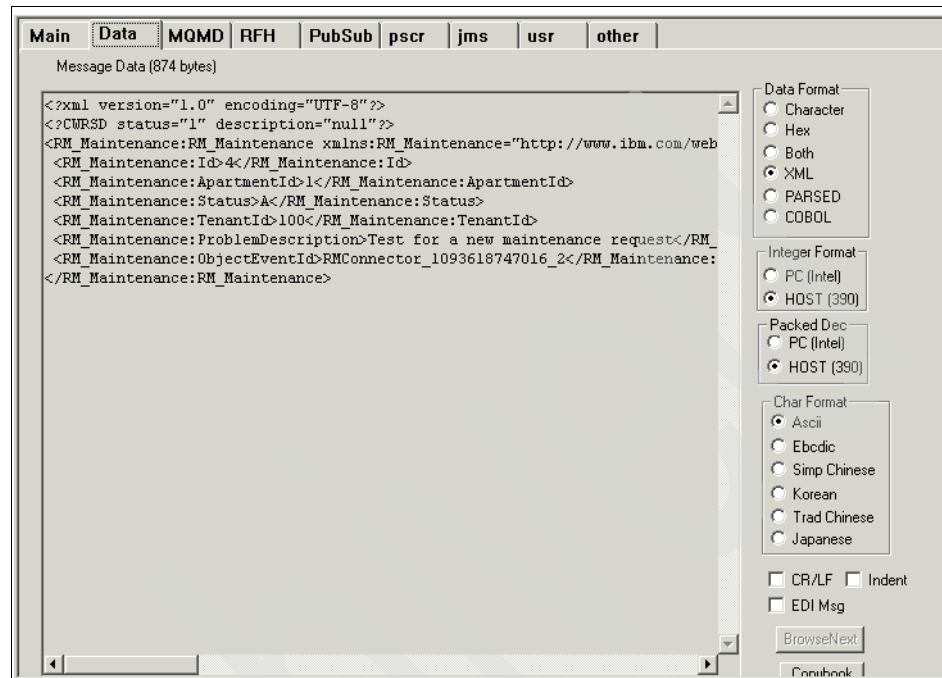


Figure 24-27 Returned create message

24.3 Testing the interaction between the connector and the application

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

Essentially, you will repeat the tests that are described in 24.2, “Testing the interaction between the connector and business objects” on page 414. This time, however, you will use the real connector so that you can interact with the real back-end application.

24.3.1 Sending a request business object (RM_Tenant.Retrieve)

To send a request business object to the RM_Tenant.Retrieve:

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 `rfhut11` as before. Remember to add the ReplyToQueue details to the Message Descriptor.
5. Check the RMConnector log for the details of the business object that is sent, which should be similar to that shown in Example 24-8.

Example 24-8 Response business object

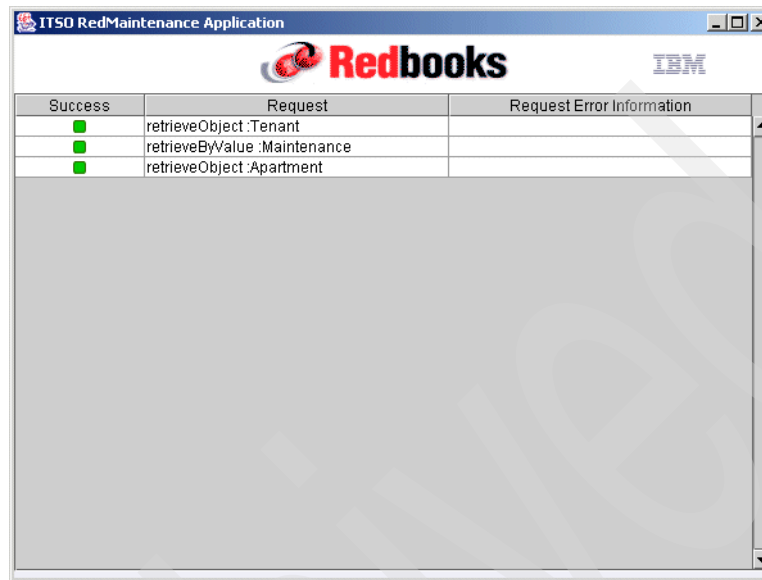
```
[Mesg: <Version = 3.0>]
[Mesg: <EndHeader>]
[Mesg: <StartBO:RM_Tenant>]
[Mesg:     BusinessObject = RM_Tenant]
[Mesg:     Verb = Retrieve]
[Mesg:     Locale = en_US]
[Mesg:     Id = 100]
[Mesg:     Name = Lee Gavin]
[Mesg:     ApartmentId = 1]
[Mesg:     EMail = leegavin@uk.ibm.com]
[Mesg: <StartChild>]
[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 = S021 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:      TenantId = 100]
[Mesg:      ProblemDescription = I have a leaking tap]
[Mesg:      StatusDescription = New part installed]
[Mesg:      ExpectedCompletion = Nov 1, 2004]
[Mesg:      ActualCompletion = Nov 4, 2004]
[Mesg:      ObjectEventId = CxIgnore]
[Mesg: <EndBO:RM_Maintenance>]
[Mesg: <StartBO:RM_Maintenance>]
[Mesg:      BusinessObject = RM_Maintenance]
[Mesg:      Verb = CxBlank]
[Mesg:      Locale = en_US]
[Mesg:      Id = 2]
[Mesg:      ApartmentId = 1]
[Mesg:      Status = P]
[Mesg:      TenantId = 100]
[Mesg:      ProblemDescription = Crack in the front window]
[Mesg:      StatusDescription = Glass delivered, size incorrect. Re-ordered.]
[Mesg:      ExpectedCompletion = Jun 28, 2005]
[Mesg:      ActualCompletion = Jan 11, 2005]
[Mesg:      ObjectEventId = CxIgnore]
[Mesg: <EndBO:RM_Maintenance>]
[Mesg: <StartBO:RM_Maintenance>]
[Mesg:      BusinessObject = RM_Maintenance]
[Mesg:      Verb = CxBlank]
[Mesg:      Locale = en_US]
[Mesg:      Id = 3]
[Mesg:      ApartmentId = 1]
[Mesg:      Status = P]
[Mesg:      TenantId = 100]
[Mesg:      ProblemDescription = High speed internet not working]
[Mesg:      StatusDescription = Cable company contacted, modem to be replaced.]
[Mesg:      ExpectedCompletion = Feb 1, 2005]
[Mesg:      ActualCompletion = Jan 10, 2005]
[Mesg:      ObjectEventId = CxIgnore]
[Mesg: <EndBO:RM_Maintenance>]
[Mesg: <EndChild>]
[Mesg:      ObjectEventId = RMConnector_1093461239688_2]
[Mesg: <EndBO:RM_Tenant> ]

```

6. Check the application log to ensure that the application is reporting successful processing (see Figure 24-28).



Success	Request	Request Error Information
■	retrieveObject :Tenant	
■	retrieveByValue :Maintenance	
■	retrieveObject :Apartment	

Figure 24-28 Application log

24.3.2 Checking the response message received

To check the response message, use `rfhuti1` to read the message that was sent to the REDMAINT.RESPONSEQUEUE. If you are using our test data, the message should be similar to that shown in Figure 24-29.

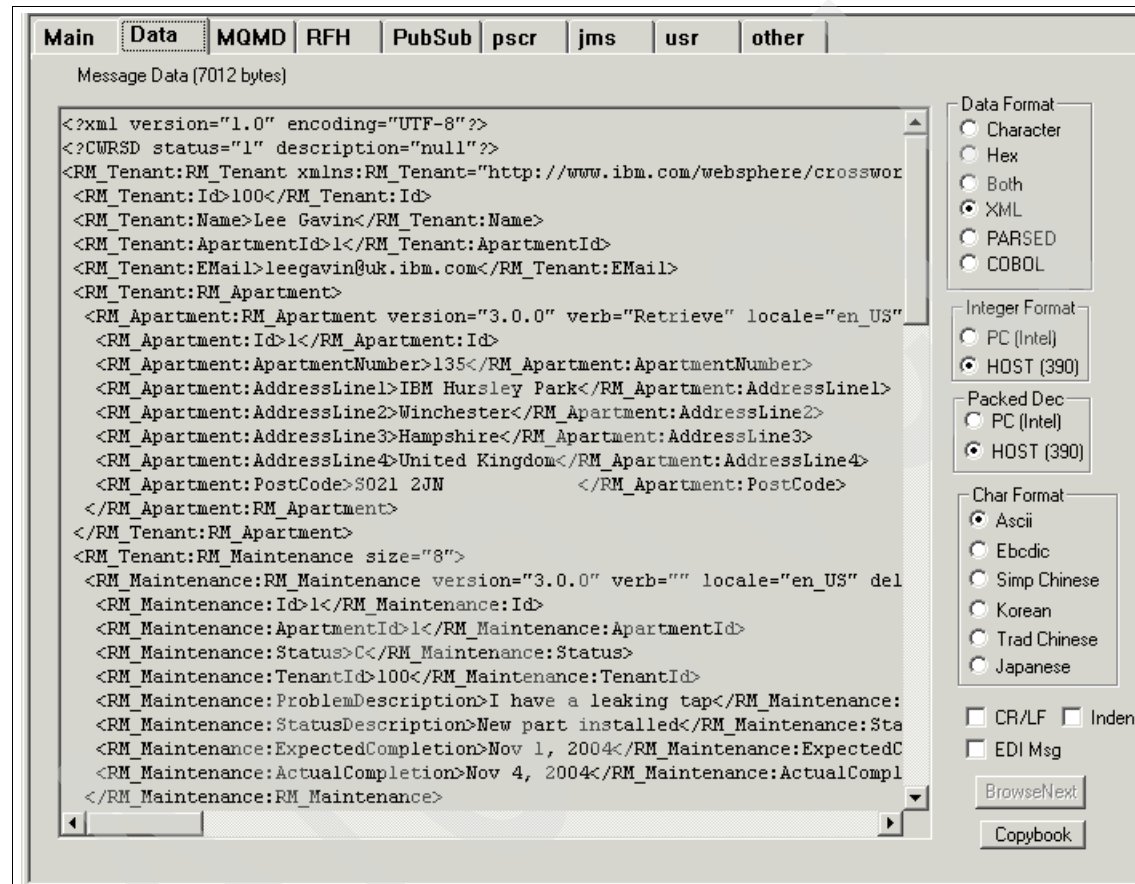


Figure 24-29 Response message

24.3.3 Sending a request business object (RM_Maintenance.Create)

The send the request business object:

1. Using the file from the RM_Maintenance.Create test, put a message on to the REDMAINT.REQUESTQUEUE using `rfhuti1`. Remember to add the ReplyToQueue details to the Message Description.
2. Check the log of the RMConnector to see the details of the business object that is being sent.

3. Verify that the message on the RESPONSEQUEUE contains the newly created maintenance record by checking that there is a new maintenance record in the database with the same key (see Figure 24-30).

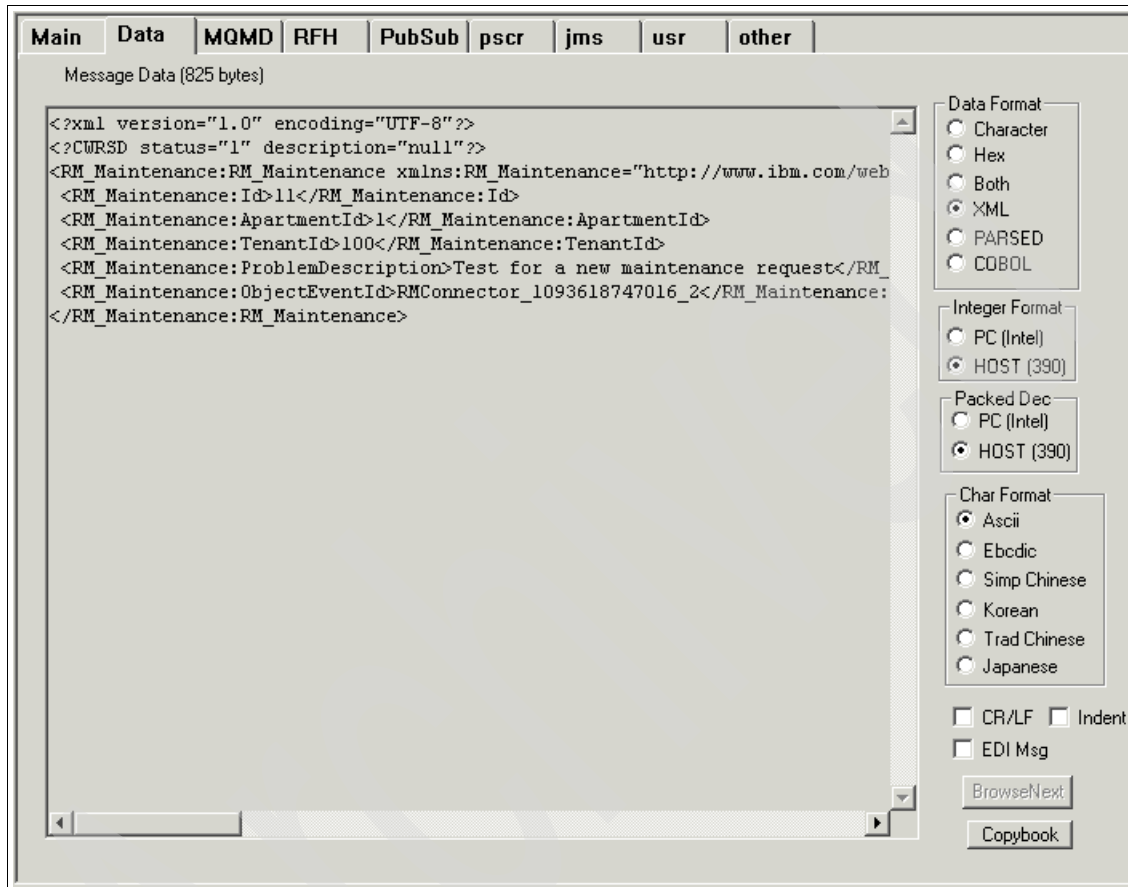


Figure 24-30 Maintenance create success

24.3.4 Creating an 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 system to manipulate the event. For example, when building the event store, the adapter calls out to the back-end system to ask for events, which are returned one at a time. In the GUI, you see these calls as multiple calls to retrieveObject. The adapter loops for a predefined number of events, which are 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 system to archive and delete events. These events are again in the GUI.

To create the event, send a create request for a new maintenance record. This action causes an event record to be written to the database by the back-end application. To create the event use your favorite DB2 tool to verify that the event record exists in the database. In our example, we used the DB2 Command Center. The event object matches the key value of the newly created maintenance record.

In our example:

- ▶ The database name is SG246345.
- ▶ The table name is itso.events.

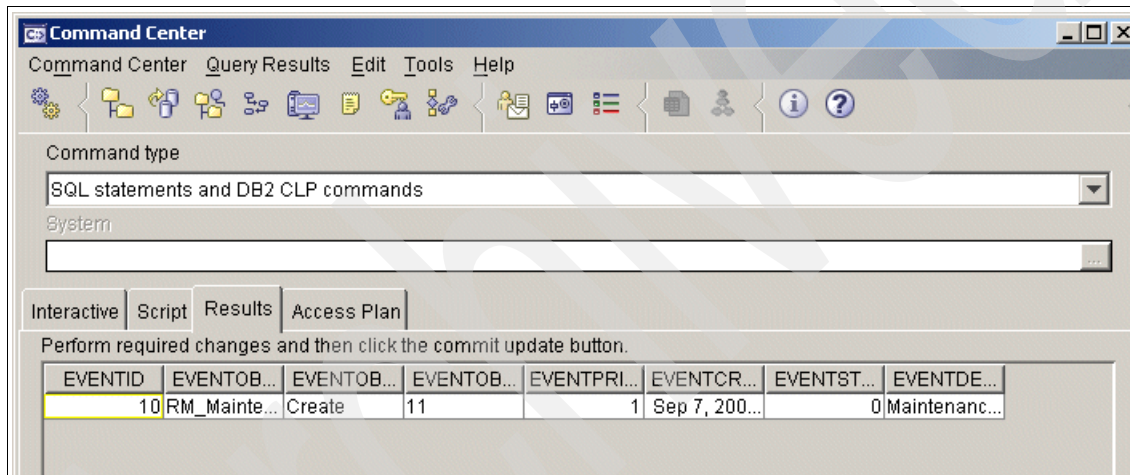


Figure 24-31 Event record

24.3.5 Checking the event message that is sent

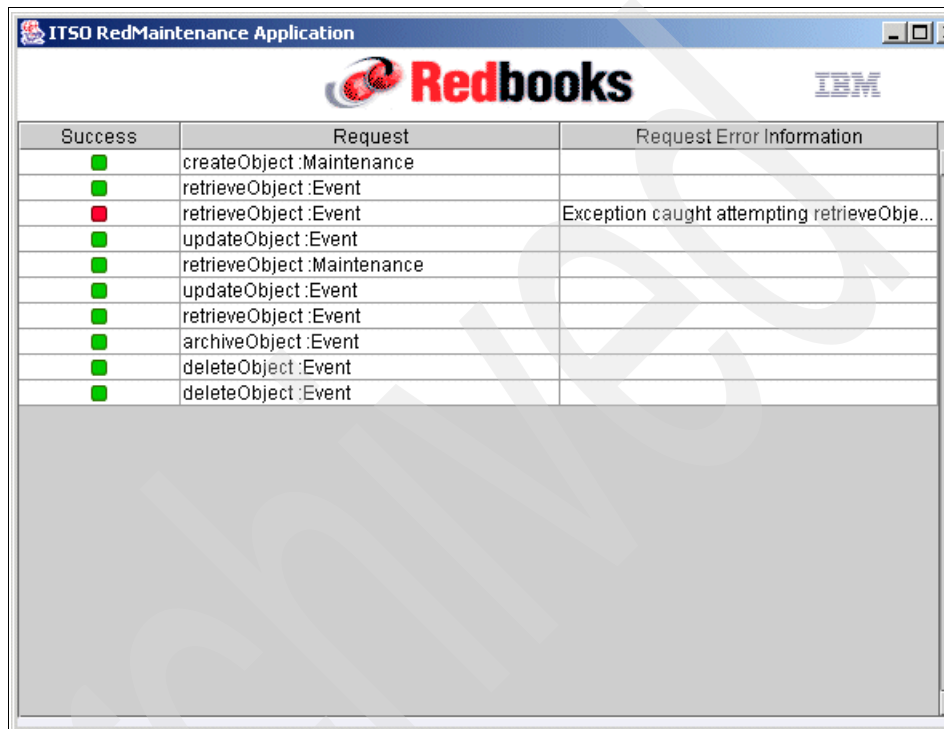
One of the connector configuration properties that controls the polling for events is PollFrequency. This value determines how often the connector polls for new events. In our example, our configured value for this property is key because it enables us to control the polling while we test. Thus, the connector only polls when we enter p in the connector command window.

The message at connector startup reads as follows:

[Mesg: Press p to poll application.]

To poll while testing:

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, as shown in Figure 24-32.



Success	Request	Request Error Information
■	createObject :Maintenance	
■	retrieveObject :Event	
■	retrieveObject :Event	Exception caught attempting retrieveObje...
■	updateObject :Event	
■	retrieveObject :Maintenance	
■	updateObject :Event	
■	retrieveObject :Event	
■	archiveObject :Event	
■	deleteObject :Event	
■	deleteObject :Event	

Figure 24-32 Application log

Check the application log. As mentioned earlier, the connector polls for a predetermined number of events. The GUI log records the following events:

1. Indicates that the new event object is retrieved when one exists.
2. When it cannot find any more new events, an error is 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 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, you can check that the message for the Create event has been delivered to the DELIVERYQUEUE by the connector, as shown in Figure 24-33.

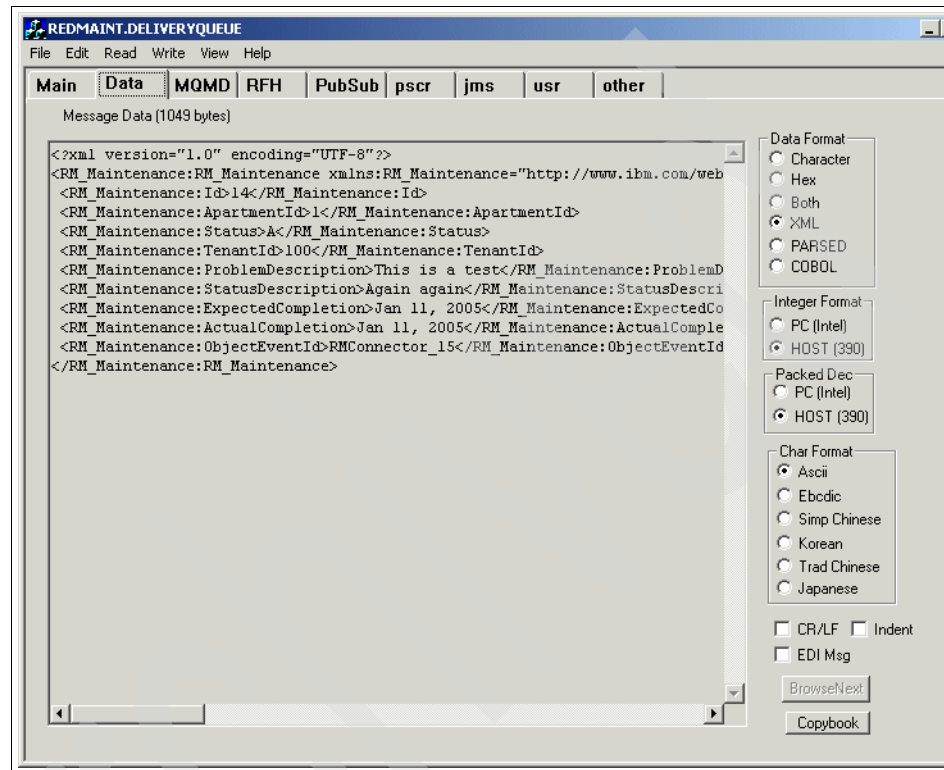


Figure 24-33 New maintenance event

- You should also double-check the archive events table to see that the archival process has completed successfully. This table is `itso.archive_events`, as shown in Figure 24-34.

Command type

SQL statements and DB2 CLP commands

System

Interactive

Script

Results

Access Plan

Perform required changes and then click the commit update button.

EVENTID	EVENTOBJ...	EVENTOBJECTVERB	EVENTOBJECTKEY	EVENTPRI...	EVENTCR...	EVENTAR...	EVENTST...	EVENTDE...
1	RM_Mainte...	Create	4	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
2	RM_Mainte...	Create	5	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
3	RM_Mainte...	Create	6	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
4	RM_Mainte...	Create	7	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
5	RM_Mainte...	Create	8	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
6	RM_Mainte...	Create	9	1	Aug 26, 20...	Aug 26, 20...	3	Maintenanc...
10	RM_Mainte...	Create	11	1	Sep 7, 200...	Sep 7, 200...	3	Maintenanc...

Figure 24-34 Archived events

24.4 Conclusions on unit testing

As you can see, we have been able to test and verify the base functionality of the custom adapter without having to create message sets or message flows for a Message Broker. All we used are the tools that are 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 run-time 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.



Part 5

Message Broker components

Archived

Deploying business objects to the Message Broker

This chapter discusses the steps that are involved in deploying the business objects from the System Manager to the Message Broker Toolkit. These business objects are required for our scenario. When the business objects are deployed, you can then migrate these message sets from the Message Broker Toolkit to the run-time Broker.

You can import a business object definition from the System Manager and repository to the Message Broker by:

- ▶ Using a tool from the System Manager.
- ▶ Using the command line utility `mqsicreatemsgsets`.
- ▶ Using a schema importer from the Message Broker Toolkit.

Each of these ways has advantages, and it is possible to use a combination of more than one method.

25.1 Importing front-end business objects

This section explains how to import the front-end business object definitions. These business objects are the most complex when you look at them, so they are a good place to start.

25.1.1 Defining the Message Set

To define the Message Set:

1. Open the Message Broker Toolkit using the desktop shortcut.
2. Select the Broker Application Development Perspective.
3. Select **File** → **New** → **Message Set Project**. The New Message Set Project window displays, as shown in Figure 25-1.
4. Enter a Project Name of WebTenant.
5. Select **Next**.

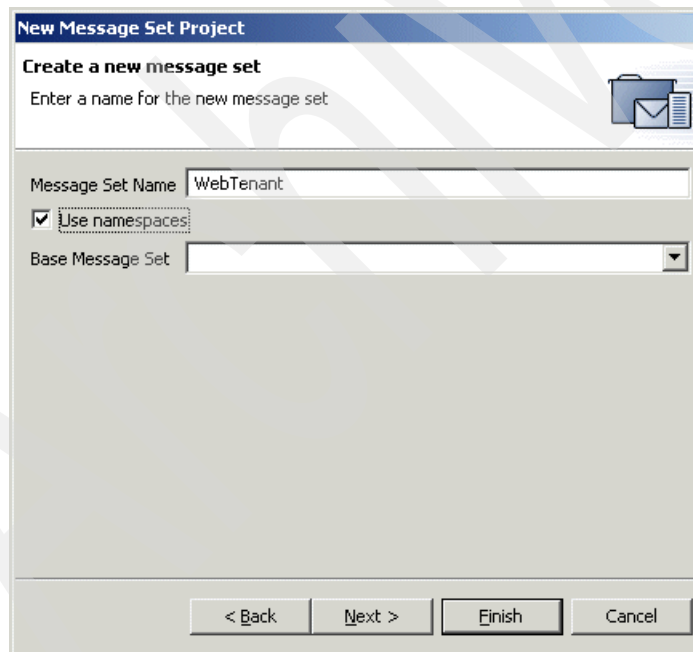


Figure 25-1 Message Set name

6. Enter WebTenant as the Message Set Name. This name matches the supported business object information in the connector configuration.

7. Select **Use namespaces**.
8. Select **Next**. The window in Figure 25-2 displays.
9. Select **XML Wire Format Name**.
10. Enter CwXML as the wire format name, which matches the connector configuration property.

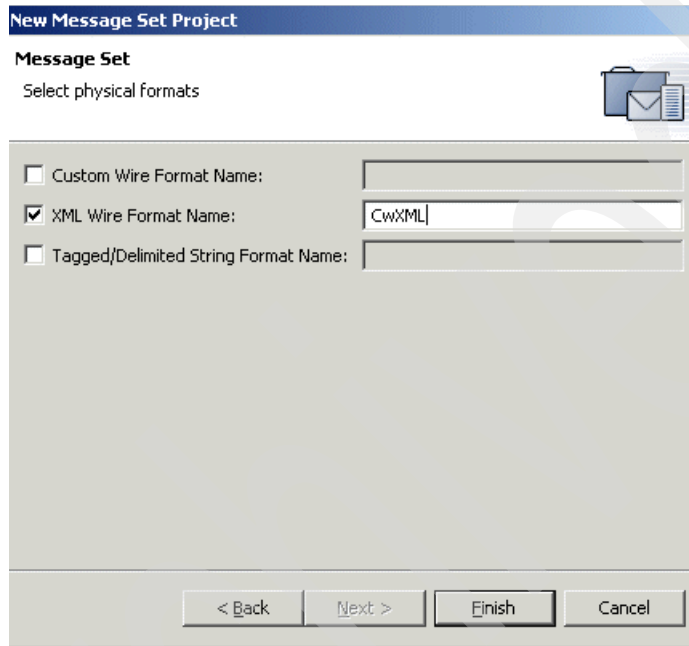


Figure 25-2 Wire format

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 25-3 on page 454:
 - Deselect **Suppress XML Declaration**.
 - Select **Suppress DOCTYPE**.
 - Remove **mrm** as the Root Tag Name.
14. Save the Message Set by pressing Ctrl+S.

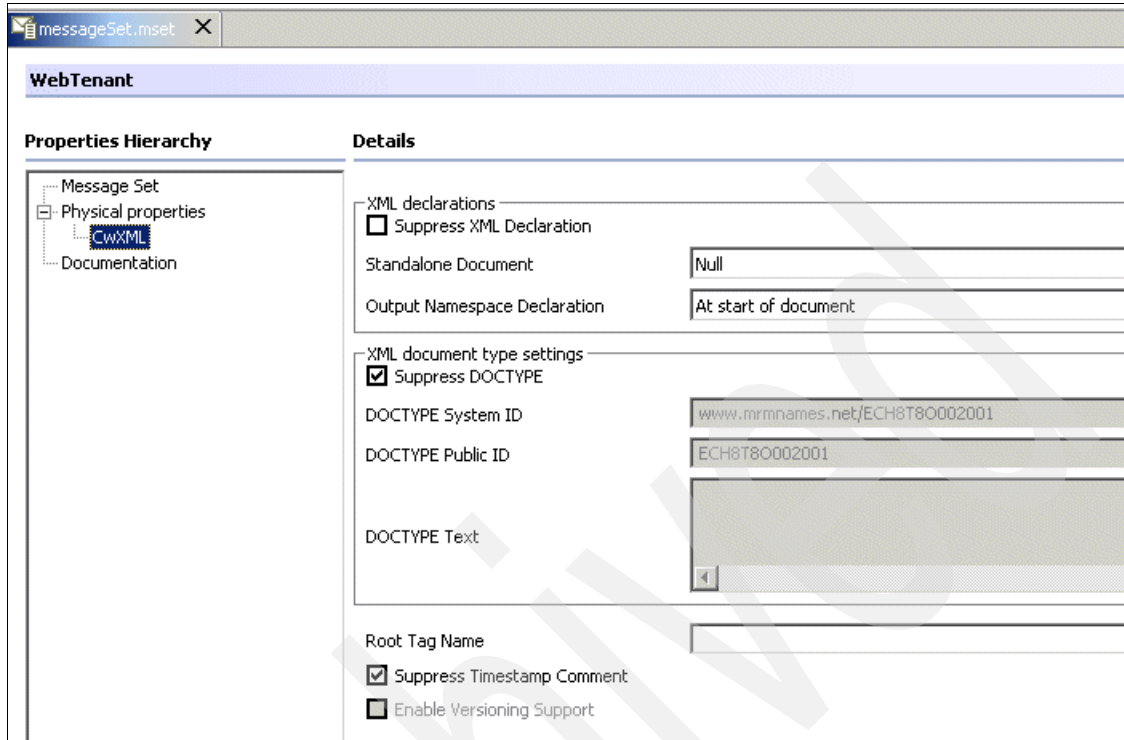


Figure 25-3 Format requirements

15. Save the Message Set definition and close the .mset file.
16. Close the Message Broker toolkit.

25.1.2 Importing schema definitions and creating message definitions

You can now import the schema files from the business object repository and create message definitions from these schemas, by following these steps:

1. Open the System Manager to set our preferences for the Importer.
2. Select **Window** → **Preferences**.
3. Select **System Manager Preferences** → **Broker Preferences**.
4. Set the preferences as shown in Figure 25-4 on page 455.
 - Message Broker importer path
C:\IBM\WBIMB501\eclipse\mqsicreatemsgdefs.exe

- Message Broker workspace directory
C:\IBM\WBIMB501\eclipse\workspace

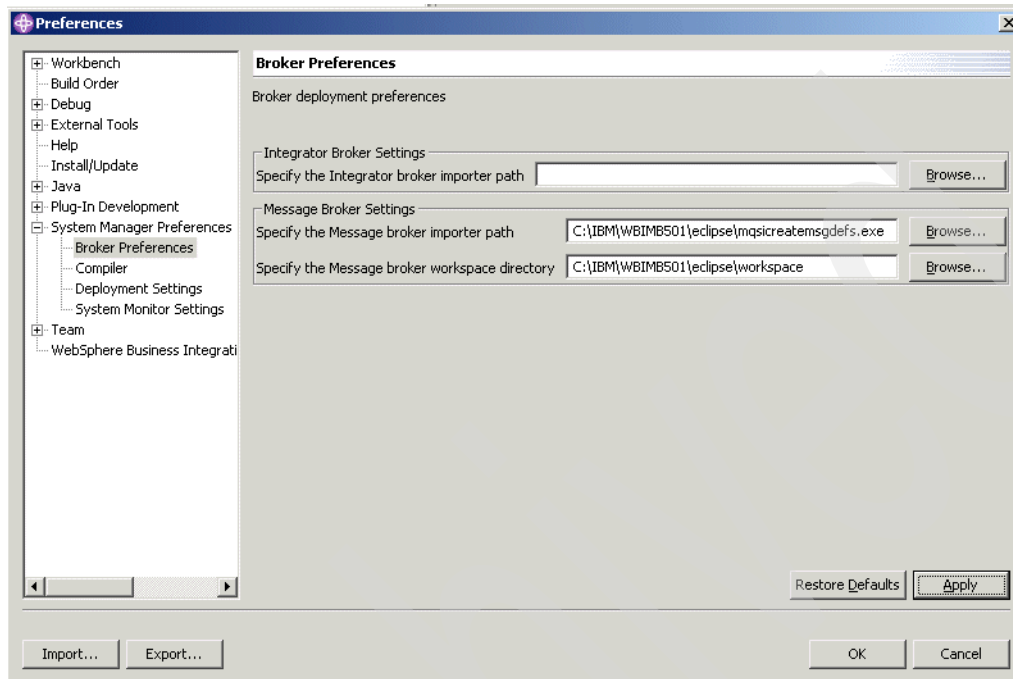


Figure 25-4 Set preferences

5. Click **Apply**.

The completed artifacts can only be deployed to an Integration Broker from the appropriate type of User Project. You now need to create a User Project for deployment, as shown in Figure 25-5.

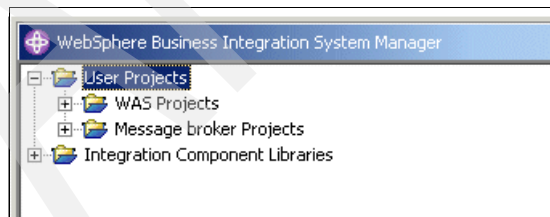


Figure 25-5 User Projects

6. From the System Manager, expand the **User Projects** folder.

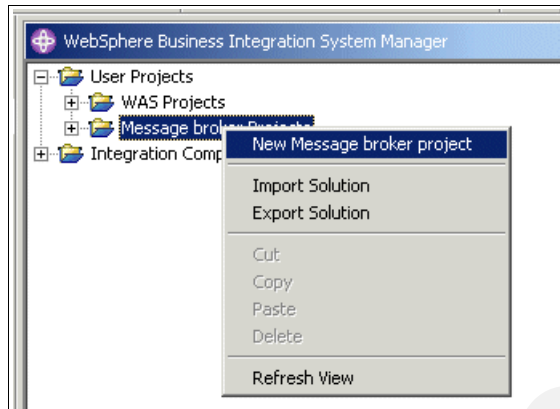


Figure 25-6 Message Broker Projects

7. Right-click **Message Broker Projects** and select **New Message Broker project** (Figure 25-7).

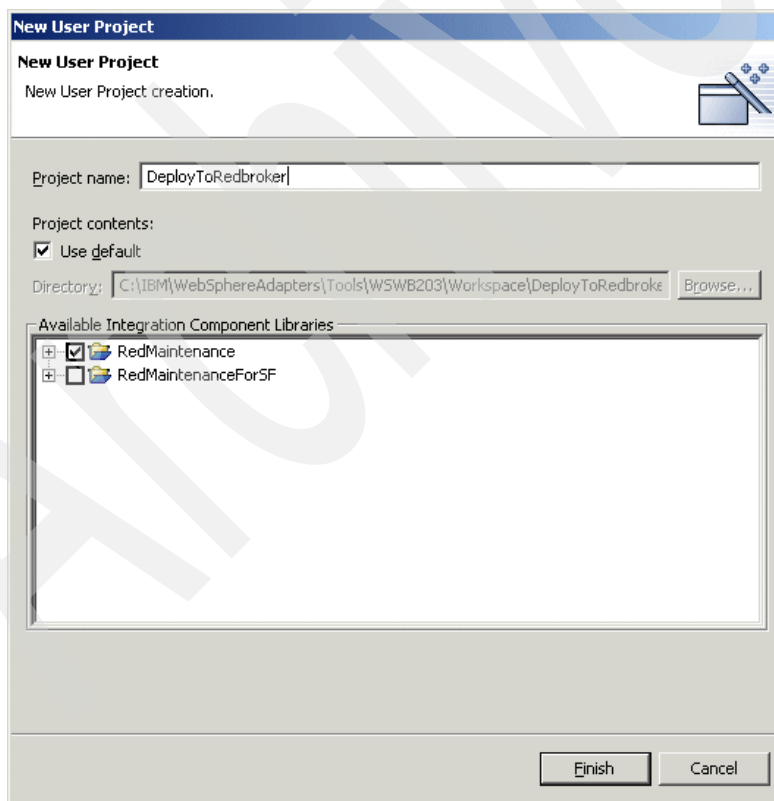


Figure 25-7 Populate project

8. Name the new project. We called ours DeployToRedbroker.
9. Click **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.
10. Click **Finish**.
11. Right-click the **new User Project**. In our case, it is **DeployToRedbroker**.
12. Select **Deploy to Message Broker workspace** (Figure 25-8).

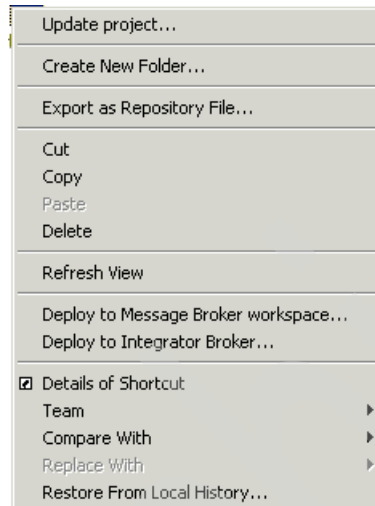


Figure 25-8 Deploy

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

13. Select all of the business objects, the renamed Web_objects, for the front-end application (Figure 25-9 on page 458).

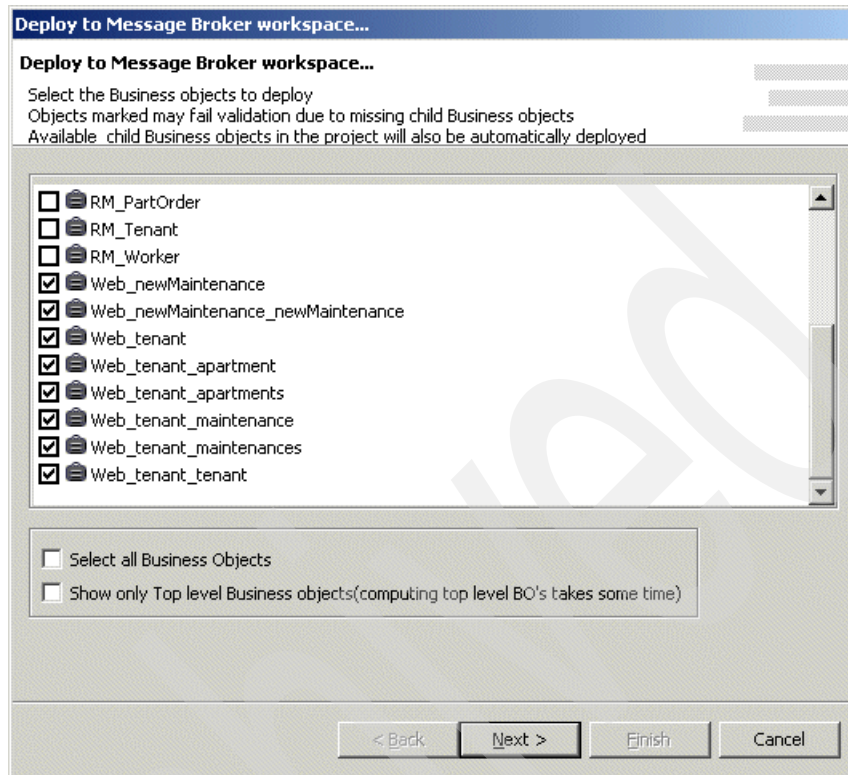


Figure 25-9 Select business objects

14. Click **Next**.

15. Enter the parameters for the import (see Figure 25-10 on page 459).

- Message set name is WebTenant.
- Select **Deploy in verbose mode**, for logging.
- Select **long** namespace support.

Select the parameters
Select the destination project name and parameters

Select the Destination Project

Enter the Message Set project name: WebTenant

Enter a base Message set project(optional):

Enter the base Message set(optional):

Select the parameters

☐ Replace existing project with the same name

☐ Namespace aware(Only applicable to new projects)

☒ Deploy in verbose mode

long Please select the xml namespace format
(Connectors in the source project will be configured accordingly).

< Back Next > Finish Cancel

Figure 25-10 Enter parameters

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 WebSphere Business Integration Adapters installation directory and is named mqsicreatemsgdefs.report.txt. Example 25-1 on page 460 shows a sample of a successful import.

Example 25-1 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

- You can now open the messages to see the imported message definitions. Figure 25-11 shows the `Web_tenant` message.



22. Add a namespace prefix for each of the business object schemas, as shown in Figure 25-12, on the CwXML physical format on the message set.

Namespace settings	
Namespace URI	Prefix
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant	Web_tenant
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment	Web_tenant_apartment
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartments	Web_tenant_apartments
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance	Web_tenant_maintenance
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenances	Web_tenant_maintenances
http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant	Web_tenant_tenant

Figure 25-12 Namespace settings

23. Save the Message Set. We chose to set the prefixes to the same as the business object names.

25.2 Importing back-end business objects

For the back-end business objects, you need to:

1. Enter a Message Set project name of RedMaintenance.
2. Enter a Message Set name of RedTenant.

Note: These two names are not the same. This is intentional.

3. Set the CwXML properties the same as for 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.

6. Select all of the business objects for your RedMaintenance application, as shown in Figure 25-13.

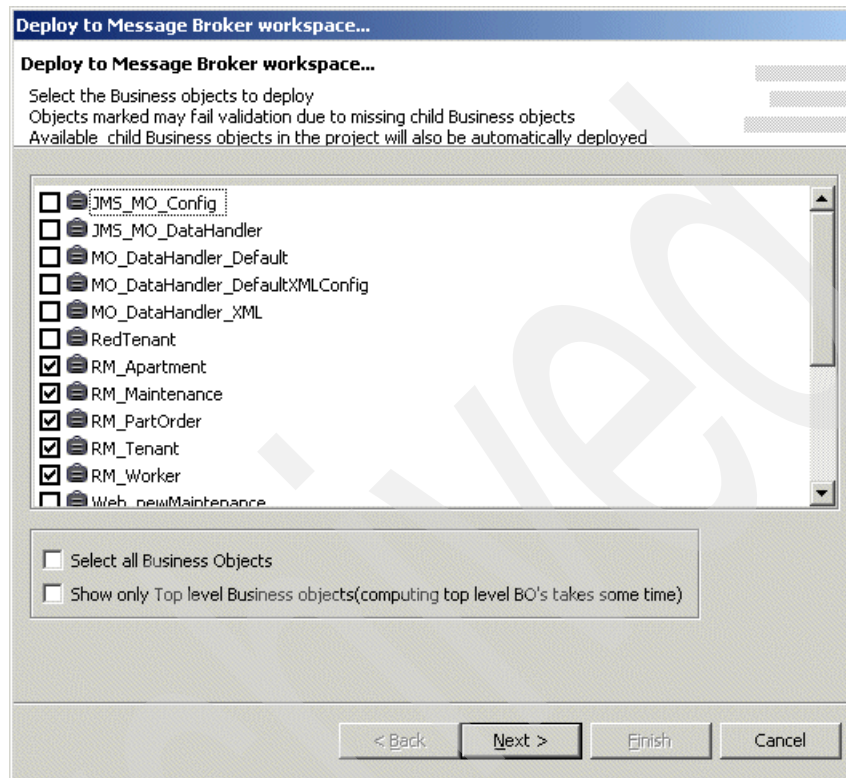


Figure 25-13 Select RM objects

7. The destination project is the RedMaintenance project, as shown in Figure 25-14.

The screenshot shows a Windows-style dialog box titled "Select the parameters". Below the title bar, the text "Select the destination project name and parameters" is displayed. The dialog is divided into two main sections. The first section, "Select the Destination Project", contains three text input fields: "Enter the Message Set project name:" (with "RedMaintenance" entered), "Enter a base Message set project(optional):", and "Enter the base Message set(optional):". The second section, "Select the parameters", contains four options: "Replace existing project with the same name" (unchecked), "Namespace aware(Only applicable to new projects)" (unchecked), "Deploy in verbose mode" (checked), and a dropdown menu for "Please select the xml namespace format" (set to "long"). A note below the dropdown states: "(Connectors in the source project will be configured accordingly).". At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 25-14 Destination project

Example 25-2 shows the successful import.

Example 25-2 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. The RM_Tenant message is similar to that shown in Figure 25-15.

Structure	Type	Min Occurs	Max Occurs
RM_Tenant.xsd			
Messages			
RM_Tenant	**ANONYMOUS**		
Id	xsd:int	1	1
Name		0	1
ApartmentId	xsd:int	0	1
Email		0	1
RM_Apartment		0	1
ANONYMOUS			
RM_Apartment:RM_Apartment		1	1
ANONYMOUS			
Id	xsd:int	1	1
ApartmentNumber	xsd:int	0	1
AddressLine1		0	1
AddressLine2		0	1
AddressLine3		0	1
AddressLine4		0	1
PostCode		0	1
ObjectEventId	xsd:string	0	1
version	xsd:token		
delta	xsd:boolean		
locale	xsd:string		
verb			
RM_Maintenance		0	1
ANONYMOUS			
RM_Maintenance:RM_Maintenance		1	-1
ANONYMOUS			
Id	xsd:int	0	1
ApartmentId	xsd:int	1	1
Status	xsd:string	0	1
TenantId	xsd:int	1	1
ProblemDescription		0	1
StatusDescription		0	1
ExpectedCompletion		0	1
ActualCompletion		0	1

Figure 25-15 RM_Tenant

10.Namespaces and prefixes should be amended as shown in Figure 25-16.

11.Save the Message Set project.

Namespace settings		
Namespace URI	Prefix	
http://www.ibm.com/websphere/crossworlds/2002/BO5chema/RM_Apartment	RM_Apartment	
http://www.ibm.com/websphere/crossworlds/2002/BO5chema/RM_Maintenance	RM_Maintenance	
http://www.ibm.com/websphere/crossworlds/2002/BO5chema/RM_Tenant	RM_Tenant	

Figure 25-16 RedMaintenance namespaces

Building and testing message flow for Retrieve

Now that you have your business objects in place, you can build the message flows that are required to transform the data from the business objects that are used by the JMS connector (for the front-end) and the custom RMConnector (for the back-end).

This chapter discusses the ESQL that is required to perform the transformation and gives some tips for troubleshooting. It also explains how to deploy the message flows to the Broker for the run-time environment.

You start by building the round trip between the front-end and back-end for the retrieval of tenant and maintenance details.

26.1 Retrieve processing scenario

As shown in Figure 26-1, 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 figure outlines the steps of the processing and which queues and message flows are used for the end-to-end retrieve processing.

To build the required message flows, you need to start by opening the Message Broker Toolkit.

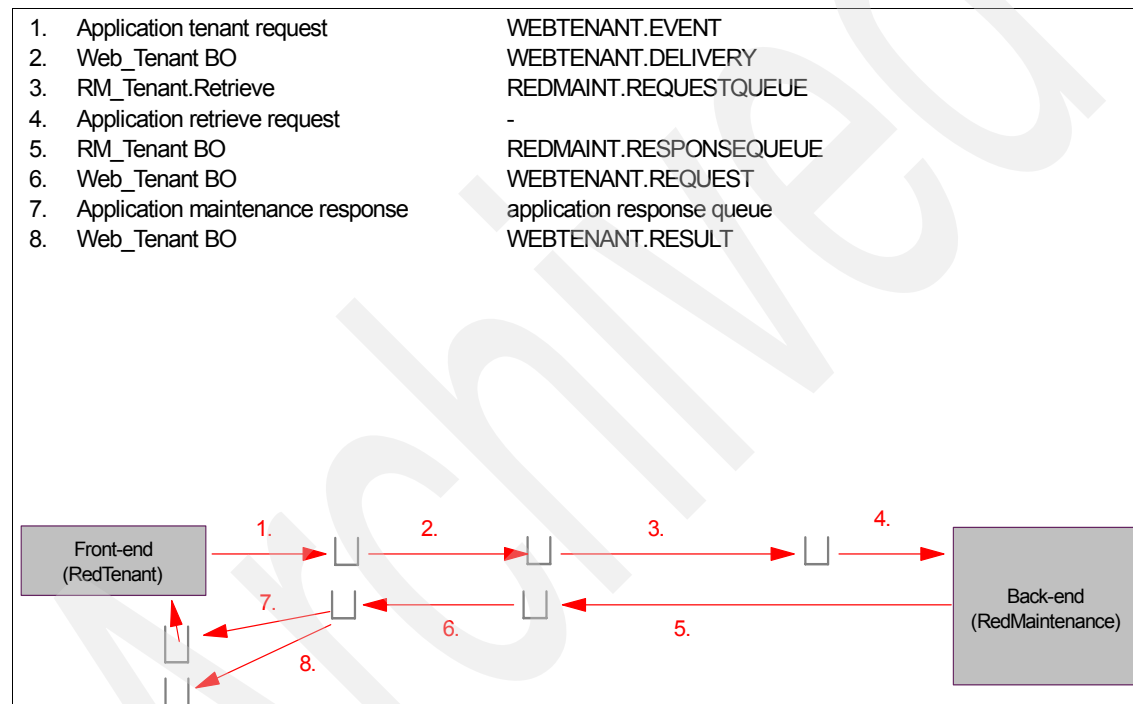


Figure 26-1 Processing flow for request

26.2 Creating a Message Flow project and Message Flows

This chapter does not discuss tracing in the flows. However, it is strongly advised that you put trace nodes everywhere you can in the initial phases of developing and testing your flows.

Tip: If you are not familiar with the tracing that is available in the broker, try using trace nodes with FILE trace properties and outputting 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, “Unit test traces” on page 867. This appendix contains samples of the information that you can obtain from file traces.

To create a Message Flow project:

1. Ensure that 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.
6. Close the new Message Flow.
7. Repeat steps 1 through 6 to create a second Message Flow. We chose RedMaint_to_RedTenant_Response as the name for the second Message Flow.
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 26-2 on page 472 to include the Message Sets that defined for the front- and back-end.
11. Save the Message Flow Project.

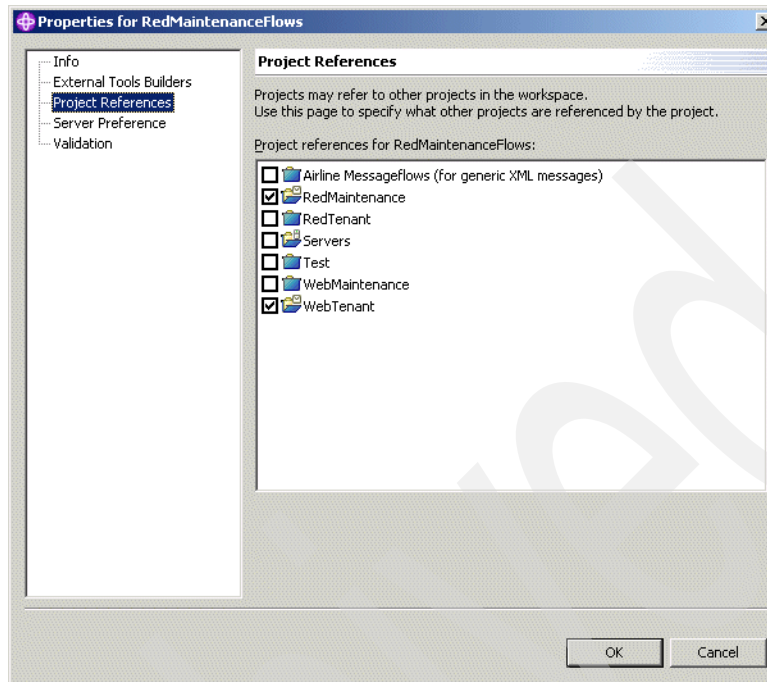


Figure 26-2 Project references

26.3 RedTenant_to_RedMaint_Request

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

Figure 26-3 on page 473 shows the fully finished request flow. We will build up this flow as we go along. You cannot add the Emitter subflow to your message flow yet. However, this information is covered in detail in the discussion of the monitoring of the run-time environment in Part 8, “Looking after the run-time environment” on page 823.

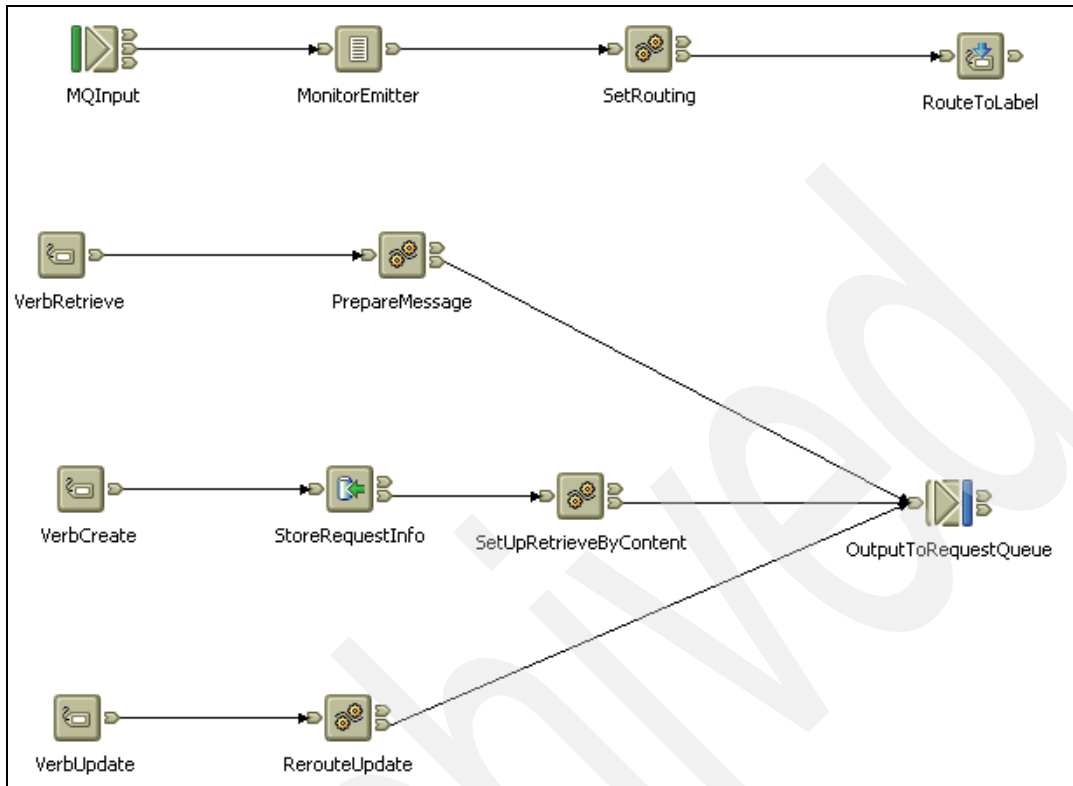


Figure 26-3 Full request message flow

Figure 26-4 shows that portion of the overall flow that we will build in this stage.

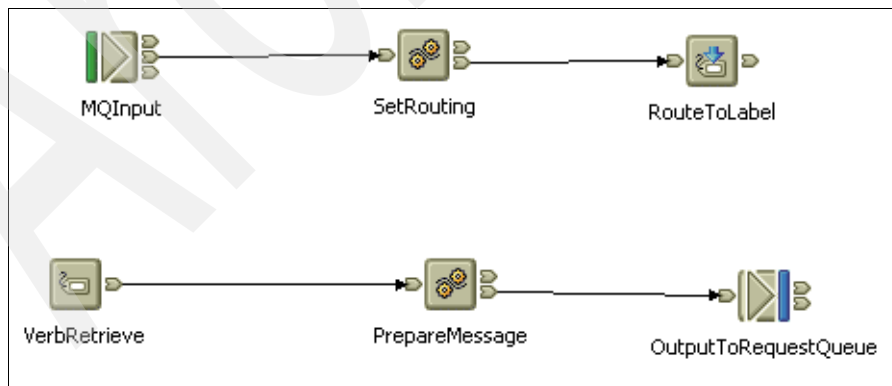


Figure 26-4 What we build for this stage

The flow for this initial stage contains the following nodes:

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

To create the flow:

1. In the Message Flow editor for the RedTenant_to_RedMaint_Request flow, drag these nodes to the canvas and rename them as shown in Figure 26-4 on page 473.
2. Wire the nodes together as shown in Figure 26-4 on page 473, omitting the MonitorEmitter subflow.
3. Open the MQInput node. Right-click the node and select **Properties**. Then, set the properties, as shown in Figure 26-5.

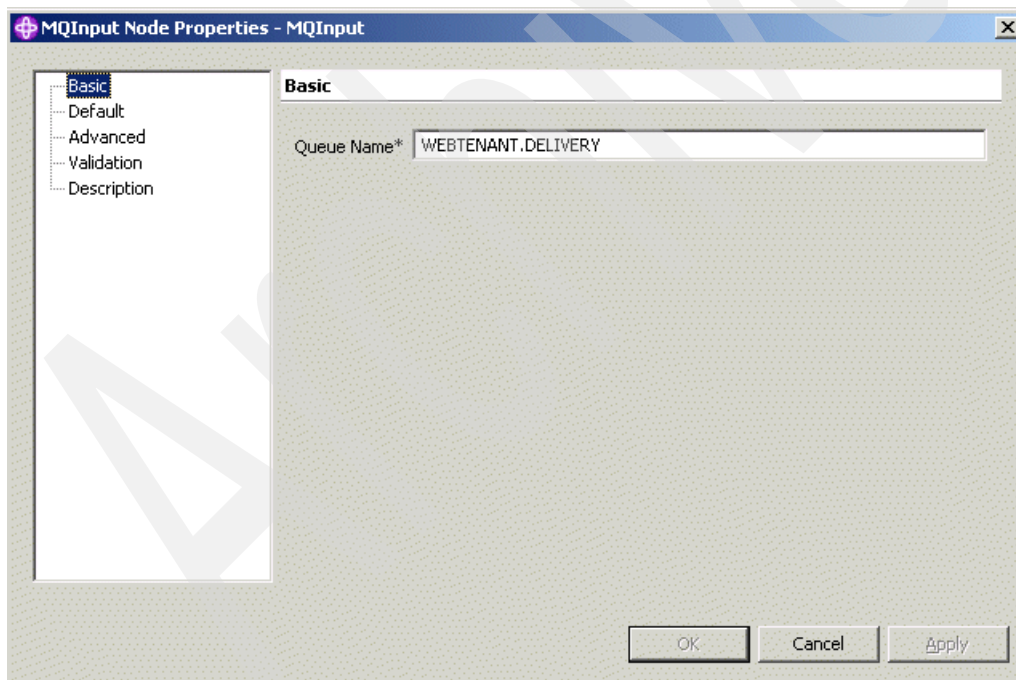


Figure 26-5 MQInput - Basic

4. Set the input queue name to WEBTENANT.DELIVERY. This input queue is the queue from the JMS connector that delivers to the broker for processing (see Figure 26-6).

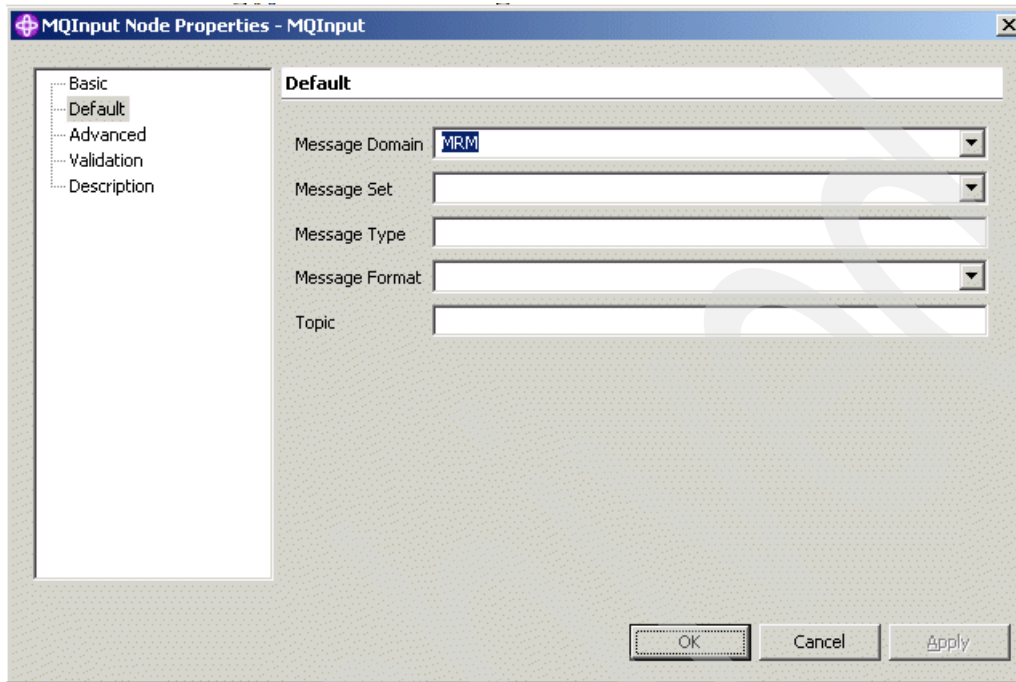


Figure 26-6 MQInput - Default

5. Set the default Message Domain to MRM, because we are expecting a message that was modeled in the broker.
6. Select the properties for the compute node and set them as shown in Figure 26-7 on page 476.
 - a. The ESQL module name is RequestRouting.
 - b. The Compute Mode is LocalEnvironment **and** Message. This is required to set the routing information in the ESQL.

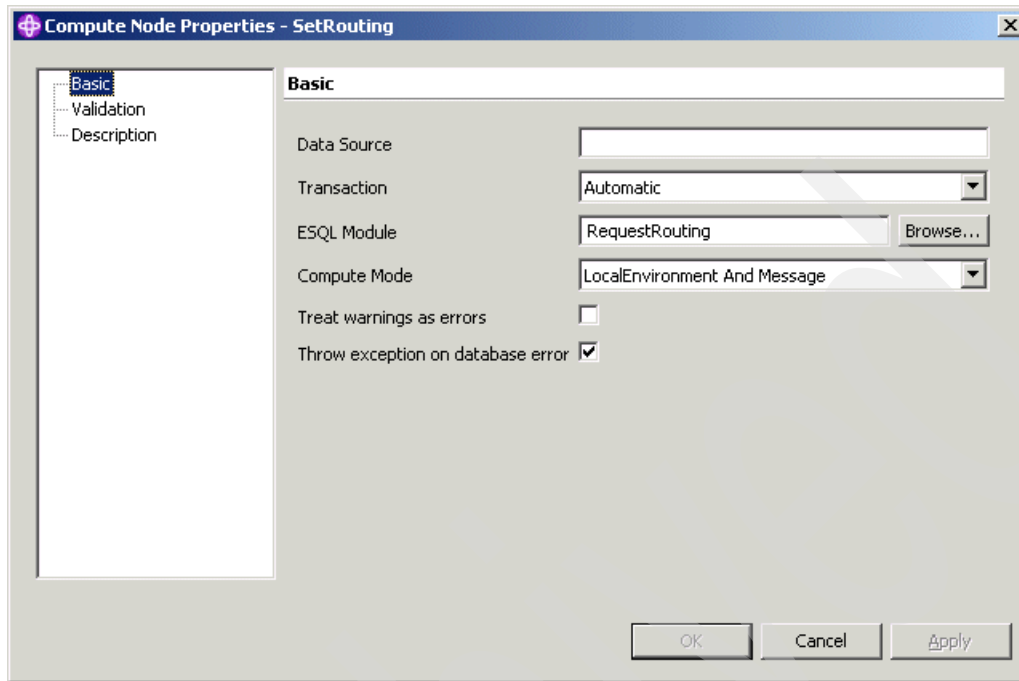


Figure 26-7 SetRouting properties

7. Click **Apply** for these changes.
8. Open the ESQL for the node and write the ESQL as shown in Example 26-1 on page 477. The ESQL does the following:
 - a. Copies the entire input message to be transmitted.
 - If the inbound message type is a Web_tenant, then it must be a retrieval request. 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, because the JMS connector uses a default verb — where the application is not able to make a distinction because it is nothing more than some application data that is dropped on a queue — you need to route to the Create processing. Set the OutputLocalEnvironment information that is required for a RouteToLabel to VerbCreate.
 - If the inbound message type is RM_Maintenance, then 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. These files are in the ESQL folder of the Additional Materials. The name of each file is *ESQL module name.txt*. Example 26-1 shows RequestRouting.txt.

You can use these files as an alternative to writing ESQL. Note, however, that so far in the process, we are performing simple tasks. 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 to give it a try.

Example 26-1 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 (Figure 26-8). You do not need to modify.

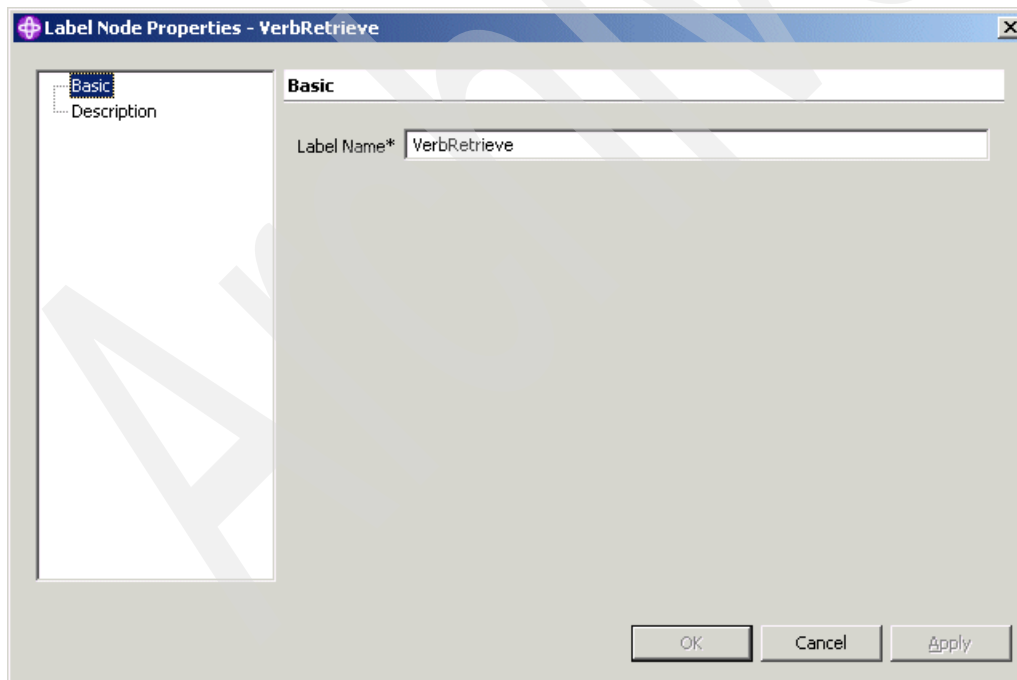


Figure 26-8 VerbRetrieve label

11. Open the properties of the Label (VerbRetrieve) node and set the label name to VerbRetrieve. This name must match the VerbRetrieve value that was set in the value that was set in the ESQL, as shown in the following example:

```
OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelName;
```

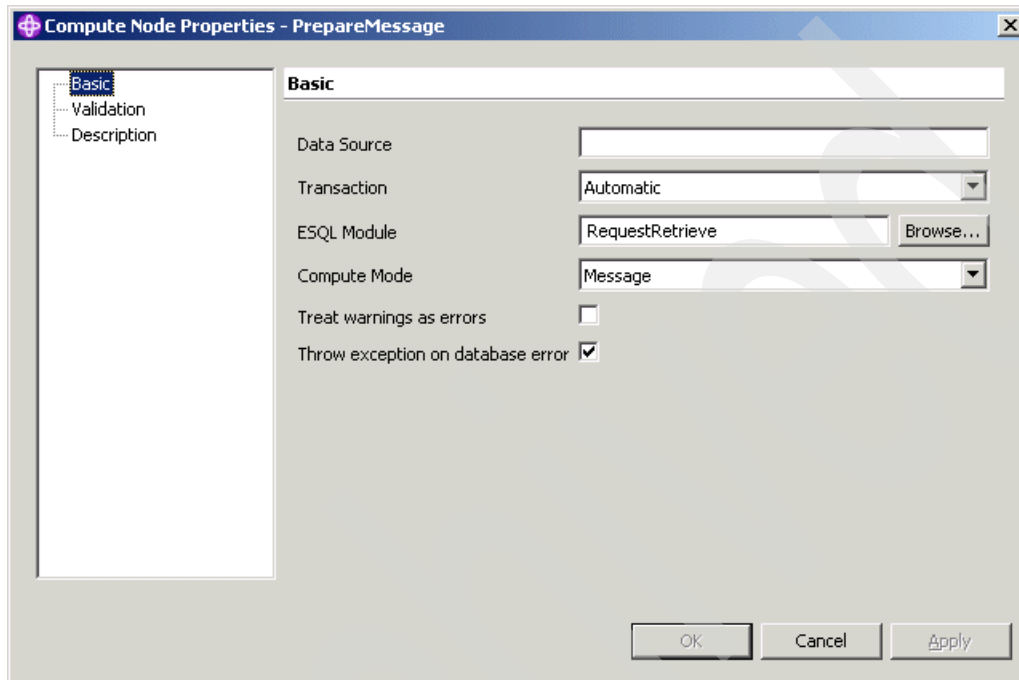


Figure 26-9 PrepareMessage

12. Open the properties of the PrepareMessage compute node.
13. Set the ESQL module to RequestRetrieve.
14. **Apply** the change.
15. Complete the ESQL. The ESQL performs the following:
 - a. Copies the message headers from input message to output message.
 - b. Sets up the namespace declarations for the front-end and back-end business object definitions.
 - c. Sets the properties of the output message.
 - d. Sets the message descriptor format to indicate an RFH header is included.

- e. Sets the following RFH header values:
 - Format = string
 - Message Domain = mrm
 - Message Set = RedTenant
 - Message Type = RM_Tenant
 - Wire Format = CwXML
 - Reply queue = Adapter Response queue
- f. Sets the following business object top-level values:
 - version = 3.0.0
 - delta = FALSE
 - verb = taken from input business object
 - locale = en_US
- g. Sets the following business object data values:
 - Output TenantId is taken from the input tenant name
 - Output ObjectEventId is taken from the input ObjectEventId

Note: If you were writing this code without 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 26-2).

Example 26-2 Without ESQL

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

(0x0300000B)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 = (

(0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant: Web_tenant_tenant = (
  (0x0300000B):version
= '3.0.0'
  (0x0300000B):verb
= ''
```

```

        (0x0300000B):locale
= 'en_US'
        (0x0300000B):delta
= FALSE

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant: schemaLocation = 'http://www.ibm.com/RedTenant tenant.xsd'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant: name = '100'
    )
)

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:0bjectE ventId =
'JMSConnector_ID:414d512052454454454e414e542020209e33654120000504'
)

```

The trace output in shows the input message. The one tiny piece of information that is required at this point is buried quite deeply in the message.

h. Sets the last of the following message properties:

- MessageDomain = mrm
- MessageFormat = CwXML.

Example 26-3 shows the full ESQL.

Example 26-3 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
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant';
    DECLARE Web_tenant_tenant NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant';

    -- Red Maintenance Back-end Application

```

```

DECLARE RM_Tenant NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';

-- ** 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 B0 fields

SET OutputRoot.MRM.RM_Tenant:Id =
InputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:name;
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;

END MODULE;
```

16. Save this ESQL.

17. Open the properties of the MQOutput node (Figure 26-10).

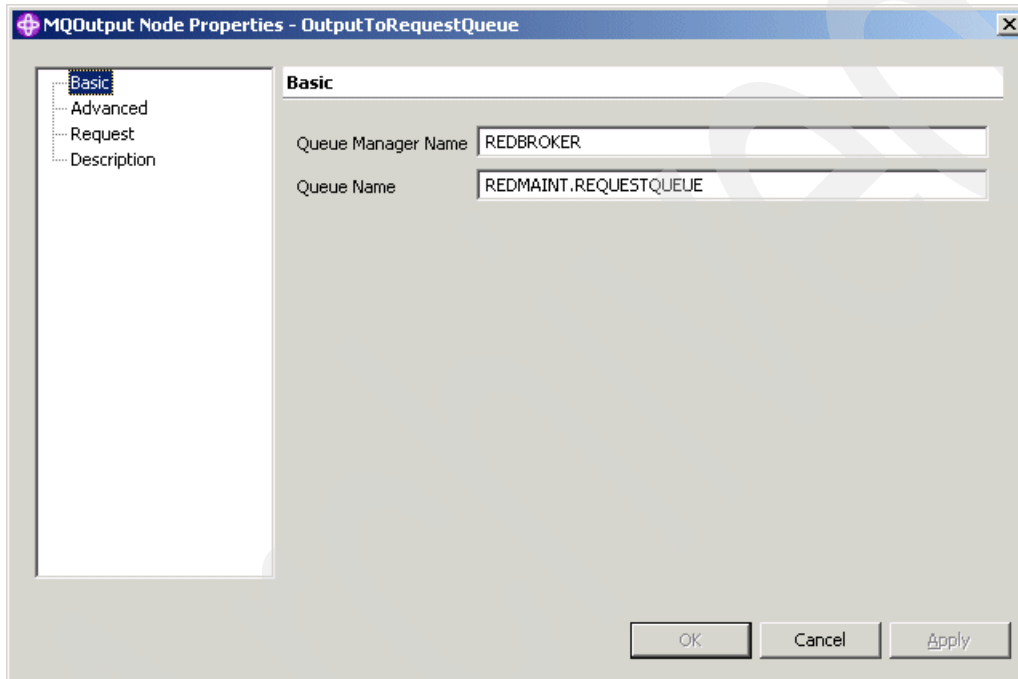


Figure 26-10 MQOutput node

18. Set the output queue for the RMConnector request queue (REDMAINT.REQUESTQUEUE).

19. Save your message flow.

Your completed flow should look similar to that shown in Figure 26-11, plus any additional trace nodes that you have added.

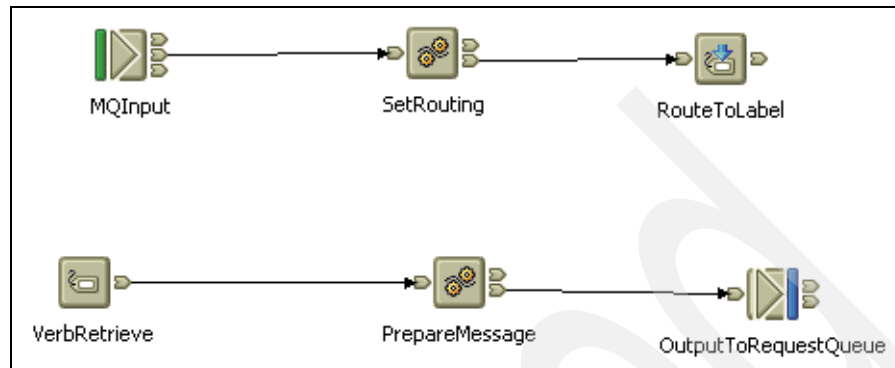


Figure 26-11 RedTenant_to_RedsMaint_Request so far

26.4 Deploying Message Sets and Message Flows

This section explains how to deploy Message Sets and the first of the Message Flows to the Broker. Before you deploy, clear the messages off any of the adapter queues so that you 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.

To start the deployment:

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. When you are connected, right-click the Broker name and select **New** → **Execution Group** (see Figure 26-12 on page 485).
5. Enter a name. We chose RedMaintenance.
6. Click **Next**.

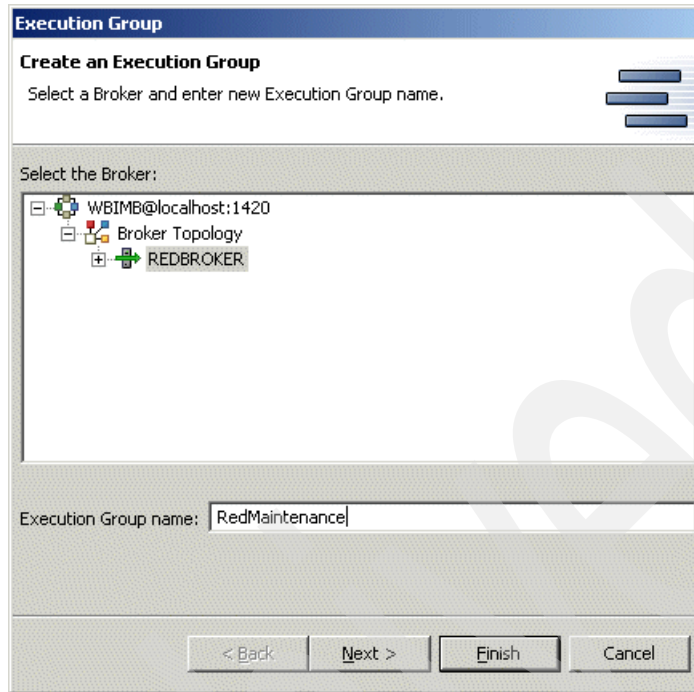


Figure 26-12 New execution group

7. Enter a description.
8. Click **Finish**.
9. In the Domains View (Figure 26-13), look for the name of the new Execution Group with a yellow exclamation mark next to it. This mark indicates that there are currently no Message Flows running in this Execution Group.

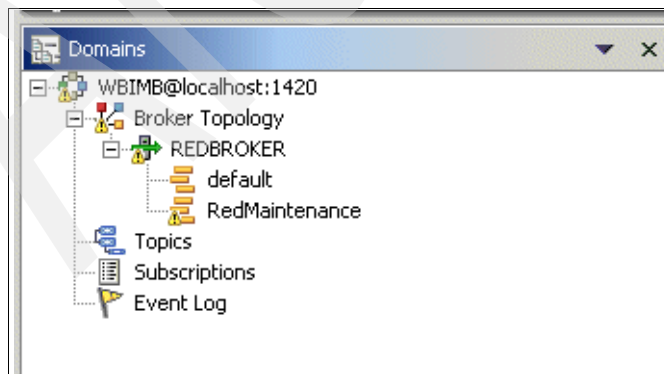
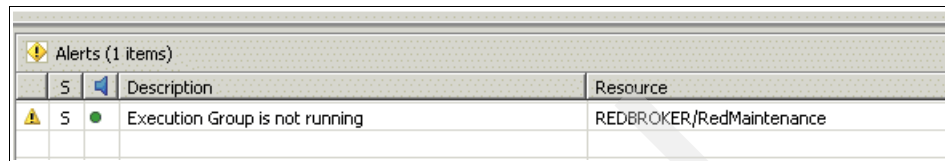


Figure 26-13 Domains view

10. The Alerts view that is shown in Figure 26-14 confirms that the Execution Group is not running a Message Flow.



Alerts (1 items)			
	S	Description	Resource
	!	Execution Group is not running	REDBROKER/RedMaintenance

Figure 26-14 Alerts view

You now need to deploy the 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 Broker Archive (BAR) file.

11. Select **File** → **New** → **Message Broker Archive** (see Figure 26-15 on page 487).

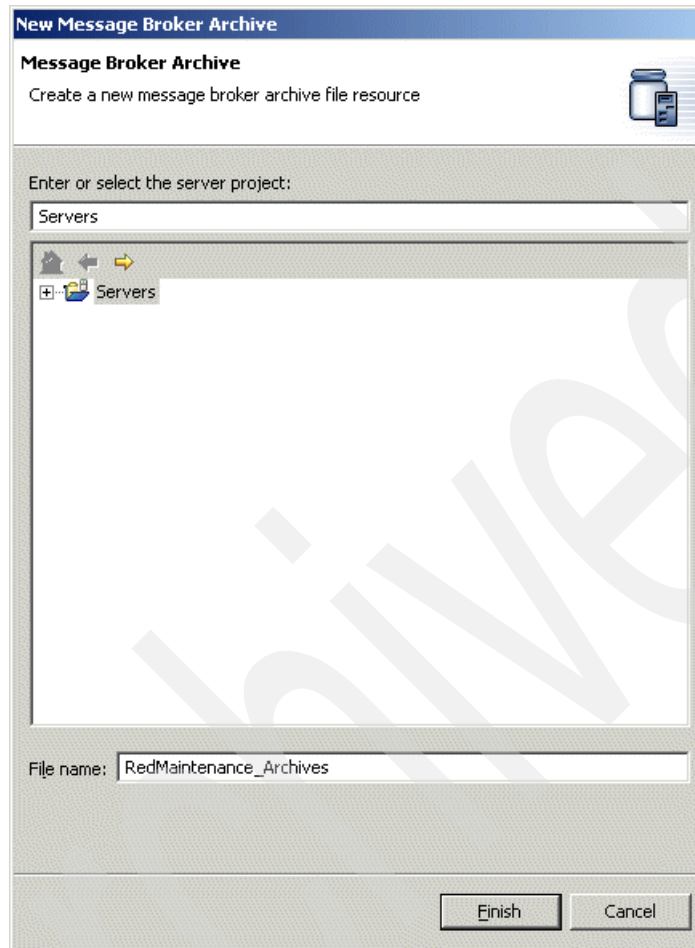


Figure 26-15 New BAR name

12. Enter a name for the new BAR file. We chose RedMaintenance_Archives.
13. Select **Finish**. The new empty BAR file is ready for you to add the artifacts (see Figure 26-16 on page 488).

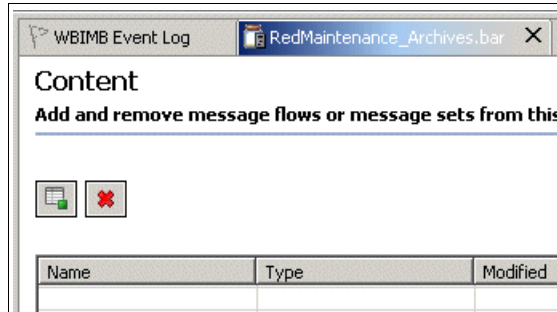


Figure 26-16 Add artifacts

14. Click **Add** (the button with the green square).

15. Select the Message Sets and Message Flows, as shown in Figure 26-17.

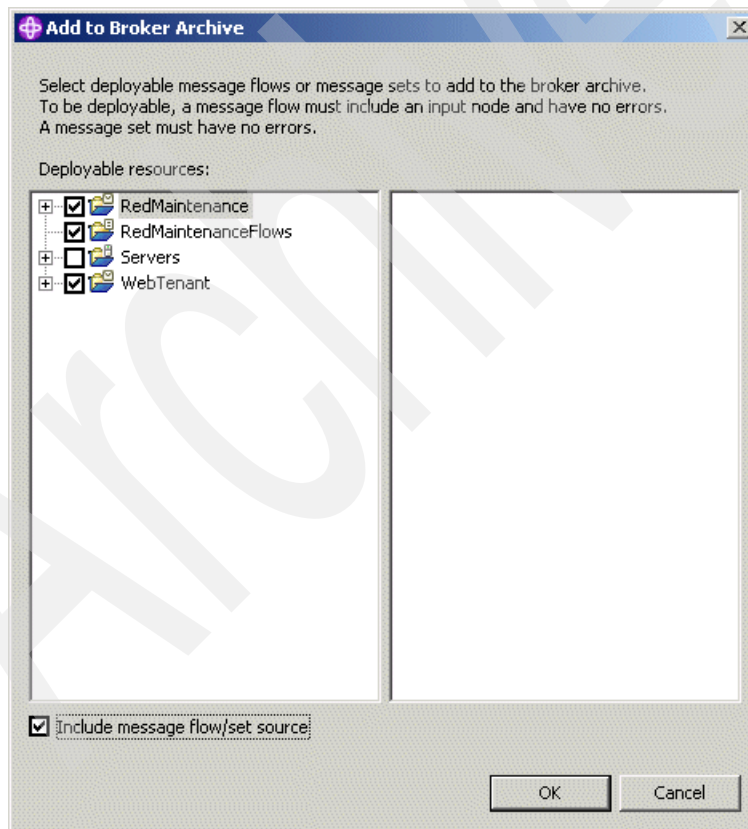


Figure 26-17 Select artifacts

16. Click **OK**. You can see the progress of the operation, shown in Figure 26-18, and the finished state, shown in Figure 26-19.

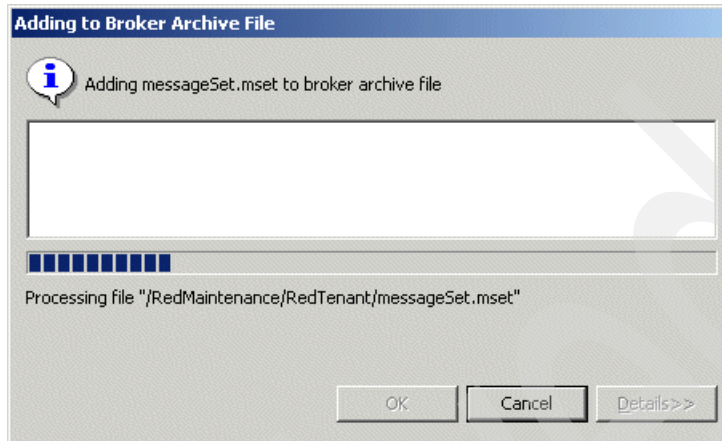


Figure 26-18 Progress window

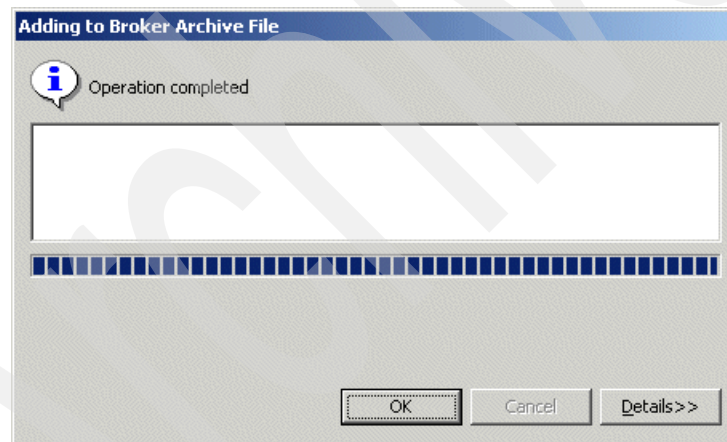


Figure 26-19 Operation complete



17. When the operation has completed, click **OK**. You can now see the contents of the BAR file, as shown in Figure 26-20 on page 490.

Important: If you are not familiar with BAR files, these files contain compiled versions of your artifacts. If you make *any* changes to these files, you must import them again to the BAR file and deploy the BAR file again. Notice a time and date stamp in Figure 26-20, which indicates the time of the last save for each file.

WBIMB Event Log *RedMaintenance_Archives.bar X

Content Servers/RedMaintenance_Archives.bar

Add and remove message flows or message sets from this archive.

Name	Type	Modified	Comment	Size	Path
Web_tenant_apartment...	XSD file	Sep 28, 2004 2:26:58 ...		3877	\WebTenant\W...
.wmqi21.mxsd	MXSD file	Jun 3, 2004 5:27:25 AM		7001	\RedMaintenan...
messageSet.mset	Message set file	Sep 28, 2004 2:26:08 ...		1656	\WebTenant\W...
.wmqi21.mxsd	MXSD file	Sep 28, 2004 2:19:55 ...		7001	\WebTenant\W...
.project	PROJECT file	Oct 7, 2004 1:53:25 PM		480	\RedMaintenan...
RedMaint_to_RedTenan...	Enhanced SQL file	Sep 27, 2004 5:15:30 ...		1037	\RedMaintenan...
RM_Apartment.xsd.rep...	TXT file	Jun 3, 2004 5:33:58 AM		317	\RedMaintenan...
messageSet.mset	Message set file	Jul 12, 2004 3:03:33 PM		1177	\RedMaintenan...
Web_tenant_tenant.xsd	XSD file	Sep 28, 2004 2:26:58 ...		4049	\WebTenant\W...
RM_Tenant.xsd	XSD file	Jun 3, 2004 5:30:35 AM		4633	\RedMaintenance
Web_tenant_apartment...	TXT file	Sep 28, 2004 2:29:08 ...		448	\WebTenant\lo...
.project	PROJECT file	Jun 3, 2004 5:27:25 AM		408	\RedMaintenance
RedMaint_to_RedTenan...	Enhanced SQL file	Sep 30, 2004 1:54:12 ...		10501	\RedMaintenan...
RM_Maintenance.xsd.re...	TXT file	Jun 3, 2004 5:32:04 AM		325	\RedMaintenan...
WebTenant.dictionary	Dictionary file	Oct 8, 2004 9:48:04 AM		46193	
.project	PROJECT file	Sep 28, 2004 2:19:55 ...		403	\WebTenant
RM_Apartment.mxsd	MXSD file	Sep 27, 2004 6:32:53 ...		8779	\RedMaintenan...
RedTenant_to_RedMain...	Enhanced SQL file	Sep 30, 2004 4:56:56 ...		2661	\RedMaintenan...
RM_Tenant.xsd.report.txt	TXT file	Jun 3, 2004 5:36:20 AM		479	\RedMaintenan...
Web_tenant_maintenan...	TXT file	Sep 28, 2004 2:27:38 ...		354	\WebTenant\lo...
Web_tenant_tenant.xs...	TXT file	Sep 28, 2004 2:29:31 ...		439	\WebTenant\lo...
RM_Maintenance.mxsd	MXSD file	Sep 15, 2004 12:48:3...		8749	\RedMaintenan...
Web_tenant.xsd	XSD file	Sep 28, 2004 2:26:58 ...		2842	\WebTenant\W...
Web_tenant_apartment...	MXSD file	Sep 28, 2004 2:28:43 ...		6420	\WebTenant\W...
RM_Tenant.mxsd	MXSD file	Jul 12, 2004 3:15:38 PM		8738	\RedMaintenan...
RedMaint_to_RedTenan...	Compiled message flow	Oct 8, 2004 9:48:06 AM		19699	
Web_tenant_apartment...	MXSD file	Sep 28, 2004 2:29:07 ...		3455	\WebTenant\W...
Web_tenant.xsd.report...	TXT file	Sep 28, 2004 2:29:55 ...		410	\WebTenant\lo...

☒ Show source files

Figure 26-20 BAR file contents

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 shown in Figure 26-20, 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 26-21).

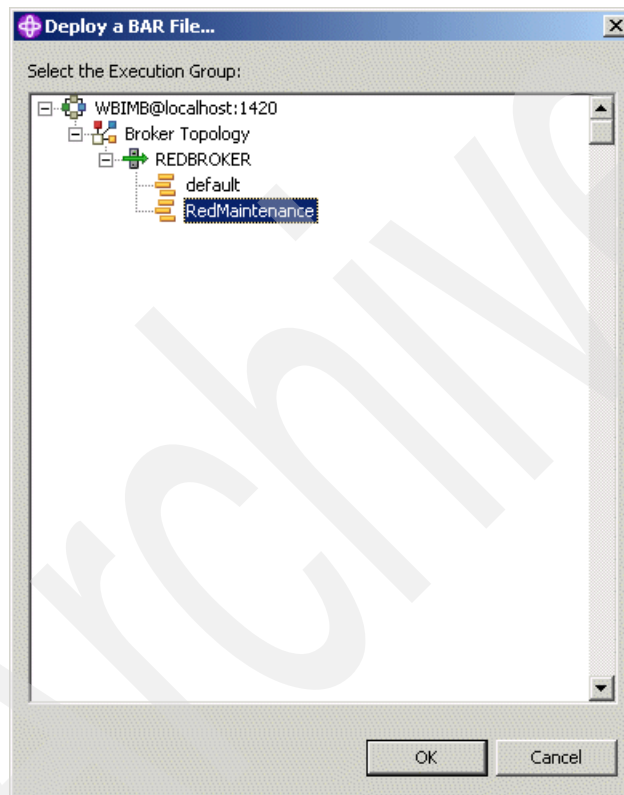


Figure 26-21 Select Execution Group

20. Select the RedMaintenance Execution Group.

21. Click **OK**. As shown in Figure 26-22 on page 492, you receive a response message from the Configuration Manager.

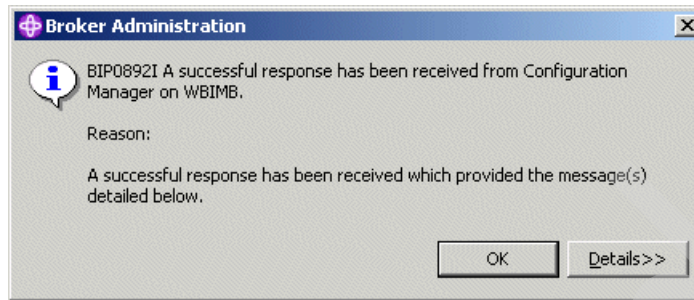


Figure 26-22 Configuration Manager response

This message 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**. This log contains messages that pertain to deployment events. Check for a successful deployment message. This is not the same as the Windows Event Viewer.

26.5 Unit testing the flow

Now that you have the first of your flows running, you can test the transformation of a front-end request to a back-end business object. Before you start, clear any left-over messages from previous unit testing by following these steps:

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 JMSConnector 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. Verify that this message exists by using `rfhuti1` in browse mode, as shown in Figure 26-23 on page 493.

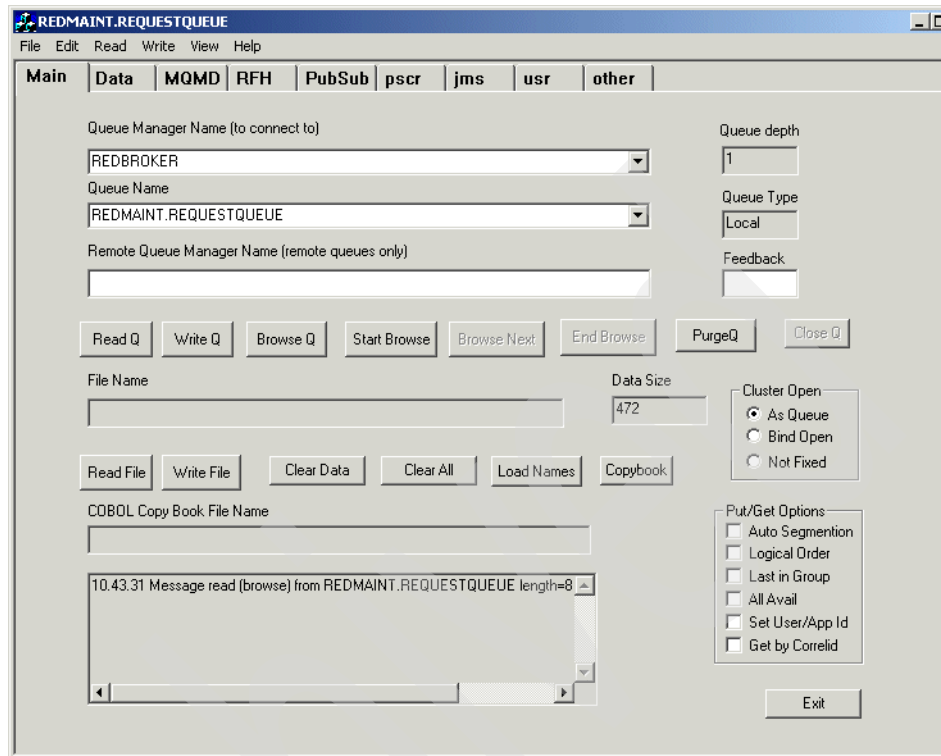


Figure 26-23 RM request queue

5. Click the Data tab to see the message for the transformed object, as shown in Figure 26-24 on page 494.

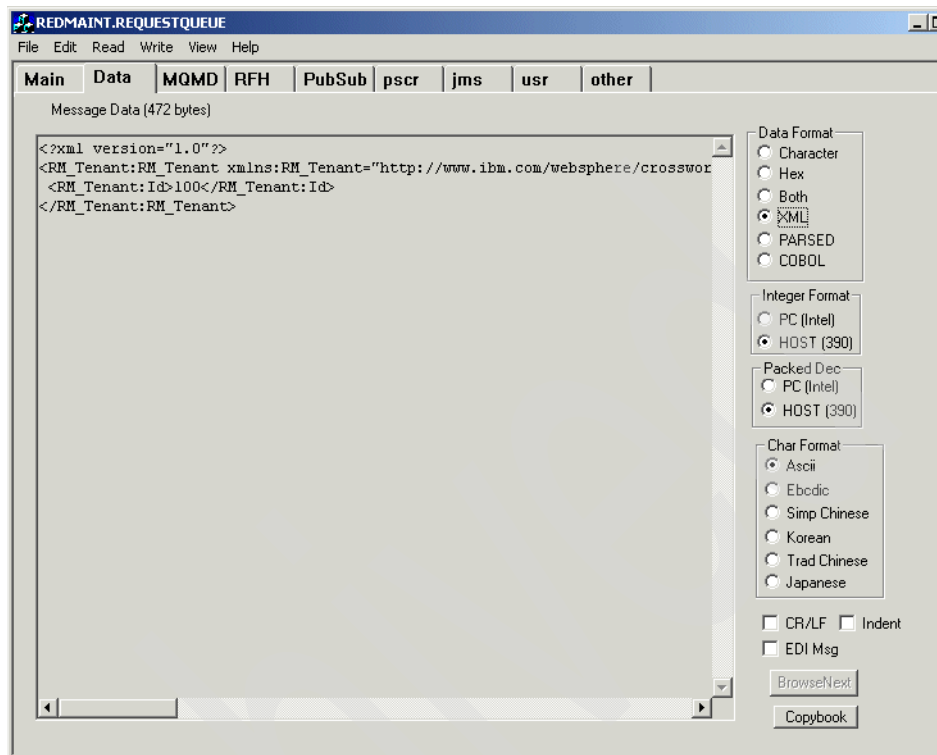


Figure 26-24 Transformed message

Tip: If you do not see this information, check the Windows Event Viewer, not the Domain Event Log. The Windows Event Viewer contains run-time errors from the broker, while the Event Log only contains deployment time errors.

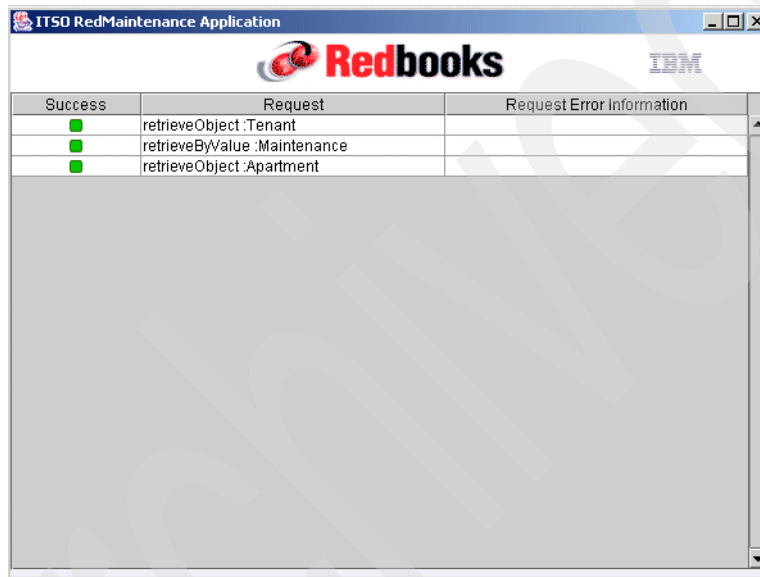
6. Check the traces that we have supplied for you if you need some clues:
 - InputMessageFromFrontEnd shows the message tree as it was delivered to the broker on the MQInput (see Example A-1 on page 867).
 - RedTenantDelivery_Retrieve shows the path that the message took after the RouteToLabel and the message tree as it was prior to the Compute (see Example A-2 on page 870).
 - RedMaintRequest shows the message tree after the Compute, that is, what should have gone to the REQUEST queue (see Example A-3 on page 872).

The final test for the first message flow is to let the RMConnector pick up your message and process it correctly by doing the following:

1. Ensure that the RedMaintenance application is running.
2. Start the RMConnector.

You should 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).

3. Check the application log for a successful retrieve (see Figure 26-25).



The screenshot shows a window titled "ITS0 RedMaintenance Application". At the top, there is a header bar with the "Redbooks" logo and the "IBM" logo. Below the header is a table with three columns: "Success", "Request", and "Request Error information". The table contains three rows of data, all with green checkmarks in the "Success" column. The "Request" column contains the following values: "retrieveObject :Tenant", "retrieveByValue :Maintenance", and "retrieveObject :Apartment". The "Request Error information" column is empty for all three rows. Below the table is a large, empty gray area, likely a scrollable log or details section.

Success	Request	Request Error information
■	retrieveObject :Tenant	
■	retrieveByValue :Maintenance	
■	retrieveObject :Apartment	

Figure 26-25 RedMaintenance log

4. Using `rfhutil`, browse the message from the REDMAINT.RESPONSEQUEUE and verify that it resembles that shown in Figure 26-26 on page 496.

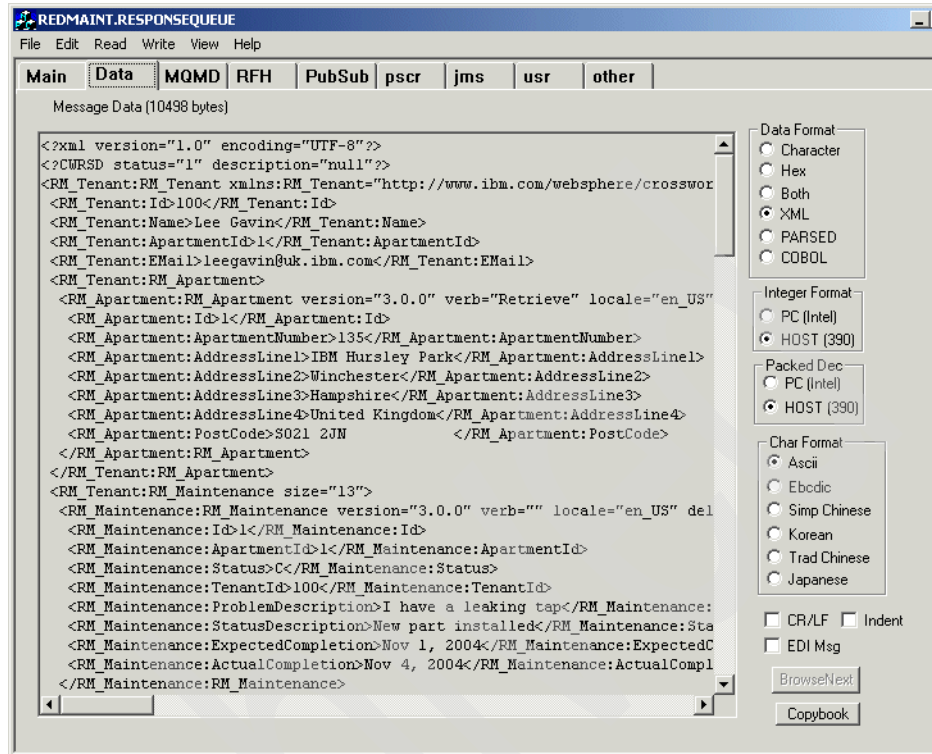


Figure 26-26 Response message

5. When all of this information is correct, you are ready to build the message flow to handle the response.
6. Shut down the JMSConnector.

26.6 RedMaint_to_RedTenant_Response

The back-end application transmits the results of the look-up to the RMConnector, 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 that is required by the JMS connector and puts the message in 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.

Figure 26-27 shows an overview of this process.

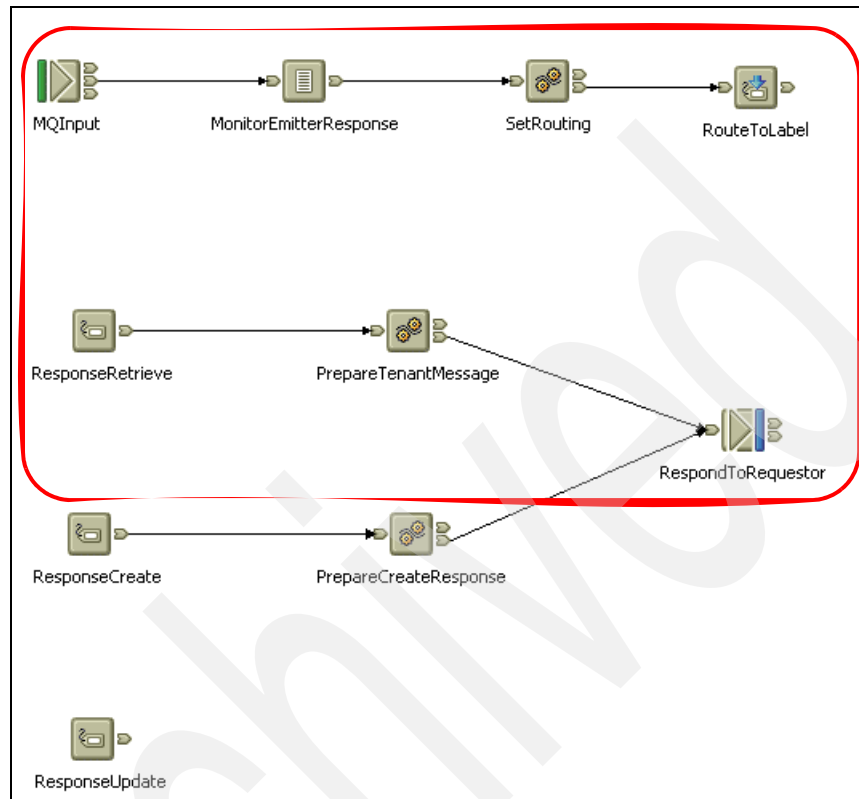


Figure 26-27 Complete response flow

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)

To create the flow:

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.

26.7 ResponseRetrieve

To create the ResponseRetrieve object:

1. Open the MQInput node properties.

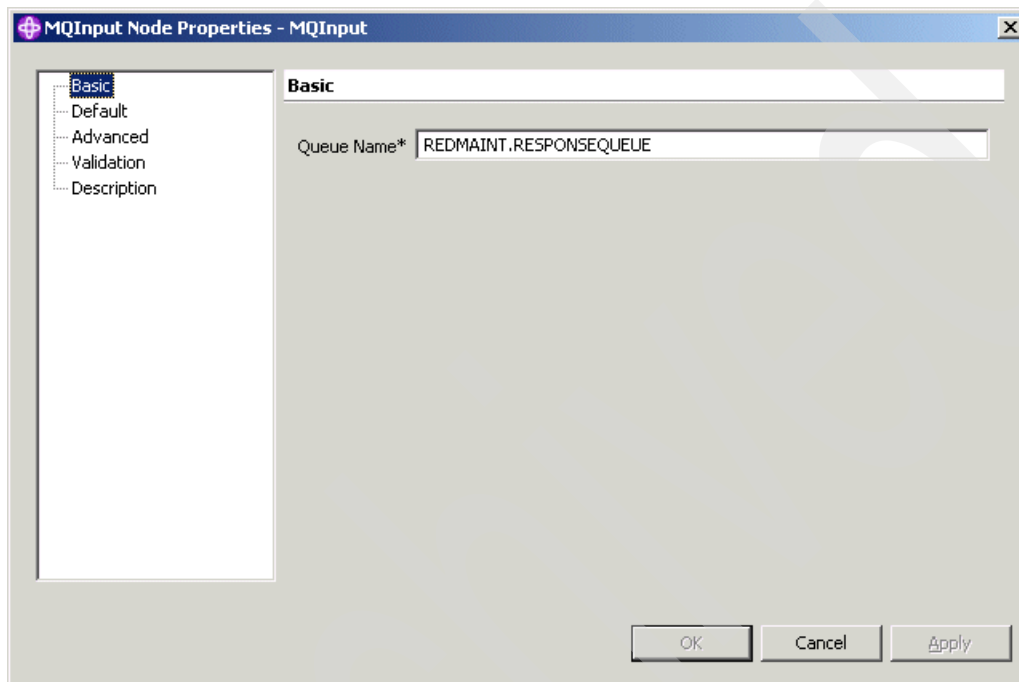


Figure 26-28 MQInput node

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 26-29 on page 499).

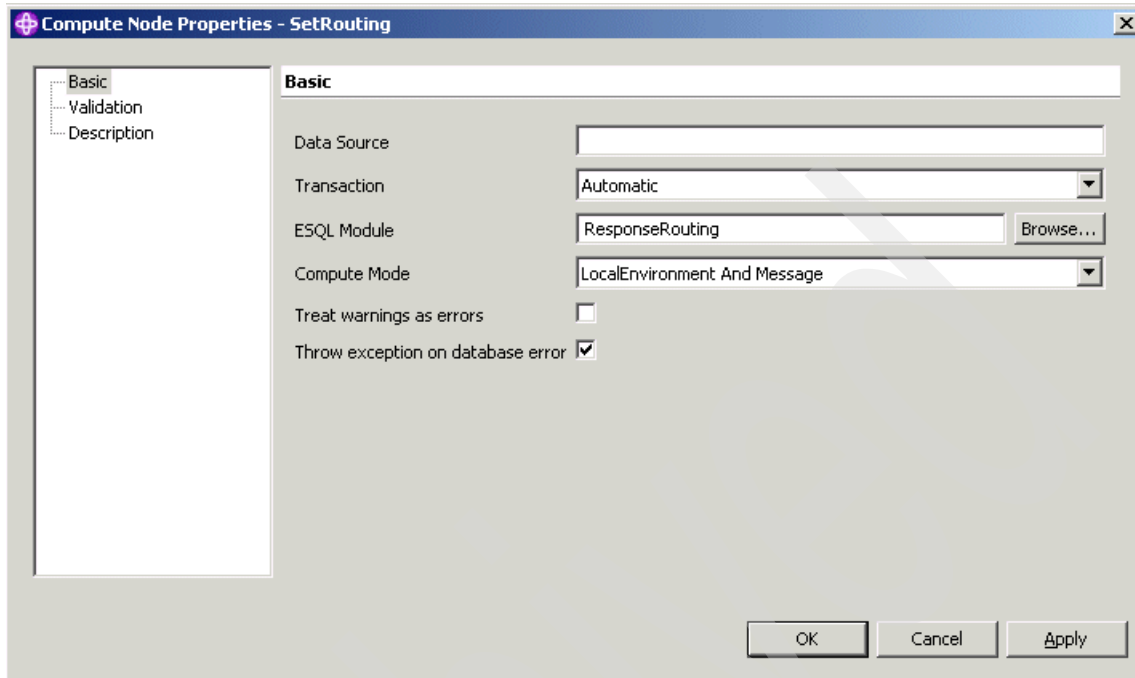


Figure 26-29 SetRouting properties

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 26-4.
The ESQL does the following:
 - a. Copies the entire input message to be passed on.
 - b. Sets the routing destination based on the Verb of the message.

Example 26-4 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 26-30 on page 501).

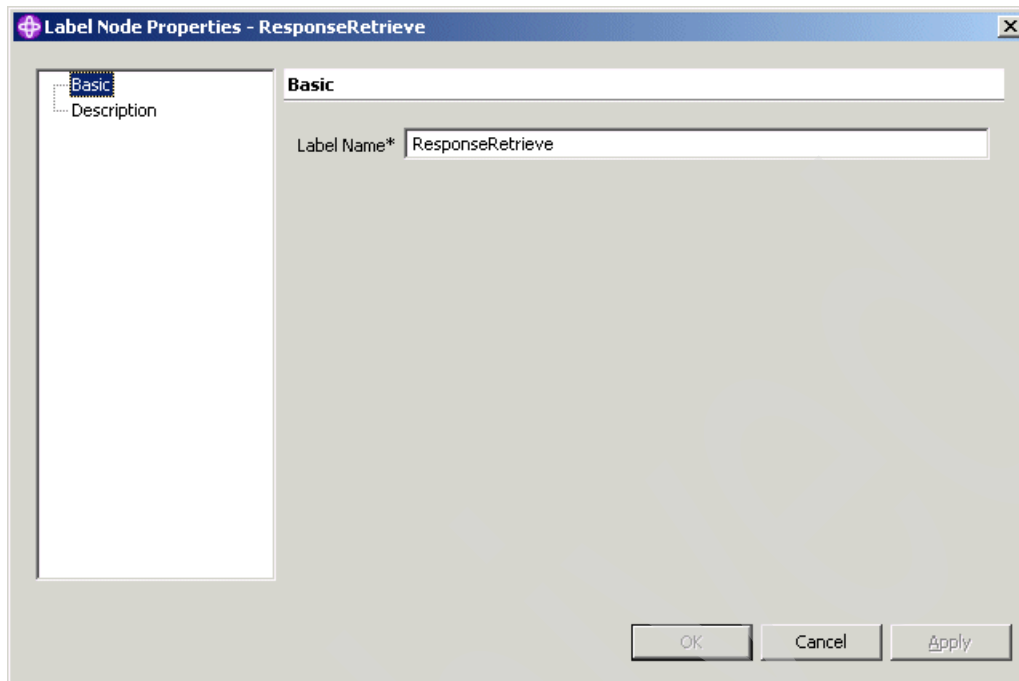


Figure 26-30 *ResponseRetrieve*

9. Set the `LabelName` to match that set in the routing ESQL.
10. Open the properties of the `PrepareTenantMessage` compute node (see Figure 26-31 on page 502).

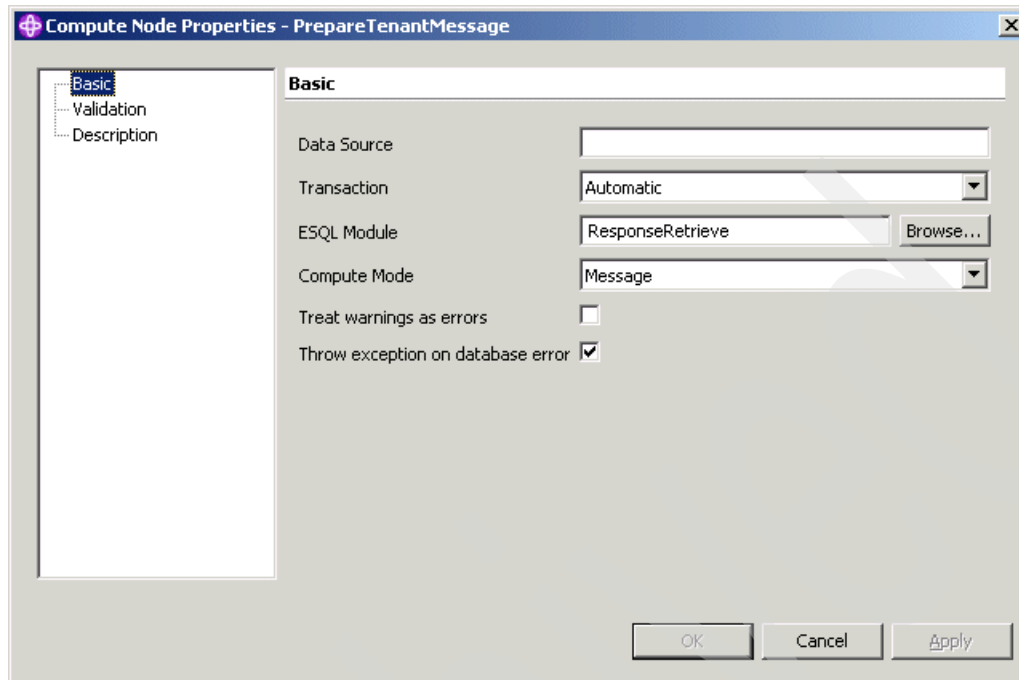


Figure 26-31 *PrepareTenantMessage* compute node

11. Set the ESQL Module name to ResponseRetrieve.
12. Complete the ESQL as shown in Example 26-5 on page 503.

The ESQL performs the following:

- a. Copies the message headers from input message to output message.
- b. Sets up the namespace declarations for the front-end and back-end business object definitions.
- c. Sets the properties of the output message.
- d. Sets the message descriptor format to indicate an RFH header is included.
- e. Sets the following RFH header values:
 - Format = string
 - Message Domain = mrm
 - Message Set = WebTenant
 - Message Type = Web_tenant
 - Wire Format = CwXML
 - Reply queue = Adapter Result queue, even though we are not going to do anything with these results.

- f. Sets the following business object top-level values:
 - version = 3.0.0
 - delta = FALSE
 - verb = taken from input business object
 - locale = en_US
- g. Sets the following business object data values:
 - XMLDeclaration
 - Schema location (The JMS connector uses it for the application data.
- h. Sets the following 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 e-mail is taken from the input tenant object e-mail.
- i. Sets the address object details.
- j. Loops through, and for each input maintenance record, creates an output maintenance record and formats the output.
- k. Sets the last of the message properties.

Example 26-5 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
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant';
    DECLARE Web_tenant_tenant NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant';
    DECLARE Web_tenant_maintenance NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance';
    DECLARE Web_tenant_maintenances NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenances';
    DECLARE Web_tenant_apartment NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment';
    DECLARE Web_tenant_apartments NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartments';
  
```

```

-- Red Maintenance Back-end Application
DECLARE RM_Tenant NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';
DECLARE RM_Maintenance NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance';
DECLARE RM_Apartment NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment';

-- ** 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 B0 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

```

```

--
Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:schemaLocation
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:schemaLoca
tion
= '"http://www.ibm.com/RedTenant_tenant.xsd"' ;

-- ** Set the tenant details

SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:id =
InputRoot.MRM.RM_Tenant:Id;
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:name =
InputRoot.MRM.RM_Tenant:Name;
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:email =
InputRoot.MRM.RM_Tenant:EMail;

-- ** Set the address

--
Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments.Web_tenant_apar
tments:Web_tenant_apartments.Web_tenant_apartments:apartment.(XML.attr)size='1'
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.size = '1';

--
Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments.Web_tenant_apar
tments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartment:Web_tenant_a
partment.Web_tenant_apartment:id
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:id
= CAST(InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:Id AS
CHARACTER) ;

--
Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments.Web_tenant_apar
tments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartment:Web_tenant_a
partment.Web_tenant_apartment:address
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments

```

```

.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:address
=

InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:ApartmentNumber ||
' ' ||
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:AddressLine1
|| ' ' ||
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:AddressLine2
|| ' ' ||
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:AddressLine3
|| ' ' ||
InputRoot.MRM.RM_Tenant:RM_Apartment.RM_Apartment:RM_Apartment.RM_Apartment:AddressLine4
;

--
Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments.Web_tenant_apa
rtments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartment:Web_tenant_a
partment.Web_tenant_apartment:postcode
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:postcode
=
InputRoot.MRM.RM_Tenant:RM_Apartment.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
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intences.Web_tenant_maintenances:maintenance.size = CAST (M AS CHARACTER);

WHILE C <= M DO

    DECLARE MaintInfo CHARACTER ;
    SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intences.Web_tenant_maintenances:maintenance.Web_tenant_maintenance:Web_tenant_maintenance[C]
.Web_tenant_maintenance:id

```

```

=
CAST(InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Id
AS CHARACTER);

CASE
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Status
WHEN 'A' THEN
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intenance:Web_tenant_maintenance:Web_tenant_maintenance[C]
.Web_tenant_maintenance:status
= 'ACTIVE';
WHEN 'P' THEN
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intenance:Web_tenant_maintenance:Web_tenant_maintenance[C]
.Web_tenant_maintenance:status
= 'IN PROGRESS';
WHEN 'C' THEN
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intenance:Web_tenant_maintenance:Web_tenant_maintenance[C]
.Web_tenant_maintenance:status
= 'COMPLETED';
ELSE
SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intenance:Web_tenant_maintenance:Web_tenant_maintenance[C]
.Web_tenant_maintenance:status
=
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Status ;
END CASE;

SET MaintInfo = ' ';

IF
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:ProblemD
escription IS NULL THEN
ELSE

```

```

        SET MaintInfo = MaintInfo ||

InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:ProblemD
escription ;

        END IF;
        IF
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:StatusDe
scription IS NULL THEN
            ELSE
                SET MaintInfo = MaintInfo || ' CURRENT STATUS INFO: ' ||

InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:StatusDe
scription ;

                END IF;
                IF
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Expected
Completion IS NULL THEN
                    ELSE
                        SET MaintInfo = MaintInfo || ' EXPECTED COMPLETION DATE: ' ||

InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Expected
Completion ;

                        END IF;
                        IF
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:ActualCo
mpletion IS NULL THEN

                            ELSE
                                IF
InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:Status =
'C' THEN
                                    SET MaintInfo = MaintInfo || ' ACTUAL COMPLETION: ' ||

InputRoot.MRM.RM_Tenant:RM_Maintenance.RM_Maintenance:RM_Maintenance[C].RM_Maintenance:ActualCo
mpletion ;

                                    END IF;
                                END IF;

                                SET
OutputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant.Web_tenant_tenant:apartments
.Web_tenant_apartments:Web_tenant_apartments.Web_tenant_apartments:apartment.Web_tenant_apartme
nt:Web_tenant_apartment.Web_tenant_apartment:maintenances.Web_tenant_maintenances:Web_tenant_ma
intenance.Web_tenant_maintenances:maintenance.Web_tenant_maintenance:Web_tenant_maintenance[C]
.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 26-32 on page 510).

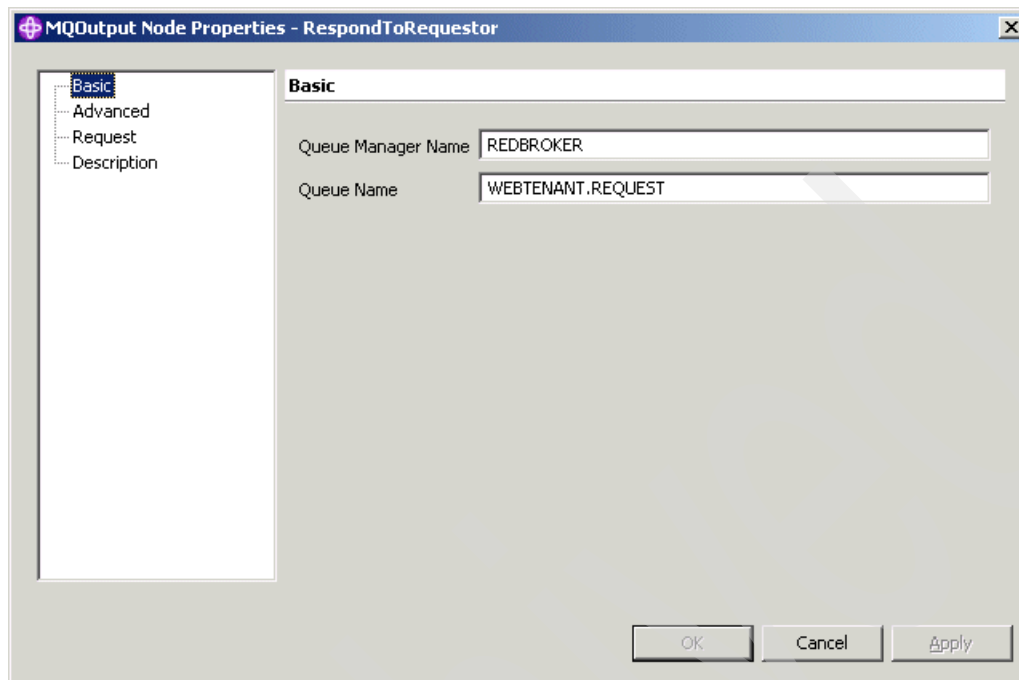


Figure 26-32 Output queue

Your complete flow up to this point should look similar to Figure 26-33, plus any trace nodes that you are using.

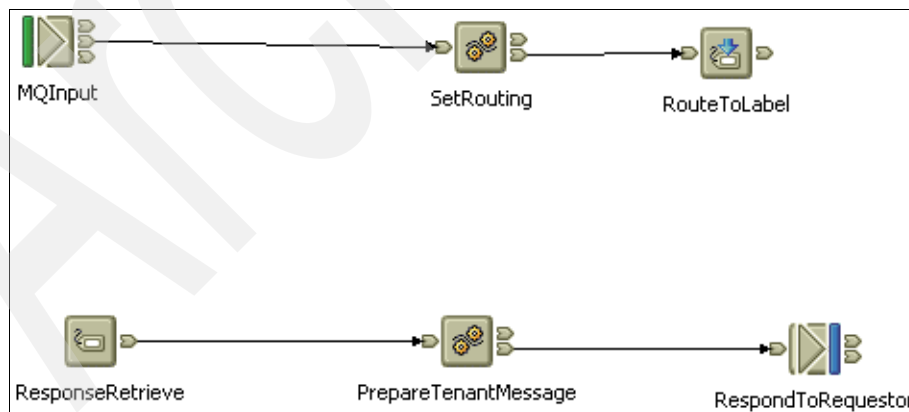


Figure 26-33 Current response flow

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.

26.8 Deploying the Message Flows

To deploy the message flows:

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.

You should see both Message Flows deployed to the broker.

26.9 Unit testing the 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 be picked up from the queue and processed. The flow of the process is as follows:

1. The message should now be on the WEBTENANT.REQUEST queue in its transformed state.
2. Use `rfhuti1` to browse the message on the queue (see Figure 26-34 on page 512).

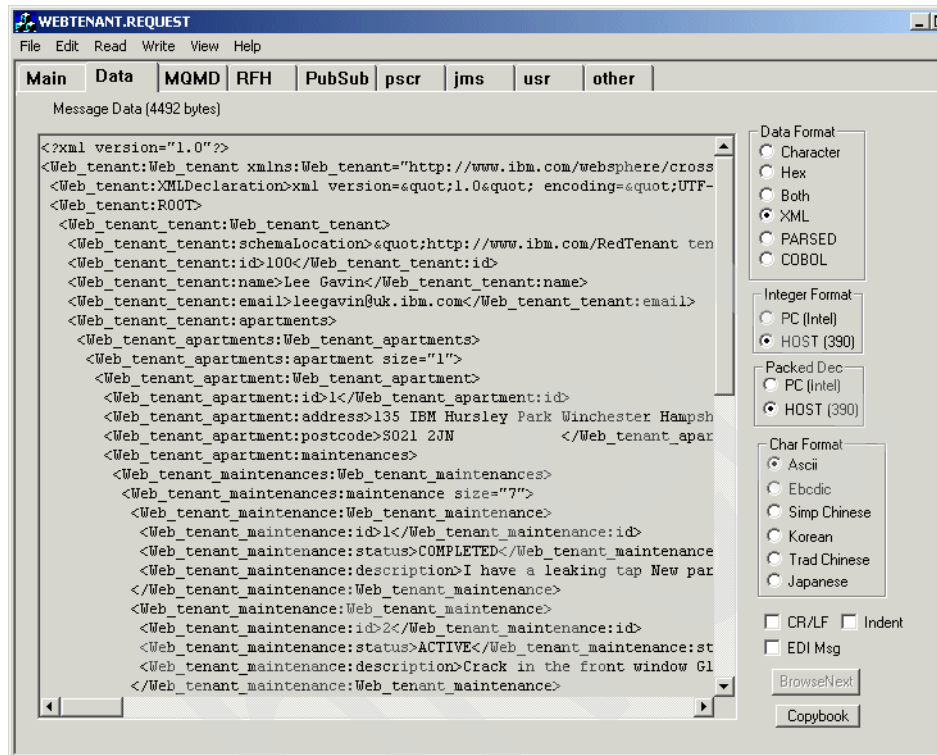


Figure 26-34 Message ready for JMSConnector

You need to test that the JMSConnector will take the 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 `rfhuti1` to browse this message and verify that it looks like a reasonably correct application message (see Figure 26-35 on page 513).

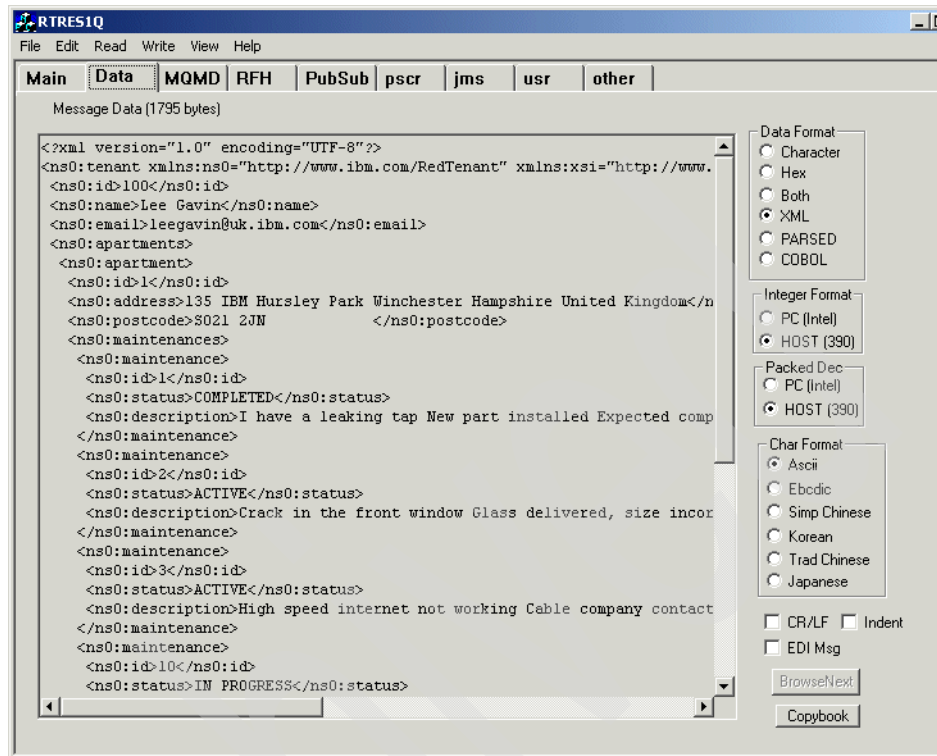


Figure 26-35 Message for the application

6. Shut down the JMSConnector. You no longer need it and will modify it prior to the full end-to-end test.
7. Navigate back to the browser window and reissue the request for tenant 100, or whichever tenant you used.

The message is picked up from the application response queue and the results are displayed correctly, as shown in Figure 26-36 on page 514.

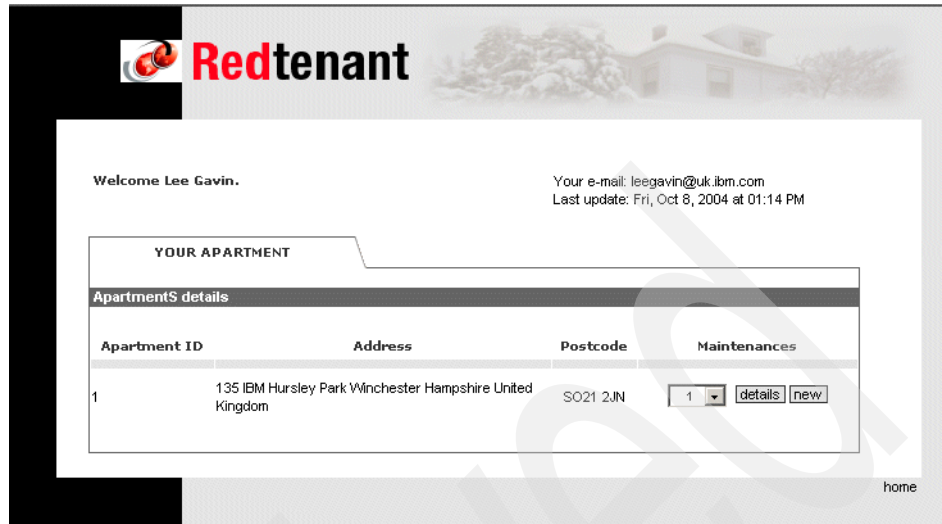



Figure 26-36 Correct response

26.10 End-to-end testing

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 test, you want to ensure that the JMS connector is polling correctly. To start the end-to-end test:

1. Check that none of the queues have leftover messages on them from 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, the application deserves decent response).
3. Save the change to the project and to file.
4. Start the JMSConnector, and ensure that the new polling has taken affect.
5. Ensure that the following 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.



Redtenant


TENANT MANAGEMENT APPLICATION

Please, enter your id to retrieve information about your apartments:

Tenant ID:

Figure 26-37 Test 101

7. Check for the correct response received, as shown in Figure 26-38.



Redtenant

Welcome ITS0 Resident.

Your e-mail: resident@us.ibm.com
Last update: Fri, Oct 8, 2004 at 01:19 PM

YOUR APARTMENT

Apartment\$ details

Apartment ID	Address	Postcode	Maintenances
2	700 Park Office Drive Raleigh NC USA	27709	<input type="button" value="14"/> <input type="button" value="details"/> <input type="button" value="new"/>

home

Figure 26-38 Response for test 101

8. Try another one. Tenant 102 has an apartment but no maintenance records. Test that then transformations handle this situation correctly.

9. Navigate to the RedTenant home page and enter a query for tenant 102.

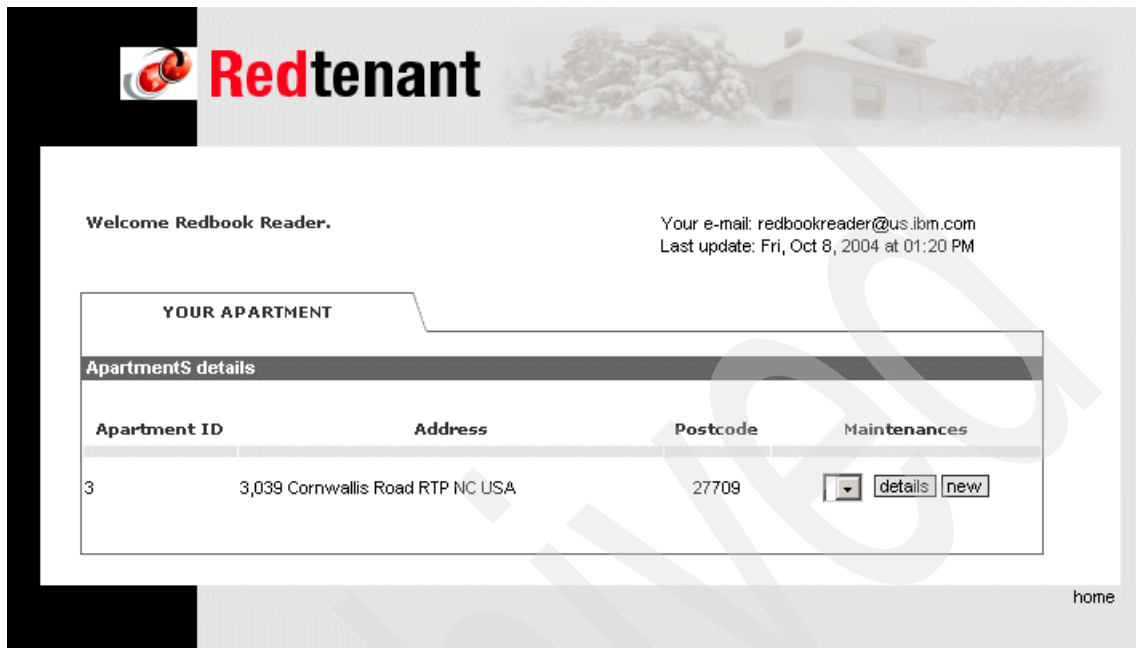


Figure 26-39 Tenant 102 test

10. In Figure 26-39, 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 26-40 on page 517).

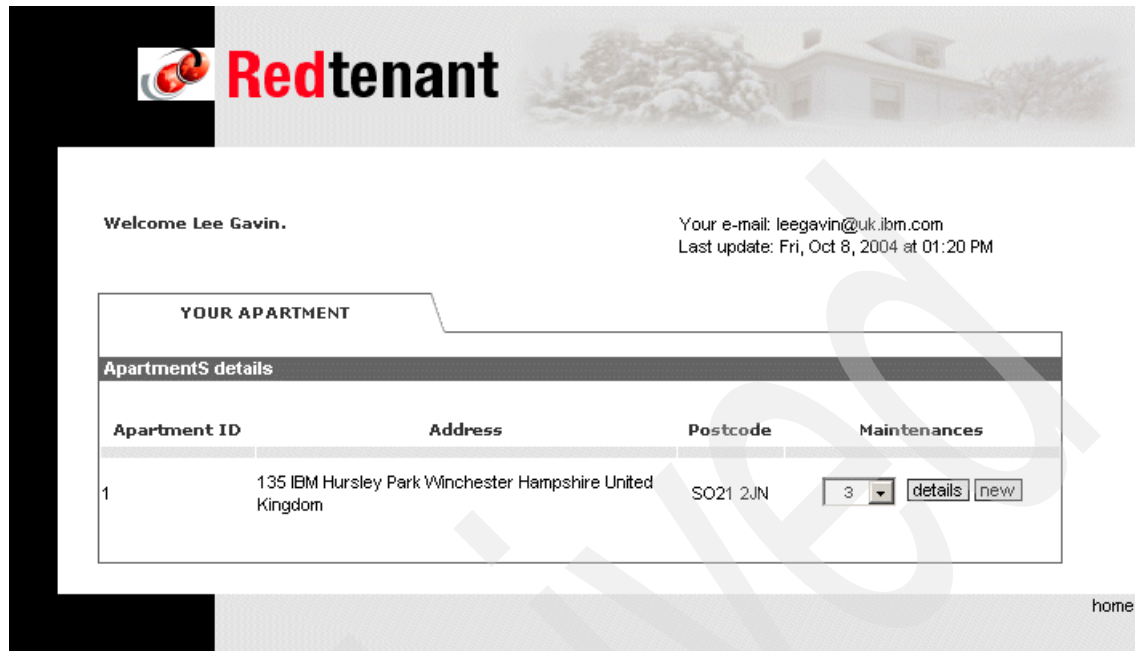


Figure 26-40 Tenant test

12. Check a maintenance record, as shown in Figure 26-41.

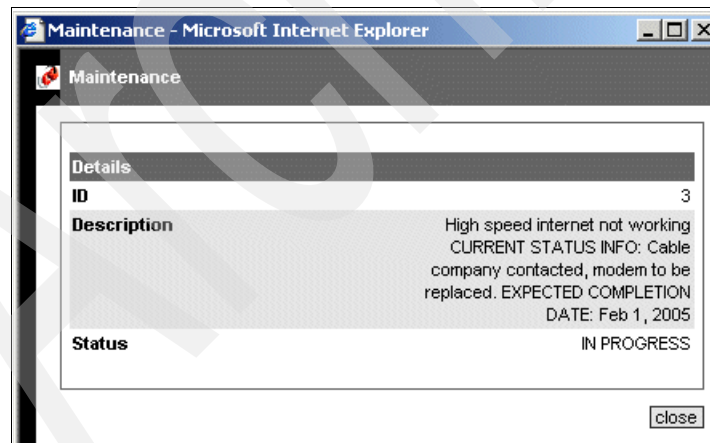


Figure 26-41 Tenant test

You now have the retrieve component of the solution working correctly and can move on to the creation of new maintenance requests.

Building and testing message flows for Create

Now that you have basic request and response flow in place, you can extend the flows to incorporate the requests for new maintenance orders from the front-end application. This chapter provides details on the nodes and ESQL that are required to perform the transformation. It also explain how to deploy the message flows to the Broker for the run-time environment.

27.1 Creating a processing scenario

As shown in Figure 27-1 and Figure 27-2 on page 521, the request processing and the responses to this processing go through a broker for data transformation and enhancement. In this case, you will 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 maintenance create processing.

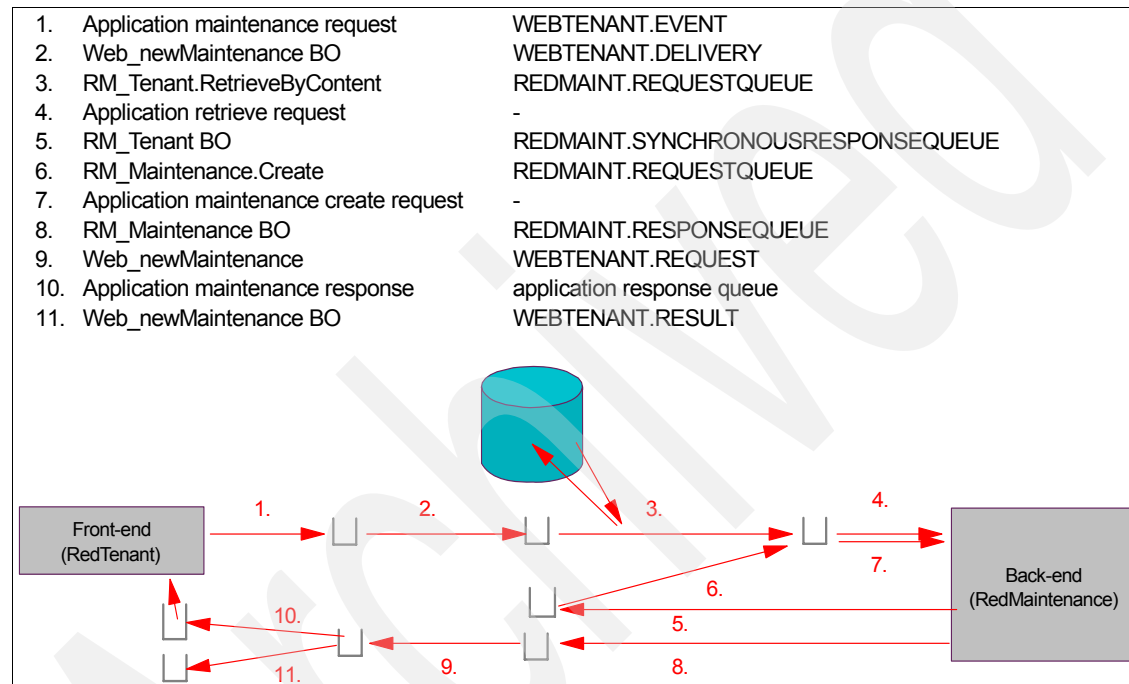


Figure 27-1 Processing flow request

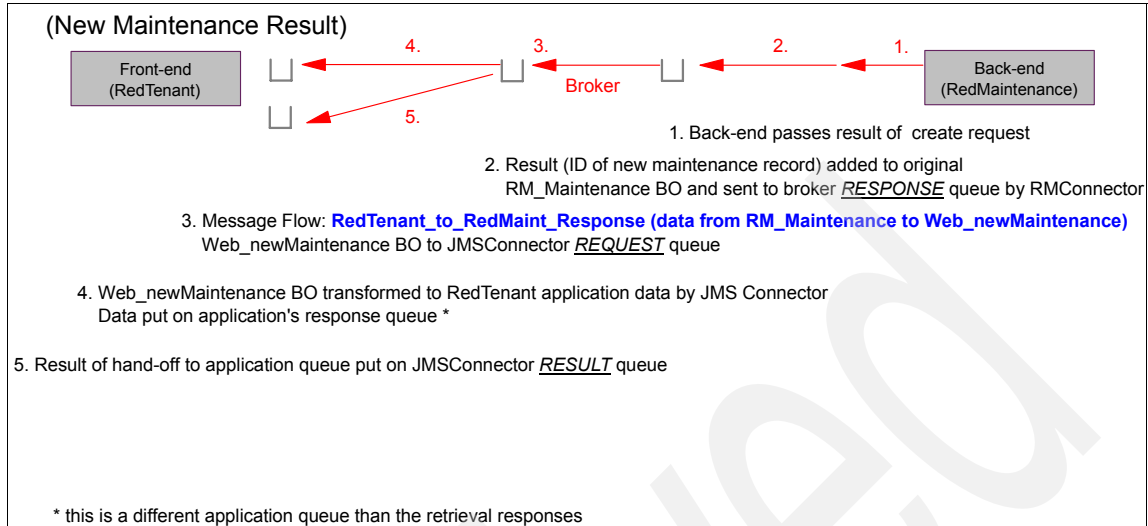


Figure 27-2 Processing flow response

27.2 RedTenant_to_RedMaint_Request

You first need to extend the message flow for the request processing. When you initially created the flow, the routing catered to Web_newMaintenance requests arriving on the queue. You 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 you will use 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 the tenant name and the details of the new request only. Previously, the entire process was mostly manual and the call center operator had screens for both applications available. However, this scenario is not enough information for the back-end application to create a maintenance request. The back-end application requires that the tenant ID and the apartment ID because these are mandatory elements of the maintenance object in the application.

We have a fairly straight forward way of retrieving data when we use the adapter. The RetrieveByContent verb retrieves the tenant details using non-key values

because this is an exposed function of the back-end API. The details of the original maintenance request are stored for later use. We send a `RetrieveByContent` based on what we know about the tenant, which is 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 send the message to the synchronous response queue, which is not being used. This action, in turn, starts another flow to build the business object data and passes it back to the request queue for adapter processing.

To start building the `RedTenant_to_RedMaint_Request` flow:

1. 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 as shown in Figure 27-3 on page 523.

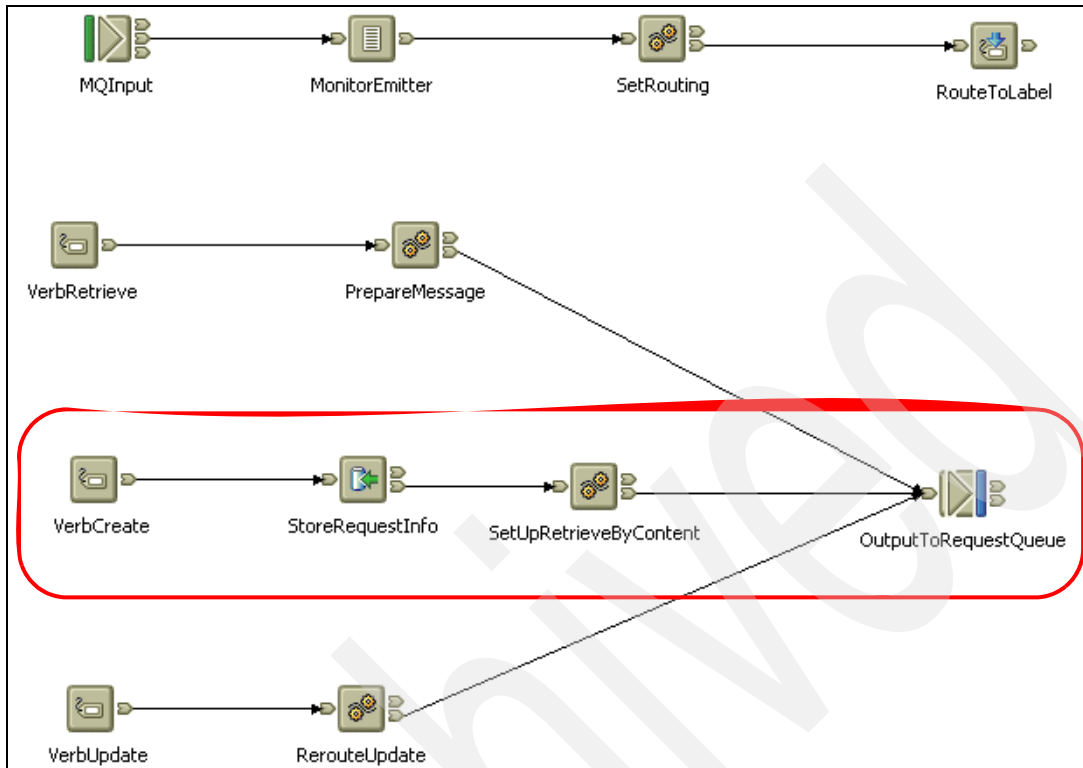


Figure 27-3 Full request message flow

3. Open the properties of the Label node and set the label to VerbCreate, as shown in Figure 27-4 on page 524.

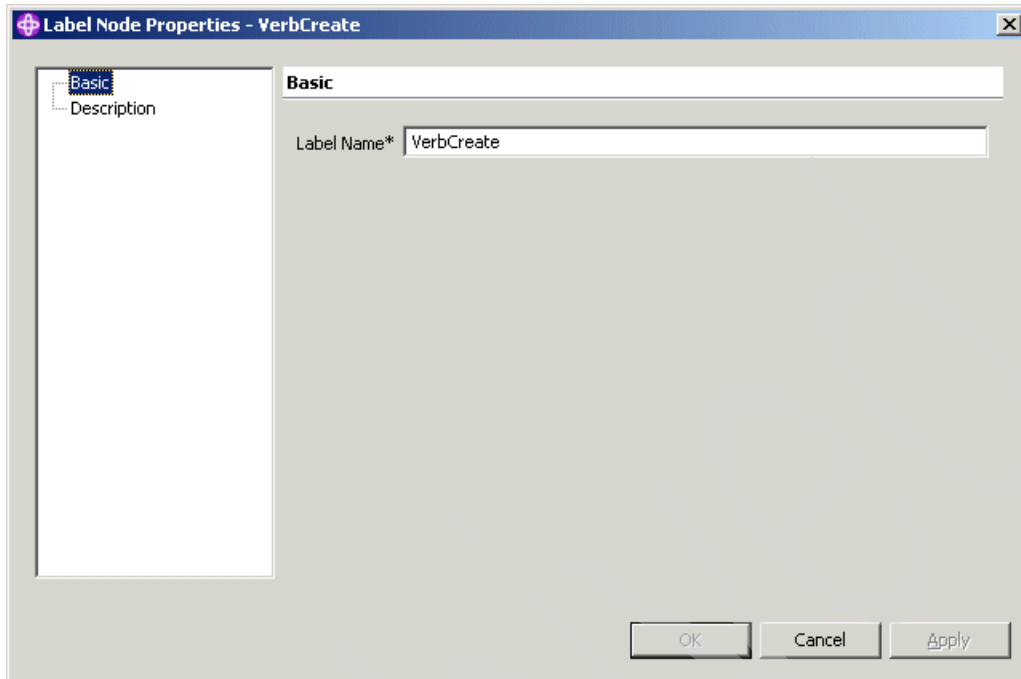


Figure 27-4 Label

4. Add the database that you are using to store the request information.

Note: The ddl for this database 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 27-5 on page 525.

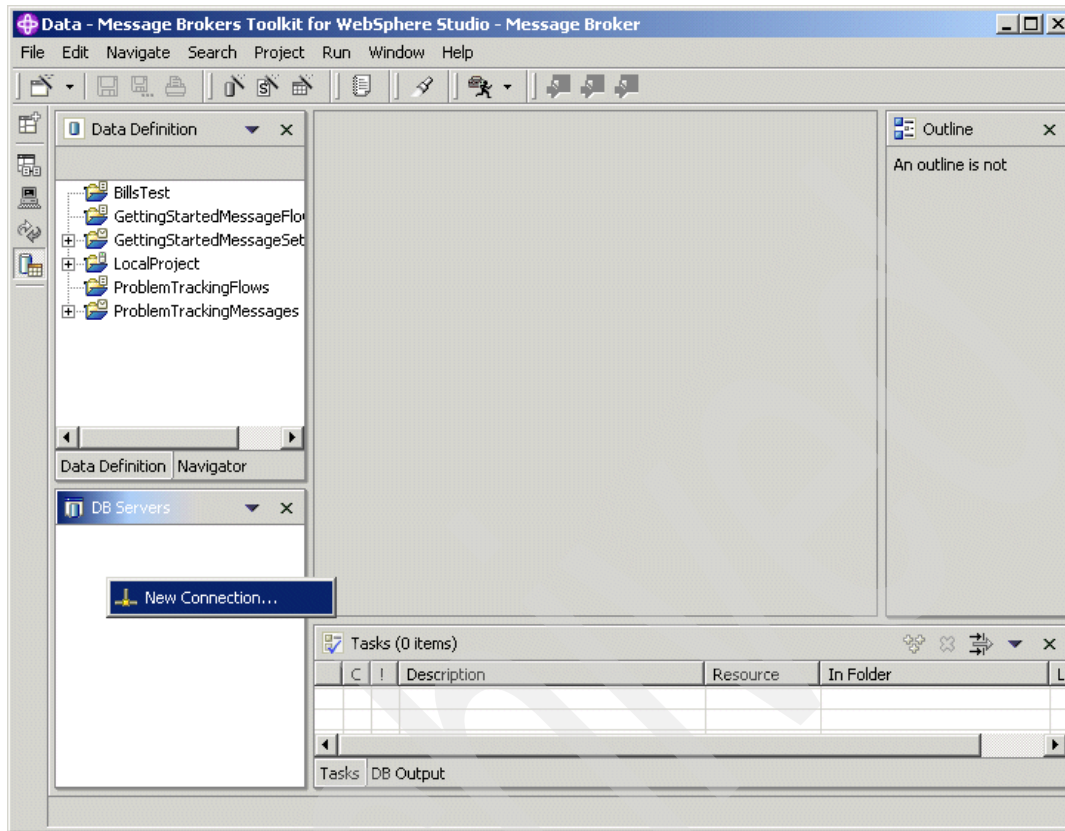


Figure 27-5 Add a new database connection

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 location is C:\sqlib\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 shown in Figure 27-6 on page 526.

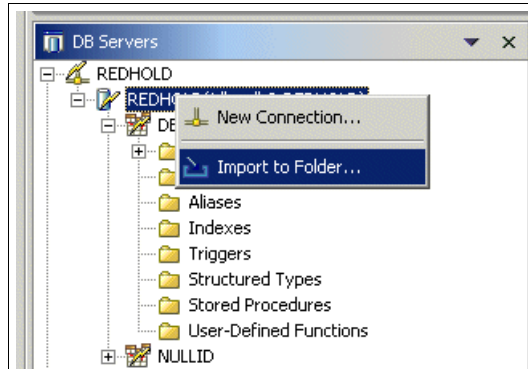


Figure 27-6 Import database to Toolkit folder

10. Select the RedMaintenanceFlows project and click **OK**.
11. Click **Finish**. You see the database in the RedMaintenanceFlows project, as shown in Figure 27-7.

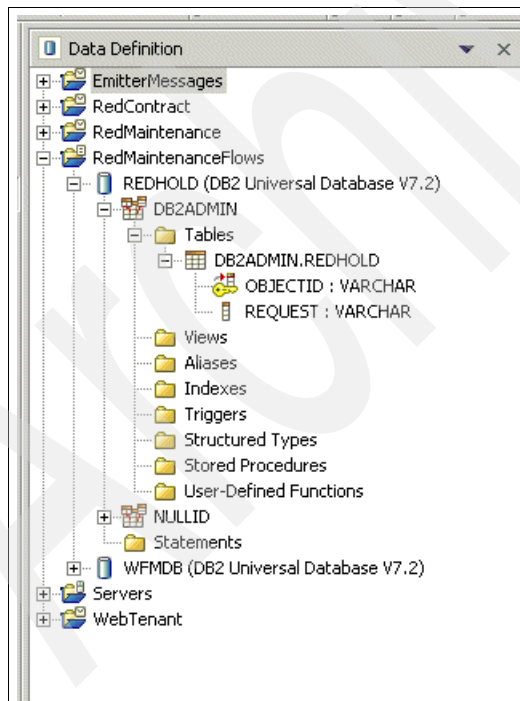


Figure 27-7 Database in Message Flow Project

12. Navigate back to the Broker Application Development perspective. You can add the database to your DataInset node.and map the insert.
13. Open the properties of the DataInsert node and add the REDHOLD data source, as shown in Figure 27-8.

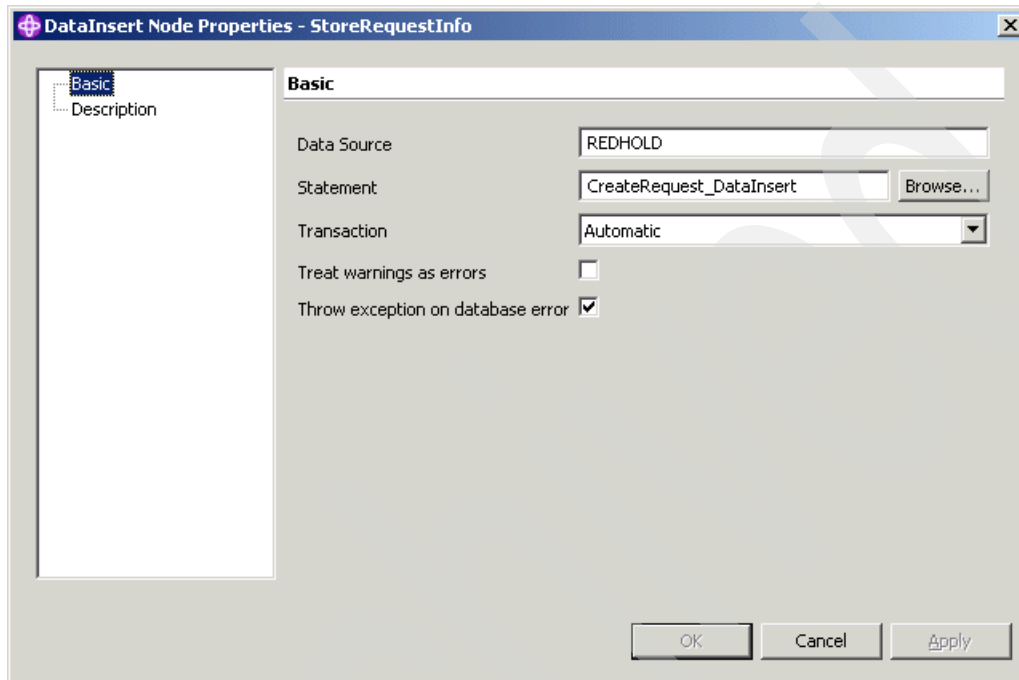


Figure 27-8 Add Data Source

14. Click **Apply**.
15. Right-click the Data Insert node and select **Open Mappings**, as shown in Figure 27-9 on page 528.

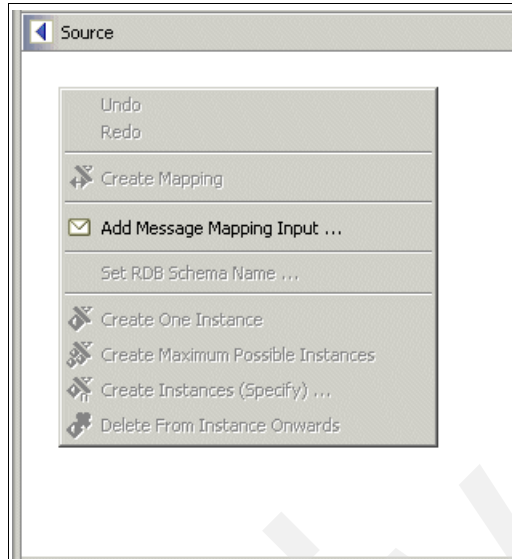


Figure 27-9 Add message mapping

16. Right-click the Source pane of the Mapping Editor.
17. Select **Add Message Mapping Input**, as shown in Figure 27-10 on page 529.

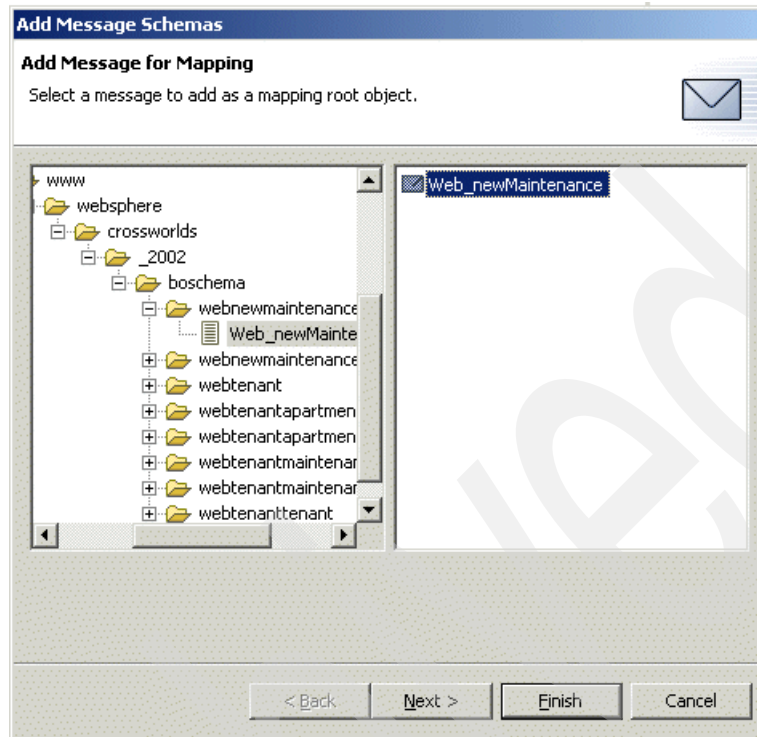


Figure 27-10 Select message

18. Select the Web_newMaintenance message.

19. Select **Finish**.

20. Right-click in the Target pane of the Mapping Editor and select **Add RDB Table Mapping Output** as shown in Figure 27-11.

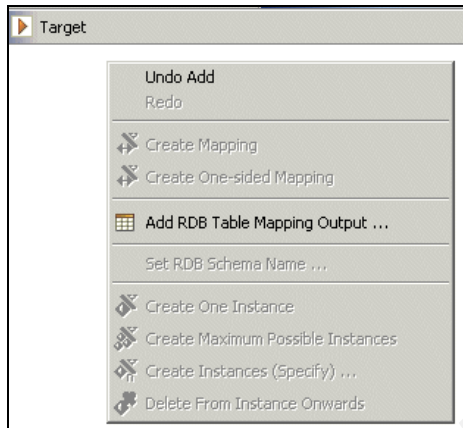


Figure 27-11 Select target

21. Select **Add database table schemas from work space**, as shown in Figure 27-12, and click **Next**.

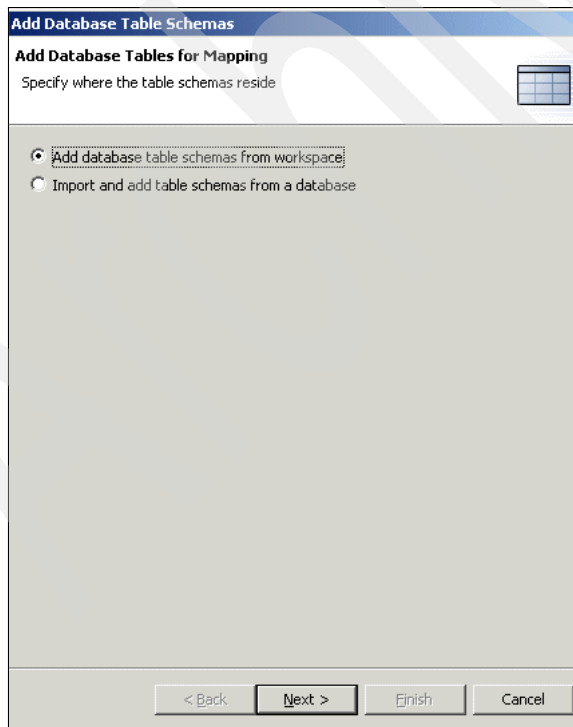


Figure 27-12 Add database table

22. Select the REDHOLD database that you added to the Message Flow Project, as shown in Figure 27-13, and click **Finish**.

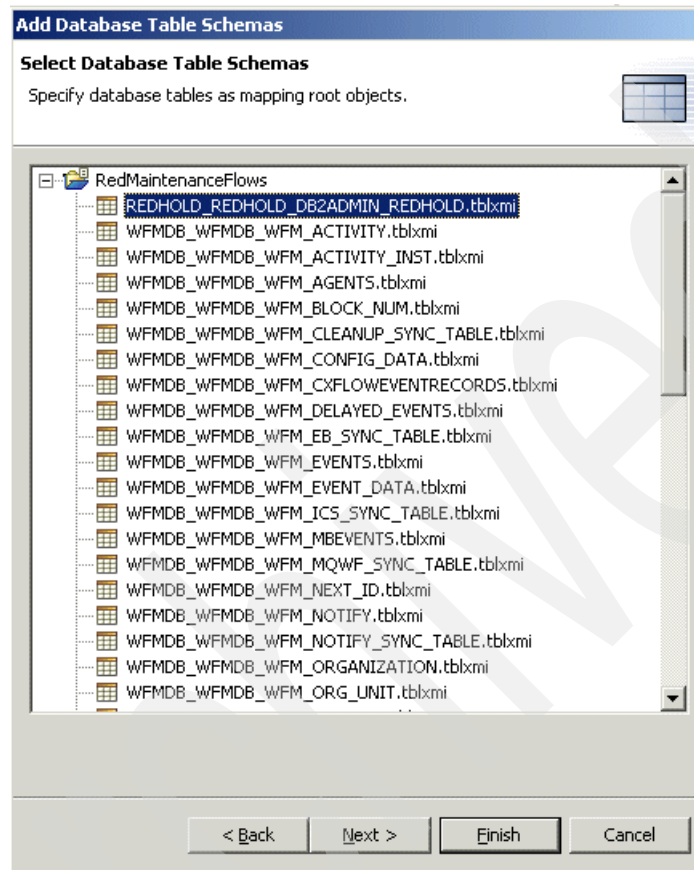


Figure 27-13 Add database

23. Your Source and Target look as shown in Figure 27-14 on page 532.
Mapping can be accomplished easily using the drag-and-drop function.

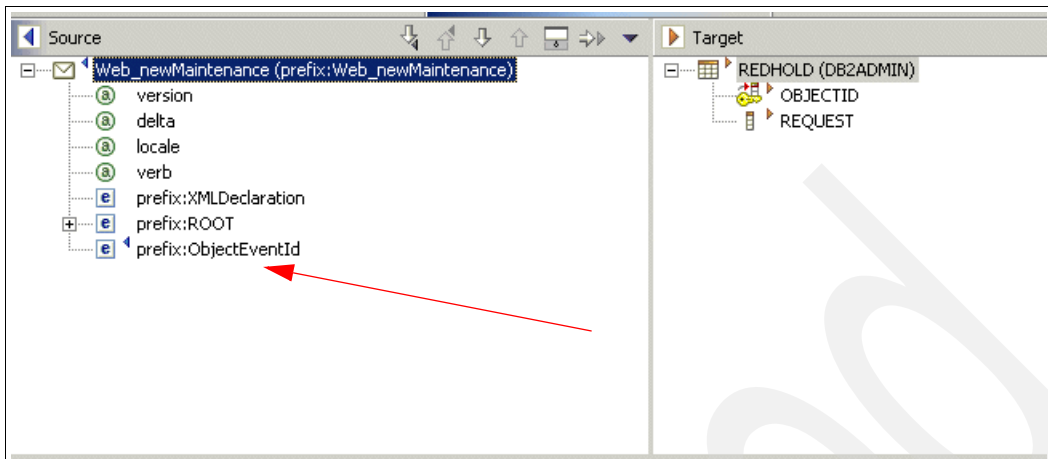


Figure 27-14 Source and Target

24. Drag the ObjectEventId from the Source to the OBJECTID of the Target, as shown in Figure 27-15.

Note: Make sure you take the ObjectEventId at prefix:ObjectEventId, not the one at prefix:ROOT.ObjectEventId.

Target	Source	Target value
REDHOLD (DB2ADMIN)	Web_newMaintenance (prefix:Web_newMaintenance)	
OBJECTID	Web_newMaintenance_newMaintenance:description, ...	s_Web_newMaintenance.prefix:ObjectEventId
REQUEST	Web_newMaintenance_newMaintenance:description, ...	s_Web_newMaintenance.prefix:ROOT.Web_newMaintenance...

Figure 27-15 Mapping field to field

25. Drag the description from the Source to REQUEST of the Target.

This action maps from one field to another and creates an insert request for each. You need a single, consolidated insert to the database.

26. In the Outline View, highlight both of the mappings, as shown in Figure 27-16. Then, right-click and select **Combine to Same Row**.

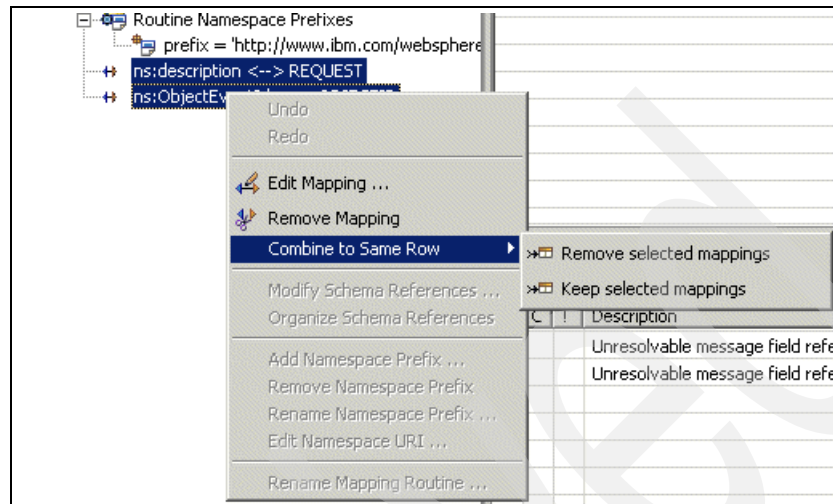


Figure 27-16 Combine mappings

27. Select **Remove Selected Mappings** so that a single insert and none of the original extraneous mappings remain.
28. You can check that your mappings are correct for each field by right-clicking each element in the Overview pane, as shown in Figure 27-17 and Figure 27-18 on page 534.

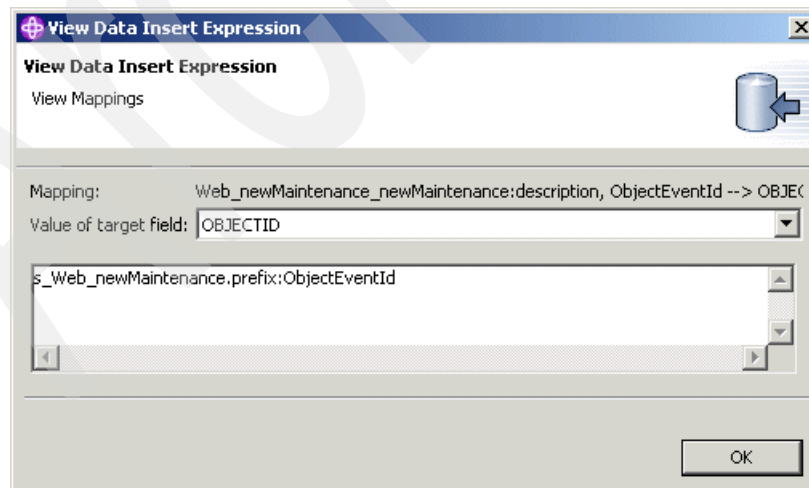


Figure 27-17 ObjectEventId

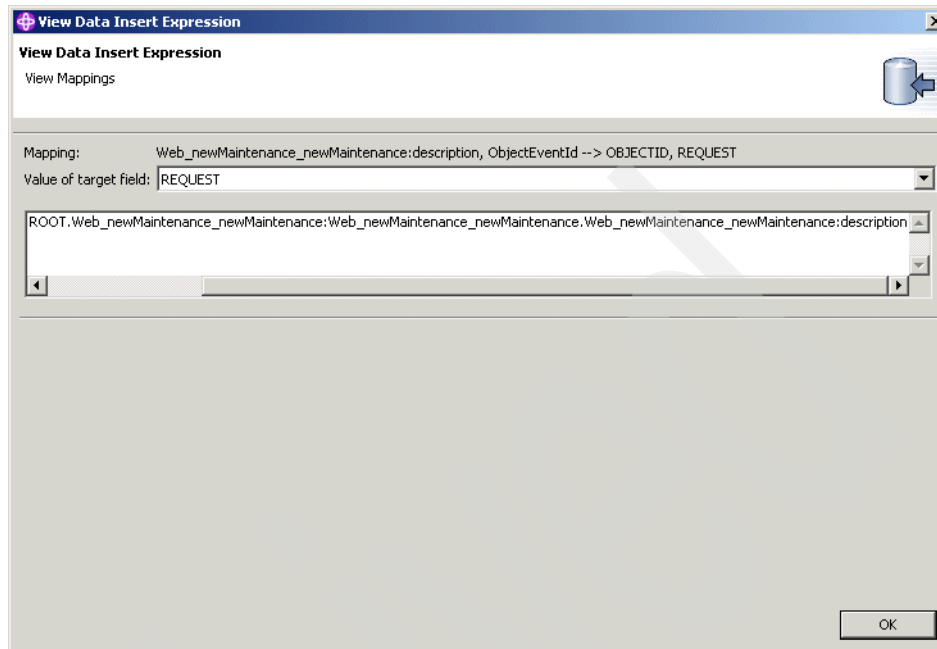


Figure 27-18 REQUEST

29. Save the mappings.

Note: We have used a database to temporarily store the request information. This storage is required to persist the information 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.

30. Open the compute node properties (Figure 27-19).
31. Set the ESQL Module to RetrieveByContent.
32. Set the Compute Mode to LocalEnvironment and Message.
33. Click **Apply** to save the changes.

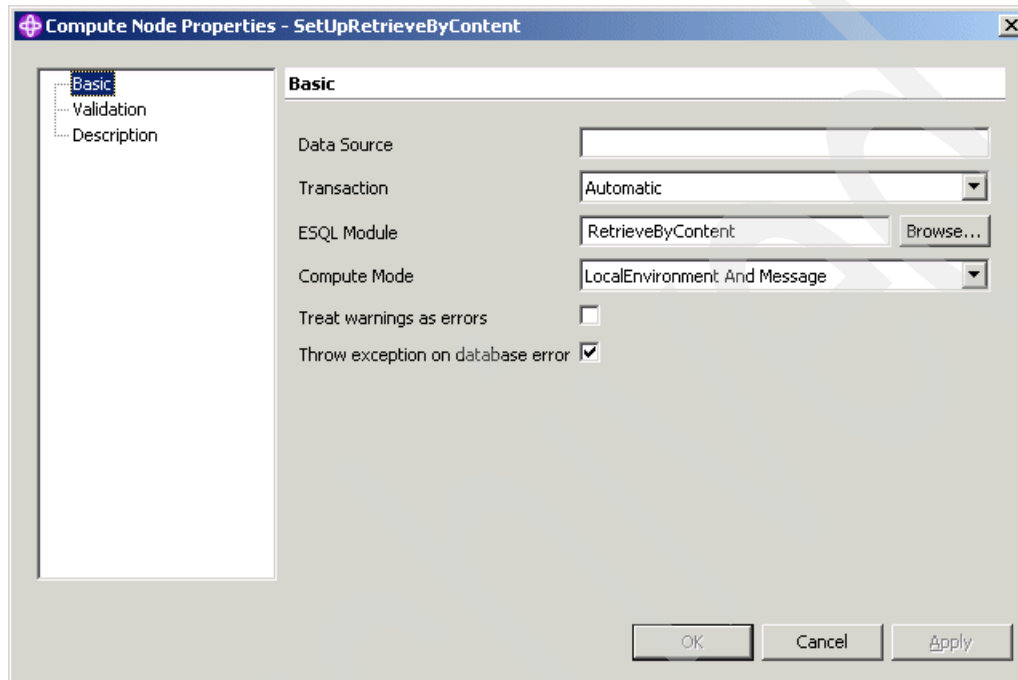


Figure 27-19 Compute node

34. Open the ESQL for this node.
35. Create the ESQL as shown in Example 27-1 on page 536.
 - The ESQL performs the following:
 - a. Copies the message headers.
 - b. Declares the required namespaces.
 - c. Sets the properties for Message Set (RedTenant) and Message Type (RM_Tenant).
 - d. Sets the MQMD for an RHF2 header.

- e. Sets the following RFH2 properties:
 - Format = MQSTR
 - Message Domain = mrm
 - Message Set = RedTenant
 - Message Type = RM_Tenant
 - Wire Format = CwXML
 - Reply queue = REDMAINT.SYNCHRONOUSRESPONSEQUEUE
- f. Sets the schema attributes.
- g. Sets the following business object fields:
 - Output Tenant Id = 0
It is a function of the API that if a mandatory field is not being used in a RetrieveByContent that it still must have a value. Setting this to zero lets 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. Sets the last of the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 27-1 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
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant';
    DECLARE Web_tenant_tenant NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant';
    DECLARE ns NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance';
    DECLARE null1 NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance_newMaintenance';

    -- Red Maintenance Back-end Application
    DECLARE RM_Tenant NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';
  
```

```

-- ** 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 B0 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:Name =
InputRoot.MRM.ns:ROOT.null1:Web_newMaintenance_newMaintenance.null1:tenantId;
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;

```

36. Save the ESQL.

The Message Flow should now look as shown in Figure 27-20, with any additional tracing nodes that you might have added.

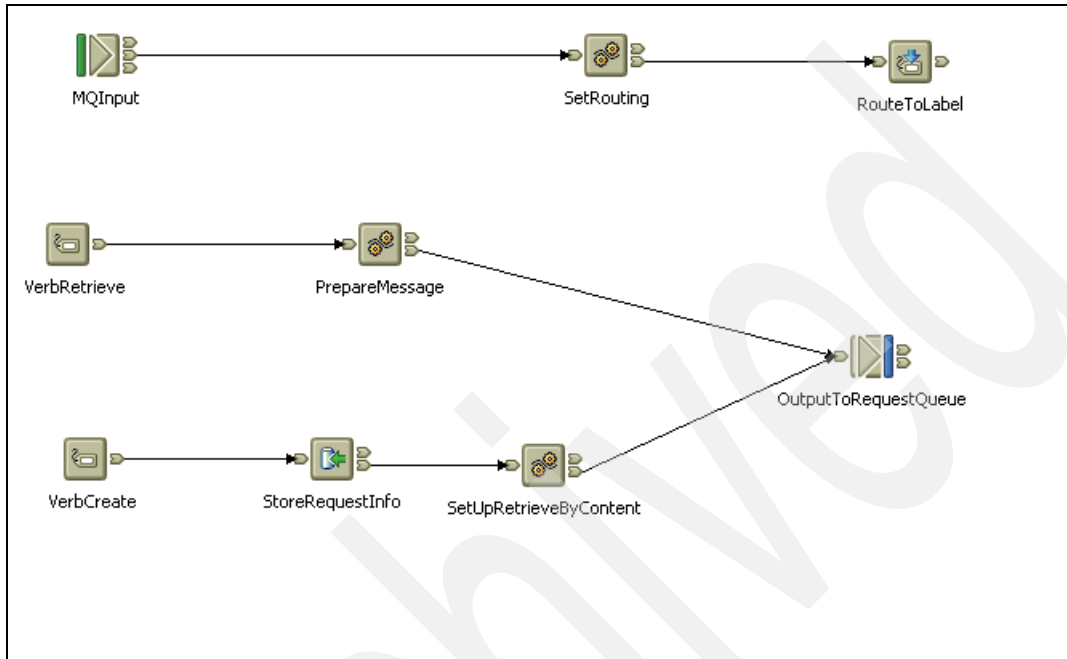


Figure 27-20 Retrieve flow so far

37. Save the Message Flow.

27.3 Deploying the Message Flows

To deploy the Message Flows:

1. Go back to the Broker Administration perspective.
2. Using the technique that is described in 26.8, “Deploying the Message Flows” on page 511, add the changed flow to the Broker Archive file.
3. Save this updated BAR file.
4. Redeploy the file to the RedMaintenance Execution Group.

27.4 Unit testing the flow

You 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. To unit test the flow:

1. Stop the RMConnector if it is running.
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. Verify that the file looks OK as a `Web_newMaintenance` business object, as shown in Figure 27-21, to be delivered to the broker. Make a note of the tenant name, maintenance description, and the `ObjectEventId`.

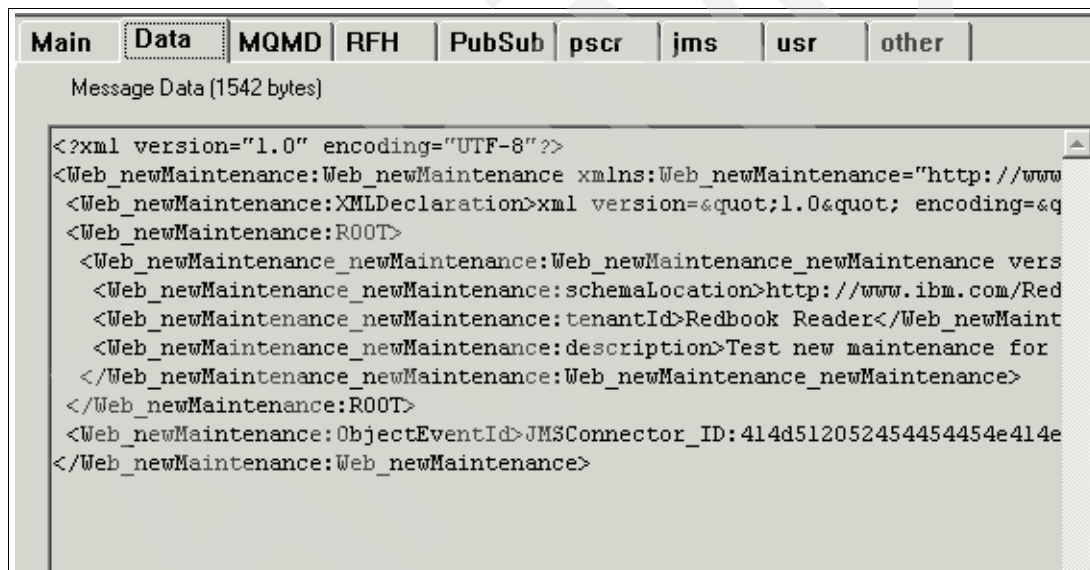


Figure 27-21 Test message

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 27-22 on page 540).

```
Message Data (641 bytes)

<?xml version="1.0"?>
<RM_Tenant:RM_Tenant xmlns:RM_Tenant="http://www.ibm.com/websphere/crosswor
<RM_Tenant:Id>0</RM_Tenant:Id>
<RM_Tenant:Name>Redbook Reader</RM_Tenant:Name>
<RM_Tenant:ObjectEventId>JMSConnector_ID:414d512052454454454e414e542020200
</RM_Tenant:RM_Tenant>
```

Figure 27-22 *RetrieveByContent* message

9. Verify that the message has the tenant fields as set in the ESQL.
10. Verify that the ObjectEventId is the same as the original. You need to preserve this information for the complete create process.
11. Verify that the verb on the message is RetrieveByContent.
12. Open a DB2 command window, as shown in Figure 27-23, and verify that the row has been successfully inserted into the REDHOLD table with the correct description and key (ObjectEventId).

Interactive	Script	Results	Access Plan
Perform required changes and then click the commit update button.			
OBJECTID		REQUEST	
JMSConnector_ID:414d512052454454454e414e5420202007bda34120000413		Test new maintenance for Redbook Reader	

Figure 27-23 *Inserted row*

13. Start the RMConnector.
14. Check the connector log for successful retrieval of the correct tenant details.
15. Using `rfhutil`, browse REDMAINT.SYNCHRONOUSRESPONSEQUEUE for the RM_Tenant business object (see Figure 27-24 on page 541).

```
Message Data (4822 bytes)

<?xml version="1.0" encoding="UTF-8"?>
<?CWRSD status="1" description="null"?>
<RM_Tenant:RM_Tenant xmlns:RM_Tenant="http://www.ibm.com/websphere/crosswor
<RM_Tenant:Id>102</RM_Tenant:Id>
<RM_Tenant:Name>Redbook Reader</RM_Tenant:Name>
<RM_Tenant:ApartmentId>3</RM_Tenant:ApartmentId>
<RM_Tenant:EMail>redbookreader@us.ibm.com</RM_Tenant:EMail>
<RM_Tenant:RM_Apartment>
  <RM_Apartment:RM_Apartment version="3.0.0" verb="Retrieve" locale="en_US"
  <RM_Apartment:Id>3</RM_Apartment:Id>
  <RM_Apartment:ApartmentNumber>3039</RM_Apartment:ApartmentNumber>
  <RM_Apartment:AddressLine1>Cornwallis Road</RM_Apartment:AddressLine1>
  <RM_Apartment:AddressLine2>RTP</RM_Apartment:AddressLine2>
  <RM_Apartment:AddressLine3>NC</RM_Apartment:AddressLine3>
  <RM_Apartment:AddressLine4>USA</RM_Apartment:AddressLine4>
  <RM_Apartment:PostCode>27709          </RM_Apartment:PostCode>
</RM_Apartment:RM_Apartment>
</RM_Tenant:RM_Apartment>
```

Figure 27-24 Tenant data

If this information is correct, you are now ready to reconstruct the create request and send it on to the RMConnector.

16. Stop the RMConnector.

27.5 RedMaint_RetrieveTenantByContent

To create RedMaint_RetrieveTenantByContent:

1. Navigate to the Broker Application Development perspective.
2. In the RedMaintenanceFlows Message Flow Project, create a new flow and name it RedMaint_RetrieveTenantByContent.
3. Drag the following nodes to the canvas and connect, as shown in Figure 27-25 on page 542:
 - MQInput
 - Compute (PrepareCreateRequest)
 - MQOutput (SendForCreate)



Figure 27-25 *RedMaint_RetrieveTenantByContent*

4. Open the MQInput node properties and set the queue name to REDMAINT.SYNCHRONOUSRESPONSEQUEUE, as shown in Figure 27-26.

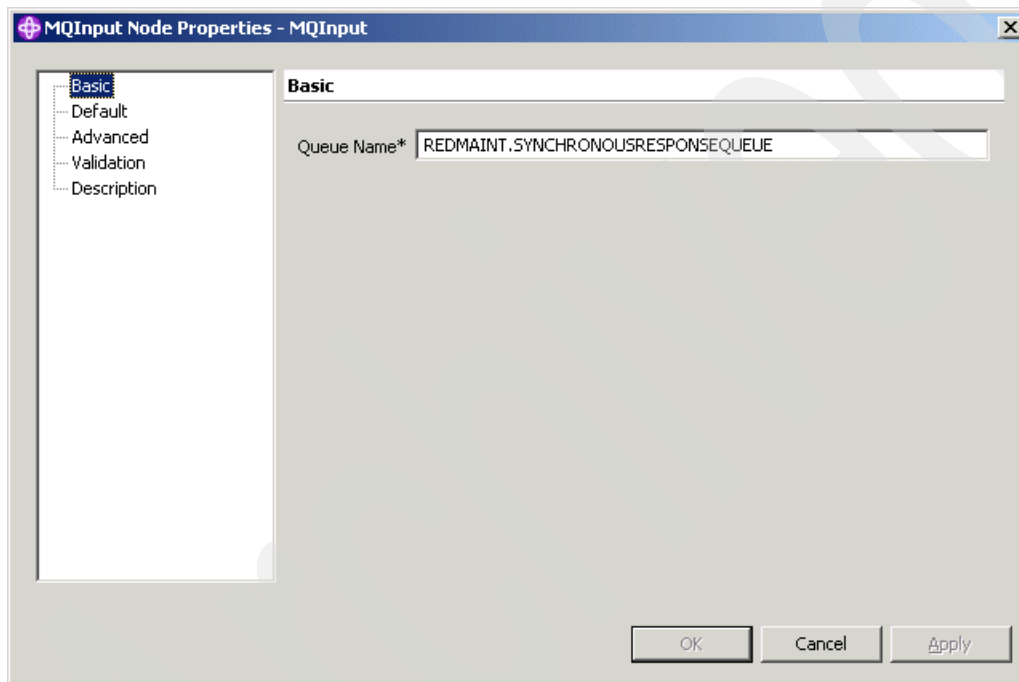


Figure 27-26 *Input queue*

5. Open the Compute node (see Figure 27-27 on page 543).

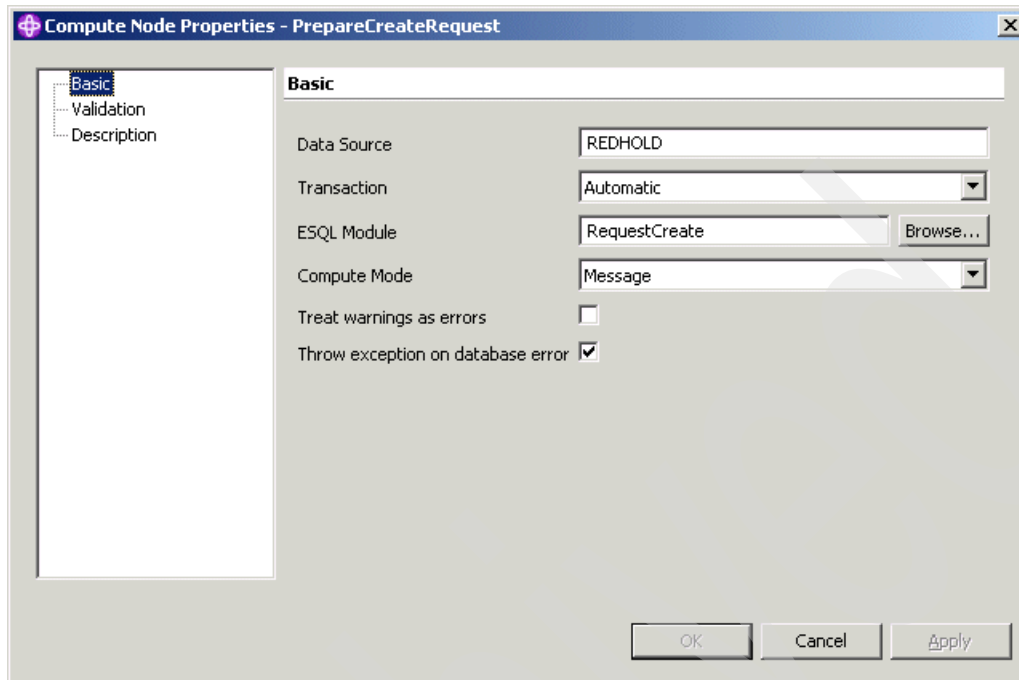


Figure 27-27 Compute node properties

6. Set the Data Source property to REDHOLD, because you will be performing database operations.
7. Set the ESQL Module to RequestCreate.
8. Click **Apply**.
9. Create the ESQL as shown in Example 27-2 on page 544.

The ESQL does the following:

- a. Copies the message headers.
- b. Sets the required namespace declarations.
- c. Sets the properties for Message Set (RedTenant) and Message Type (RM_Maintenance).
- d. Sets the RFH2 header in the Message Descriptor.

- e. Sets the following 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. Sets the schema attributes (remember the verb is now Create).
- g. Sets the following business object data from the message:
 - Output apartment ID is the Input apartment ID.
 - Output tenant ID is the Input tenant ID.
- h. Sets the business object data that was previously stored (using the ObjectEventId as the key). Output problem description is the REQUEST column data from the table.
- i. Sets the Output ObjectEventId from the Input ObjectEventId.
- j. Deletes the record from the table because it is no longer required.
- k. Sets the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 27-2 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
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';
    DECLARE RM_Maintenance NAMESPACE
    'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance';

    -- ** 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.
-- ** 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_Maintenance';
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" = 'Create';
SET OutputRoot.MRM."locale" = 'en_US';

-- Set B0 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];
    SET I = I + 1;
END WHILE;
END;

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

```

10. Save the ESQL.

11. Open the properties for the MQOutput node, as shown in Figure 27-28.

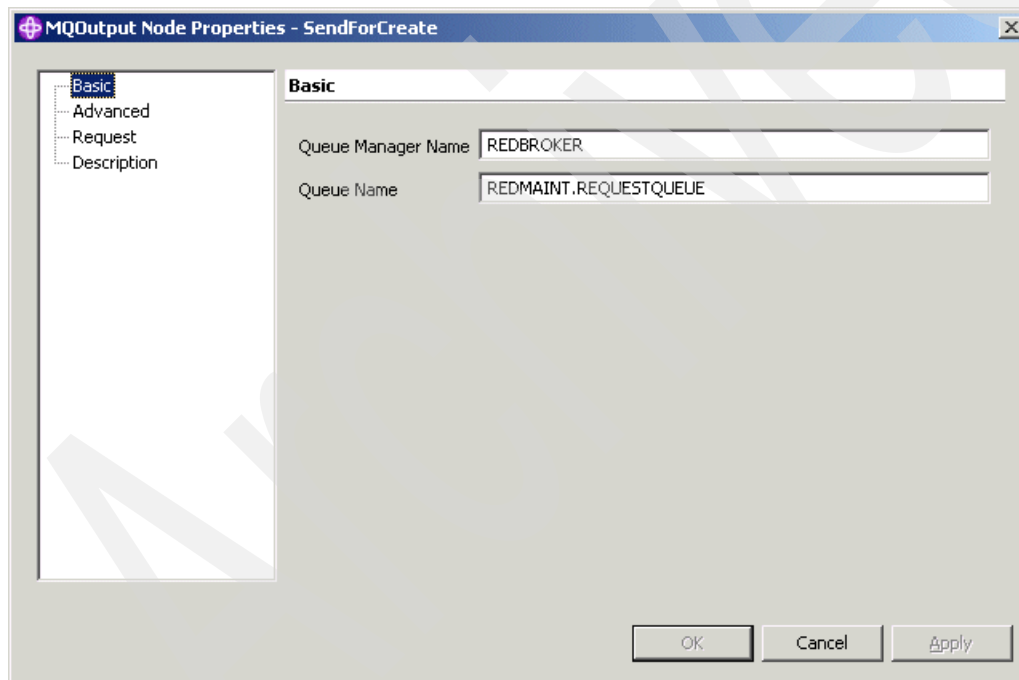


Figure 27-28 MQOutput node

12. Set the queue to the REDMAINT.REQUESTQUEUE.

13. Save the flow.

27.6 Deploying RedMaint_RetrieveTenantByContent

To deploy this flow:

1. Go back to the Broker Administration perspective.
2. Using the technique that is described in 26.8, “Deploying the Message Flows” on page 511, add the changed flow to the Broker Archive file.
3. Save this updated BAR file.
4. Redeploy to the RedMaintenance Execution Group.

27.7 Unit testing RedMaint_RetrieveTenantByContent

If your Message Flow is functioning correctly, as soon as it is deployed successfully, the message that is on it from the previous test will be picked up by the flow. The new RM_Maintenance message should now be sitting on the REDMAINT.REQUESTQUEUE waiting for the RMConnector (Figure 27-29).

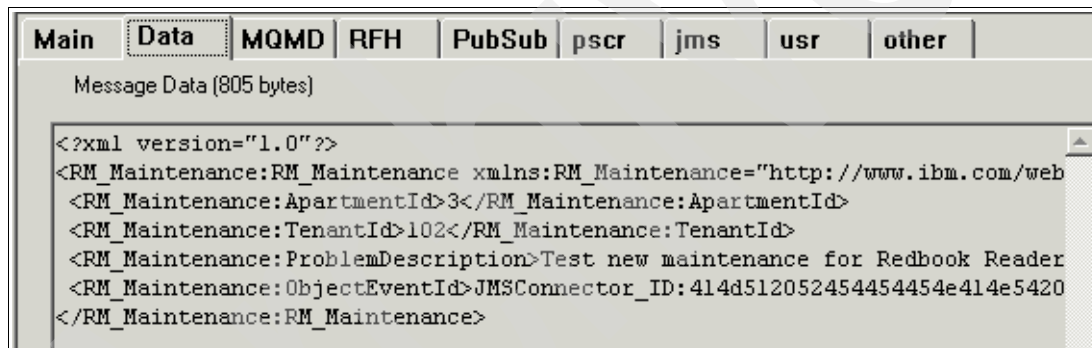


Figure 27-29 Request queue

To test the message flow:

1. Verify 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. Verify the message on the response queue for the new maintenance ID on the business object (Figure 27-30).

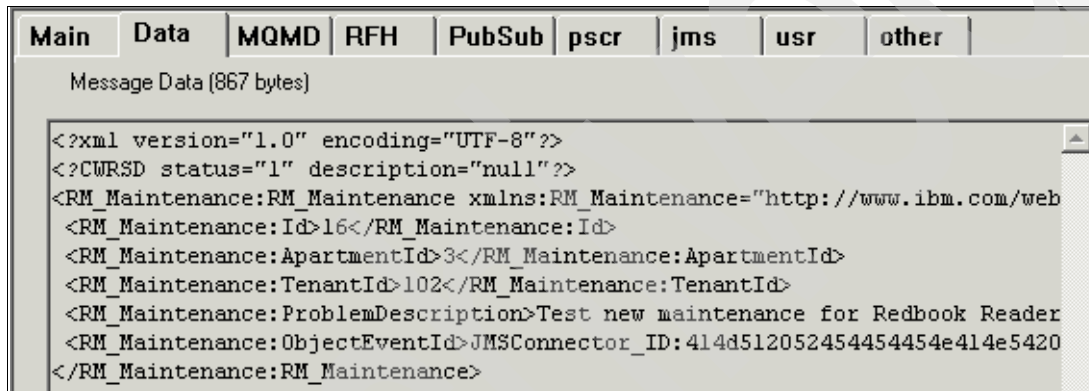


Figure 27-30 Response message

You are now ready to complete the round trip for a creation request.

27.8 RedMaint_to_RedTenant_Response

To create RedMaint_to_RedTenant_Response:

1. Go to the Broker Development perspective and open the RedMaint_to_RedTenant_Response flow (see Figure 27-31).

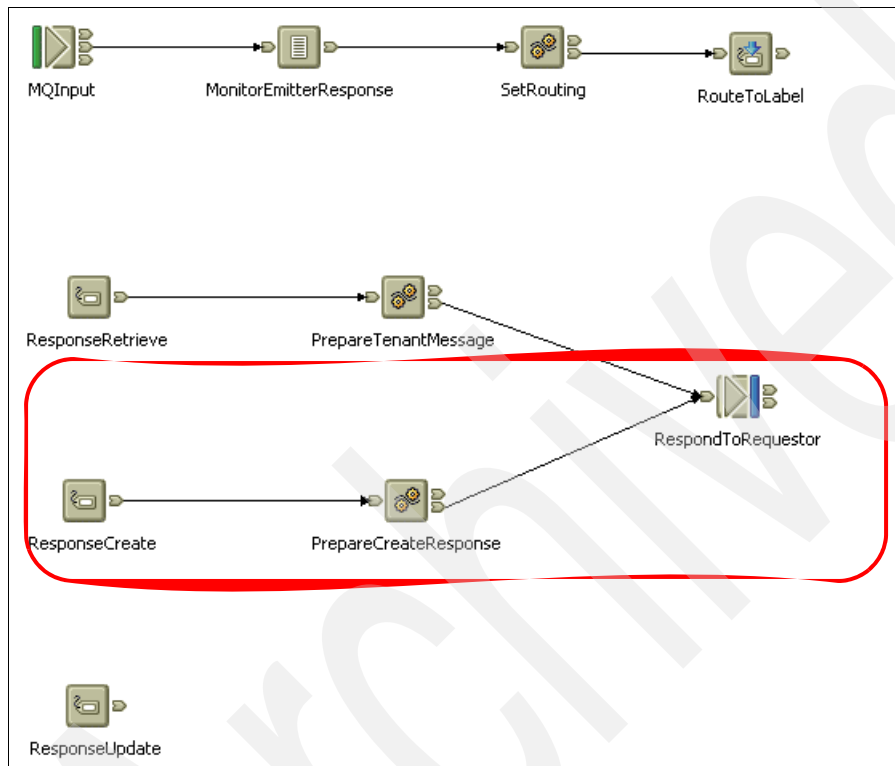


Figure 27-31 Response flow

2. Add the following nodes and connect as shown in Figure 27-31.
 - Label (ResponseRetrieve)
 - Compute (PrepareCreateResponse).

3. Open the properties for the Label node, and set the LabelName to ResponseCreate, as shown in Figure 27-32.

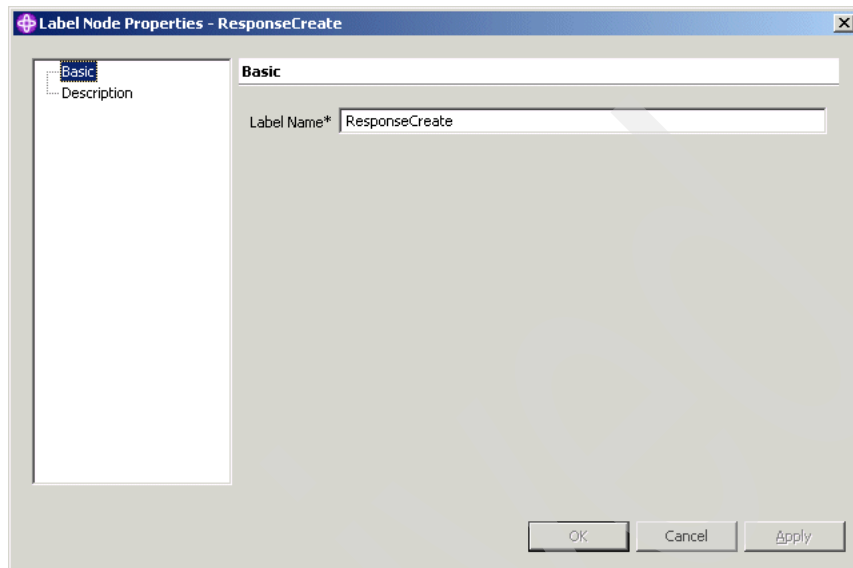


Figure 27-32 Label

4. Open the properties for the Compute node, and set the ESQL module to ResponseCreate, as in Figure 27-33.

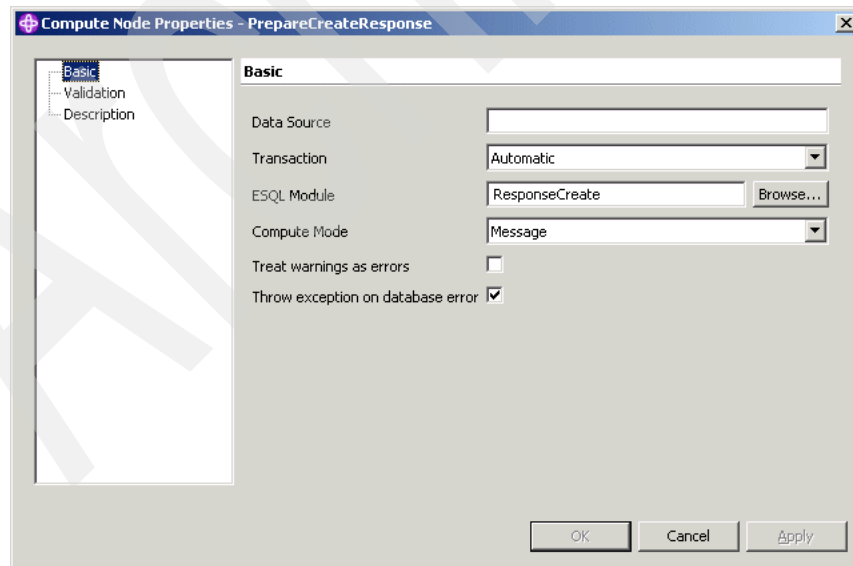


Figure 27-33 Compute node

5. Click **Apply**.
6. Complete the ESQL as shown in Example 27-3.

The ESQL does the following:

 - a. Copies the message headers.
 - b. Sets any required namespace declarations.
 - c. Sets the properties of Message Set (WebTenant) and Message Type (Web_newMaintenance).
 - d. Sets the Message Descriptor to contain an RFH2 header.
 - e. Sets the following RFH header properties:
 - Format = MQSTR
 - Message Domain = mrm
 - Message Set = WebTenant
 - Message Type = Web_newMaintenance
 - Wire Format = CwXML
 - Reply q is WEBTENANT.RESULT
 - f. Sets the schema attributes.
 - g. Sets the following business object data:
 - XLM Declaration
 - Schema location for the application schema
 - Set the Output message maintenance ID to the Input message maintenance ID.
 - h. Sets the properties for Message Domain (MRM) and Wire Format (CwXML).

Example 27-3 ResponseCreate

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

    -- *** Schema Declarations ***

    -- Red Tenant Web Application
    DECLARE Web_newMaintenance NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance';
    DECLARE Web_newMaintenance_newMaintenance NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance_newMaintenance';
```

```

-- Red Maintenance Back-end Application
DECLARE RM_Maintenance NAMESPACE
'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance';

-- ** 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 = '  ';

-- ** 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_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 B0 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

```

```

SET
OutputRoot.MRM.Web_newMaintenance:ROOT.Web_newMaintenance_newMaintenance:Web_newMaintenance_new
Maintenance.Web_newMaintenance_newMaintenance:schemaLocation =
  '"http://www.ibm.com/RedMaintenance maintenance.xsd"' ;

-- ** Set the maintenance details

SET
OutputRoot.MRM.Web_newMaintenance:ROOT.Web_newMaintenance_newMaintenance:Web_newMaintenance_new
Maintenance.Web_newMaintenance_newMaintenance:id =
  InputRoot.MRM.RM_Maintenance:Id;

-- ** 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;

```

7. Save the ESQL.

The Message Flow should now look similar to that shown in Figure 27-34 on page 554.

8. Save the Message Flow.

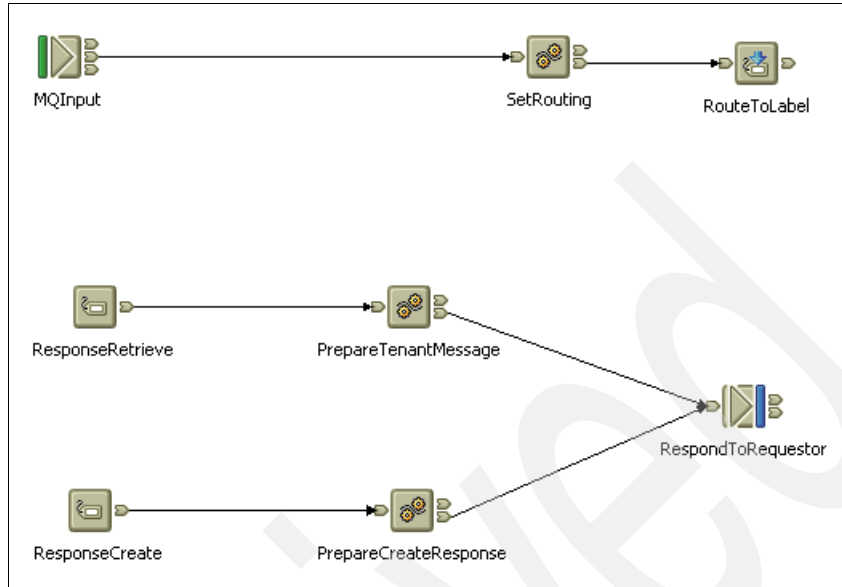


Figure 27-34 Message Flow so far

27.9 Deploying RedMaint_to_RedTenant_Response

To deploy Redmaint_to_RedTenant_Response:

1. Go back to the Broker Administration perspective.
2. Using the technique that is described in 26.8, “Deploying the Message Flows” on page 511 to add the changed flow to the Broker Archive file.
3. Save this updated BAR file.
4. Redeploy to the RedMaintenance Execution Group.

27.10 Unit testing RedMaint_to_RedTenant_Response

Using the techniques learned so far, you can unit test the transformation of the response message to the front-end application response queue. Check the metaobject configuration, if necessary, to see where the JMSConnector should put the application response for a create request. This business object is the JMS_MO_Config business object.

27.11 End-to-end testing

You are now ready for the final test of the integration with the front-end application. The front-end application is responsible for the retrieve and create operations, so we will run through the entire process. To test end-to-end:

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 27-35), enter a tenant ID.

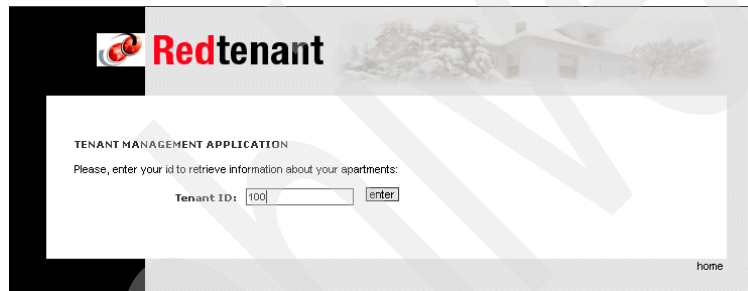
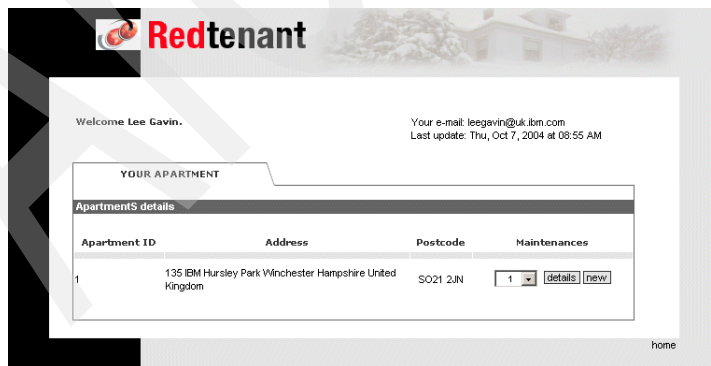


Figure 27-35 Enter an ID

7. Wait for the tenant details, as shown in Figure 27-36.



Apartment ID	Address	Postcode	Maintenances
1	135 IBM Hursley Park Winchester Hampshire United Kingdom	SO21 2JN	1 <input type="button" value="details"/> <input type="button" value="new"/>

Figure 27-36 Tenant details

8. When the response from the request returns, create a new maintenance request, as shown in Figure 27-37.

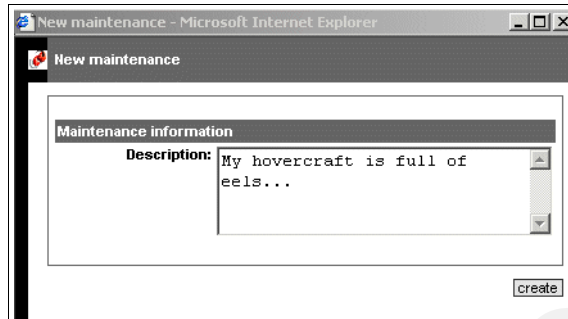


Figure 27-37 Maintenance request

9. Wait for the response from the application to notify that the request has been received, as shown in Figure 27-38.

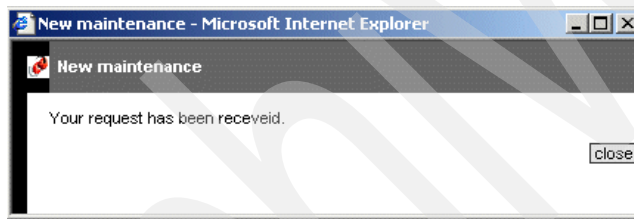


Figure 27-38 Request received

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.

Building and testing 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 monitor this component with the WebSphere Business Integration Monitor in the same way that we monitor our retrieve and create processing.

Because the current version of the Monitor does not accept feeds from Server Foundation, we will use a special node for the Message Broker that captures information for the Monitor at the start of both the Request and Response flows. To capture this information for the update processing, we set up a pass-through of sorts and catch the messages from Server Foundation.

This chapter explains how to modify the flows to enable this pass-through. (We will describe the pass-through function in Part 7, “Server Foundation components” on page 631.)

28.1 RedTenant_to_RedMaint_Request

We need to add the last of the label processing for the request flow, as shown in Figure 28-1.

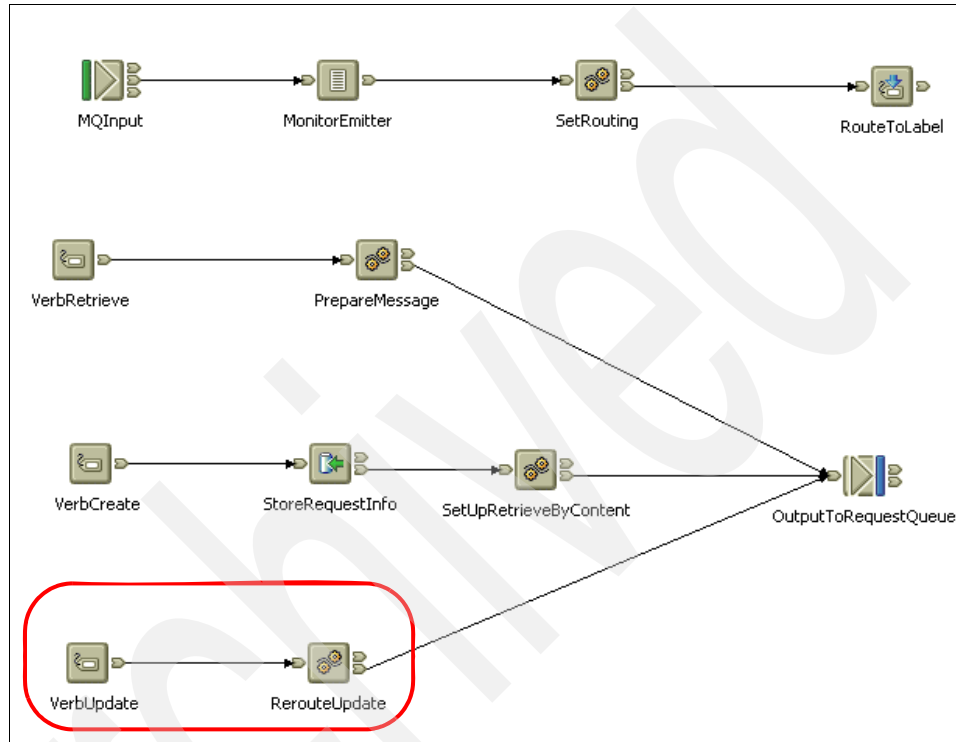


Figure 28-1 Request flow

To add the last label processing:

1. Open the Broker Application Development perspective.
2. Add the following nodes to the canvas and connect as shown in Figure 28-1:
 - Label (VerbUpdate)
 - Compute (RerouteUpdate)
3. Open the Label node and set the Label Name to VerbUpdate, as shown in Figure 28-2 on page 559.

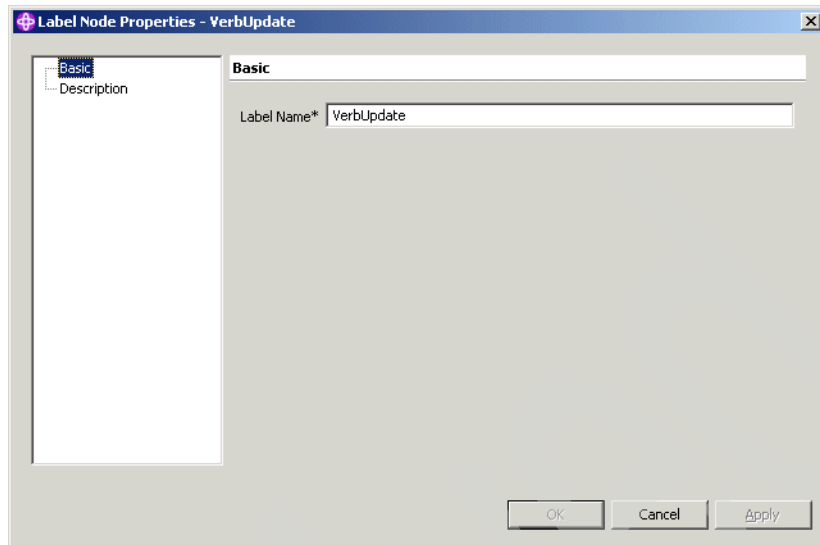


Figure 28-2 Label node

4. Open the Compute node properties, as shown in Figure 28-3.

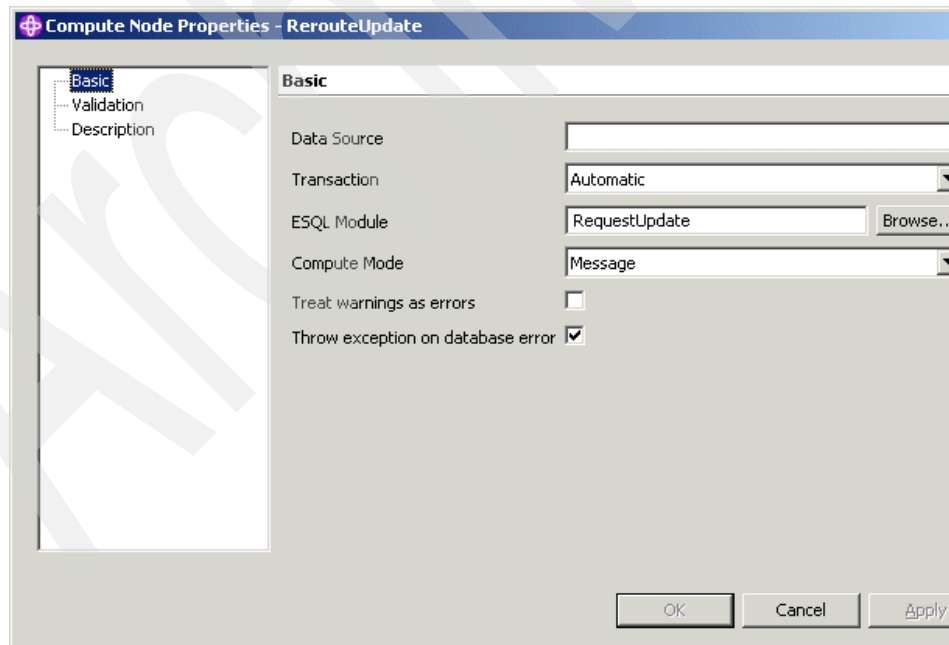


Figure 28-3 Compute node

5. Set the ESQL Module Name to RequestUpdate.
6. Complete the ESQL as shown in Example 28-1.
The ESQL does the following:
 - a. Copies the Input message to the Output message
 - b. Sets the reply queue in the JMS header to REDMAINT.RESPONSEQUEUE

Example 28-1 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.
9. Deploy the BAR file.

28.2 RedMaint_to_RedTenant_Response

In the response flow, you do not need to do anything with the response message from the adapter, because the original message was an Agent Delivery from the Server Foundation process. We merely allow the response to pass through to capture the emitter data that is required. For completeness, we add a label node that does not perform any processing.

To add the node:

1. Open the Label node properties and give the label a name of ResponseUpdate, as shown in Figure 28-4.

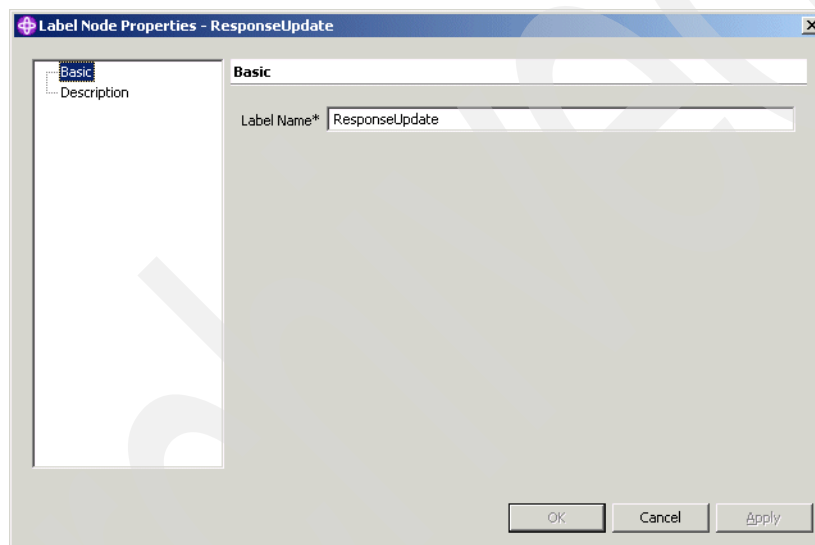


Figure 28-4 Label node

2. Save the Message Flow.
3. Add the modified flow to the BAR file.
4. Deploy the BAR file.



Part 6

Business-to-business components

Archived

Integrating external contractors using WebSphere BI Connect

We have now built the basic infrastructure for the interaction between the front-end and back-end applications in the enterprise. As shown in Figure 19-2 on page 310, we potentially have external contractors (business-to-business trading partners) which carry out the maintenance to integrate into our overall WebSphere BI solution. We use WebSphere Business Integration Connect to integrate these external contractors.

Note: For more detailed information regarding WebSphere Business Integration Connect see the following resources:

- ▶ *B2B Solutions using WebSphere BI Connect Version 4.2.2*, SG24-6355
- ▶ *EDI Solutions using WebSphere BI Connect Version 4.2.2*, SG24-6355

For this part of our scenario, we install WebSphere Business Integration Connect Advanced (for our enterprise, known as RedMaint) and WebSphere Business Integration Connect Express (for our trading partner, known as RedContract).

For WebSphere Business Integration Connect Advanced, we install Fix Pack 3. For the Express edition, we install Fix Pack 1. The installation of these products is a standard installation that fits a single machine.

To limit the number of servers for our solution, we install both editions of WebSphere Business Integration Connect on the same computer. Thus, the business-to-business exchange between the RedMaint and the RedContract companies does not really leave the actual machine. In a real implementation, the two versions of WebSphere Business Integration Connect would be installed on separate computers.

This chapter explains a number of configuration steps, such as creating community participants. It then defines the XML format to the server and configures a business-to-business exchange where the document that is to be sent is retrieved from a directory and delivered into the target directory. When that configuration is verified, we adjust the configuration of WebSphere Business Integration Connect Advanced to read documents from a queue and send them to the company RedContract. Next, the documents that are sent by RedContract to RedMaint are delivered in a queue by WebSphere Business Integration Connect Advanced.

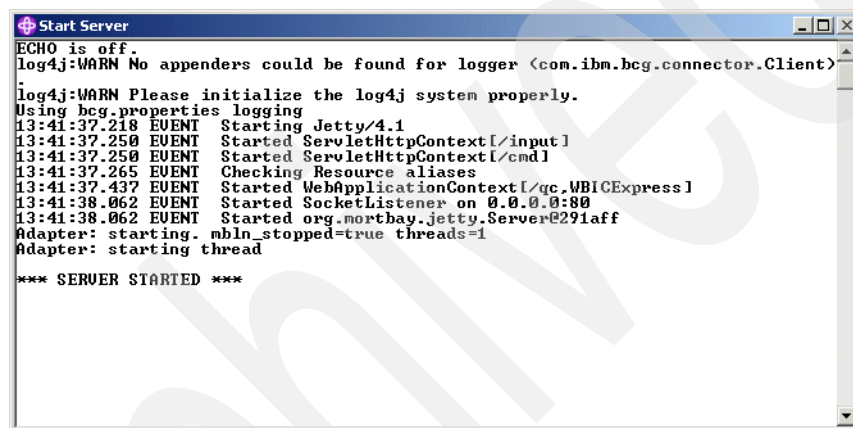
At the end of this chapter, we provide a working business-to-business solution that can then be integrated with the EAI solution that we build in WebSphere Business Integration Server Foundation.

29.1 Initial configuration of the Express server

To configure the WebSphere Business Integration Connect Express server:

1. Start the server via the Start menu (**Start** → **Programs** → **Business Integration Connect** → **Express** → **Start Server**).
2. You managing the server (including stopping it) through a browser-based console, as shown in Figure 29-1.

To open this console, a shortcut has been added to the Start Programs menu. Select **Start** → **Programs** → **Business Integration Connect** → **Express** → **Console**.



```
Start Server
ECHO is off.
log4j:WARN No appenders could be found for logger <com.ibm.bcg.connector.Client>
log4j:WARN Please initialize the log4j system properly.
Using bcg.properties logging
13:41:37.218 EVENT Starting Jetty/4.1
13:41:37.250 EVENT Started ServletHttpContext[/input]
13:41:37.250 EVENT Started ServletHttpContext[/cmd]
13:41:37.265 EVENT Checking Resource aliases
13:41:37.437 EVENT Started WebApplicationContext[/qc.WBICExpress]
13:41:38.062 EVENT Started SocketListener on 0.0.0.0:80
13:41:38.062 EVENT Started org.mortbay.jetty.Server@291aff
Adapter: starting. mbln_stopped=true threads=1
Adapter: starting thread
*** SERVER STARTED ***
```

Figure 29-1 Online logging in console window

3. When the logon panel appears, log on with the user admin and password admin, as shown in Figure 29-2.

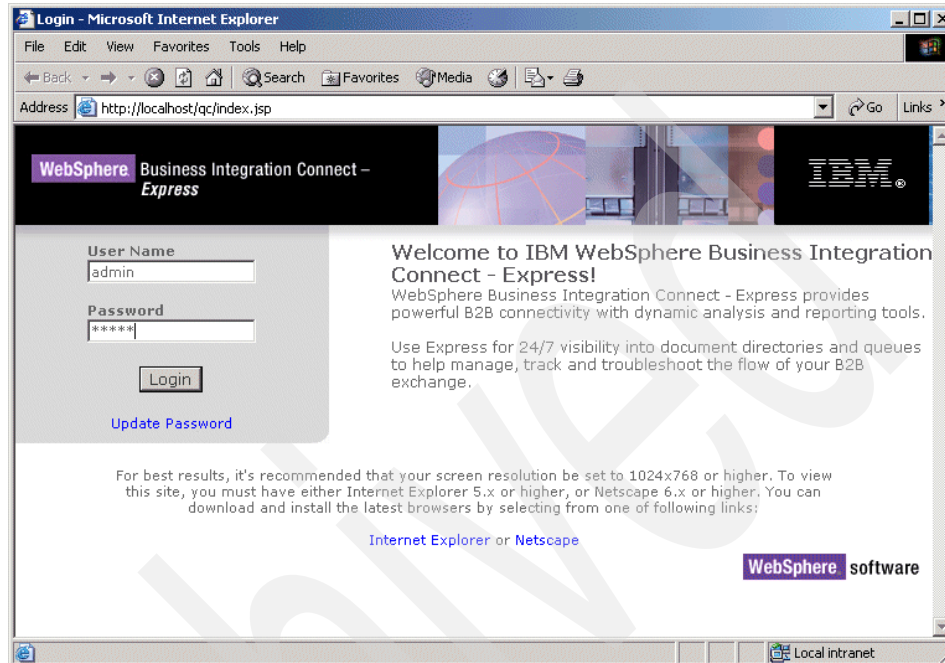


Figure 29-2 Initial logon to WebSphere Business Integration Connect Express

4. During this very first log on, you are requested to change the password of the user admin ID (Figure 29-3). You also need to change the password of another built-in user ID called Guest. Provide new passwords (we chose itso4you) for both users, and click **Save**.

WebSphere Business Integration Connect - Express

Initialize Passwords

User Name
Admin

New Password

Retype New Password

User Name
Guest

New Password

Retype New Password

Save

Passwords must be at least six alphanumeric characters in length.

Welcome to IBM WebSphere Business Integration Connect - Express!
WebSphere Business Integration Connect - Express provides powerful B2B connectivity with dynamic analysis and reporting tools.
Use Express for 24/7 visibility into document directories and queues to help manage, track and troubleshoot the flow of your B2B exchange.

For best results, it's recommended that your screen resolution be set to 1024x768 or higher. To view this site, you must have either Internet Explorer 5.x or higher, or Netscape 6.x or higher. You can download and install the latest browsers by selecting from one of following links:
[Internet Explorer](#) or [Netscape](#)

WebSphere software

Done Local intranet

Figure 29-3 Initialize passwords during initial logon

5. After setting the new passwords, you are taken back to the original main logon window. This time, log on using the user admin ID and the new password.

When the logon is complete, you are taken immediately to the Create Participant option. You need to provide some basic information about the first participant with whom the owner of this server of WebSphere Business Integration Connect Express wants to communicate and what options or features we want to use in that exchange. Except for the first field, shown in Figure 29-4 (Participant Name), all parameters can be changed later.

6. Provide a participant name, such as RedMaint, and select the inbound protocols that you want to support: HTTP or HTTPS.

The screenshot shows the 'Create Participant' page in the WebSphere Business Integration Connect Express interface. The page has a header with the product name and an IBM logo. The main content area contains several form fields and sections:

- Create Participant** (Page title) and **Logout** (Link).
- Participant Name:** A text input field containing 'RedMaint' with a '(Required field)' label.
- Document Receipt Protocol (Select at least one if receiving):** Two radio button options: **HTTP** (selected) and **HTTPS** (unselected).
- User Alerts:** A section with an **Enabled:** label and two radio button options: **Yes** (unselected) and **No** (selected).
- E-Mail Host:** A text input field with a placeholder '(eg. 12.3.12.1)'.
- Authentication Name:** A text input field.
- Authentication Password:** A text input field.
- E-Mail Recipients:** A text input field with a placeholder '(Separate addresses with commas)'.

Figure 29-4 Create participant - part 1

7. In the lower section of the Create Participant window, provide the AS2 identifier of RedMaint, which should be redmaint (Figure 29-5).
8. Select the content types that you will use. For the purposes of our scenario, select at least XML. EDI content types can be selected as well but will not be used in this scenario.
9. Click **Save** to store the new profile.

Capabilities		Protocol	Can Send	Can Receive
		HTTP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
AS2		Participant ID:	<input type="text" value="redmaint"/>	
Content Type				
	EDI-X12	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	EDIFACT	<input type="checkbox"/>	<input type="checkbox"/>	
	EDI-Consent	<input type="checkbox"/>	<input type="checkbox"/>	
	XML	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	Binary	<input type="checkbox"/>	<input type="checkbox"/>	
Binary Content Type		<input type="text"/>		
<input type="button" value="Save"/>				
<p>NOTES:</p> <ul style="list-style-type: none"> • Participant ID is required if any AS2 capabilities are enabled. • E-mail host and e-mail recipients are required if User Alerts are enabled. • Binary Content Type is required if AS2 binary capabilities are enabled. 				

Figure 29-5 Create participant - part 2

10. The Manage Participants window displays (Figure 29-6) and shows that more configuration work is required for AS2, which we explain in 29.2.4, “Creating the participant connection” on page 605.

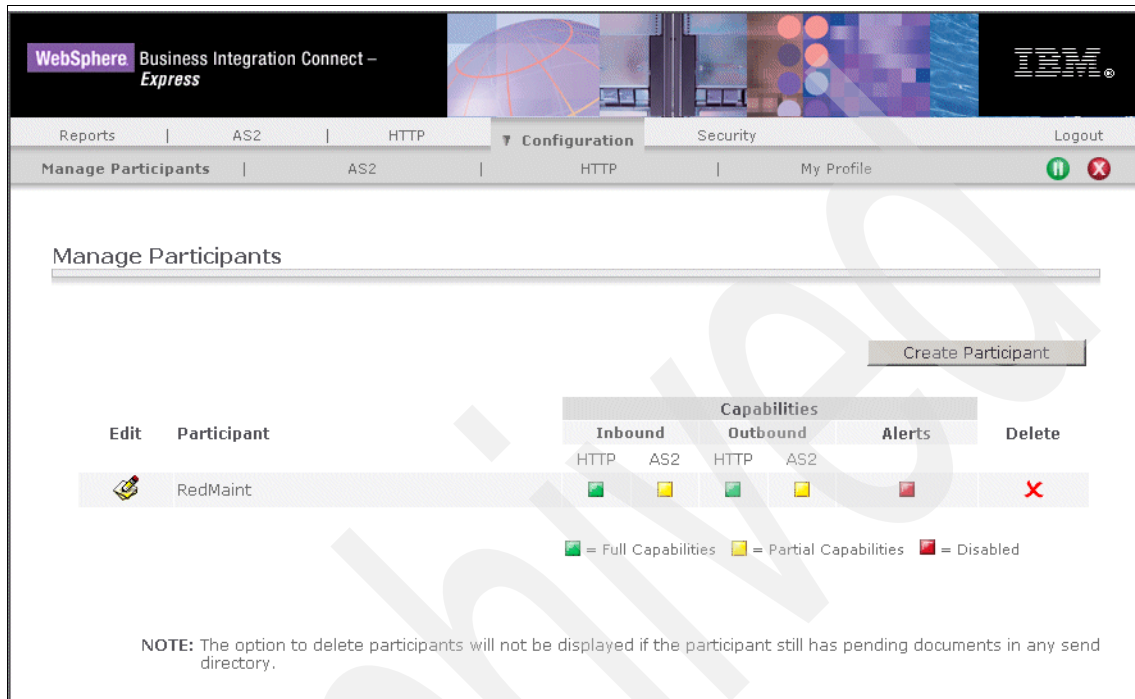


Figure 29-6 Initial configuration of WebSphere Business Integration Connect Express

29.2 Initial configuration of the Advanced server

The configuration consists of four steps:

- ▶ Defining the global profile (Operator), which includes:
 - Creating targets.
 - Creating the document flow using custom XML.
- ▶ Customizing the profile for the local company RedMaint.
- ▶ Customizing the profile for the trading partner RedContract.
- ▶ Defining the participant connection, which brings everything together.

29.2.1 Updating the hubadmin profile

To update the hubadmin profile:

1. Start the console component of WebSphere Business Integration Connect.

We found that is a good idea to put shortcuts on to the desktop for the starting and stopping of the three main components.

Start the console, router, and receiver using the following commands:

```
C:\WBICConnect\console\was\bin\startServer.bat server1
C:\WBICConnect\router\was\bin\startServer.bat server1
C:\WBICConnect\receiver\was\bin\startServer.bat server1
```

2. Open a browser and go to <http://studentx:58080/console> (where *studentx* is the host name of your server).
3. Log on as hubadmin with the installed default password (Pa55word) and company Operator. These values are case-sensitive. You are required to change this password. (We chose itso4you.)

Creating targets

To create targets for the global profile:

1. Select **Hub Admin** → **Hub Configuration** → **Targets**.
2. The browser shows a list of targets (Figure 29-7), which is empty. Click **Create Target**.

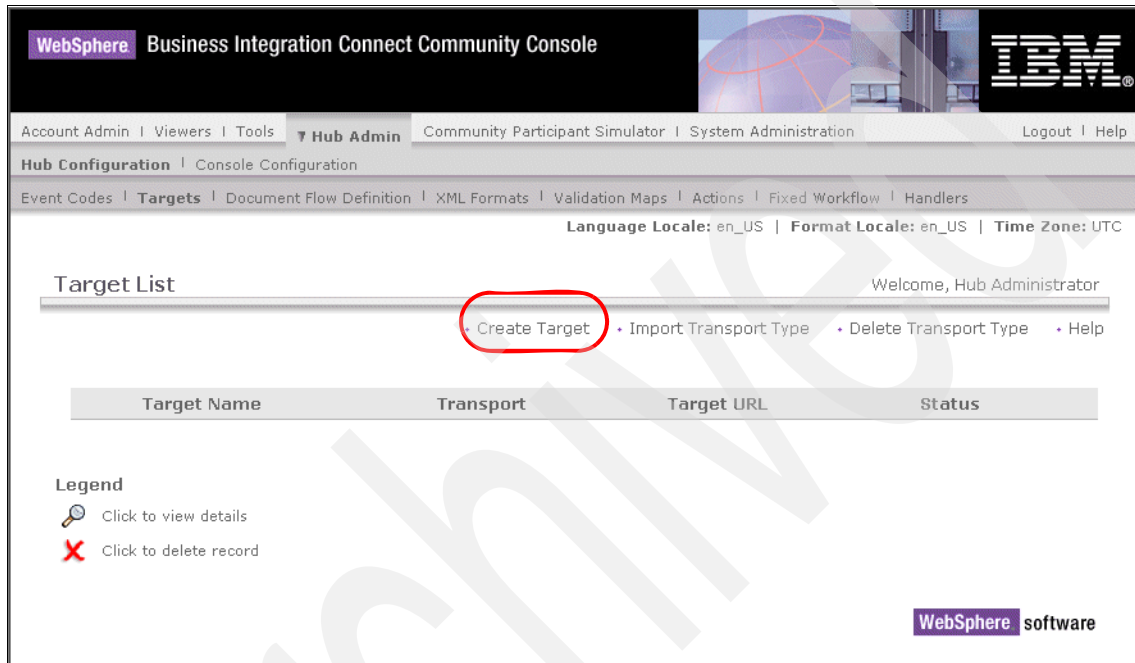


Figure 29-7 List of targets

3. Create the HTTP target first, which is the target that partners of RedMaint use to send documents to RedMaint. Provide a name, for example HTTPTarget, and select the transport as HTTP/S, as shown in Figure 29-8.
4. Selecting HTTP/S causes new parameters to appear. Set the URI to /bcgreceiver/input. The value /bcgreceiver is mandatory. Beyond /bcgreceiver, the URI can be anything.

Target Details Welcome, Hub Administrator

[List](#) [Help](#)

Target Name: HTTPTarget *

Status: ☒ Enabled ☐ Disabled

Description:

Transport: HTTP/S *

Target Configuration

Gateway Type: Production * [New](#) [Edit](#)

URI: /bcgreceiver/input *

Sync Routing: (Changes applies to all http/s receivers)

Max Sync Timeout: ms

Max Sync Sim Conn:

Configuration Point Handlers: Select One

[Save](#) [Cancel](#)

WebSphere software

Figure 29-8 Create new HTTP target

5. Click **Save**.
6. When the new target is saved, click **List** to return to the list of targets (Figure 29-9).

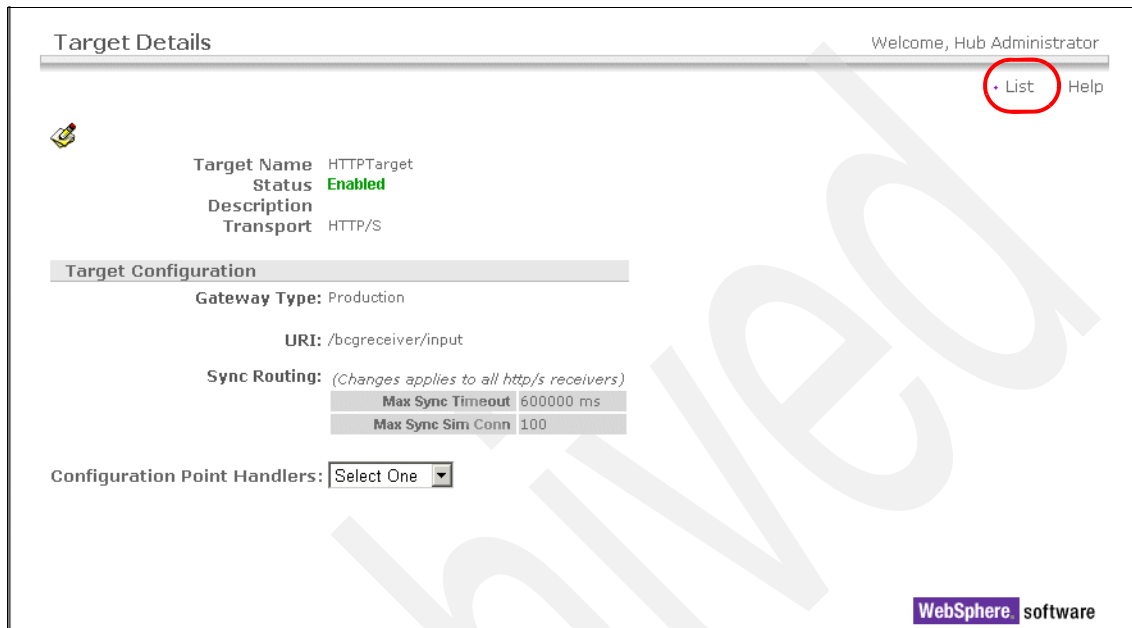


Figure 29-9 New target is created and enabled

7. The list of targets should now have one entry. Click **Create Target** again to create the target that the back-end applications of RedMaint use to send documents to WebSphere Business Integration Connect so that WebSphere Business Integration Connect can transmit them to the partners.
8. Before creating the target, use Explorer to create the folder data in C:\WBICconnect.

9. Return to the browser and complete the target details:

- Target Name: FileTarget
- Transport: File Directory
- Document Root Path: \WBICconnect\data\output

10. Click **Save**.

The screenshot shows a web interface titled "Target Details" with a "Welcome, Hub Administrator" message in the top right. Below the title bar are links for "List" and "Help". The form contains the following fields and controls:

- Target Name:** A text input field containing "FileTarget" with a red asterisk indicating it is required.
- Status:** Radio buttons for "Enabled" (selected) and "Disabled".
- Description:** A large text area for additional information.
- Transport:** A dropdown menu set to "File Directory" with a red asterisk.
- Target Configuration:** A section header for the configuration options.
- Default Gateway Type:** A dropdown menu set to "Production" with a red asterisk, accompanied by "New" and "Edit" buttons.
- Document Root Path:** A text input field containing "\WBICconnect\data\output" with a red asterisk.
- Poll Interval:** A numeric input field set to "5".
- File Unchanged Interval:** A numeric input field set to "3", followed by the text "seconds".
- Thread Nbr:** A numeric input field set to "1".

Figure 29-10 Creating a new file directory target

Creating the community manager

The WebSphere Business Integration Connect server can be used by a number of community participants. One of them can be defined as the community manager, which is usually the company that owns the WebSphere Business Integration Connect server.

To create the community manager:

1. While logged as hub administrator, select **Account Admin** → **Profiles** → **Community Participant**.
2. Click **Create** to add a new participant, as shown in Figure 29-11.

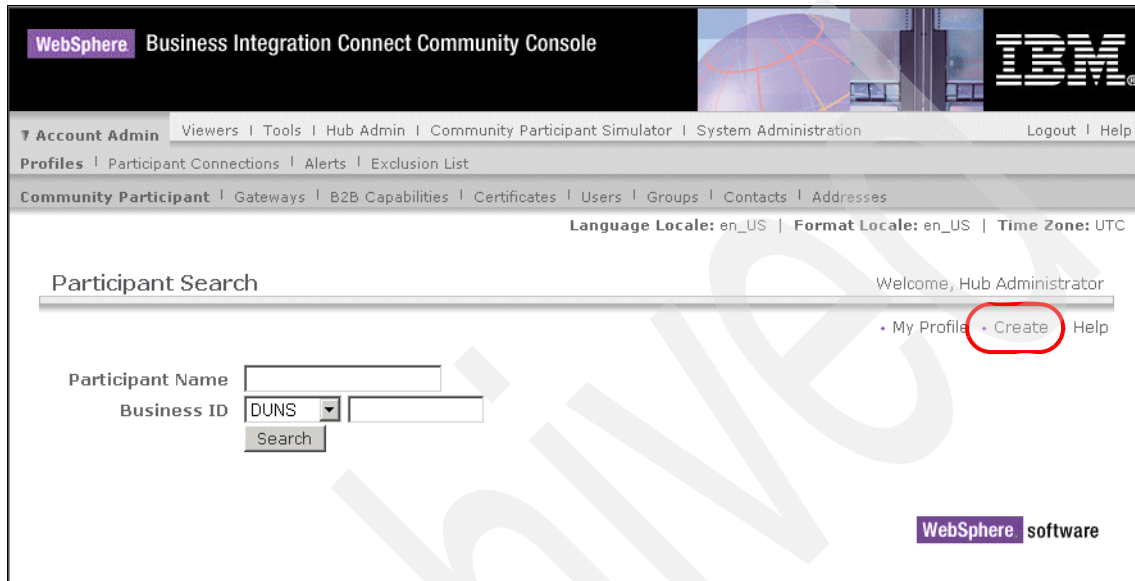


Figure 29-11 Creating a new participant - step 1

3. Provide a login name for the company, for example RedMaint. This name is case sensitive and cannot contain blanks.
4. Provide a participant name, for example RedMaint. This value is used in the user interface.
5. Select Community Manager as the type of participant.
6. Select Other as vendor type.

7. Click **New** in the section Business ID, as shown in Figure 29-12.

The screenshot displays the WebSphere Business Integration Connect Community Console interface. The top navigation bar includes 'WebSphere Business Integration Connect Community Console' and the IBM logo. Below this, a secondary navigation bar lists 'Account Admin', 'Viewers', 'Tools', 'Hub Admin', 'Community Participant Simulator', and 'System Administration'. A third navigation bar shows 'Profiles', 'Participant Connections', 'Alerts', and 'Exclusion List'. The main content area is titled 'Community Participant' and includes links for 'Gateways', 'B2B Capabilities', 'Certificates', 'Users', 'Groups', 'Contacts', and 'Addresses'. The language and time zone are set to 'en_US' and 'UTC' respectively. The main heading is 'Profile > New Participant'. A 'Welcome, Hub Administrator' message is visible. The form contains the following fields: 'Participant Login Name' (RedMaint), 'Participant Name' (RedMaint), 'Participant Type' (Community Manager), 'Status' (Enabled), 'Vendor Type' (Other), and 'Web Site'. 'Save' and 'Cancel' buttons are at the bottom of the form. Below the form is a 'Business ID' section with a table header: 'Type', 'Identifier', and 'Remove'. A 'New' button is located under the 'Type' column.

Type	Identifier	Remove
New		

Figure 29-12 Creating a new participant - step 2

8. Select Freeform as the type of business identifier, as shown in Figure 29-13. Set the identifier to redmaint. This value is used in the XML document for routing purposes.
9. In the IP Address or Host Name section, select the gateway type and set the host name to *studentx* (where *studentx* is the host name of your WebSphere Business Integration Connect Server).
10. Click **Save**.

Profile > New Participant Welcome, Hub Administrator

[Search](#) [Help](#)

Participant Login Name *

Participant Name *

Participant Type *

Status ☒ Enabled ☐ Disabled

Vendor Type

Web Site

Business ID

Type	Identifier	Remove
<input type="text" value="Freeform"/>	<input type="text" value="redmaint"/>	<input type="checkbox"/>

IP Address or Host Name

Gateway Type	IP Address or Host Name	Remove
<input type="text" value="Production"/>	<input type="text" value="studentx"/>	<input type="checkbox"/>

Figure 29-13 Creating a new participant - step 3

When the new profile is created, WebSphere Business Integration Connect generates a temporary password that needs to be passed to the administrator for the community manager company (Figure 29-14 on page 581). In this scenario, we do not exploit the WebSphere Business Integration Connect feature that

performs some kind of self-definition and self-management. The admin user of RedMaint could now edit his own configuration. However, for the purposed of our scenario, we perform all tasks to all profiles with one single user ID, hubadmin of company Operator.

Profile : RedMaint Welcome, Hub Administrator

[Search](#) [Help](#)

Participant Login Name RedMaint

Participant Name RedMaint

Participant Type Community Manager

Status Enabled

Vendor Type Other

Web Site

Administrator's Password **!Rq#w7fg** (Send this password to the participant)

Business ID

Type	Identifier
Freeform	redmaint

IP Address or Host Name

Gateway Type	IP Address or Host Name
Production	studentx

Legend

Click to edit record

WebSphere software

Figure 29-14 New profile created

Creating a community participant

You follow the same steps to create the community manager that you did to create a community participant. The participant is RedContract, which uses WebSphere Business Integration Connect Express, and the login name is RedContract. The business identifier is redcontract, as shown in Figure 29-15.

Note: In this scenario, both instances of WebSphere Business Integration Connect are running on the same machine. Therefore, the gateway host name is always the same, (*studentx* is our host name). In reality, the gateway host name will not be the always the same.

Profile > New Participant Welcome, Hub Administrator

[Search](#) [Help](#)

Participant Login Name *

Participant Name *

Participant Type *

Status ☒ Enabled ☐ Disabled

Vendor Type

Web Site

Business ID		
Type	Identifier	Remove
<input type="text" value="Freeform"/>	<input type="text" value="redcontract"/>	<input type="checkbox"/>
<input type="button" value="New"/>		

IP Address or Host Name		
Gateway Type	IP Address or Host Name	Remove
<input type="text" value="Production"/>	<input type="text" value="studentx"/>	<input type="checkbox"/>
<input type="button" value="New"/>		

Figure 29-15 Create new participant

Creating document flow interaction

The WebSphere Business Integration Connect server needs to learn about the new custom XML format (our message format that we will exchange with our partners) so that it can process, route, package, and unpack the message correctly.

To create document flow interaction:

1. Log on as hubadmin to the console of WebSphere Business Integration Connect and select **Hub Admin** → **Hub Configuration** → **Document Flow Definition**.
2. When the existing document flow definitions are shown, click **Create Document Flow Definition**.

The document flow definition we create is at the Protocol Level, which means that this XML document is defined at the same level as an EDI standard.

Use the values in Table 29-1 for this new document flow definition.

Table 29-1 Values to use when creating a protocol document flow

Attribute	Value
Document flow type	Protocol
Code	customXML
Name	customXML
Version	1.0
Document level	No
Status	Enabled

The Create Document Flow Definitions window opens, as shown in Figure 29-16 on page 584.

Create Document Flow Definitions

Welcome, Hub Administrator

[Manage Document Flow Definitions](#)
[Help](#)

Document flow type
Protocol
*

Code
customXML
*

Name
customXML
*

Version
1.0
*

Description

Document level
☐ Yes
☒ No

Status
☒ Enabled
☐ Disabled

Visibility

Community Operator
☒ Yes
☐ No

Community Manager
☒ Yes
☐ No

Community Participant
☒ Yes
☐ No

Validation maps
No maps found

☐ Top level

☒ Package: AS (N/A): AS

☐ Protocol: Binary (1.0): Binary

☐ Protocol: EDI-Consent (ALL): EDI-Consent

☐ Protocol: EDI-EDIFACT (ALL): EDI-EDIFACT

☐ Protocol: EDI-X12 (ALL): EDI-X12

☒ Package: None (N/A): None

☒ Package: Backend Integration (1.0): Backend Integration

Save

Cancel

Figure 29-16 Create document flow definition at protocol level

3. Set Visibility to Yes for the following:
 - Community Operator
 - Community Manager
 - Community Participant
4. Add this new Protocol to the following:
 - Package: AS
 - Package: None
 - Package: Backend Integration

5. When all values are entered, click **Save**.
6. Create a document flow definition at the document flow level. Click **Manage Document Flow Definitions**, or select **Hub Admin** → **Hub Configuration** → **Document Flow Definition**. Then, click **Create Document Flow Definition**.

Use the values in Table 29-2 to complete the form (shown in Figure 29-17 on page 586).

Table 29-2 Values to use when creating a document flow

Attribute	Value
Document flow type	Document Flow
Code	ContractorRequest
Name	ContractorRequest
Version	1.0
Document level	Yes
Status	Enabled

7. Set Visibility to Yes for the three roles.

The definition of this document flow can be compared to adding the definition of an 850 purchase order document to the EDI standard. As such, if other XML documents need to be defined, we can add them directly at the document flow level. You do not need to create another document flow at the protocol level.

Create Document Flow Definitions Welcome, Hub Administrator

[Manage Document Flow Definitions](#)
[Help](#)

Document flow type *

Code *

Name *

Version *

Description

Document level ☒ Yes ☐ No

Status ☒ Enabled ☐ Disabled

Visibility

Community Operator	<input checked="" type="radio"/> Yes	<input type="radio"/> No
Community Manager	<input checked="" type="radio"/> Yes	<input type="radio"/> No
Community Participant	<input checked="" type="radio"/> Yes	<input type="radio"/> No

Validation maps No maps found

☐ Top level

- ☐ Package: AS (N/A): AS
- ☐ Package: None (N/A): None
- ☐ Package: Backend Integration (1.0): Backend Integration

Figure 29-17 Create document flow at document flow level

8. At the bottom of Figure 29-17, the expandable tree structure of packages and protocols are shown. Expand this structure and add this new document flow to the following:

- Package: AS
 - Protocol: customXML
- Package: None
 - Protocol: customXML
- Package: Backend Integration
 - Protocol: customXML

Figure 29-18 shows the completed structure.

9. Click **Save**.

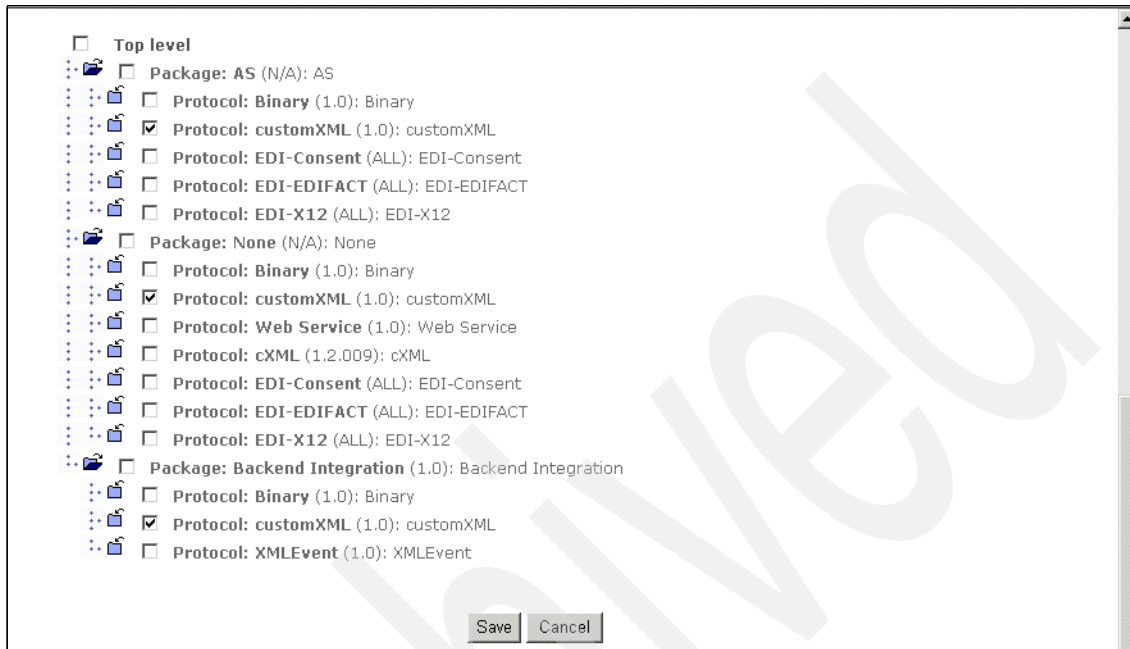


Figure 29-18 Link document flow to packages and protocols

Creating a new XML format

You should define the XML format to WebSphere Business Integration Connect to make sure that the server knows how to route the XML document (that is, use the data in the XML document to obtain to which partner a document should be sent).

To create a new XML format:

1. While logged on as hubadmin, select **Hub Admin** → **Hub Configuration** → **XML Formats** to open the Manage XML Formats window, as shown in Figure 29-19.

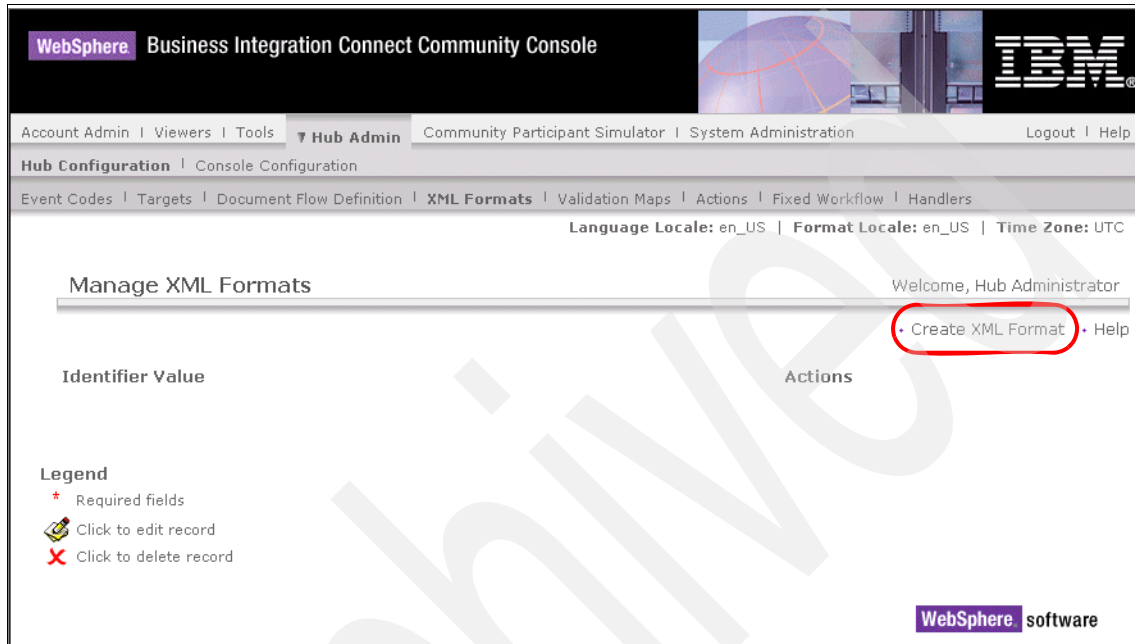


Figure 29-19 Manage XML formats in WebSphere Business Integration Connect

2. Click **Create XML Format**. A new XML format form is presented. A new XML format needs to be linked to an existing protocol, which is what you defined earlier.
3. Select **customXML 1.0** in the drop-down box that is labeled Routing Format. The file type is XML.
4. Select Identifier Type as Root Tag. The root tag in our XML documents is Maintenance. Other ways of identifying an XML document are by using a DTD or name space.
5. In the section Schema Attributes, indicate to WebSphere Business Integration Connect how to identify the source and target business identifiers. These identifiers could be set as a constant, but in most cases, you use an element in the XML document as the carrier for information about business identifiers. That way, it becomes easier to send this kind of XML document to multiple partners.

Figure 29-20 shows a sample XML document that indicates the element names that store business identifiers. Figure 29-21 on page 590 shows the form for creating a new XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<Q1:Maintenance xmlns:Q1="http://external.request.redmaint.com">
  <Q1:From>redmaint</Q1:From>
  <Q1:To>redcontract</Q1:To>
  <Q1:ExpectedCompletion>1110600000000</Q1:ExpectedCompletion>
  <Q1:ActualCompletion></Q1:ActualCompletion>
  <Q1:StatusDescription></Q1:StatusDescription>
  <Q1:Tenant>100</Q1:Tenant>
  <Q1:Apartment>100</Q1:Apartment>
  <Q1:MaintenanceId>2</Q1:MaintenanceId>
  <Q1:CreateDate>1110500000000</Q1:CreateDate>
  <Q1:Description>leak in roof</Q1:Description>
</Q1:Maintenance>
```

Figure 29-20 Sample XML file (edited)

View XML Format

Welcome, Hub Administrator

[Manage XML Formats](#)
[Help](#)

Routing Format

customXML 1.0 *

File Type

XML *

Identifier Type

Root Tag *

Maintenance

Schema Attributes

Name	Type	Value
Source BusinessId	Element Path	/Maintenance/From *
Target BusinessId	Element Path	/Maintenance/To *
Source Document Flow	Constant	ContractorRequest *
Source Document Flow Version	Constant	1.0 *
Document Identifier	Element Path	
Document Timestamp	Element Path	
Duplicate Check Key 1	Element Path	
Duplicate Check Key 2	Element Path	
Duplicate Check Key 3	Element Path	
Duplicate Check Key 4	Element Path	
Duplicate Check Key 5	Element Path	

Save

Cancel

Legend

*

Required fields

Click to edit record

Click to delete record

WebSphere

software

Figure 29-21 Create a new XML format

- Besides information about business identifiers, WebSphere Business Integration Connect needs to know what document flow should be invoked for XML documents of this format. This information could again be carried in the XML document itself. However, we choose to make this information a constant. The source document flow is ContractorRequest and the version is 1.0. The document flow was defined earlier, see Figure 29-17 on page 586.
- Click **Save** to store the new format.

In our scenario, the ContractorResponse message is actually the same as the ContractorRequest message. Only a few more elements are needed, but the root element is the same. Also, the business identifiers are carried in the same elements for the response as for the request. Therefore, you do not need to define another document flow definition at the document flow level or to define another XML format. However, in real-life situations, this is possible. We should

then repeat the steps outlined above, so that a document flow definition and XML format is created to handle ContractorResponse messages.

Creating interaction

Earlier, we created the protocol customXML that could be packaged in AS2, None, or Backend Integration. We then created the document flow ContractorRequest that was linked to the protocol customXML for each packaging method. We defined the XML format linked to that document flow.

The next step is to create the interactions that tell WebSphere Business Integration Connect how to move from one packaging, protocol, or document to another. To create this interaction:

1. While logged on as hubadmin, select **Hub Admin** → **Hub Configuration** → **Document Flow Definition**.
2. Click **Manage Interactions**. Then, click **Create Interaction**.
3. We need an interaction from AS/customXML/ContractorRequest to None/customXML/ContractorRequest, which is the interaction shown in Figure 29-22 on page 592. Select **Pass Through** as the action and click **Save**.

Manage Interactions

Welcome, Hub Administrator

[Manage Document Flow Definitions](#)
[Help](#)

Select one document flow definition each from the Source and Target column, and then fill in the data fields.

Source *

0 1 2 3 4 All

Package: AS

Protocol: Binary (1.0)

Protocol: customXML (1.0)

Document Flow: ContractorRequest (1.0)

Protocol: EDI-Consent (ALL)

Protocol: EDI-EDIFACT (ALL)

Protocol: EDI-X12 (ALL)

Package: None

Package: Backend Integration (1.0)

Target *

0 1 2 3 4 All

Package: AS

Package: None

Protocol: Binary (1.0)

Protocol: customXML (1.0)

Document Flow: ContractorRequest (1.0)

Protocol: Web Service (1.0)

Protocol: cXML (1.2.009)

Protocol: EDI-Consent (ALL)

Protocol: EDI-EDIFACT (ALL)

Protocol: EDI-X12 (ALL)

Package: Backend Integration (1.0)

Transform map

Transform map description

Action *

Figure 29-22 Creating interaction from packaging AS to None for customXML

- Repeat steps 1 on page 591 through 3 on page 591 to create the reverse interaction, from None/customXML/ContractorRequest to AS/customXML/ContractorRequest. This interaction is shown in Figure 29-23 on page 593. Select **Pass Through** and click **Save**.

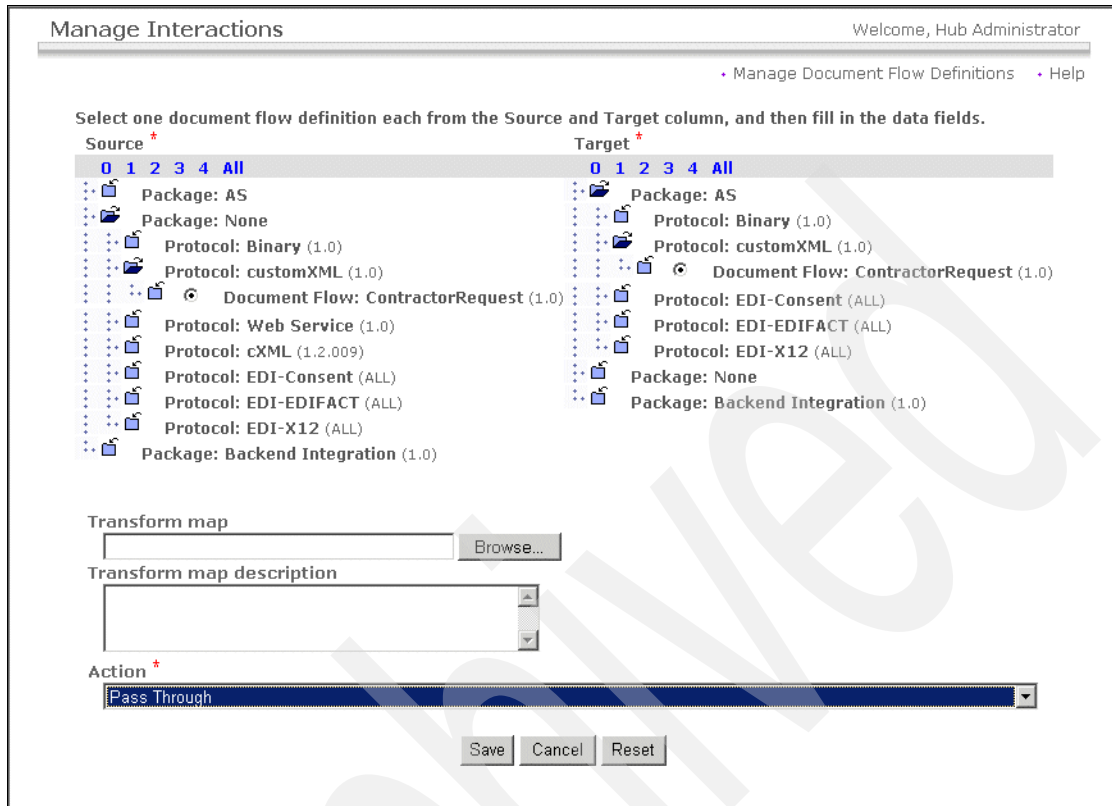


Figure 29-23 Create interaction from packaging AS to None for customXML

The server is now configured to handle incoming ContractorRequest documents that are not packaged. The server can package the document in an AS2 package. The server can now also handle the reverse operation. That is, the server can receive an AS2 packaged ContractorRequest document and remove the AS2 packaging layer.

29.2.2 Configuring the community manager's profile

So far, we have performed some global setup of the server, and we have created two profiles for actual trading partners. We are now going to work with these trading partner profiles.

Opening the profile of RedMaint

To manage the profile, we could log on as the manager of the profile (user = admin, company = RedMaint, and password = generated). However, for simplicity we configure everything while logged on as hubadmin. To open the profile:

1. Select **Account Admin** → **Profiles** → **Community Participant**.
2. Click **Search**.
3. Click the magnifying glass next to RedMaint.

The screenshot shows the WebSphere Business Integration Connect Community Console interface. The top navigation bar includes 'WebSphere Business Integration Connect Community Console' and the IBM logo. Below this, a secondary navigation bar shows 'Account Admin' and various tools. The main content area is titled 'Participant Search' and displays a search form with fields for 'Participant Name' and 'Business ID' (with a dropdown menu set to 'DUNS'). A 'Search' button is present. Below the search form, a table lists participants:

Participant Name	Participant Type	Status
Hub Operator	Community Operator	Enabled
RedContract	Community Participant	Enabled
RedMaint	Community Manager	Enabled

Below the table, a legend indicates that the magnifying glass icon is used to 'Click to view details' and the red X icon is used to 'Click to delete record'. The bottom right corner of the console shows the 'WebSphere software' logo.

Figure 29-24 List of participants

You should now see the main page of the profile of RedMaint, as shown in Figure 29-25 on page 595.

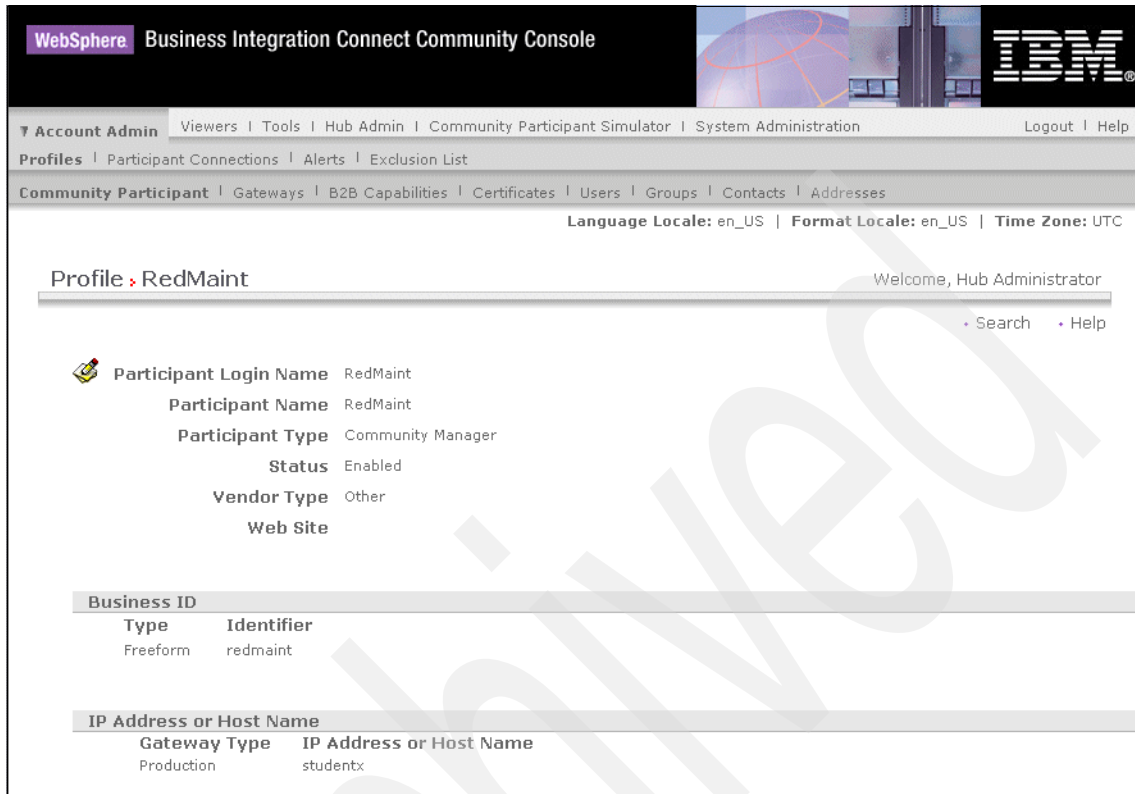


Figure 29-25 Managing the profile of RedMaint

Defining gateways

To define gateways:

1. Before proceeding, use Explorer to create the C:\WBICconnect\data\input directory (where *WBICconnect* is the installation directory).
2. Return to the WebSphere Business Integration Connect Console. While you are still in the profile of RedMaint, select **Gateways**.

Attention: Verify that you are indeed working with the profile of RedMaint (as shown in Figure 29-26 on page 596).

3. Click **Create**.

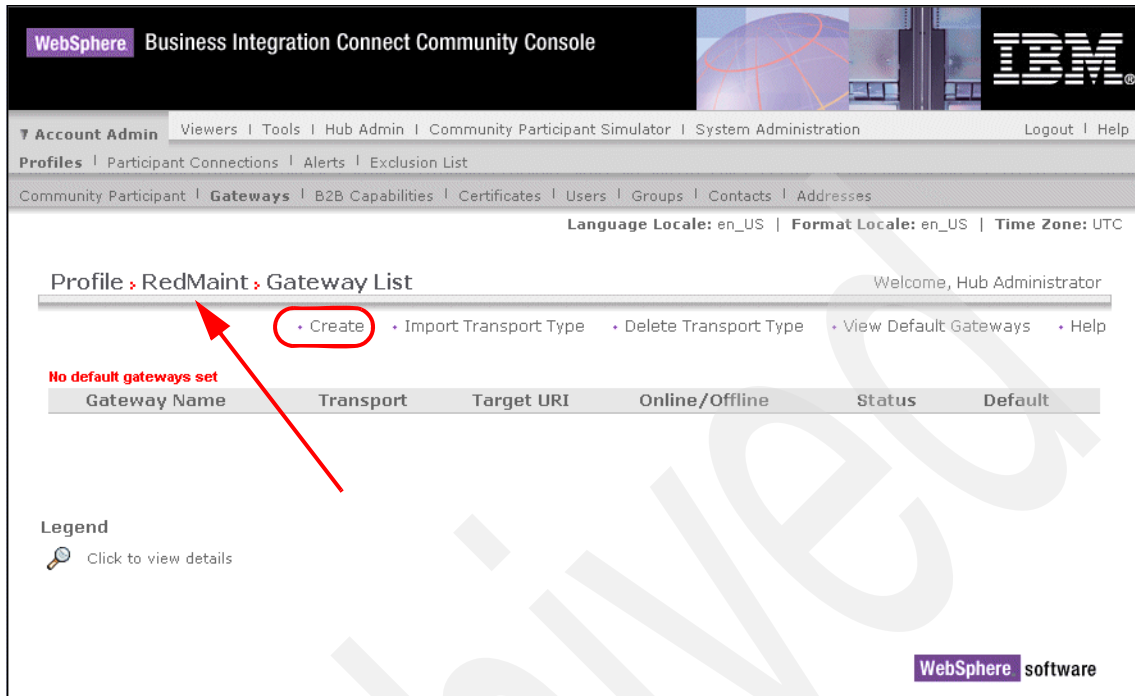


Figure 29-26 List of gateways for redmaint

4. In the form that displays (Figure 29-7 on page 574), provide a name for the gateway. We use FileSystemGateway.
5. Select the transport, which is File Directory for our scenario.
6. Provide the complete directory structure in the URI format. In our case, there are three forward slash (/) characters following the colon character. If a disk letter is required, add C: between the second and third slash character. In the URI format, the usual Windows directory separator backslash (\) is not used.
7. Click **Save** to store the new gateway in the database.

WebSphere Business Integration Connect Community Console

Account Admin Viewers Tools Hub Admin Community Participant Simulator System Administration Logout Help

Profiles Participant Connections Alerts Exclusion List

Community Participant Gateways B2B Capabilities Certificates Users Groups Contacts Addresses

Language Locale: en_US Format Locale: en_US Time Zone: UTC

Profile > RedMaint > Gateway List Welcome, Hub Administrator

List Help

Gateway Name

Status ☒ Enabled ☐ Disabled

Online/Offline ☒ Online ☐ Offline

Description

Gateway Configuration

Transport

Target URI

Retry Count

Retry Interval seconds

Number of Threads

Validate Client IP ☒ No ☐ Yes

Auto Queue ☒ No ☐ Yes

Use Unique File Name ☒

Configuration Point Handlers:

Save Cancel

Figure 29-27 Create a new gateway

8. Click **List** to return to the list of defined gateways.

Figure 29-28 on page 598 lists the new gateway. It shows also that we have not yet provided a default gateway.

9. To label a gateway as the default gateway, click **View Default Gateways**.

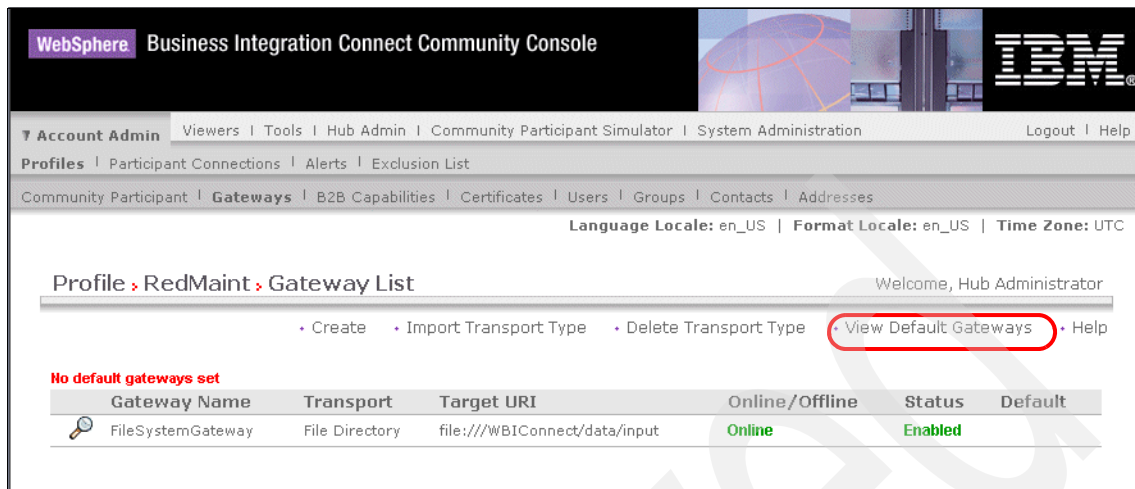


Figure 29-28 List of gateways without a default gateway

10. Select **FileSystemGateway** as the default production gateway and click **Save**.

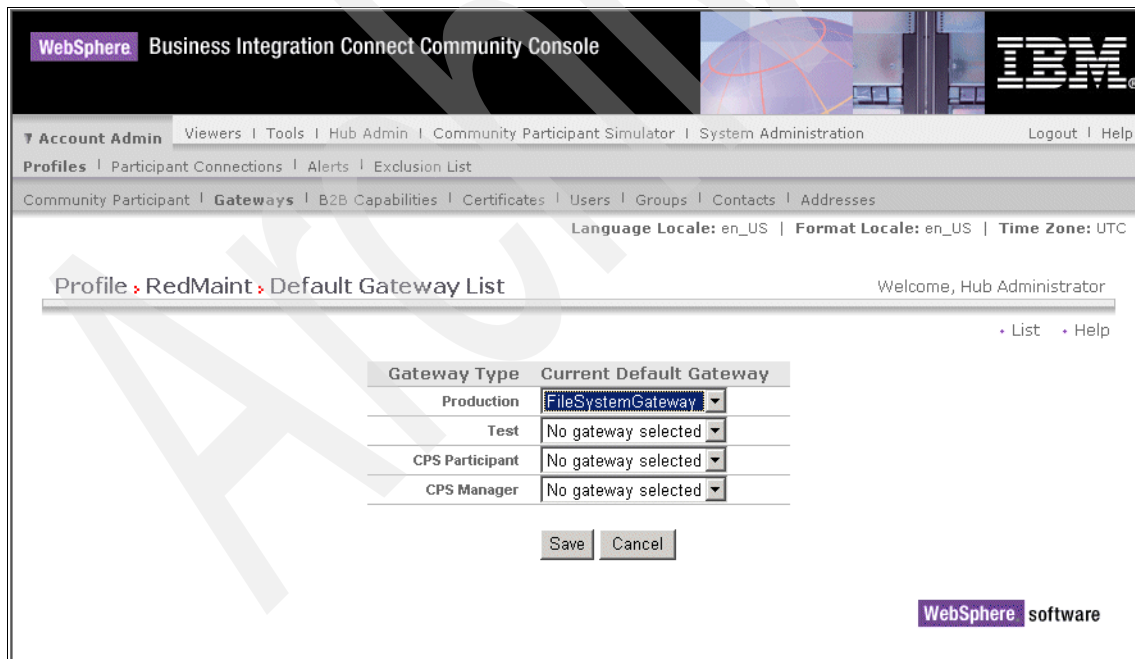
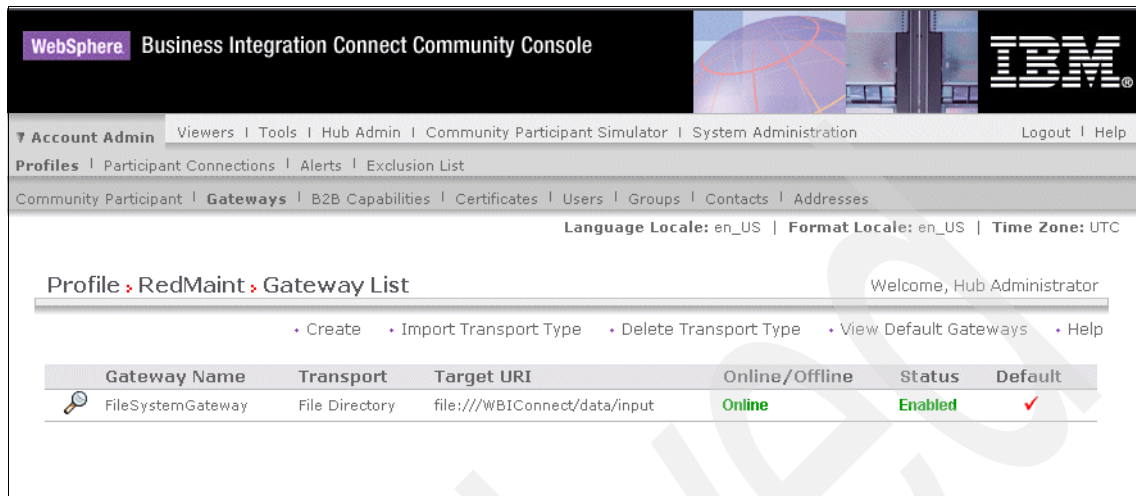


Figure 29-29 Update the default gateway

Figure 29-30 shows the final gateway list for RedMaint.



The screenshot displays the WebSphere Business Integration Connect Community Console. The top navigation bar includes 'WebSphere Business Integration Connect Community Console' and the IBM logo. Below this, a secondary navigation bar lists 'Account Admin', 'Viewers', 'Tools', 'Hub Admin', 'Community Participant Simulator', and 'System Administration'. A third bar shows 'Profiles', 'Participant Connections', 'Alerts', and 'Exclusion List'. The main content area is titled 'Profile: RedMaint: Gateway List' and includes a 'Welcome, Hub Administrator' message. A table lists the defined gateways, with one entry shown: 'FileSystemGateway' using 'File Directory' transport, pointing to 'file:///WBICconnect/data/input', which is 'Online' and 'Enabled'.

Gateway Name	Transport	Target URI	Online/Offline	Status	Default
FileSystemGateway	File Directory	file:///WBICconnect/data/input	Online	Enabled	✓

Figure 29-30 List of defined gateways

Defining business-to-business capabilities

The next step is to define the business-to-business capabilities of RedMaint. As a community participant, you can indicate that you are capable of handling certain types of documents. The server must support the type of document. However, a server can be configured to handle more types of documents than a community participant can handle.

Therefore, you need to set the business-to-business capabilities of a community participant as follows:

1. While logged on as hubadmin, open the profile for RedMaint. Select **B2B Capabilities**.

WebSphere

Business Integration Connect Community Console

Account Admin
Viewers
Tools
Hub Admin
Community Participant Simulator
System Administration
Logout
Help

Profiles
Participant Connections
Alerts
Exclusion List

Community Participant
Gateways
B2B Capabilities
Certificates
Users
Groups
Contacts
Addresses

Language Locale: en_US
Format Locale: en_US
Time Zone: UTC

Profile
RedMaint
B2B Capabilities

Welcome, Hub Administrator

Help

Set Source	Set Target	Enabled	Edit	Document Flow Definition					
				0	1	2	3	4	All
				Package: AS					
				Package: None					
				Package: Backend Integration (1.0)					

Legend

- Edit attributes
- Tree is expanded; click to collapse.
- Tree is collapsed; click to expand.
- Role is active, click to deactivate.
- Role is not active; click to create role.
- Role is inactive; cannot activate while the capability is disabled.

WebSphere software

Figure 29-31 List of business-to-business capabilities

- Click the icon underneath Set Source for Package: AS to enable it. Click the icon underneath Set Target for Package: AS to enable it.
- Click the icon next to Package: AS to drill down. Click the icon for Protocol: customXML (1.0) for both source and target.
- Click the icon next to Protocol: customXML(1.0) to drill down. Finally, click the icon for Document Flow: ContractorRequest for both source and target.
- Repeat this sequence for the Package: None. At the end, you will see something similar to Figure 29-32 on page 601.

WebSphere

Business Integration Connect Community Console

Account Admin

Viewers | Tools | Hub Admin | Community Participant Simulator | System Administration

Logout | Help

Profiles

Participant Connections | Alerts | Exclusion List

Community Participant

Gateways | B2B Capabilities | Certificates | Users | Groups | Contacts | Addresses

Language Locale: en_US | Format Locale: en_US | Time Zone: UTC

Profile > RedMaint > B2B Capabilities

Welcome, Hub Administrator

Help

Set Source	Set Target	Enabled	Edit	Document Flow Definition					
				0	1	2	3	4	All
		Enabled		Package: AS					
				Protocol: Binary (1.0)					
		Enabled		Protocol: customXML (1.0)					
		Enabled		Document Flow: ContractorRequest (1.0)					
				Protocol: EDI-Consent (ALL)					
				Protocol: EDI-EDIFACT (ALL)					
				Protocol: EDI-X12 (ALL)					
		Enabled		Package: None					
				Protocol: Binary (1.0)					
		Enabled		Protocol: customXML (1.0)					
		Enabled		Document Flow: ContractorRequest (1.0)					
				Protocol: Web Service (1.0)					
				Protocol: cXML (1.2.009)					
				Protocol: EDI-Consent (ALL)					
				Protocol: EDI-EDIFACT (ALL)					
				Protocol: EDI-X12 (ALL)					
				Package: Backend Integration (1.0)					

Figure 29-32 Business-to-business capabilities for participant RedMaint

29.2.3 Configuring the community participant's profile

So far, we have performed some global set-up of the server and we have updated the profile of RedMaint. We are now going to work with the profile of RedContract.

Opening the profile of redcontract

To open the profile:

1. Select **Account Admin** → **Profiles** → **Community Participant**.
2. Click **Search**.
3. Click the magnifying glass next to RedContract.

We now see the main page of the profile of RedContract.

The screenshot displays the WebSphere Business Integration Connect Community Console. The top navigation bar includes 'WebSphere Business Integration Connect Community Console' and the IBM logo. Below this, a secondary navigation bar lists 'Account Admin', 'Viewers', 'Tools', 'Hub Admin', 'Community Participant Simulator', and 'System Administration'. A third bar shows 'Profiles', 'Participant Connections', 'Alerts', and 'Exclusion List'. The main content area is titled 'Community Participant' and lists 'Gateways', 'B2B Capabilities', 'Certificates', 'Users', 'Groups', 'Contacts', and 'Addresses'. A language and locale settings bar shows 'Language Locale: en_US', 'Format Locale: en_US', and 'Time Zone: UTC'. The main profile section is titled 'Profile > RedContract' and includes a 'Welcome, Hub Administrator' message. Below this, a list of fields for the RedContract participant is shown: Participant Login Name (RedContract), Participant Name (RedContract), Participant Type (Community Participant), Status (Enabled), Vendor Type (Other), and Web Site. A 'Business ID' section contains a table with columns 'Type' and 'Identifier', showing 'Freeform' and 'redcontract'. An 'IP Address or Host Name' section contains a table with columns 'Gateway Type' and 'IP Address or Host Name', showing 'Production' and 'studentx'.

Business ID	
Type	Identifier
Freeform	redcontract

IP Address or Host Name	
Gateway Type	IP Address or Host Name
Production	studentx

Figure 29-33 Managing the profile of RedContract

Defining the gateway

To define the gateway:

1. While working with the profile of RedContract, select **Gateways**.
2. When the empty list of gateways is shown, click **Create**.
3. Provide a name for the gateway, for example HTTPGateway.
4. Select the protocol, which is HTTP/1.1 for our scenario.
5. Set the target URI to: `http://studentx:88/input/AS2` (where *studentx* is the host name of our machine).
6. Set the user name to redmaint and password to itso4you.

Note: This user ID and password is created on the WebSphere Business Integration Connect Express server so that redmaint can access it.

7. Click **Save**.

Profile » RedContract » Gateway List Welcome, Hub Administrator

[List](#) [Help](#)

Gateway Name *

Status ☒ Enabled ☐ Disabled

Online/Offline ☒ Online ☐ Offline

Description

Gateway Configuration

Transport

Target URI *

User Name

Password

Retry Count

Retry Interval seconds

Number of Threads

Validate Client IP ☒ No ☐ Yes

Auto Queue ☒ No ☐ Yes

Connection Timeout seconds

Configuration Point Handlers:

Figure 29-34 Create new gateway

8. Return to the list of gateways again and select **View Default Gateways**.
9. Make the HTTPGateway the default production gateway for RedContract.

Figure 29-35 on page 604 shows the list of gateways when this configuration is completed.

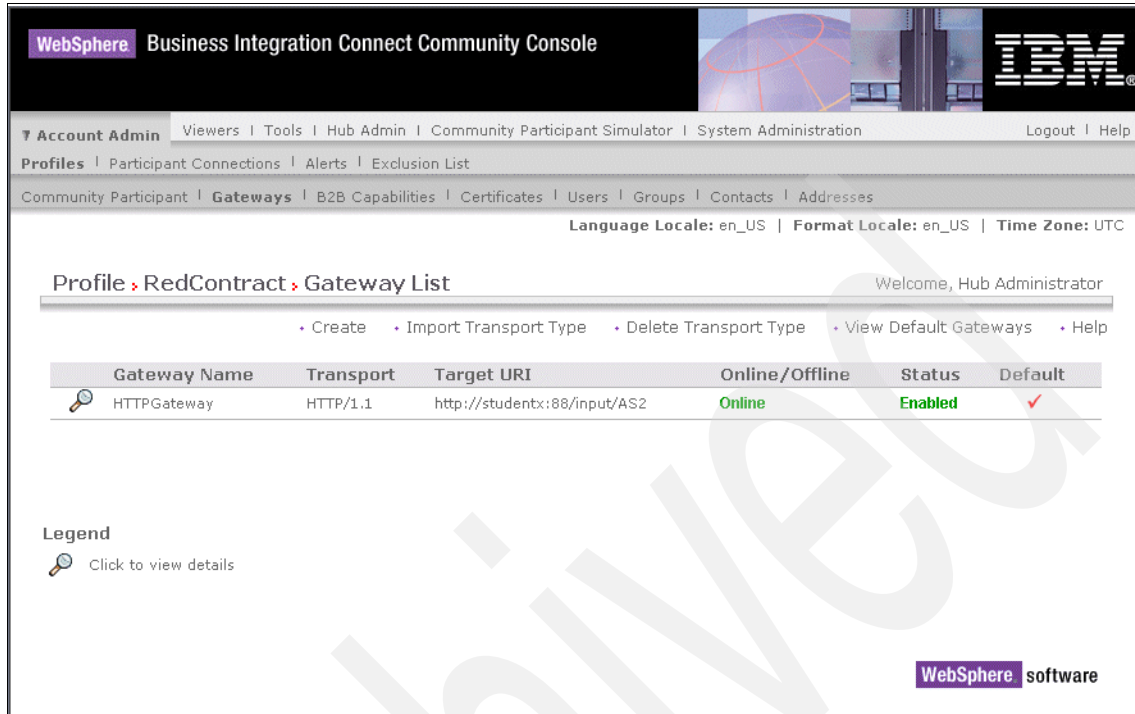



Figure 29-35 List of gateways for redcontract

Defining business-to-business capabilities

To define business-to-business capabilities:

1. While logged on as hubadmin, select **Account Admin** → **Profiles** → **Community Participant**. In the Participant Search window, click **Search**.
2. When the list of profiles is shown, click the  icon in front of the RedContract profile.
3. Select **B2B Capabilities**. Again, verify that you are working with the profile of RedContract.
4. Enable AS/customXML/ContractorRequest as source and target. Also, enable None/customXML/ContractorRequest for source and target, as shown in Figure 29-36 on page 605.

WebSphere

Business Integration Connect Community Console

Account Admin

Viewers

Tools

Hub Admin

Community Participant Simulator

System Administration

Logout

Help

Profiles

Participant Connections

Alerts

Exclusion List

Community Participant

Gateways

B2B Capabilities

Certificates

Users

Groups

Contacts

Addresses

Language Locale: en_US

Format Locale: en_US

Time Zone: UTC

Profile > RedContract > B2B Capabilities

Welcome, Hub Administrator

Help

Set Source	Set Target	Enabled	Edit	Document Flow Definition
				0 1 2 3 4 All
✓	✓	Enabled		Package: AS
				Protocol: Binary (1.0)
✓	✓	Enabled		Protocol: customXML (1.0)
✓	✓	Enabled		Document Flow: ContractorRequest (1.0)
				Protocol: EDI-Consent (ALL)
				Protocol: EDI-EDIFACT (ALL)
				Protocol: EDI-X12 (ALL)
✓	✓	Enabled		Package: None
				Protocol: Binary (1.0)
✓	✓	Enabled		Protocol: customXML (1.0)
✓	✓	Enabled		Document Flow: ContractorRequest (1.0)
				Protocol: Web Service (1.0)
				Protocol: cXML (1.2.009)
				Protocol: EDI-Consent (ALL)
				Protocol: EDI-EDIFACT (ALL)
				Protocol: EDI-X12 (ALL)
				Package: Backend Integration (1.0)

Figure 29-36 Set business-to-business Capabilities of RedContract on server of RedMaint

29.2.4 Creating the participant connection

You can now tie everything together in a participant connection by following these steps:

1. Select **Account Admin** → **Participant Connections**.
2. From the Manage Connections screen, select the Source as RedMaint and the Target as RedContract.
3. Click **Search**.

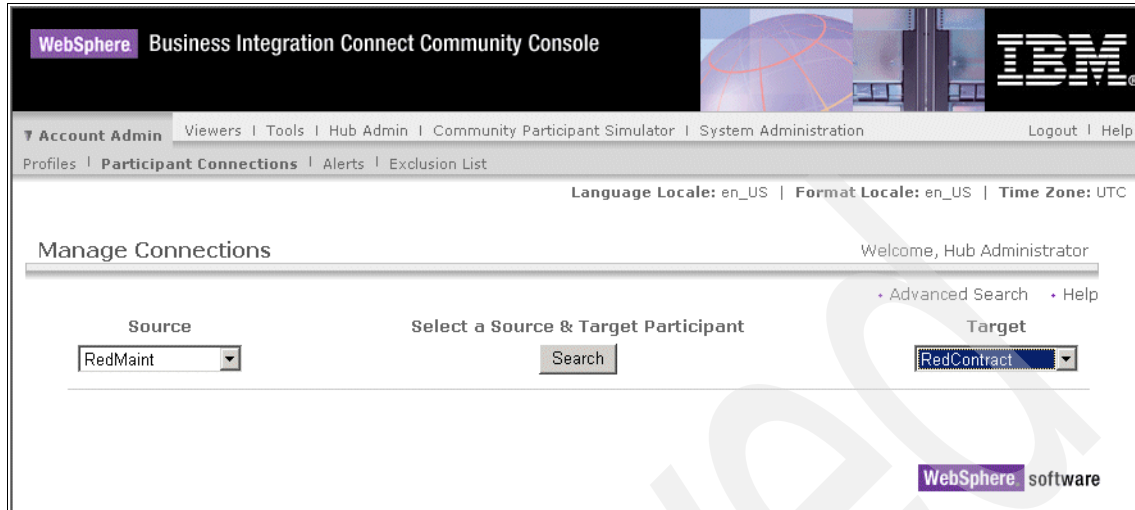


Figure 29-37 Search participants for a connection

4. When the list of connections is shown, click **Activate** for the connection that takes None/customXML documents to AS2/customXML documents, as shown in Figure 29-38.

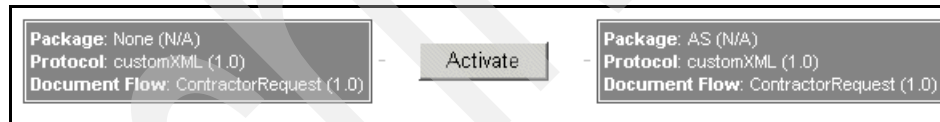


Figure 29-38 Activate connection to send AS2-XML to redcontract

5. The connection is now activated, but needs some customization. Click **Attributes** next to Package AS to make changes to the connection attributes (Figure 29-39 on page 607).

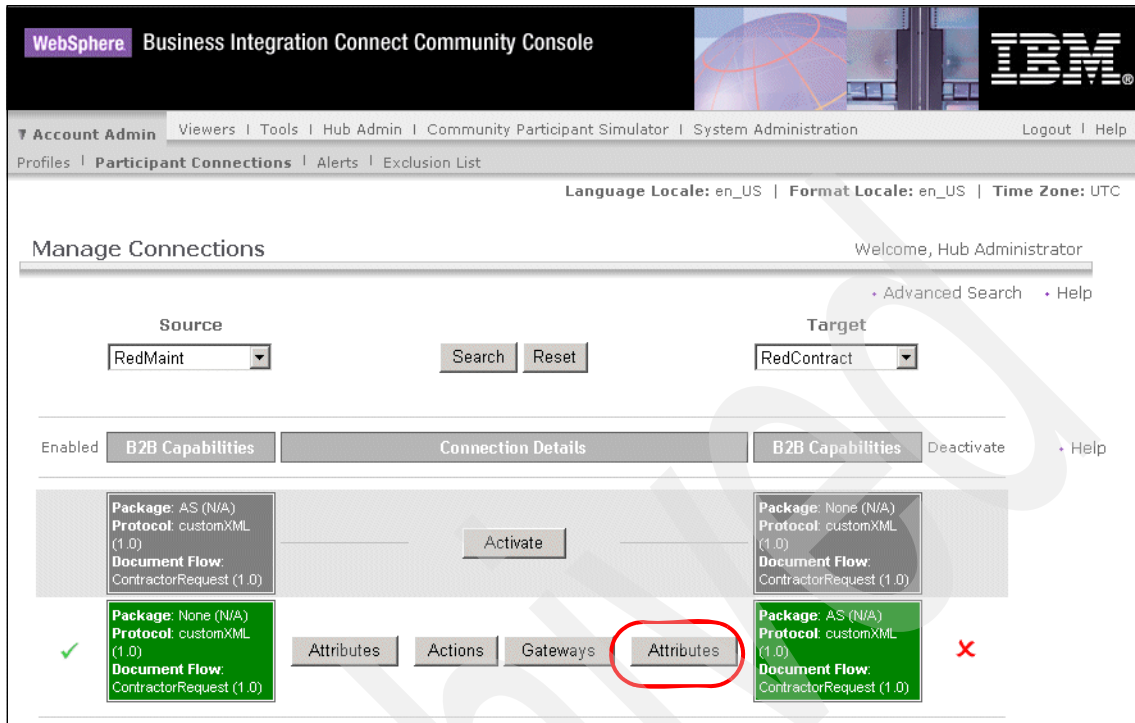


Figure 29-39 Updates attributes of activated connection

6. The connection attributes are now shown.
7. Scroll down the Connection attributes screen and click the blue folder icon next to Package: AS (N/A).

Connection Attributes

Welcome, Hub Administrator

[Return](#)
[Help](#)

Source

RedMaint

Package: None (N/A)
Protocol: customXML (1.0)
Document Flow: ContractorRequest (1.0)

Target

RedContract

Package: AS (N/A)
Protocol: customXML (1.0)
Document Flow: ContractorRequest (1.0)

Scope: Connection

Connection Summary

Attribute	Description	Current Value	Inheritance
Time To Acknowledge	Time To Acknowledge	30	Inherited from: Scope: Global Type: Time To Acknowledge
Retry Count	Retry Count	3	Inherited from: Scope: Global Type: Retry Count
AS Compress Before Sign	AS Compress Before Sign	Yes	Inherited from: Scope: Global Type: AS Compress Before Sign
AS Compressed	AS Compressed	No	Inherited from: Scope: Global Type: AS Compressed
AS Encrypted	AS Encrypted	No	Inherited from: Scope: Global Type: AS Encrypted
AS MDN Http Url	AS MDN Http Url		No value provided
AS MDN Email Address	AS MDN Email Address		No value provided
AS MDN Asynchronous	AS MDN Asynchronous	Yes	Inherited from: Scope: Global Type: AS MDN Asynchronous
AS MDN Requested	AS MDN Requested	Yes	Inherited from: Scope: Global Type: AS MDN Requested
AS Message Digest Algorithm	AS Message Digest Algorithm	sha1	Inherited from: Scope: Global Type: AS Message Digest Algorithm
AS MDN Signed	AS MDN Signed	No	Inherited from: Scope: Global Type: AS MDN Signed
AS Signed	AS Signed	No	Inherited from: Scope: Global Type: AS Signed
AS Business Id	AS Business Id		No value provided

Package: AS (N/A)

Figure 29-40 Connection attributes

- Enter `http://studentx:57080/bcgreceiver/input` as the AS MDN HTTP URL (AS2) (where *studentx* is the host name of our machine).
- Enter `mdn@redmaint.com` as the AS MDN e-mail Address
- Click **Save**.

RedMaint

Package: None (N/A)
 Protocol: customXML (1.0)
 Document Flow: ContractorRequest (1.0)

RedContract

Package: AS (N/A)
 Protocol: customXML (1.0)
 Document Flow: ContractorRequest (1.0)

Scope: Connection

Connection Summary

Package: AS (N/A)

Attribute	Description	Current Value	Inheritance	Update	Reset
Time To Acknowledge	Time To Acknowledge	30	Inherited from: Scope: Global Type: Time To Acknowledge	<input type="text"/>	
Retry Count	Retry Count	3	Inherited from: Scope: Global Type: Retry Count	<input type="text"/>	
AS Compress Before Sign	AS Compress Before Sign	Yes	Inherited from: Scope: Global Type: AS Compress Before Sign	Select one to update ▼	
AS Compressed	AS Compressed	No	Inherited from: Scope: Global Type: AS Compressed	Select one to update ▼	
AS Encrypted	AS Encrypted	No	Inherited from: Scope: Global Type: AS Encrypted	Select one to update ▼	
AS MDN Http Url	AS MDN Http Url	No value provided	No value provided	<input type="text" value="7080/bcgreceiver/input"/>	
AS MDN Email Address	AS MDN Email Address	No value provided	No value provided	<input type="text" value="mdn@redmaint.com"/>	
AS MDN Asynchronous	AS MDN Asynchronous	Yes	Inherited from: Scope: Global Type: AS MDN Asynchronous	Select one to update ▼	
AS MDN Requested	AS MDN Requested	Yes	Inherited from: Scope: Global Type: AS MDN Requested	Select one to update ▼	
AS Message Digest Algorithm	AS Message Digest Algorithm	sha1	Inherited from: Scope: Global Type: AS Message Digest Algorithm	Select one to update ▼	
AS MDN Signed	AS MDN Signed	No	Inherited from: Scope: Global Type: AS MDN Signed	Select one to update ▼	
AS Signed	AS Signed	No	Inherited from: Scope: Global Type: AS Signed	Select one to update ▼	
AS Business Id	AS Business Id	No value provided	No value provided	Select one to update ▼	

Save Close

Figure 29-41 Updating connection attributes

11. Click **Return**.
12. Now search for the reverse connection — RedContract as the source and RedMaint as the target. Activate that connection as well. This connection does not require further configuration.

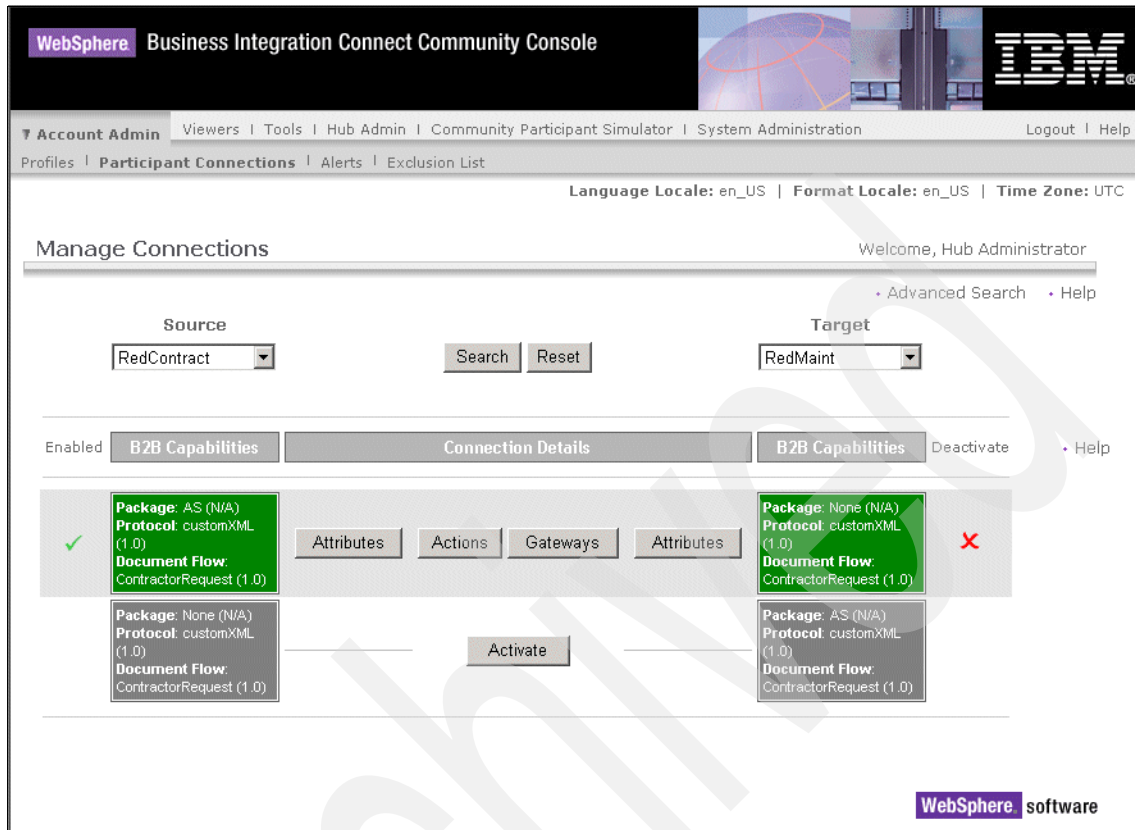


Figure 29-42 Activate connection from RedContract to RedMaint

This completes the configuration of the WebSphere Business Integration Connect server of RedMaint.

29.3 Configuring the Express server of RedContract

At the end of 29.1, “Initial configuration of the Express server” on page 567, we had a running WebSphere Business Integration Connect Express instance. However, the configuration was not finished. We will now complete it so that RedContract and RedMaint can exchange XML documents over AS2.

To configure the server:

1. Assuming that the server of redcontract is still running, start the console by selecting **Start** → **Programs** → **Business Integration Connect Express** → **Console**.
2. Log on as admin.

Updating the profile for RedMaint

To update the profile:

1. Select **Configuration** → **HTTP** from the main menu of the console.
2. Make sure that RedMaint is selected in the drop-down box labeled Selected Participant and click **Edit**.

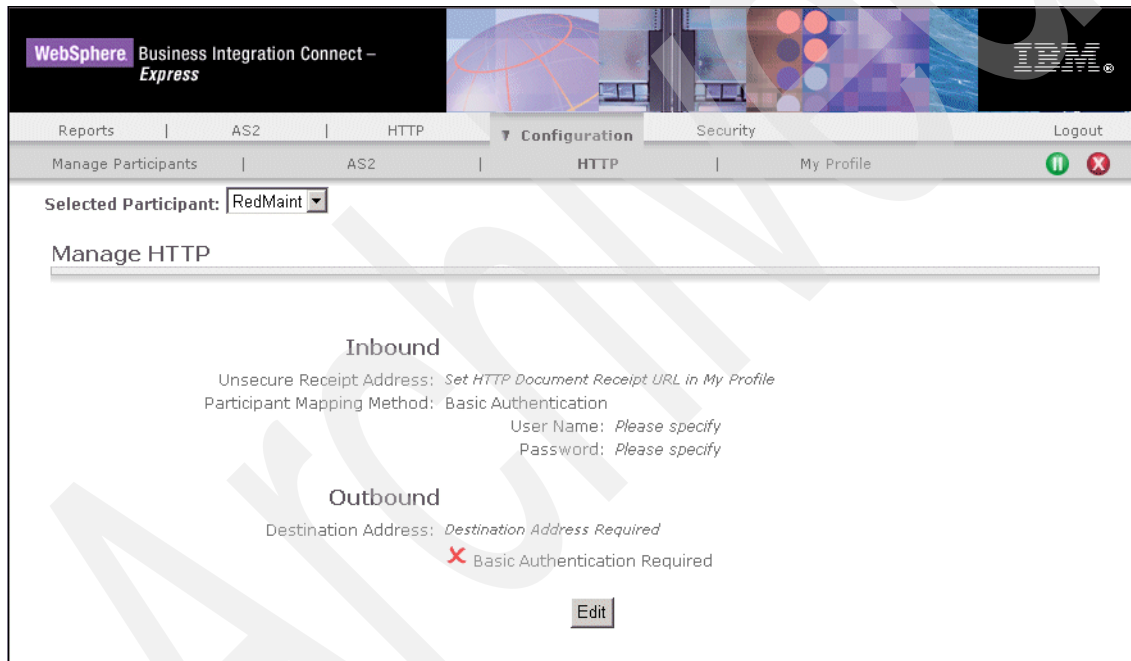


Figure 29-43 HTTP settings for RedMaint

3. In the section labeled Inbound, provide a user name and password that RedMaint should use when exchanging documents with RedContract. You used this user name and password previously when creating the gateway on the server of RedMaint. Make sure to use the same values, for example redmaint and itso4you.

4. In the section labeled Outbound, provide the URL that the server of RedMaint uses: `http://studentx:57080/bcgreceiver/input`. This URL corresponds to the HTTP target on the server of RedMaint.
5. Click **Save**.

The screenshot shows the 'Manage HTTP' configuration page for the 'RedMaint' participant in the WebSphere Business Integration Connect Express console. The page is divided into 'Inbound' and 'Outbound' sections. In the 'Inbound' section, the 'Unsecure Receipt Address' is set to 'Set HTTP Document Receipt URL in My Profile' and the 'Participant Mapping Method' is 'Basic Authentication'. The 'User Name' is 'redmaint' and the 'Password' is 'itso4you'. In the 'Outbound' section, the 'Destination Address' is 'http://studentx:57080/bcgreceiver/input'. There is an unchecked checkbox for 'Basic Authentication Required'. Below this, there are empty fields for 'User Name' and 'Password'. At the bottom of the 'Outbound' section are 'Save' and 'Cancel' buttons. The top navigation bar includes 'Reports', 'AS2', 'HTTP', 'Configuration' (selected), 'Security', and 'Logout'. A secondary bar shows 'Manage Participants', 'AS2', 'HTTP', and 'My Profile'. The 'Selected Participant' dropdown is set to 'RedMaint'.

Figure 29-44 Edit HTTP settings for redmaint

6. Select **Configuration** → **AS2** to update AS2-related settings.
7. Make sure that RedMaint is selected in the drop-down box labeled Selected Participant and click **Edit**.
8. In the section labeled Outbound, provide the URL that the server of RedMaint uses: `http://studentx:57080/bcgreceiver/input`.
9. Select the option to request on asynchronous MDN.
10. Click **Save**.

The screenshot shows the 'Configuration' tab in the WebSphere Business Integration Connect Express interface. The 'Selected Participant' is 'RedMaint'. The 'Manage AS2' section is active, showing 'Inbound' and 'Outbound' settings. The 'Outbound' settings are expanded, showing fields for 'Participant' (RedMaint), 'Destination Address' (http://studentx:57080/bcgreceiver/input), and 'Sender ID (your ID)' (Set ID in My Profile for Document Receipt). There are several checkboxes for 'Request MDN' (checked), 'Request Signed MDN', 'Sign Documents', 'Encrypt Documents', and 'Compress Documents'. Radio buttons are used for 'Synchronous' vs 'Asynchronous' and 'HTTPS' vs 'HTTP' (selected). 'Save' and 'Cancel' buttons are at the bottom.

WebSphere Business Integration Connect – Express

Reports | AS2 | HTTP | **Configuration** | Security | Logout

Manage Participants | **AS2** | HTTP | My Profile

Selected Participant: RedMaint

Manage AS2

Inbound

Unsecure Receipt Address: Set HTTP Document Receipt URL in My Profile (* MDN receipt address)
Participant: redmaint

Outbound

Participant: RedMaint

Destination Address: http://studentx:57080/bcgreceiver/input

Sender ID (your ID): Set ID in My Profile for Document Receipt

☒ Request MDN

☐ Synchronous ☒ Asynchronous

☐ HTTPS ☒ HTTP

☐ Request Signed MDN

☐ Sign Documents

☐ Encrypt Documents

☐ Compress Documents

Save Cancel

Figure 29-45 Edit AS2 settings related to partner redmaint

Updating My Profile

To update My Profile:

1. Select **Configuration** → **My Profile** and click **Edit**.
2. Provide the host name for unsecure (HTTP) receipt.
3. Provide also an AS2 ID, for example redcontract.
4. Click **Save**.

WebSphere Business Integration Connect – Express **IBM**

Reports | AS2 | HTTP | **Configuration** | Security | Logout

Manage Participants | AS2 | HTTP | **My Profile** | [Status Icons]

Manage My Profile

Receipt Address (At least one required)

	Domain	Port
Unsecure:	<input type="text" value="studentx"/>	<input type="text" value="88"/>
Secure:	<input type="text"/>	<input type="text"/>

Company AS2 ID (Required for AS2 Transmission)

Sender ID:

Company Details (Optional)

Company Name:

Address:

Business ID Type: Identifier:

Vendor Type:

Web Site:

Figure 29-46 Manage My Profile for RedContract

This completes the configuration of the WebSphere Business Integration Connect Express server of RedContract.

29.4 Trading documents

Before you can validate the communication, you need to make sure that all three server components of WebSphere Business Integration Connect are running. To validate the communication and begin trading documents:

1. Start the router (or document manager) and the receiver.
2. To validate the communication from RedMaint to RedContract, drop an XML document in the following folder:

`C:\WBICConnect\data\output\Documents\Production`

There is a sample document, `ContractorRequest.xml`, in the Additional Materials.

The document arrives in the following folder:

`C:\WBICConnect-Express\data\FileSystemAdapter2\partners\RedMaint\received\AS2\XML`

3. To validate the communication from RedContract to RedMaint, drop an XML document in the following folder:

`C:\WBICConnect-Express\data\FileSystemAdapter2\partners\RedMaint\send\AS2\XML`

There is a sample document, `ContractorResponse.xml`, in the Additional Materials.

The document should arrive in the `C:\WBICConnect\data\input` folder.

Instead of browsing folders, you can use the built-in viewers. When the files do not arrive, you will find error messages by using the Event Viewer in WebSphere Business Integration Connect Advanced.

- When documents are traded, select **Viewers** → **AS1/AS2 Viewer** in the console of WebSphere Business Integration Connect Advanced. The default search arguments look for documents sent or received in the last hour.

Figure 29-47 shows two AS2 exchanges, one in either direction.

The screenshot displays the WebSphere Business Integration Connect Community Console. The top navigation bar includes links for Account Admin, Viewers, Tools, Hub Admin, Community Participant Simulator, System Administration, Logout, and Help. The main content area is titled 'Results' and shows a list of AS2 transactions. The first transaction is from RedContract to RedMaint, and the second is from RedMaint to RedContract. Both transactions are marked as 'Production' and 'MDN Status' is 'N/A'.

WebSphere Business Integration Connect Community Console

Account Admin | **Viewers** | Tools | Hub Admin | Community Participant Simulator | System Administration | Logout | Help

Event Viewer | **AS1/AS2 Viewer** | RosettaNet Viewer | Document Viewer | Gateway Queue

Language Locale: en_US | Format Locale: en_US | Time Zone: UTC

Welcome, Hub Administrator

• Show Criteria • Help

Page 1 of 1 Total Rows: 2

Participants	Time Stamps	Document Flow	Gateway Type	Synchronous	MDN Status
Source Participant: RedContract Target Participant: RedMaint	Source: 1/27/05 8:48:51 PM	AS (N/A) customXML (1.0) ContractorRequest: Production	Production		N/A
Source Participant: RedMaint Target Participant: RedContract	Source: 1/27/05 8:43:29 PM	None (N/A) customXML (1.0) ContractorRequest: Production	Production		N/A

Page 1 of 1 Total Rows: 2

Legend

- N/A MDN Not Required
- MDN Processed
- Waiting for MDN
- Retries Exceeded

WebSphere software

Figure 29-47 List of AS2 transactions between RedMaint and RedContract



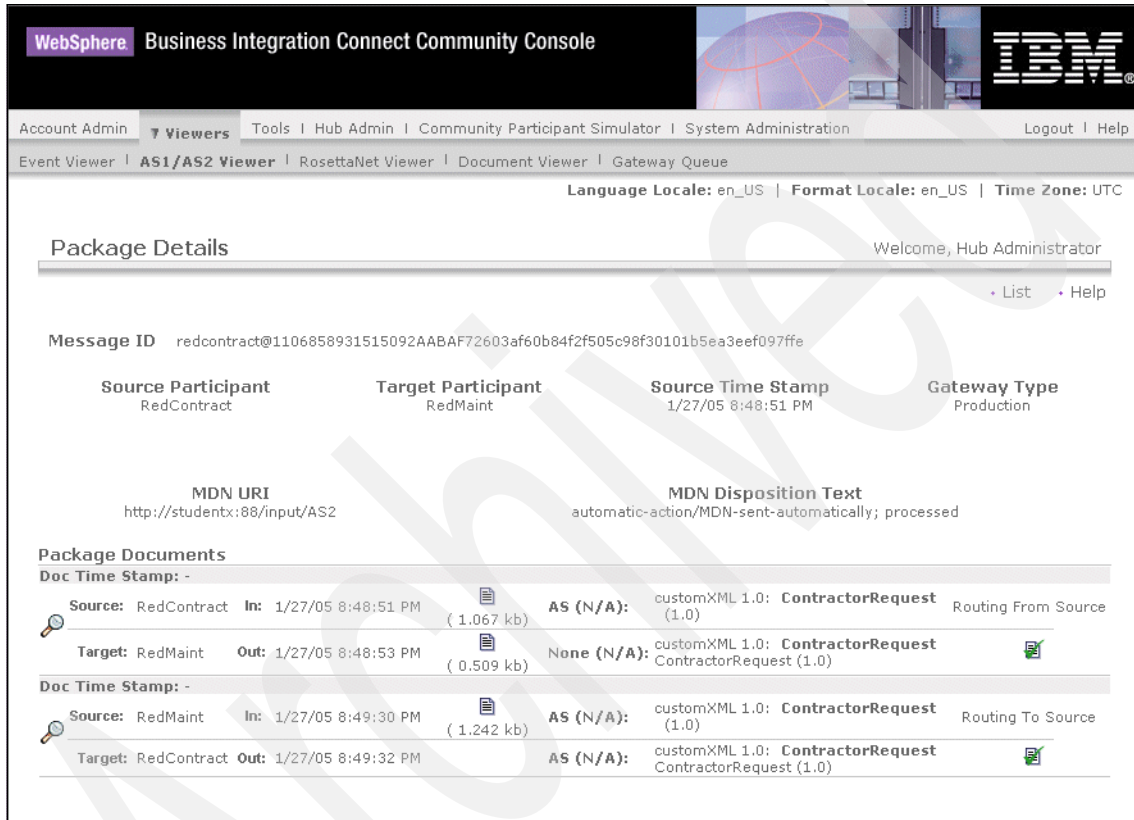
- Click the  icon next to a transaction to look at the details.

Figure 29-48 shows the incoming HTTP/AS2/XML document, the unpackaged XML document, and the MDN.

- Select one of the  icons (shown in Figure 29-48) to look at the actual document.



The screenshot displays the WebSphere Business Integration Connect Community Console interface. The top navigation bar includes links for Account Admin, Viewers, Tools, Hub Admin, Community Participant Simulator, System Administration, Logout, and Help. The main content area is titled 'Package Details' and shows the following information:

- Message ID:** redcontract@1106858931515092AABAF72603af60b84f2f505c98f30101b5ea3eef097ffe
- Source Participant:** RedContract
- Target Participant:** RedMaint
- Source Time Stamp:** 1/27/05 8:48:51 PM
- Gateway Type:** Production
- MDN URI:** http://studentx:88/input/AS2
- MDN Disposition Text:** automatic-action/MDN-sent-automatically; processed

Below this information, the 'Package Documents' section shows two document exchanges:

Doc Time Stamp: -	
Source: RedContract In: 1/27/05 8:48:51 PM Target: RedMaint Out: 1/27/05 8:48:53 PM	AS (N/A): customXML 1.0: ContractorRequest (1.0) None (N/A): customXML 1.0: ContractorRequest (1.0)
Doc Time Stamp: -	
Source: RedMaint In: 1/27/05 8:49:30 PM Target: RedContract Out: 1/27/05 8:49:32 PM	AS (N/A): customXML 1.0: ContractorRequest (1.0) AS (N/A): customXML 1.0: ContractorRequest (1.0)

Figure 29-48 Document exchanges for a single AS2 transaction

Figure 29-49 on page 618 shows the incoming document including the AS2 packaging.

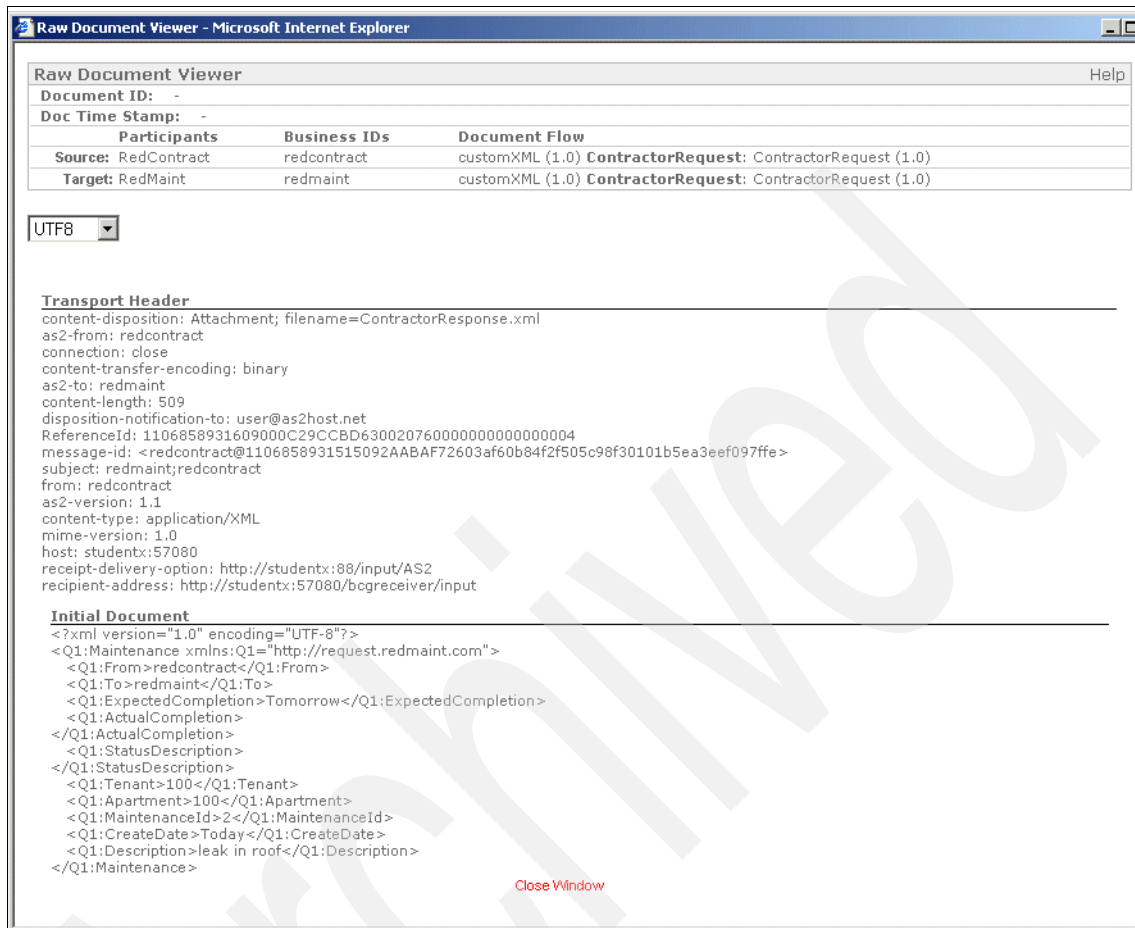


Figure 29-49 XML document received by RedMaint via AS2

7. Browse to the input folder where received documents are stored (Figure 29-50 on page 619). Notice the file names and types.

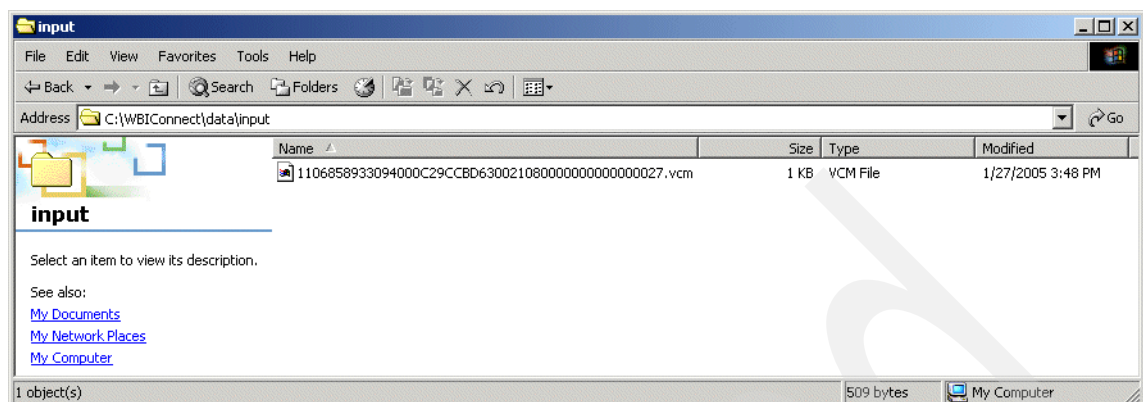


Figure 29-50 Directory contents on RedMaint

8. You can also review the logs and status on the server of RedContract (Figure 29-51). Log on to the console as user admin and select **AS2** → **Sent**.

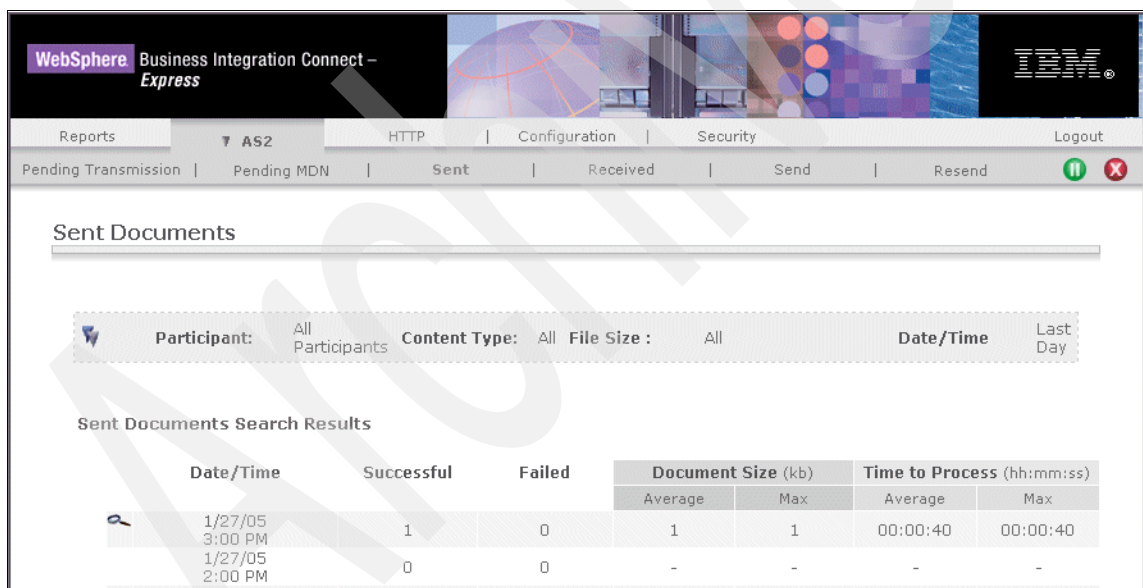


Figure 29-51 AS2 documents sent by RedContract

9. Use the **AS2** → **Received** option to look at received documents (Figure 29-52).

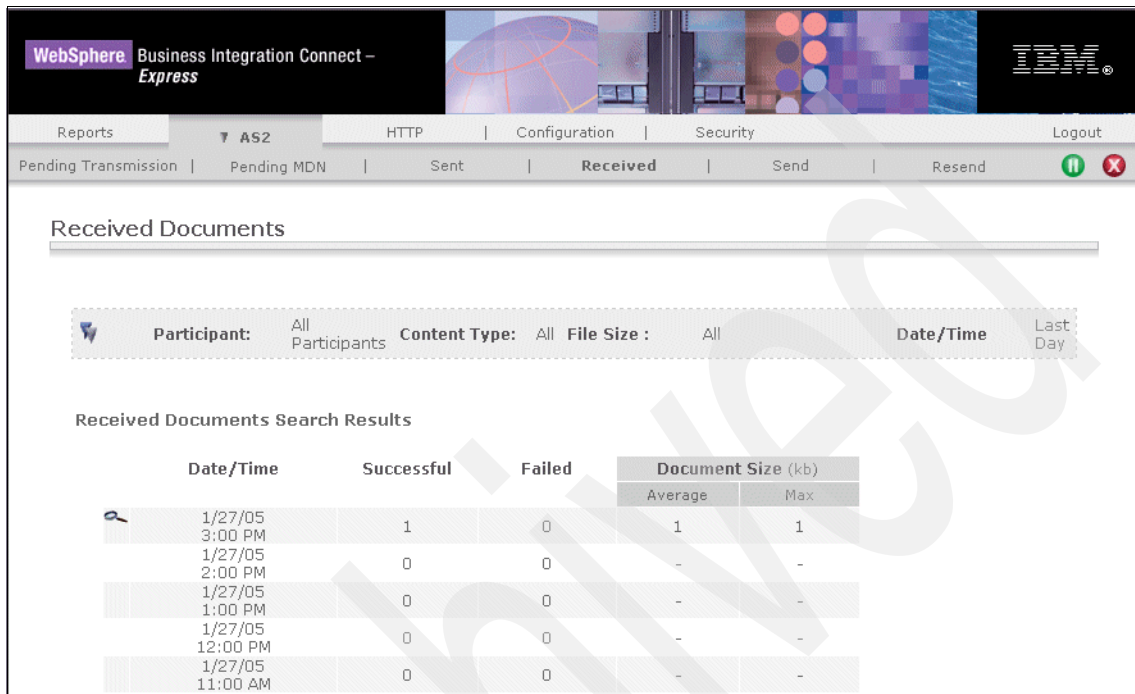


Figure 29-52 AS2 documents received by RedContract

10. On the file system, every received XML document is stored in a folder that is specific for the sender (Figure 29-53 on page 621). Thus, there can be many folders that the back-end application needs to monitor: one per partner and per format. For a system with 5 partners and each partner sends two types of documents, there are 10 inbound and 10 outbound folders. In the Advanced edition, you can change the routing functionality in many ways. In Express, you have to use the built-in routing.

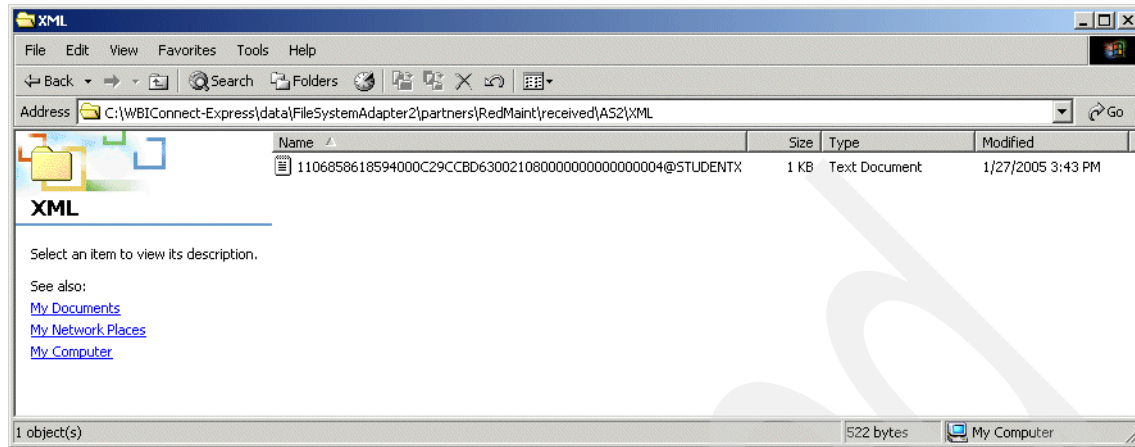


Figure 29-53 Contents of folder where received XML documents are stored

29.5 JMS setup for WebSphere Business Integration Connect

This section discusses the setup of JMS for use by WebSphere Business Integration Connect. We start with the setup of WebSphere MQ itself, so that messages can be stored in a queue and received from a queue. Next, we define the mapping between MQ resources and JMS resources. These JMS resources are then used when defining a JMS gateway and JMS target in WebSphere Business Integration Connect.

29.5.1 Defining MQ resources

For the queue manager `partner_a.bcg.queue.manager`, execute the following commands using the interactive command interface `runmqsc` of WebSphere MQ:

1. Messages that are received by WebSphere Business Integration Connect need to arrive at the queue `XML_IN`. Use the following command:

```
define qlocal(XML_IN)
```

Messages that need to be sent to RedContract need be retrieved by WebSphere Business Integration Connect from a local queue on queue manager `partner_a.bcg.queue.manager`.

Note: This queue manager was created as part of our initial installation and setup of WebSphere Business Integration Connect)

2. To create this queue, use the following command:

```
define qlocal(XML_OUT)
```

Note: Each of these MQ objects could have also be created using the WebSphere MQ Explorer application.

29.5.2 Enabling JMS

To enable JMS:

1. Open the file `jmsadmin.config` in the `C:\WMQ\Java\bin` folder in a text editor. Uncomment the setting of `INITIAL_CONTEXT_FACTORY` for the JNDI provider of the file-based JNDI provider (as shown in Example 29-1).
2. Each JNDI provider needs to be addressed in a different way, which is expressed via the parameter `PROVIDER_URL`. Update the file-based URL to point to a valid directory on the file system, for example `C:\WMQ\Java\JNDI`. Notice that the syntax of `PROVIDER_URL` uses the forward slash instead of the normal backward slash of Windows.

Note: Make sure that only one setting of each parameter is uncommented. If you want to use multiple providers, create multiple configuration files and pass the name of the configuration file as a parameter when running the JMSAdmin tool.

Example 29-1 JMSAdmin configuration file

```
# The following line specifies which JNDI service provider is in use.
# It currently indicates an LDAP service provider. If a different
# service provider is used, this line should be commented out and the
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.ReffFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#
# The following line specifies the URL of the service provider's initial
# context. It currently refers to an LDAP root context. Examples of a
# file system URL and WebSphere's JNDI namespace are also shown, commented
# out.
#
#PROVIDER_URL=ldap://polaris/o=ibm,c=us
PROVIDER_URL=file:/C:/WMQ/Java/JNDI
#PROVIDER_URL=iiop://localhost/
```

3. Use the JMSAdmin tool to create the mapping between JMS objects and MQ objects. Start JMSAdmin in a command window with C:\WMQ\Java\bin as the current directory.
4. Create a mapping between a queue connection factory object and a queue manager. You also need two mappings between the JMS queues for inbound and outbound messages and the MQ queues that were defined as described in 29.5.1, “Defining MQ resources” on page 621.
5. Figure 29-54 lists the commands that are used in JMSAdmin to set up the mapping.

The first command creates a context, called WBIC_JMS. Think of a context as a folder to store related objects. Strictly speaking, it is not required to create a context. However, it is always a good practice to group common objects together. Defining all JMS objects in the root context can make things confusing and difficult to manage.

6. After creating the context, switch to that context using the **chg ctx(WBIC_JMS)** command.
7. Create the three JMS mappings within this context.
 - The first object is called a queue connection factory.
 - The next object is a JMS queue that points to the MQ queue called XML_IN.
 - The last object, JMS queue XML_OUT, refers to the local queue XML_OUT hosted by queue manager partner_a.bcg.queue.manager.
8. To end the interactive command session in JMSAdmin, use the **end** command.

```
5648-C60, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2002. All Rights Reserved.  
Starting WebSphere MQ classes for Java(tm) Message Service Administration  
  
InitCtx> def ctx(WBIC_JMS)  
  
InitCtx> chg ctx(WBIC_JMS)  
  
InitCtx/WBIC_JMS> def qcf(WBIC_QCF) qmanager(partner_a.bcg.queue.manager)  
  
InitCtx/WBIC_JMS> def q(XML_IN) qmanager(partner_a.bcg.queue.manager) queue(XML_IN)  
  
InitCtx/WBIC_JMS> def q(XML_OUT) qmanager(partner_a.bcg.queue.manager) queue(XML_OUT)  
  
InitCtx/WBIC_JMS>end
```

Figure 29-54 Creating JMS objects using JMSAdmin

29.5.3 Creating the JMS gateway

With the JMS definitions in place, you can proceed with the definition of a JMS gateway for the WebSphere Business Integration Connect Community Manager by following these steps:

1. Log on as hubadmin to the console of redmaint.
2. Select **Account Admin** → **Profiles** → **Community Participant** and click **Search**. When the list of community participants is shown, open the profile of RedMaint.
3. Select **Gateways**. You should see the list of gateways that are currently defined, which includes the FileSystemGateway that you defined earlier.
4. Click **Create** to add a new gateway.
5. When the new gateway window appears, provide a name for the gateway (for example, JMSGateway), and select JMS as the transport. to bring up new attributes that are JMS-specific.
6. The attribute Target URI needs to have the provider URL for JNDI, which is file://C:/WMQ/Java/JNDI/WBIC_JMS in our setup.
7. The implementation class of the JNDI interface is com.sun.jndi.fscontext.RefFSContextFactory, which needs to be specified as the value for the attribute JMS JNDI Factory Name.
8. Set the attribute JMS Message Class to TextMessage.
9. The queue connection factory WBIC_QCF is the value for attribute JMS Factory Name and the queue XML_IN is the value for the attribute JMS Queue Name.

Figure 29-55 on page 625 shows the completed form.

10. Click **Save** to store the new gateway.

Note: The Target URI includes the context that is defined in JMSAdmin. However, this is not required. Alternatively, you can prefix the JMS Factory Name and the JMS Queue Name attributes with the context. Thus:

- ▶ JMS Queue Name: WBIC_JMS/XML_IN
- ▶ JMS Factory Name: WBIC_JMS/WBIC_QCF

Profile
RedMaint
Gateway List

Welcome, Hub Administrator

List
Help

Gateway Name
JMSGateway

Status
☒ Enabled
☐ Disabled

Online/Offline
☒ Online
☐ Offline

Description

Gateway Configuration

Transport
JMS

Target URI
file://C:/WMQ/Java/JNDI/WBIC_JMS

User Name

Password

Retry Count
3

Retry Interval
300
seconds

Number of Threads
3

Validate Client IP
☒ No
☐ Yes

Auto Queue
☒ No
☐ Yes

Authentication Required
☒ No
☐ Yes

JMS Factory Name
WBIC_QCF

JMS Message Class
TextMessage

JMS Message Type

Provider URL Packages

JMS Queue Name
XML_IN

JMS JNDI Factory Name
j.sun.jndi.fscontext.RefFSContextFactory

Connection Timeout
120
seconds

Configuration Point Handlers:
Select One

Save
Cancel

Figure 29-55 Creating a new JMS gateway

Before you can use the new JMS gateway, you need to update the participant connection that was used earlier for receiving AS2 documents from RedContract by following these steps:

1. Select **Account Admin** → **Participant Connections**. Select **RedContract** as the source and **RedMaint** as the target, and click **Search**.
2. Locate the connection that has AS as source package and None as target package. Click **Gateways** for that connection and change the attribute Target Gateways to JMSGateway (see Figure 29-56 on page 626). Click **Save**.

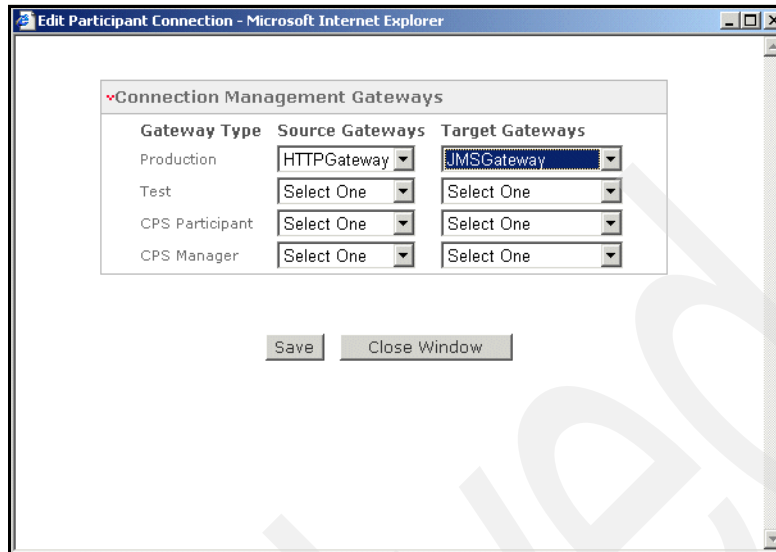


Figure 29-56 Change gateway on participant connection

29.5.4 Creating the JMS target

Creating a JMS gateway allows you to receive XML documents in a JMS queue. You now want to create a JMS target, which allows you to send JMS messages that contain XML documents to WebSphere Business Integration Connect. In 29.5.1, “Defining MQ resources” on page 621, we discussed how to define the queue XML_OUT for that purpose. You now create a JMS target mapping to that queue by following these steps:

1. Log on to the console of WebSphere Business Integration Connect as hubadmin and select **Hub Admin** → **Hub Configuration** → **Targets**.
2. When the list of targets appears, click **Create Target**.
3. When the form appears that allows we to create a new target, provide a name for it (for example, JMSTarget), and select **JMS** as the value for the attribute Transport.
4. More attributes appear, as shown in Figure 29-57 on page 627. The attribute JMS Provider URL needs to have the provider URL for JNDI, which is the file `//C:/WMQ/Java/JNDI/WBIC_JMS` in our setup.
5. The implementation class of the JNDI interface is `com.sun.jndi.fscontext.RefFSContextFactory`, which needs to be specified as the value for the attribute JNDI Factory Name.
6. Finally, the queue connection factory WBIC_QCF is the value for attribute JMS Factory Name and the queue XML_OUT is the value for the attribute

JMS Queue Name. Figure 29-57 shows the completed form. Click **Save** to store the new target.

Target Details

Welcome, Hub Administrator

List Help

Target Name: JMSTarget *

Status: ☒ Enabled ☐ Disabled

Description:

Transport: JMS *

Target Configuration

Gateway Type: Production * New Edit

JMS Provider URL: file:///C:/WMQ/Java/JNDI/WBIC_JMS *

User Id:

Password:

JMS Queue Name: XML_OUT

JMS Factory Name: WBIC_QCF

Provider URL Package:

JNDI Factory Name: com.sun.jndi.fscontext.RefFSContextFactory

Time Out: 1 ms

Thread Nbr: 1

Configuration Point Handlers: Select One

Save Cancel

Figure 29-57 Create a JMS target

Before you can use the new JMS target, you need to update the participant connection that was used earlier for sending XML documents to RedContract. Previously, these XML documents were retrieved from the \WBICConnect\data\output directory, or the target called FileSystemTarget. Now, you want to retrieve these documents in the XML_OUT queue that is represented by the JMS target. To update the participant connection:

1. Select **Account Admin** → **Participant Connections**. Select **RedMaint** as the source and **RedContract** as the target, and click **Search**.
2. Locate the connection that has None as source package and AS as target package. Click **Gateways** for that connection and change the attribute

Source Gateways to JMSGateway (see Figure 29-58 on page 628). Click **Save**.

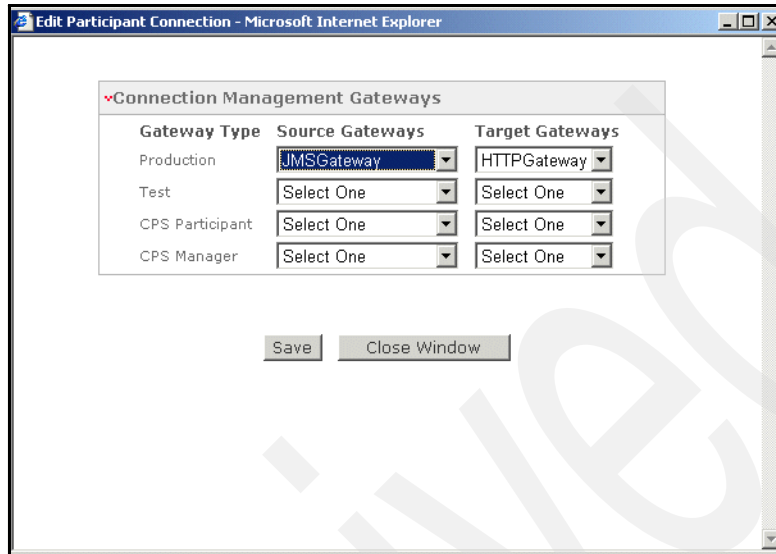


Figure 29-58 Changing gateway on participant connection

Note: There is not really a link between a target and a connection. A document can arrive on any target. Further processing depends on its characteristics.

29.5.5 Validation of the JMS and MQ configuration

The JMS configuration is now complete, so that received XML documents are now stored in a queue and XML documents to be sent can be retrieved from a queue as well (see Figure 29-59 on page 629) . Note that on the side of RedContract, everything is still file based.

To validate the configuration:

1. Drop the XML file ContractorResponse.xml in the following directory:
C:\WBICConnect-Express\data\FileSystemAdapter2\partners\redmaint\send\AS2\XML

If all goes well, the XML document is received by redmaint and is sent onto the XML_IN queue.

2. To verify the message contents, use a utility such as **rfhutil**.

3. Start **rfhuti1**. Select the correct queue manager and queue (XML_IN) and click **Read Q**.
4. When a message is read, select the Data tab.

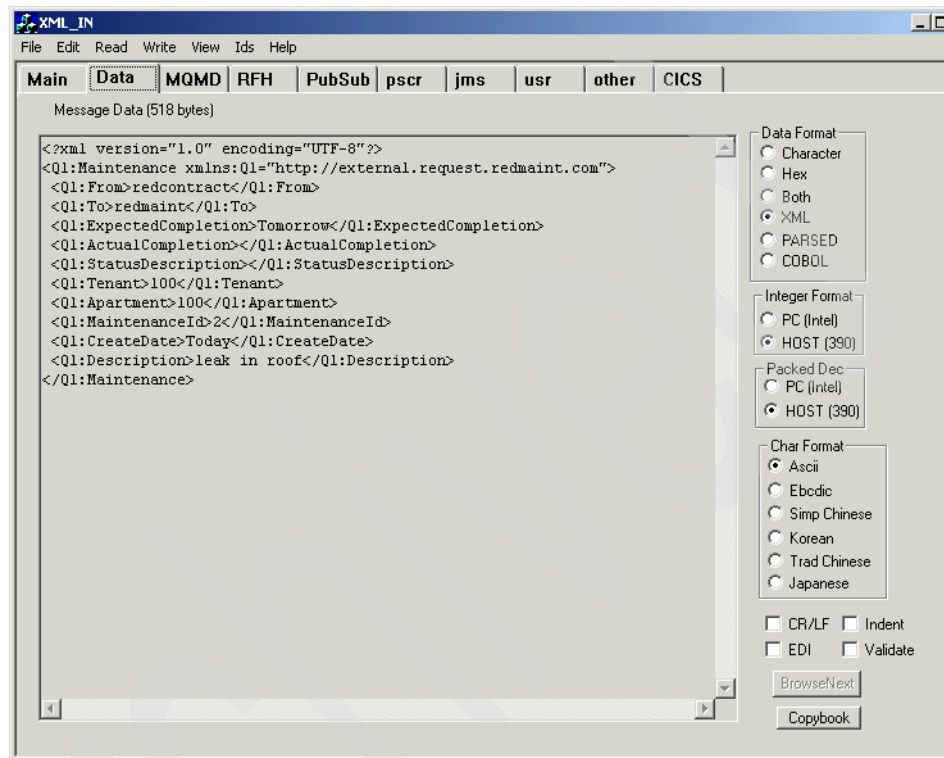


Figure 29-59 Received XML message

5. If no message was available, make sure that all WebSphere Business Integration Connect components are running. Use the Event Viewer or Document Viewer to find any error messages.

To test the outbound flow, you need to write a message to the XML_OUT queue. You then need to verify that the message gets picked up by WebSphere Business Integration Connect and that it arrives at the intended partner. You can use again the tool **rfhuti1** for such a test by following these steps:

1. On the tab Main of **rfhuti1**, select **Read File** and locate the file ContractorRequest.xml in the Additional Materials folder.
2. Select the correct XML_OUT queue and click **Write Q**.

Figure 29-60 on page 630 shows a file that is read and then sent to the XML_OUT queue.

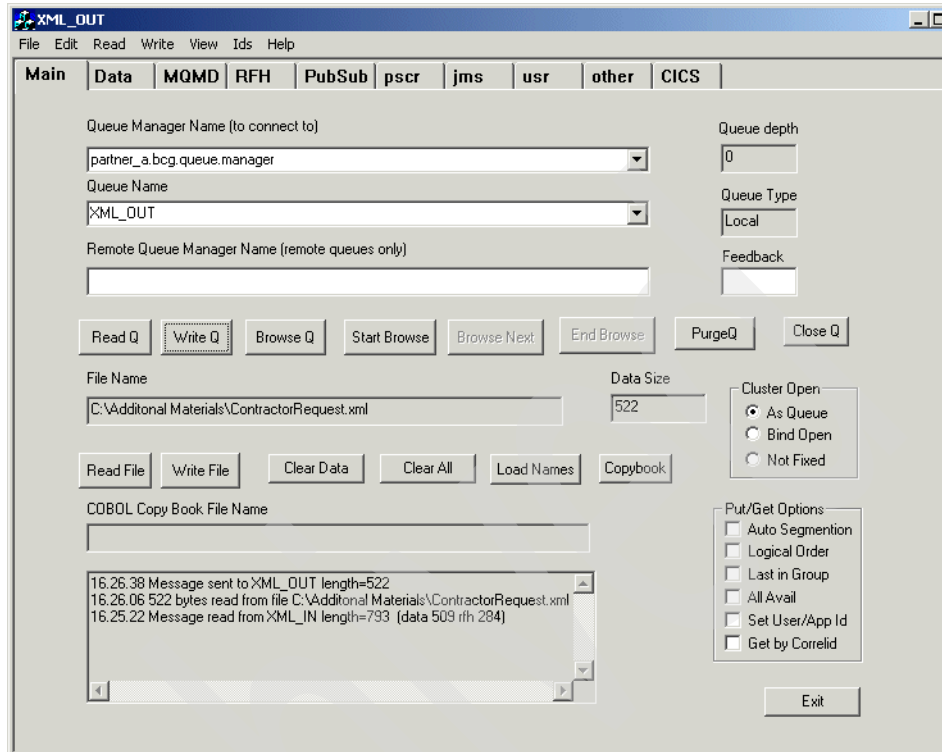


Figure 29-60 Read from file and write to queue

You now have an environment where you can use MQ messages to send and to receive XML documents to and from partners.



Part 7

Server Foundation components

Adding human interaction using WebSphere BI Server Foundation

This chapter extends the integration solution by adding a simple workflow to manage maintenance requests from tenants. During the execution of the workflow, reports are sent back to the maintenance application about the progress of the work request.

The human interaction is reasonably simple, where someone reviews the maintenance request and determines whether it is possible to perform the maintenance request in-house. If it is possible to perform the maintenance request in-house, a second human step consists of reporting the completion of the work request.

We develop this workflow in WebSphere Studio Application Developer Integration Edition, Version 5.1.1 with FixPack 4 applied. We test the solution in a test server that is modelled after WebSphere Business Integration Server Foundation.

30.1 Developing the services

The first step is to develop two services:

- ▶ A service that accepts a maintenance request document and sends it to a JMS destination.
- ▶ A service that interacts with the RMConnector developed in previous chapters. This service is used to report to the RedMaintenance application about the progress of the maintenance request. It is also used to kick-off the process itself. That is, the creation of maintenance request is an event that will kick off a process in Server Foundation (using an instance of the RM adapter which has been configured for use with a WebSphere Application Server, we call this the RM2 Connector).

30.1.1 Step 1: Developing the InhouseSend service

The InhouseSend service accepts a maintenance request document and sends it to a JMS destination. To develop this service:

1. Start Studio using the shortcut in the Start menu and use the default workspace.
2. Switch to the Business Integration perspective if needed. Select **Window → Open Perspective → Business Integration**.
3. Create a new service project. Select **File → New → Service Project**.
4. Provide a name for the project (for example, Maintenance_Processes), and click **Finish** (Figure 30-1 on page 635).

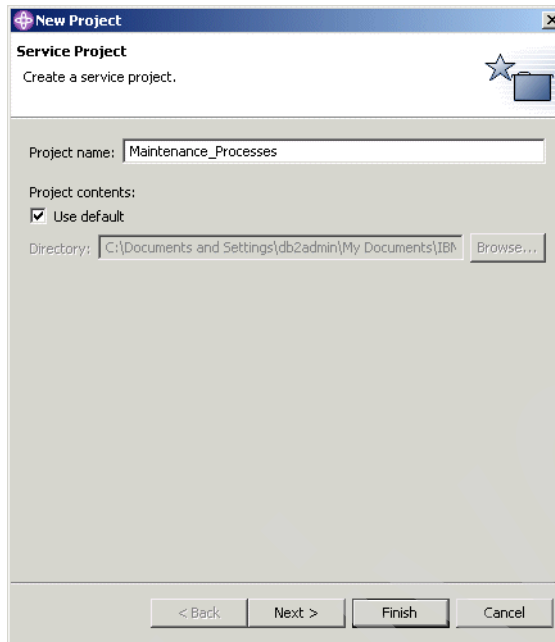


Figure 30-1 Creating a new service project

5. Select **File** → **Import** to import the schema definition of the message that you will send.
6. Select File System and click **Next** (Figure 30-2 on page 636).

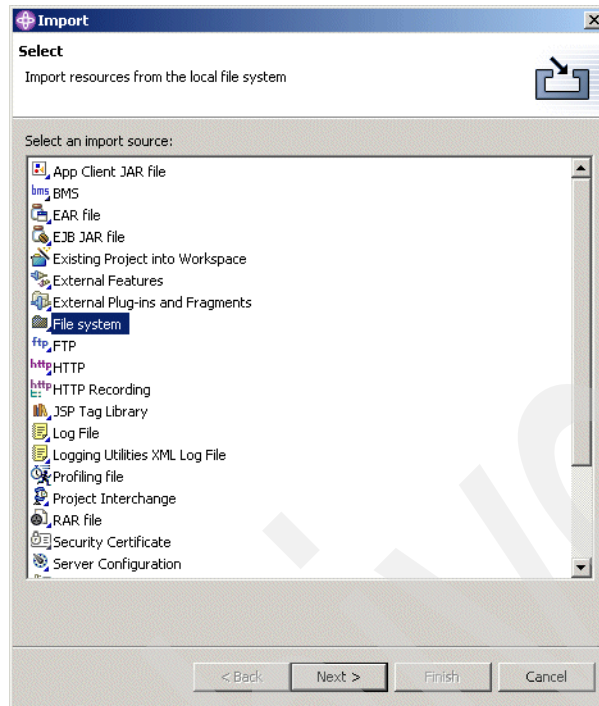


Figure 30-2 Importing resources from the file system in the project

7. Set the directory to C:\Additional Materials, where the schema is provided for you.
8. Select the file MaintenanceRequest.xsd.
9. Make sure that the folder name is set to Maintenance_Processes, and click **Finish** (Figure 30-3 on page 637).

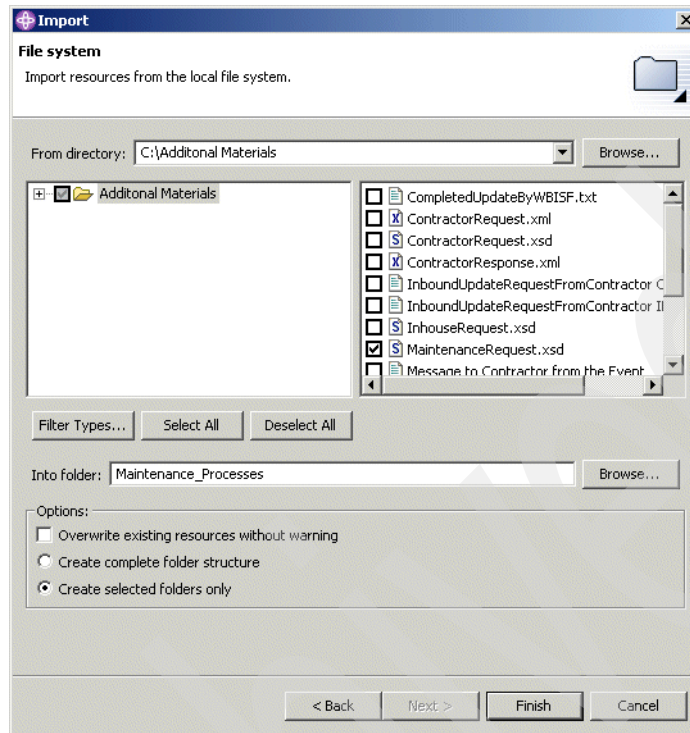


Figure 30-3 Importing schema in service project

10. Open the schema in Studio and review its structure (Figure 30-4 on page 638).

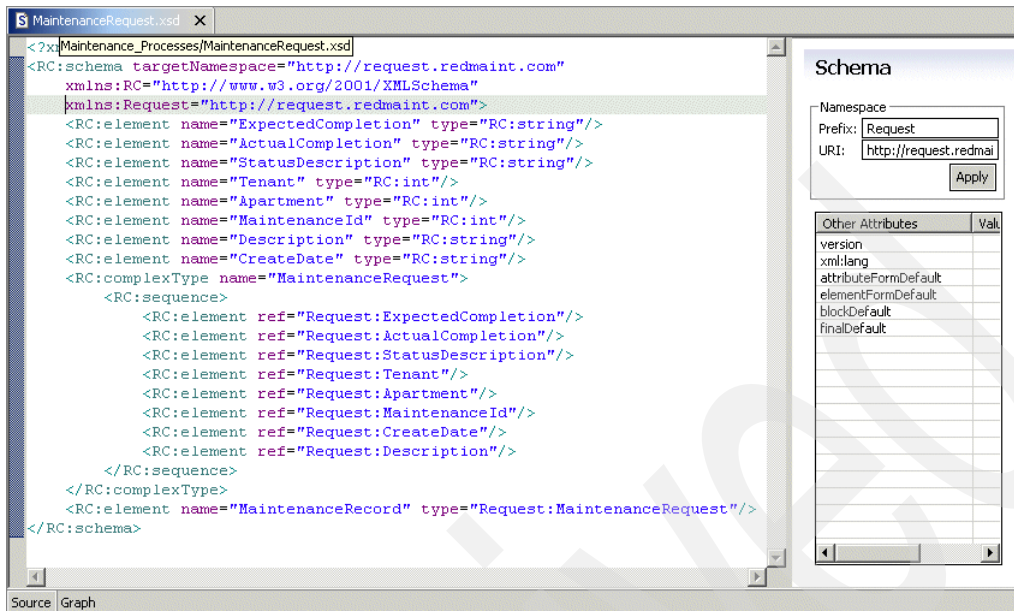


Figure 30-4 Review schema definition

11. Save the schema and close it.
12. Right-click the service project Maintenance_Processes and select **New** → **Empty Service**.
13. Set the package name to inhouse.send, set the file name to InhouseSend, and click **Finish** (Figure 30-5).

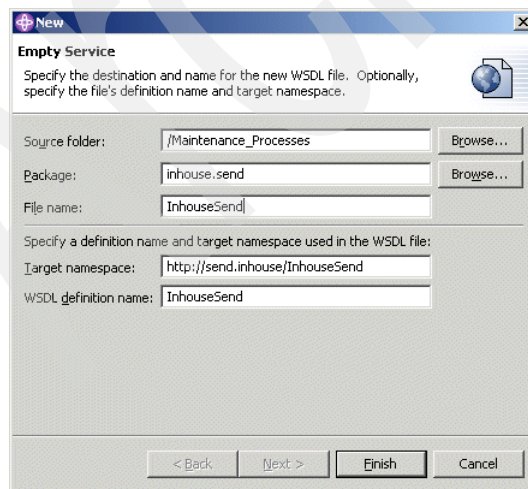


Figure 30-5 Creating a new empty service

The WSDL editor opens, either in the graph view (as shown in Figure 30-6) or in the source view. You can change the view by selecting the appropriate tab at the bottom of the WSDL editor.

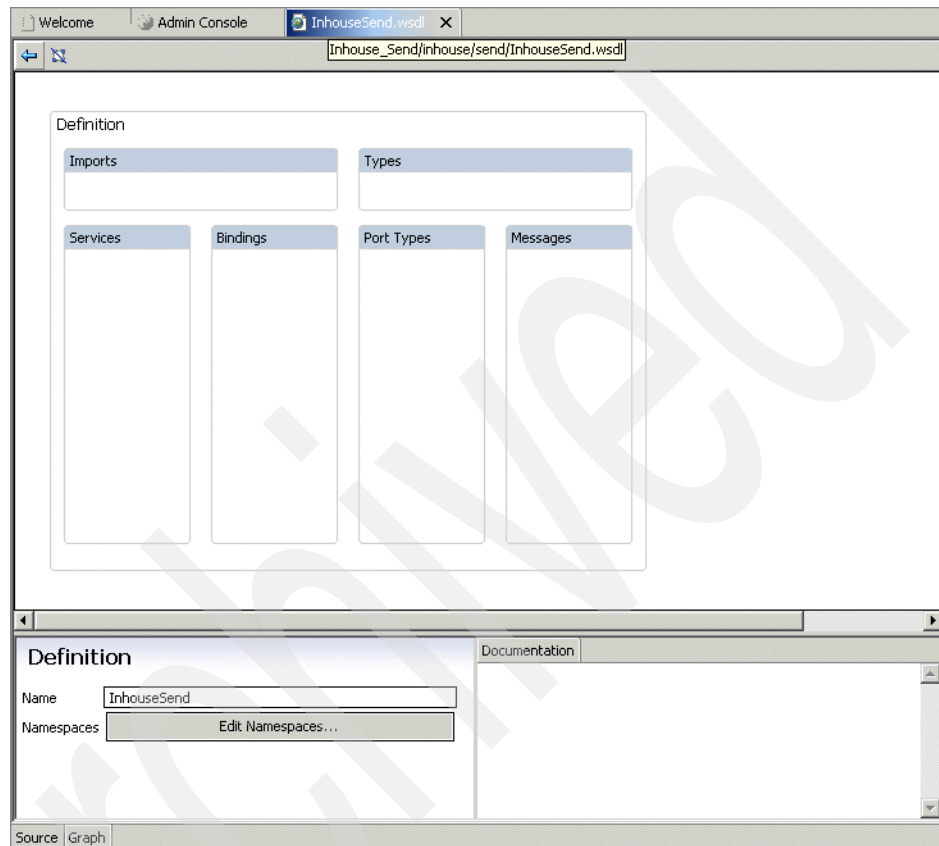


Figure 30-6 WSDL editor

30.1.2 Exploring the WSDL editor

This section introduces you to the WSDL editor. It does not contain any steps to be performed. If you are familiar with the WSDL editor, you can skip this section.

This section describes the elements of the WSDL editor and lists the some of the elements in a WSDL file. The first sample WSDL file is not used in later chapters of this book. However, building the file introduces you to the major features of the WSDL editor.

Editor views

The editor contains two views:

- ▶ Source view, where you can see the XML source code that is generated in the editor. You can directly edit the source code of the WSDL in this view.
- ▶ Graph view, where you can see a graphical representation of the WSDL file that allows you to create or change a WSDL without needing to edit the source XML directly.

Editor panes

Figure 30-7 is an example of the graph view in the new WSDL editor.

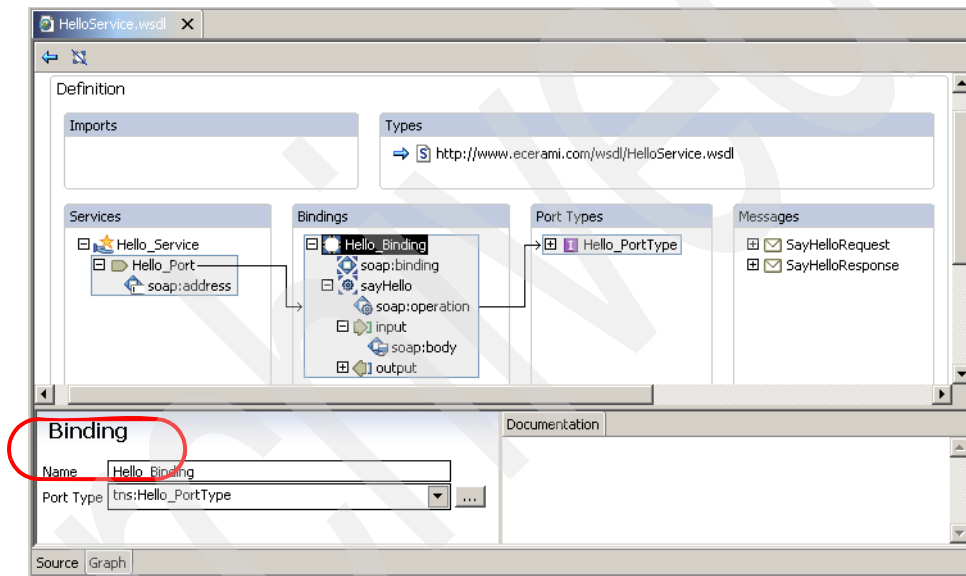


Figure 30-7 Panes in the WSDL editor

There are three panes that are associated with the editor:

- ▶ Elements pane, which is the top pane and holds the graphical representation of the elements of the WSDL.
- ▶ Details pane, which is the bottom left-hand pane and generally shows the attributes of the selected element in the elements view. Only attributes from the active WSDL can be changed here. Attributes from imported WSDLs must be changed in their respective file.
- ▶ Documentation pane, which is the bottom right-hand pane that allows you to set and view documentation about the selected element.

Elements pane

The elements pane consists of seven containers that correspond to the elements of a WSDL file. The containers are:

- ▶ Imports
- ▶ Type
- ▶ Services
- ▶ Bindings
- ▶ Port type
- ▶ Messages
- ▶ User-defined types

Imported files

When another file is imported, you see the definitions from that file in the relevant container as though it was defined in the active WSDL file. You cannot update or change the elements from an imported file. They are essentially opened as read only.

In the editor, you can tell the elements from an imported file because they are greyed out, as shown in Figure 30-8 where the only elements that are actually defined in the active WSDL are under the services container.

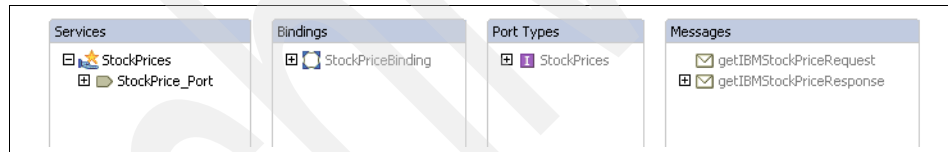


Figure 30-8 Import elements are greyed out in the active WSDL

Tasks in the WSDL editor

The following list contains some of standard tasks that you can perform within the WSDL editor. These tasks are grouped by the element pane in which the task is generally done.

- ▶ Port Types
 - Create port type
 - Add an operation to a port type
 - Add input/output/fault to an operation
 - Associating a message to input/output/fault
- ▶ Messages
 - Create new message
 - Add a part to a message
 - Assigning an XSD element to a part
 - Setting the type of a part

- Bindings
 - Create binding
 - Set a port type
- Services
 - Create a service
 - Add port to a service

30.1.3 Step 2: Developing the InhouseSend service

Now that you have a better understanding of the WSDL editor, you can continue developing the service. Follow these steps:

1. Create a port type called InhouseSend_PortType by right-clicking in the port type container and choosing **Add Child** → **portType**.
2. Type in the name InhouseSend_PortType and click **OK**.
3. Create an operation against this new port type by right-clicking **InhouseSend_PortType** and choosing **Add Child** → **operation**.
4. Call the operation send and click **OK**.
5. Give the operation an input element by right-clicking the send operation and choosing **Add Child** → **input**.

The WSDL should look similar to Figure 30-9.

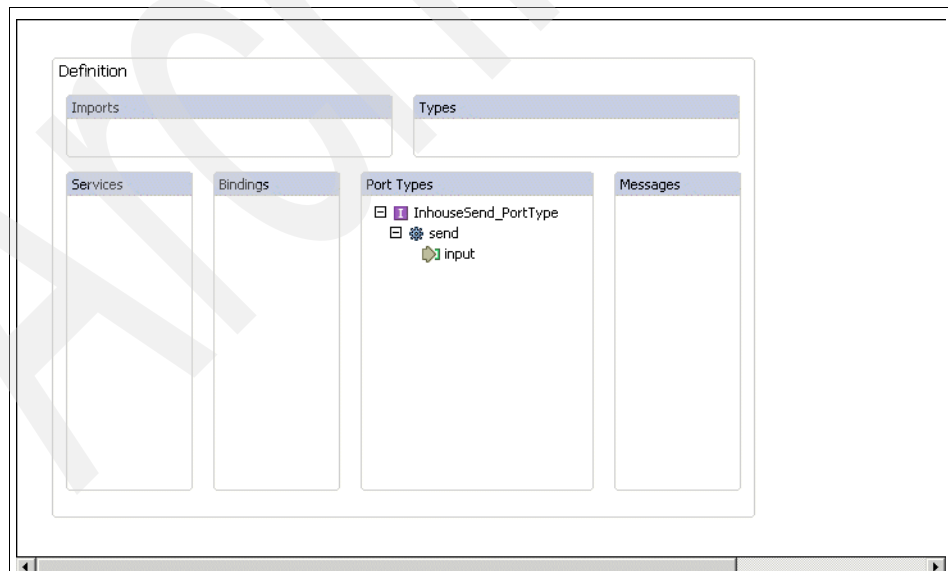


Figure 30-9 Sample WSDL after creating operation

Next, you create a message to go with the input. There is no output message that is associated with this service. To create a message:

1. Right-click input and choose **Set Message**.
2. Take the default of create message and leave the default name of sendRequest and click **Finish** (Figure 30-10).

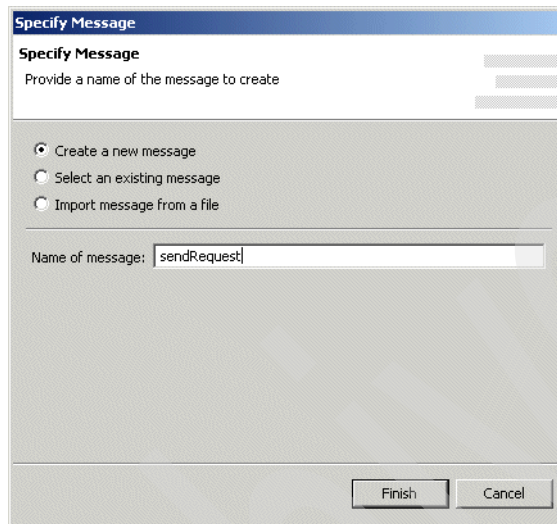


Figure 30-10 Creating a message for the input of the service

The WSDL should look similar to Figure 30-11.

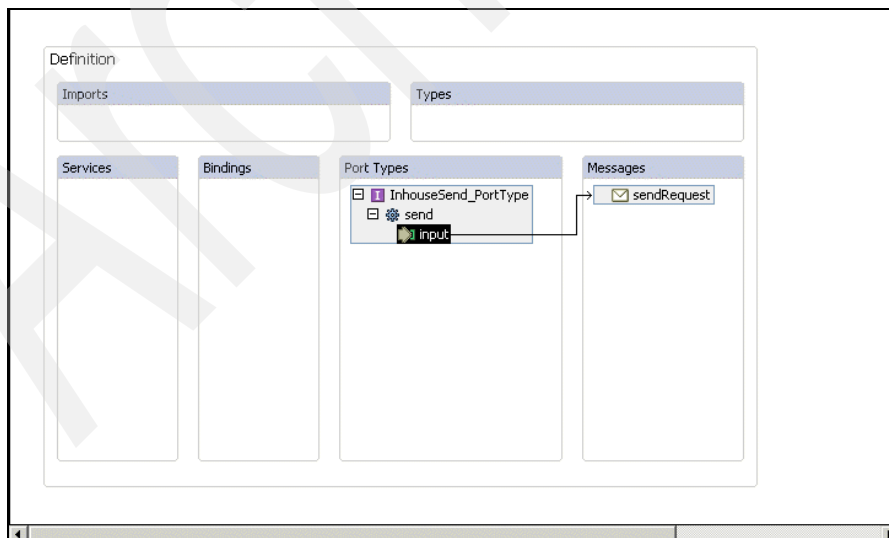


Figure 30-11 Sample WSDL after creating port type and operation

Next, you add some substance to the messages.

3. Right-click sendRequest and choose **Add Child** → **part**. Give the part a name of sendRequestPart and click **OK**.
4. Right-click sendRequestPart and select **Set Type**.
5. Select the option to import from file and click **Browse**.
6. Locate the MaintenanceRequest.xsd file and click **OK** (Figure 30-12).

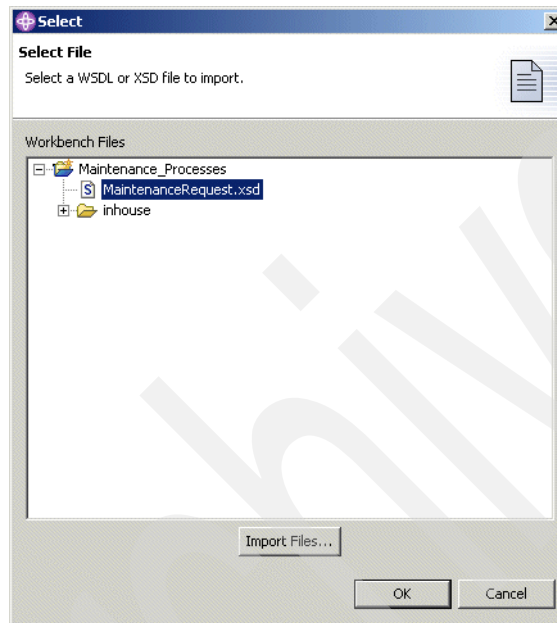


Figure 30-12 Associate schema definition with WSDL message

7. Select the type MaintenanceRequest and click **Finish** (Figure 30-13).

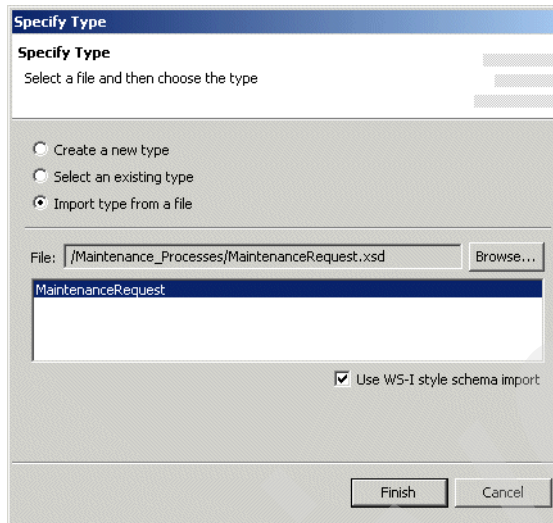


Figure 30-13 Set the type of the WSDL message

Figure 30-14 on page 646 shows how the WSDL editor should look. If you select the input of the operation, it should point to the message from where you should see the schema structure. You might need to scroll to the right to see the schema.

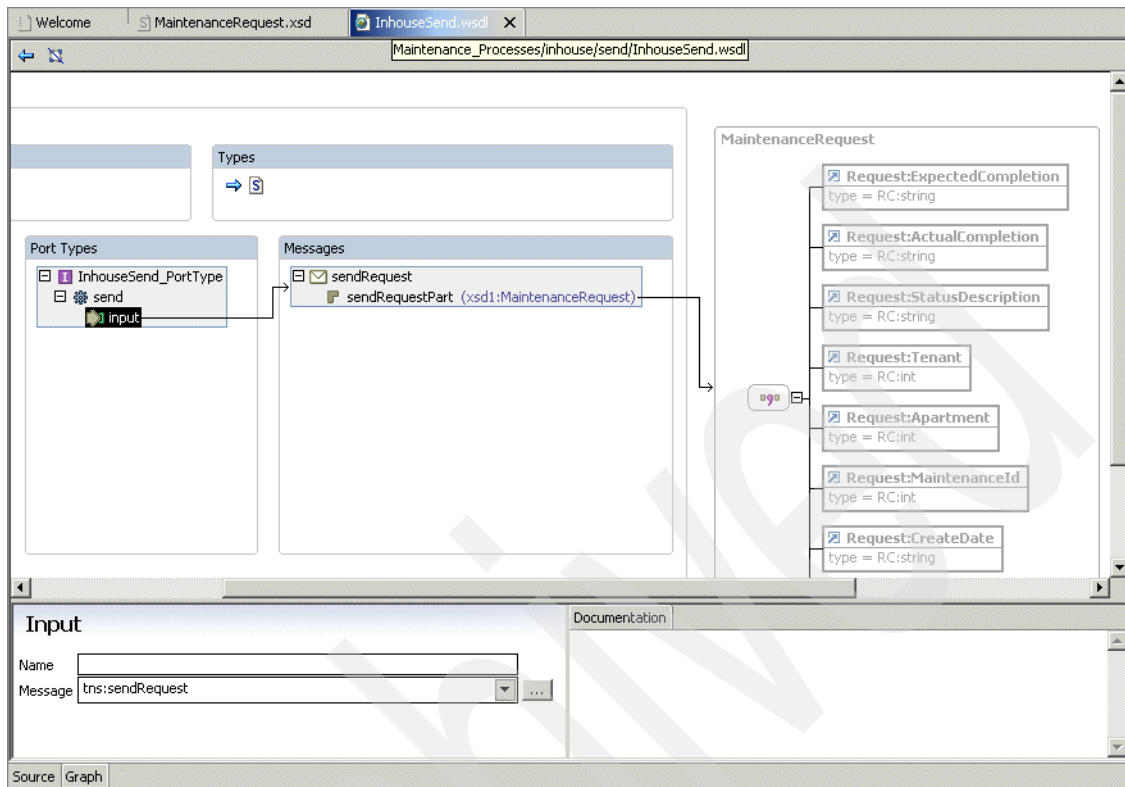


Figure 30-14 Sample WSDL after setting the type of the message

Next, you need to create a binding. You could add the binding and port to the same WSDL file. However, to separate the abstract and concrete part of the WSDL, we will create separate files.

8. Right-click the package and select **New** → **Empty Service**.
9. Name the service `InhouseSendService` and click **Finish** (Figure 30-15).

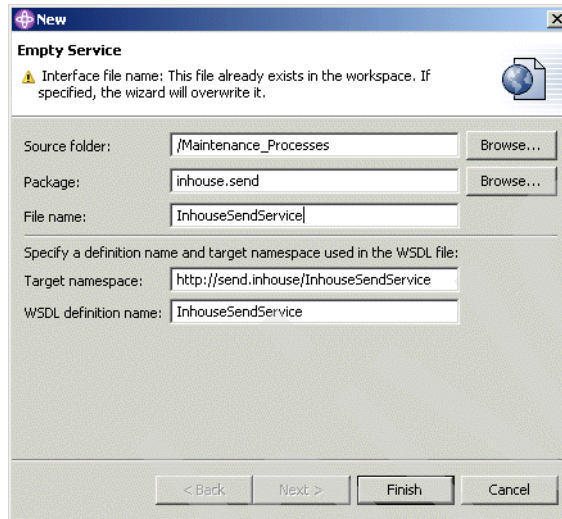


Figure 30-15 Create a new service

10. When the WSDL editor is opened, import the schema definition. Right-click in the import container and select **Add Child** → **Import**.

11. Select the button below and load the XSD file, as shown in Figure 30-16.

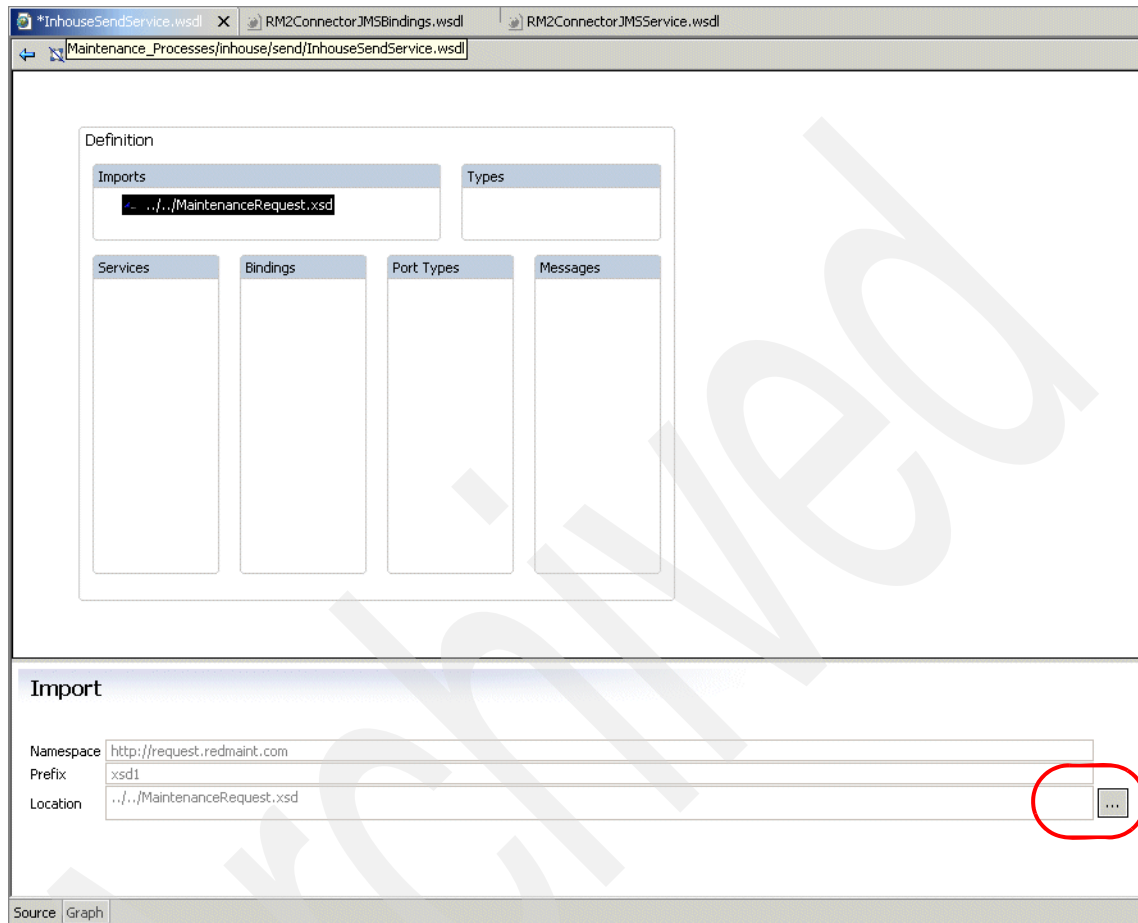


Figure 30-16 WSDL file with an import

12. Import, in a similar way, the abstract WSDL file InhouseSend.wSDL, as shown in Figure 30-17.

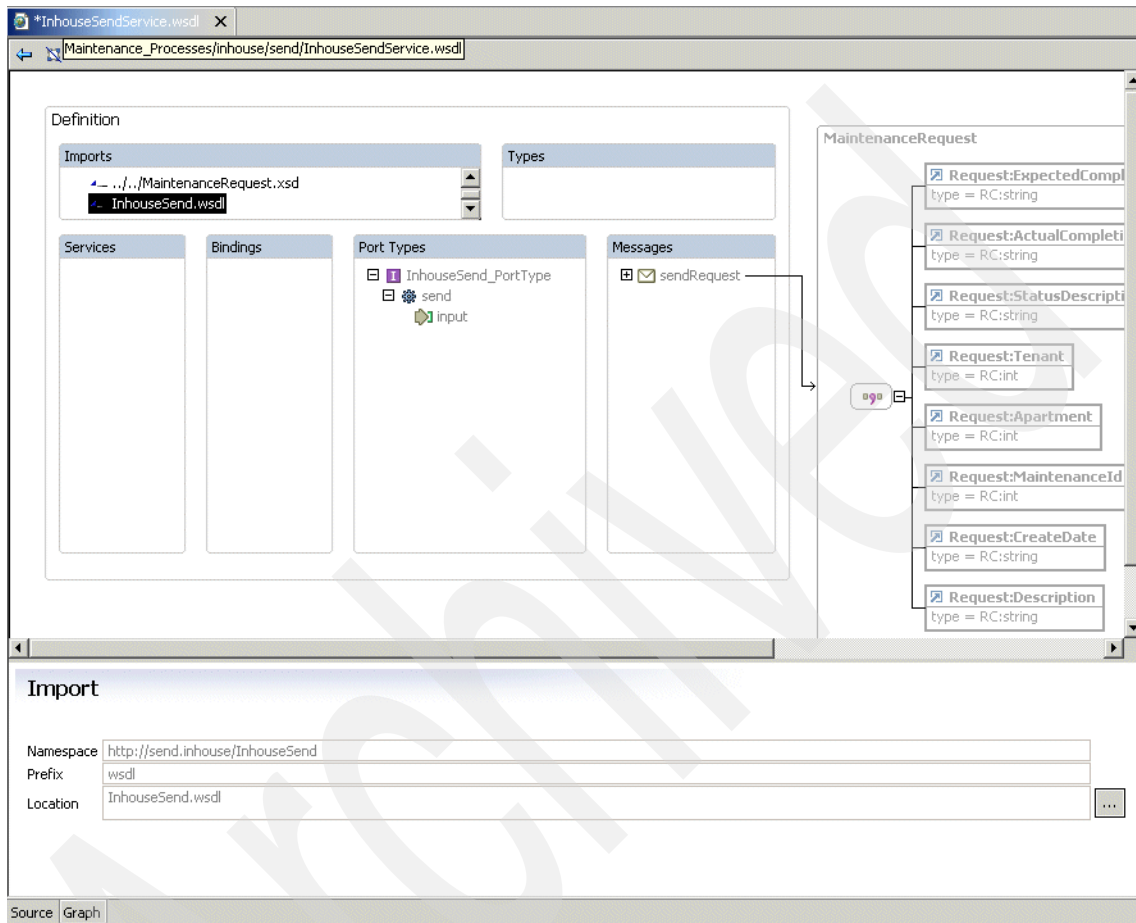


Figure 30-17 Import the abstract WSDL

13. Right-click the bindings container and choose **Add Child** → **binding**.
14. Complete the following in the Specify Binding Details window (Figure 30-18 on page 650):
 - a. Give it the name **InhouseSend_Binding**.
 - b. Choose **wsdl:InhouseSend_PortType** in the Port Type menu.
 - c. Choose **JMS** in the protocol menu to activate more options.
 - d. Set the Message Type to **TextMessage**.
 - e. Click **Input message properties**.

You can further specify the properties of the JMS message. By default, the body of the message is set to the WSDL message part, which is what we need. However, you can also set JMS header fields or custom properties if desired.

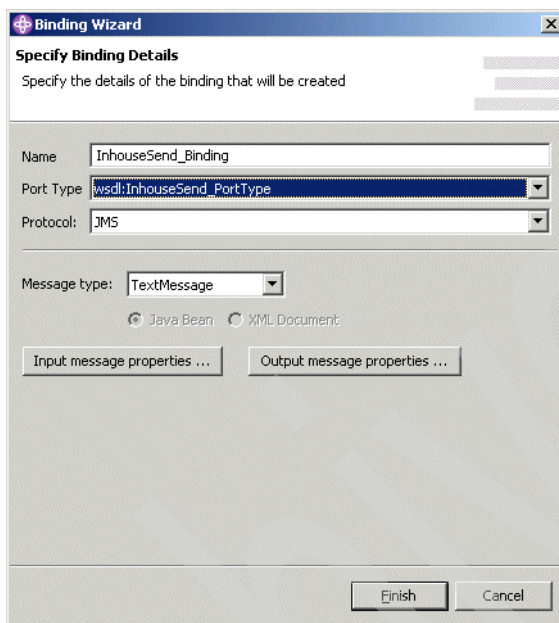


Figure 30-18 Create a binding to the port type for JMS

15. Click **Finish** to close the input message properties window
16. Click **Finish** again to close the binding window.

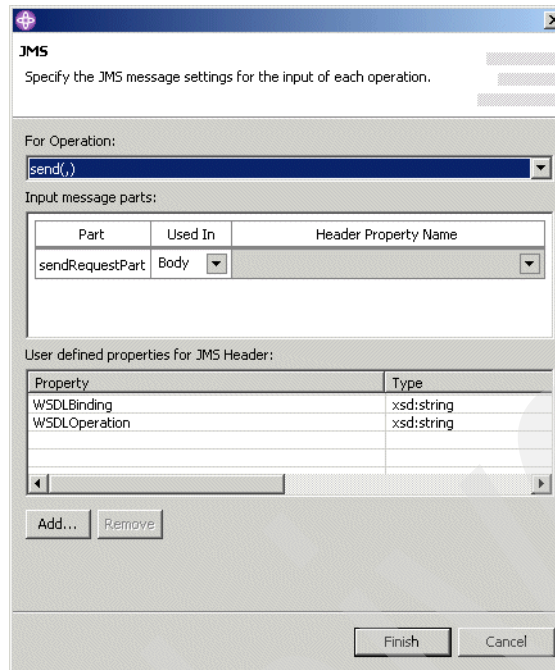


Figure 30-19 Review JMS message properties

The WSDL should look similar to that shown in Figure 30-20 on page 652. Because you chose the port type when creating the binding, it has created the binding operation and attached it to the `send` operation in the port type automatically. Also, a format mapping has been defined that determines how the XML message should look and that is used by the tools in WebSphere Studio Application Developer to generate helper classes that build the XML message from a Java object.

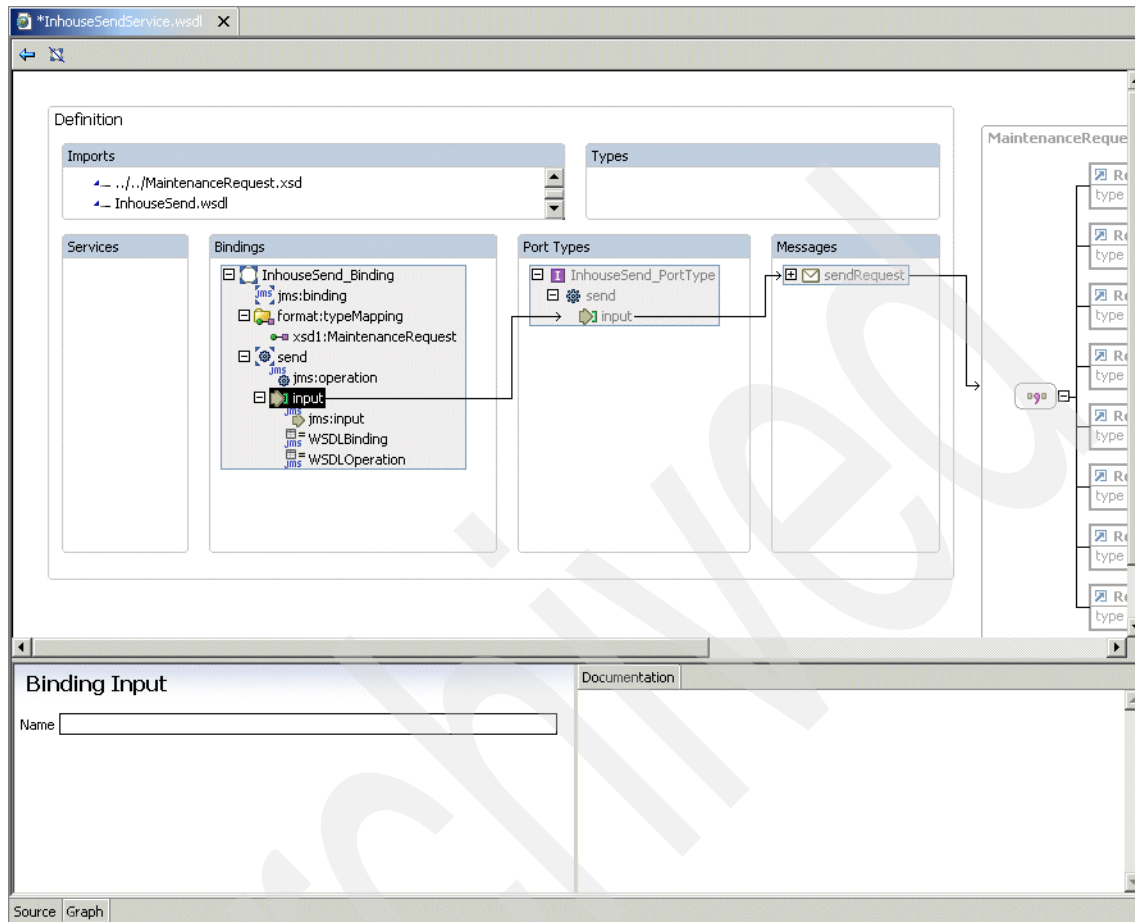


Figure 30-20 WSDL with binding defined

You now need to create a service, as described in the following steps:

1. Right-click the services container and choose **Add Child** → **service**.
2. Call the service `InhouseSend_Service` and click **OK**.
3. Right-click the `InhouseSend_Service` and choose **Add Child** → **port**.
4. Complete the following in the Specify Port Details window (Figure 30-21 on page 653):
 - a. Call the port `InhouseSend_Port`.
 - b. From the binding drop-down choose **InhouseSend_Binding**.
 - c. From the protocol drop-down choose **JMS** to match the protocol of the binding chosen.

- d. Change the field JNDI connection factory name to `jms/wbisf.queue.manager`.

Note: We created this queue manager for use with our business process.

- e. Change the field JNDI destination name to `jms/inhouse.maintenance.request`.

The screenshot shows a 'Port Wizard' dialog box with the title 'Specify Port Details'. Below the title is the instruction 'Specify the details of the port that will be created'. The dialog contains several input fields and dropdown menus:

- Name:** A text field containing 'InhouseSend_Port'.
- Binding:** A dropdown menu showing 'tns:InhouseSend_Binding'.
- Protocol:** A dropdown menu showing 'JMS'.
- JNDI connection factory name:** A text field containing 'jms/wbisf.queue.manager'.
- JNDI destination name:** A text field containing 'jms/inhouse.maintenance.request'.
- JNDI provider URL:** A text field containing 'iiop://localhost:2809'.

At the bottom right of the dialog are two buttons: 'Finish' and 'Cancel'. The 'Finish' button is highlighted with a blue border.

Figure 30-21 Set JMS parameters for the port

- 5. Click **Finish**.
- 6. Select **File** → **Save** or press CTRL+S to save the WSDL file.

The WSDL should look similar to that shown in Figure 30-22. You can close the WSDL file at this point.

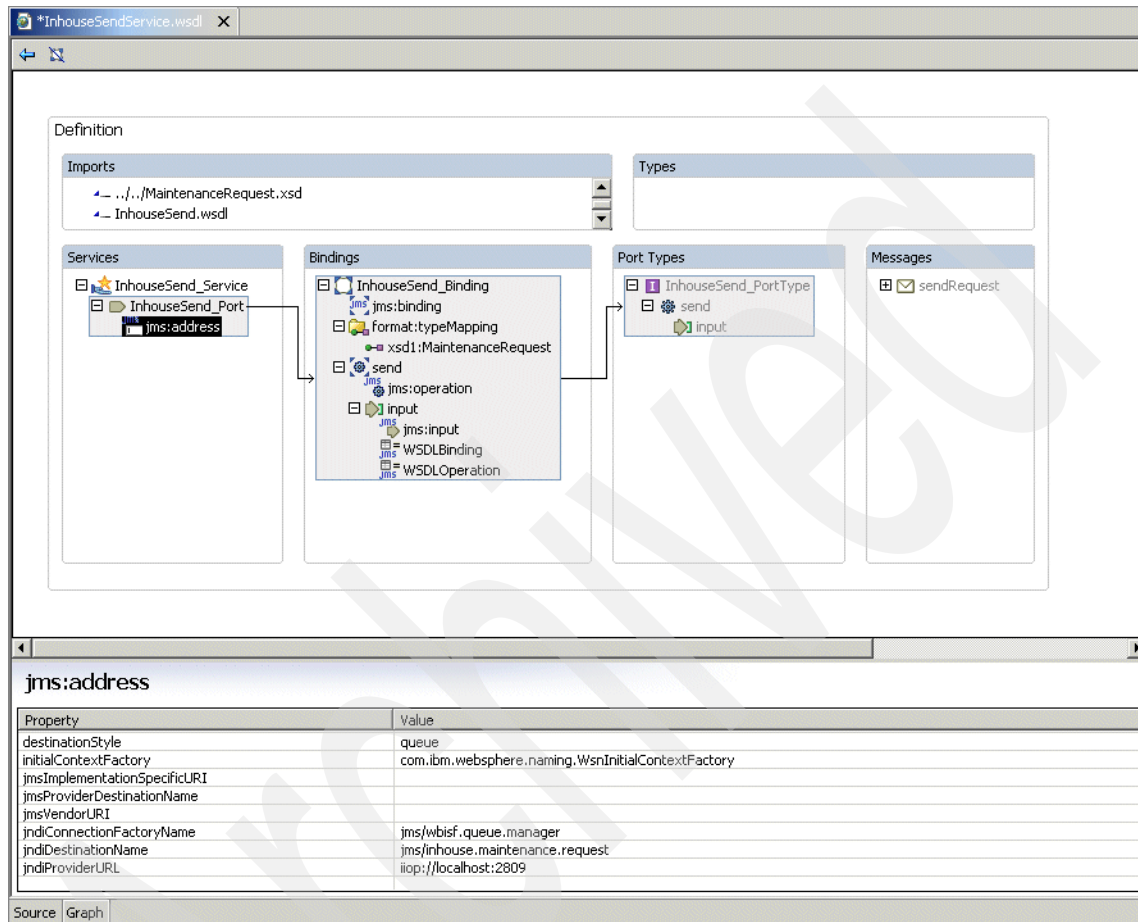


Figure 30-22 Completed WSDL

30.1.4 Generating deploy code for the JMS service

We will use the WSDL that we created to generate an EJB™. This EJB performs the JMS send based on the specifications of the WSDL. This EJB can be considered as the implementation of the service.

To generate the deploy code for the JMS service:

1. In the pane Services, right-click InhouseSendService.wsdl and select **Enterprise Services** → **Generate Deploy Code**.
2. Make sure that the option Generate Helper Classes is set.
3. Set the inbound binding type to EJB and click **Next** (Figure 30-23).

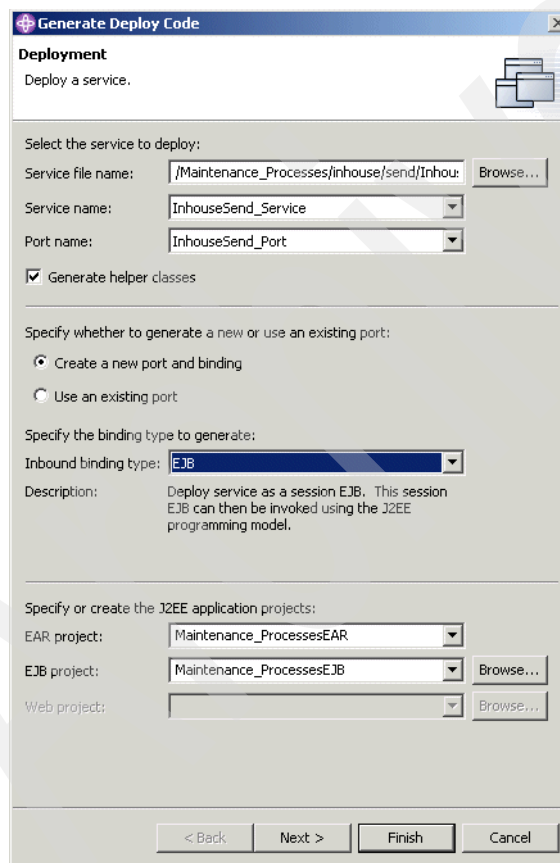
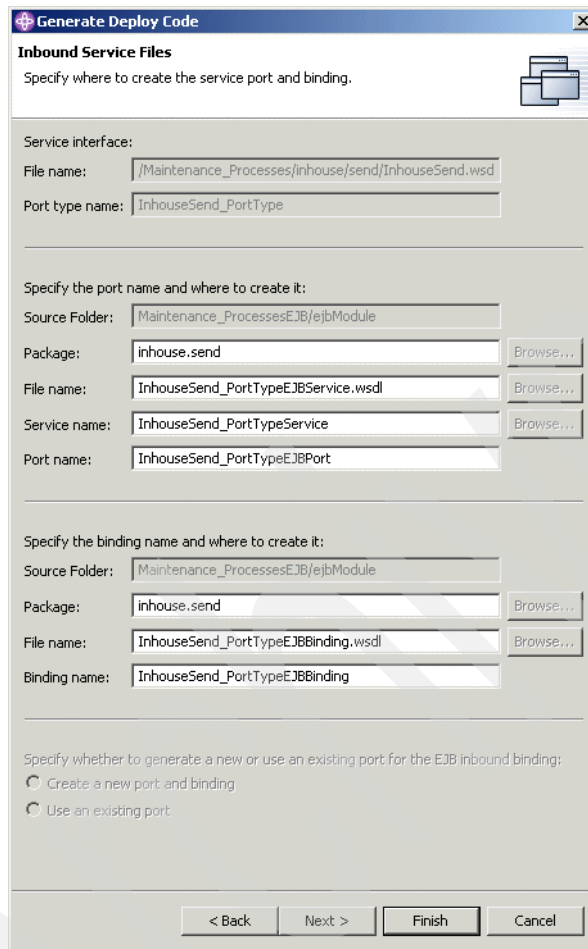


Figure 30-23 Generate deploy code for the JMS service

4. Review the generation options (Figure 30-24). Note that new WSDL files are going to be created for the EJB. Click **Next**.



The dialog box is titled "Generate Deploy Code" and contains the following sections:

- Inbound Service Files**: Specify where to create the service port and binding.
- Service interface**:
 - File name: /Maintenance_Processes/inhouse/send/InhouseSend.wsd
 - Port type name: InhouseSend_PortType
- Specify the port name and where to create it:**
 - Source Folder: Maintenance_ProcessesEJB/ejbModule
 - Package: inhouse.send
 - File name: InhouseSend_PortTypeEJBService.wsdl
 - Service name: InhouseSend_PortTypeService
 - Port name: InhouseSend_PortTypeEJBPort
- Specify the binding name and where to create it:**
 - Source Folder: Maintenance_ProcessesEJB/ejbModule
 - Package: inhouse.send
 - File name: InhouseSend_PortTypeEJBBinding.wsdl
 - Binding name: InhouseSend_PortTypeEJBBinding
- Specify whether to generate a new or use an existing port for the EJB inbound binding:**
 - ☒ Create a new port and binding
 - ☐ Use an existing port
- Navigation buttons**: < Back, Next >, Finish, Cancel

Figure 30-24 Generate deploy code for the JMS service

5. Set the JNDI name for the EJB. Click **Finish**.

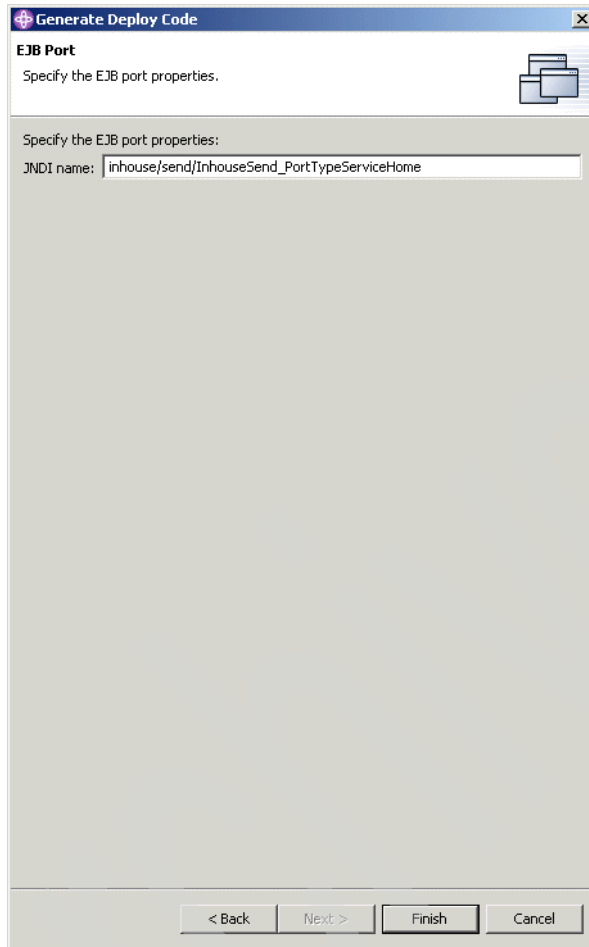


Figure 30-25 Generate deploy code for the JMS service

6. Review the Services pane in Studio. It should look similar to that shown in Figure 30-26 on page 658.

Note: The classes that are generated to manipulate XML documents and JMS messages. Note also that the deployable service project has an error currently, which will be corrected.

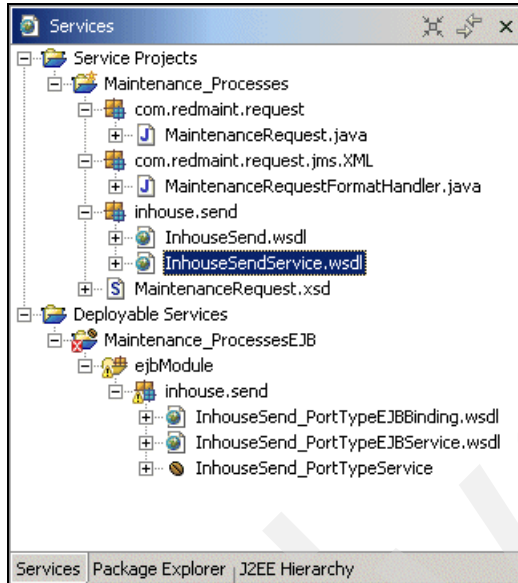


Figure 30-26 Contents of the service project after deployment code generation

7. Right-click the deployable service Maintenance_ProcessesEJB and select **Properties**.
8. Select Java Build Path in the left pane.
9. Select the Libraries tab and locate the error (Figure 30-27 on page 659). The path for the jar querymd.jar is wrong.

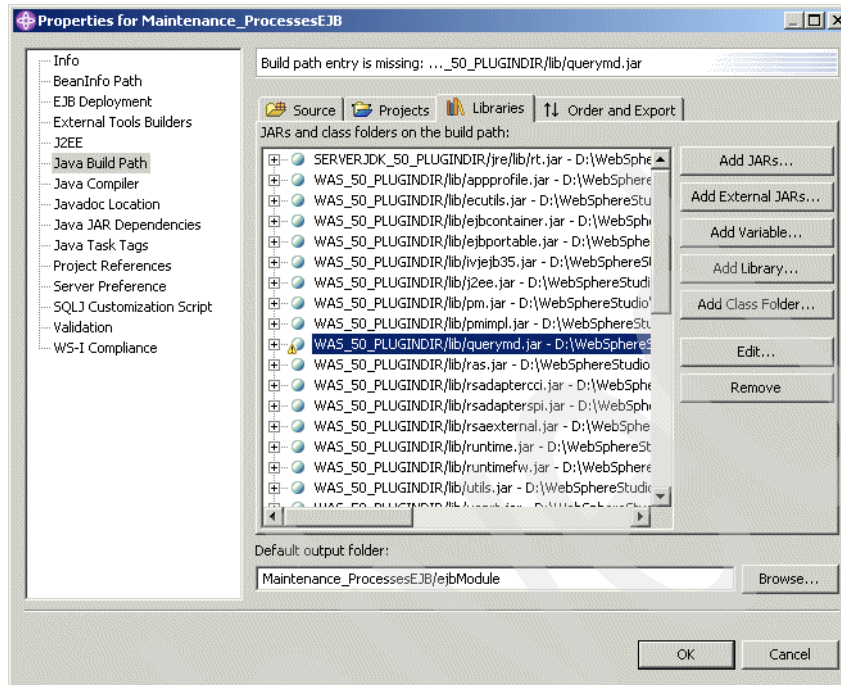


Figure 30-27 Review project properties

10. Click **Edit** and correct the entry, as shown in Figure 30-28.

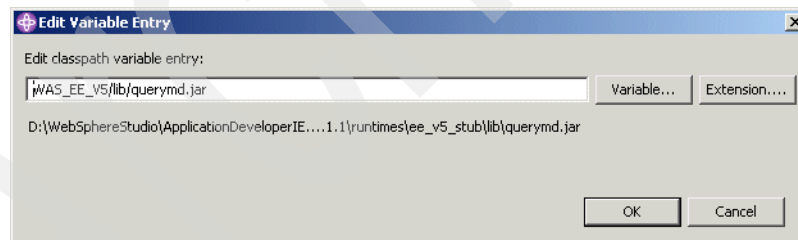


Figure 30-28 Correct the definition of the jar file

11. The error should be solved, as shown in Figure 30-29 on page 660. Expand the deployable service to locate the EJB.

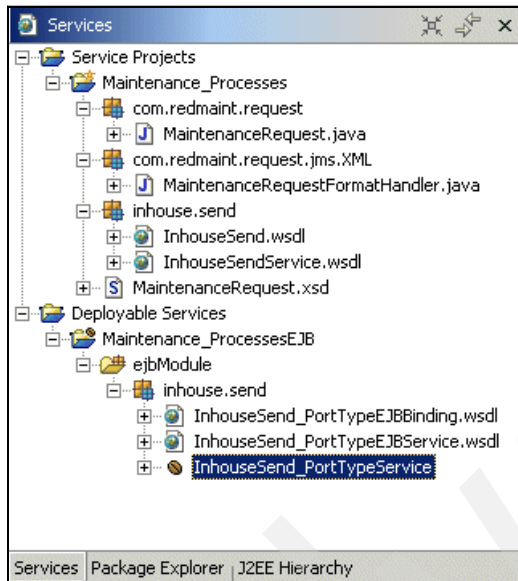


Figure 30-29 Contents of the deployable service project

12. Right-click the EJB and select **Open With** → **Deployment Descriptor Editor**.
13. Select the References tab and expand the bean in the left pane.
14. Select the resource environment reference. Change the setting for WebSphere Bindings as shown in Figure 30-30 on page 661. The value for WebSphere Bindings is the JNDI name of the JMS resource as defined in WebSphere Application Server.

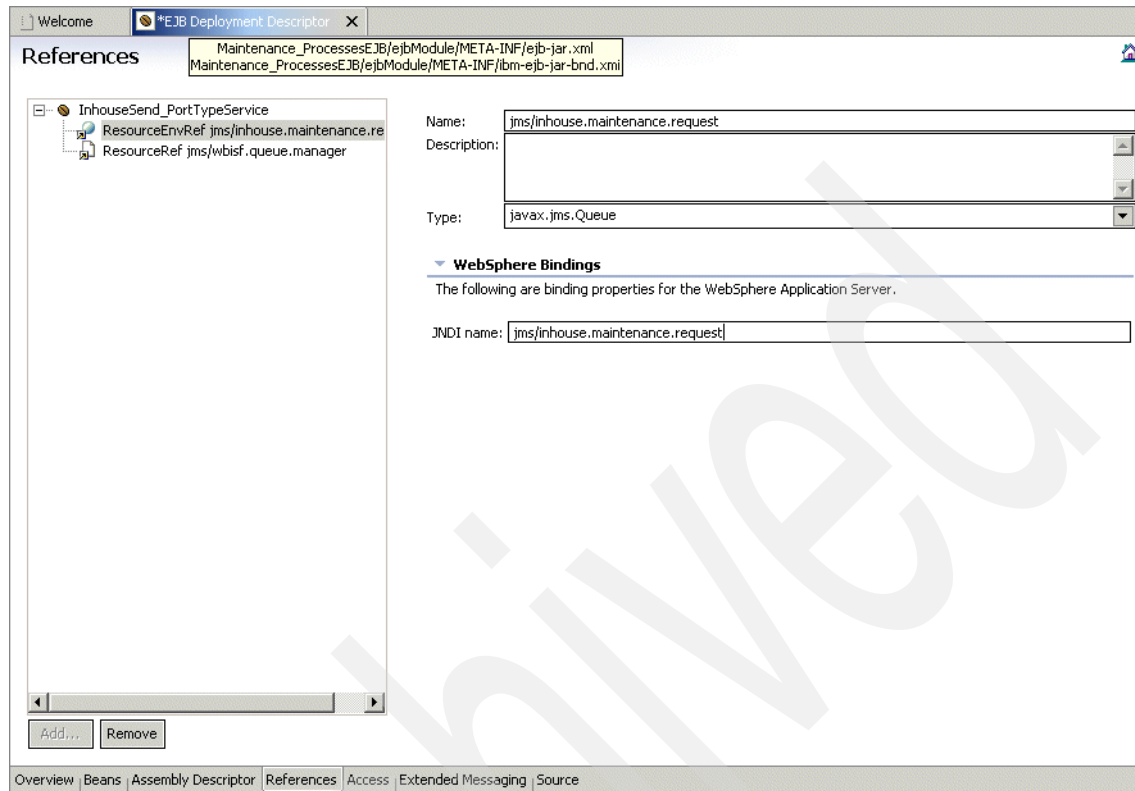


Figure 30-30 Change the resource environment reference

15. Select the resource reference. Change the setting for WebSphere Bindings as shown in Figure 30-31 on page 662. The value for WebSphere Bindings is the JNDI name of the JMS resource as defined in WebSphere Application Server.

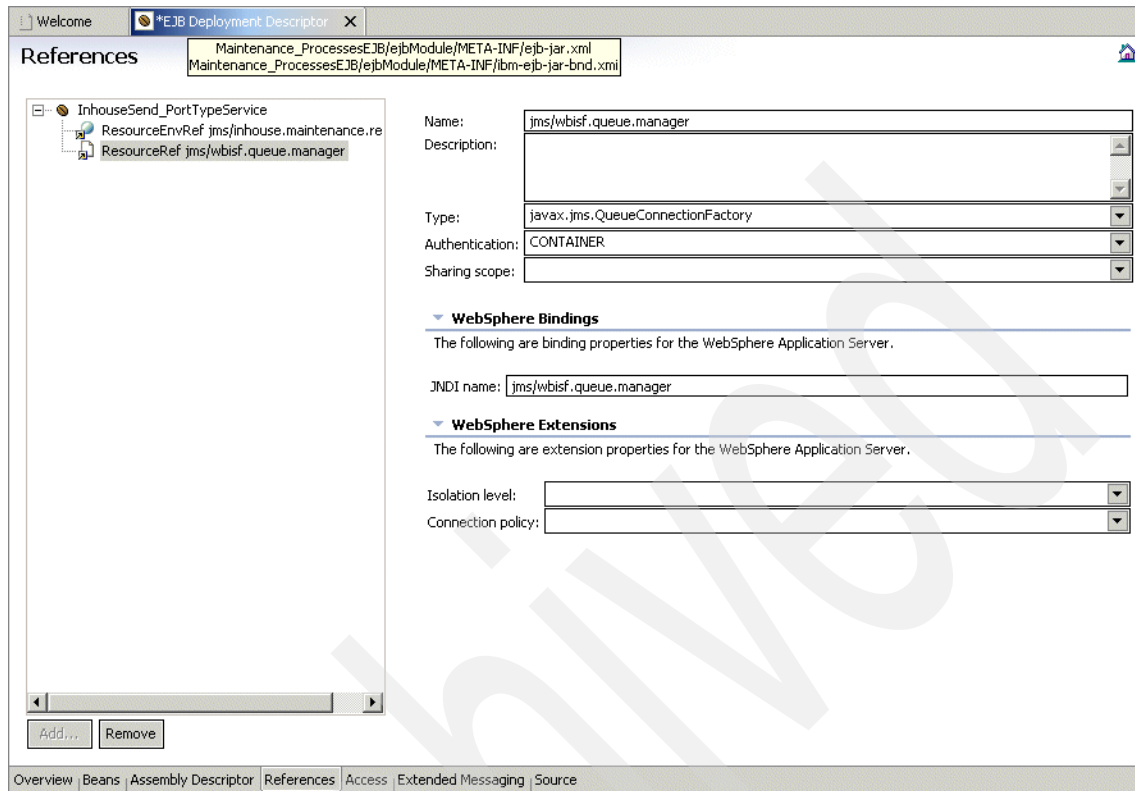


Figure 30-31 Change the resource reference

16. Save and close the EJB deployment descriptor.

30.1.5 Deploying and testing the new service

You can now deploy the service to the server. You need to configure a test server within WebSphere Studio Application Developer. (You can find details of how to do this in *WebSphere Business Integration for SAP*, SG24-6354.) To deploy and test the new service:

1. Switch to the Server perspective.
2. Right-click the server in the Server Configuration pane and select **Add and remove projects**.
3. Select the project Maintenance_ProcessesEAR and click **Add**.
4. Click **Finish**.

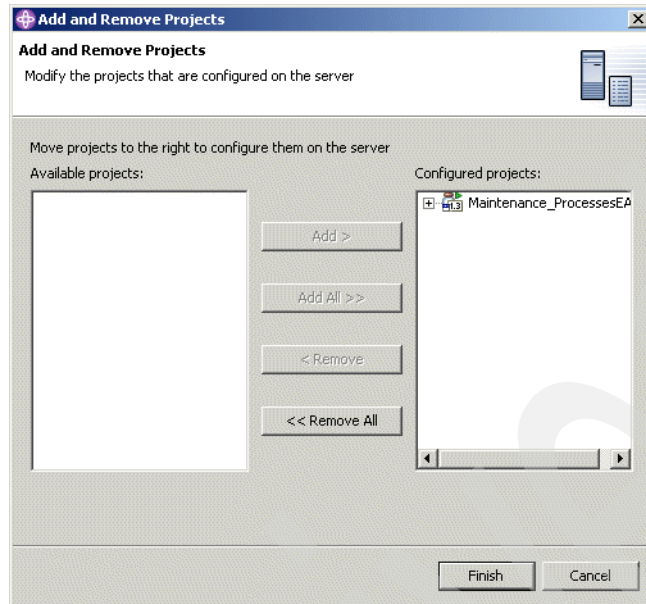


Figure 30-32 Add project to server

5. Start or restart the server.

The restart or start of the server might take a minute or two. To test the service, use the built-in Universal Test Client, which acts as a client of the EJB.

6. When the server is started, right-click it in the Servers pane and select **Run universal test client**.
7. When the client is started, select **JNDI Explorer**.

8. Expand the JNDI tree structure to locate the home interface of the bean, as shown in Figure 30-33.

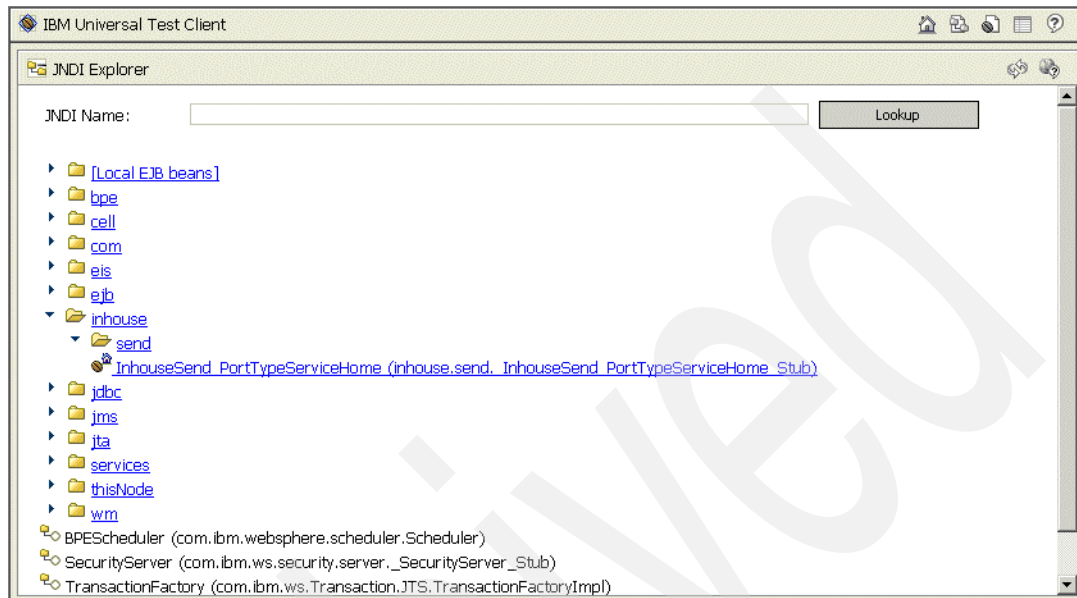


Figure 30-33 Locate the home interface of the bean

9. Expand the bean under the heading EJB References. Locate the create method and select it.
10. The signature of the create method is shown in the right pane. Click **Invoke**.

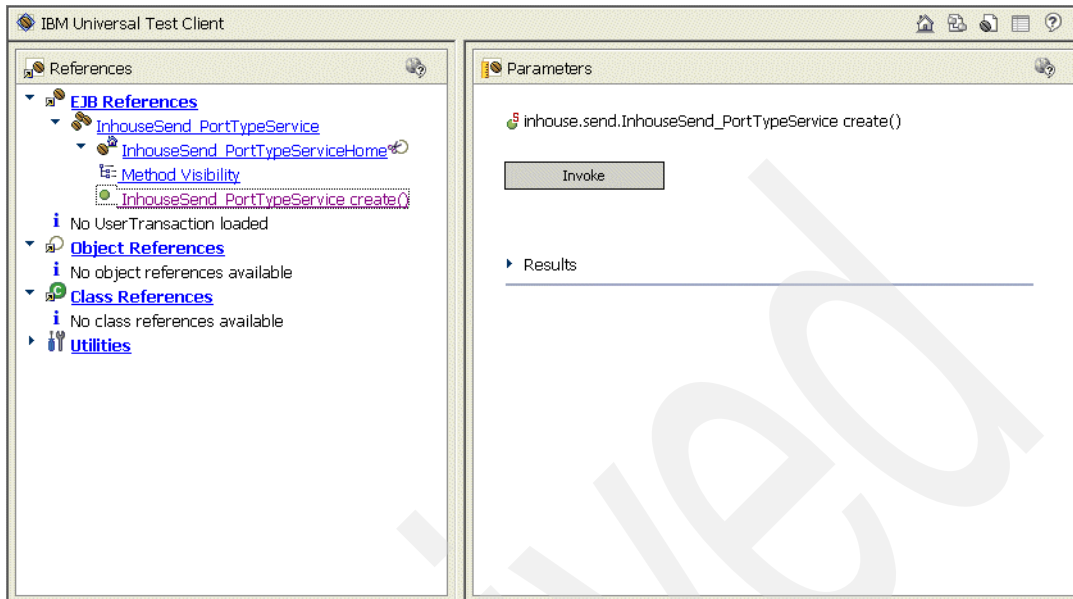


Figure 30-34 Invoke the create method

11. Click **Work with Object** to add the instance under the heading EJB References in the left pane.

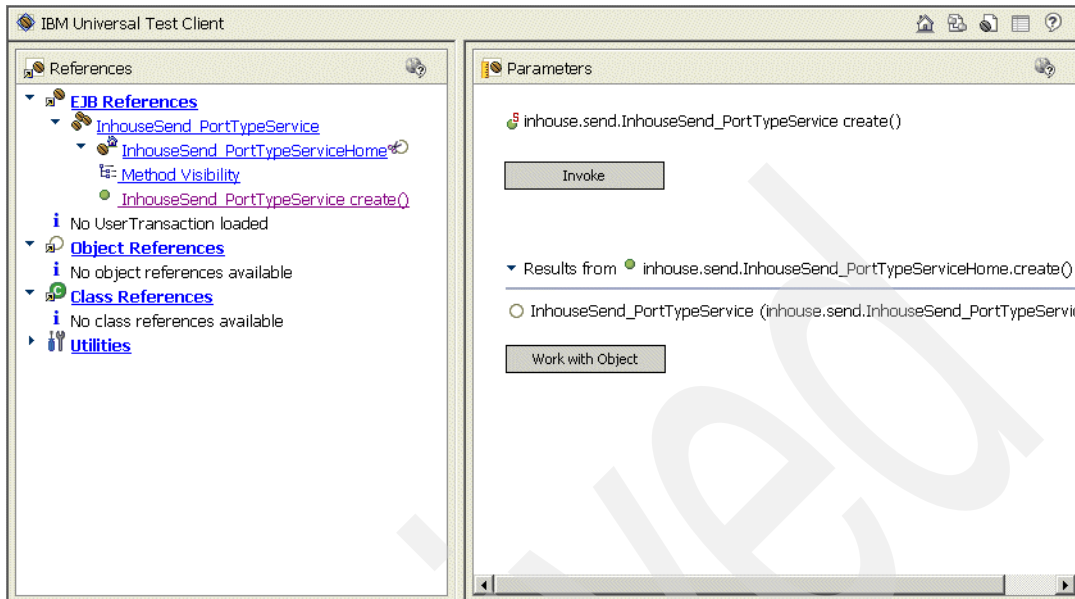


Figure 30-35 Instance created

12. Locate the send method of this new instance and click it. The signature of the send method is shown in the right pane. Expand the input parameter MaintenanceRequest.

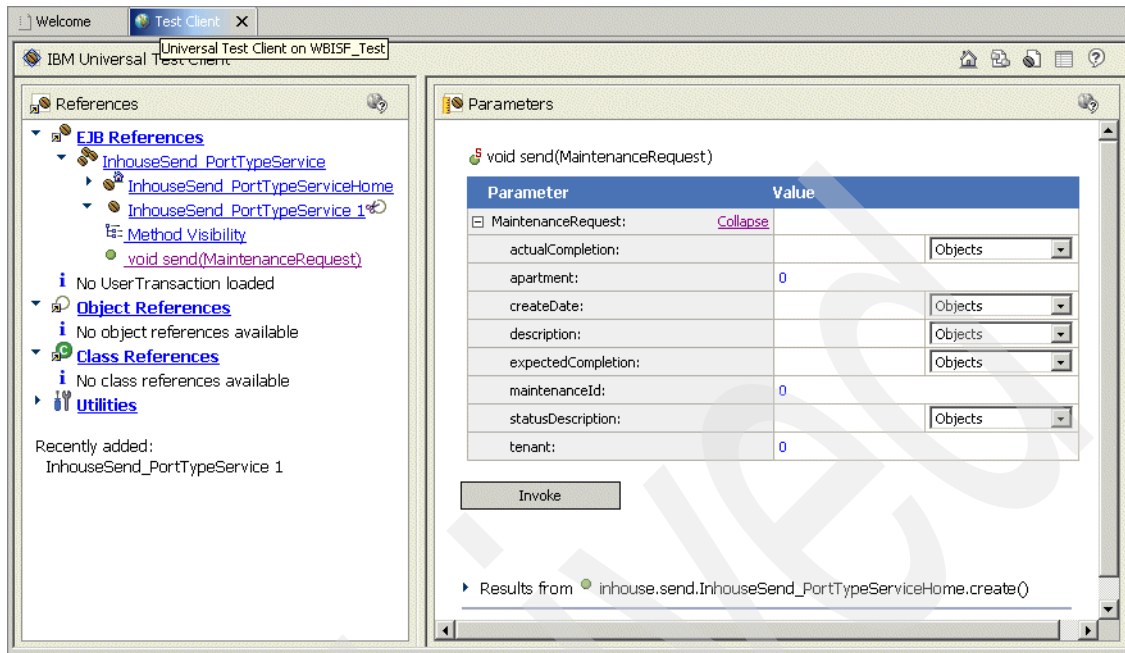


Figure 30-36 Using the send method of the EJB

13. Provide appropriate input values, as shown in Figure 30-37 on page 668, and click **Invoke**.

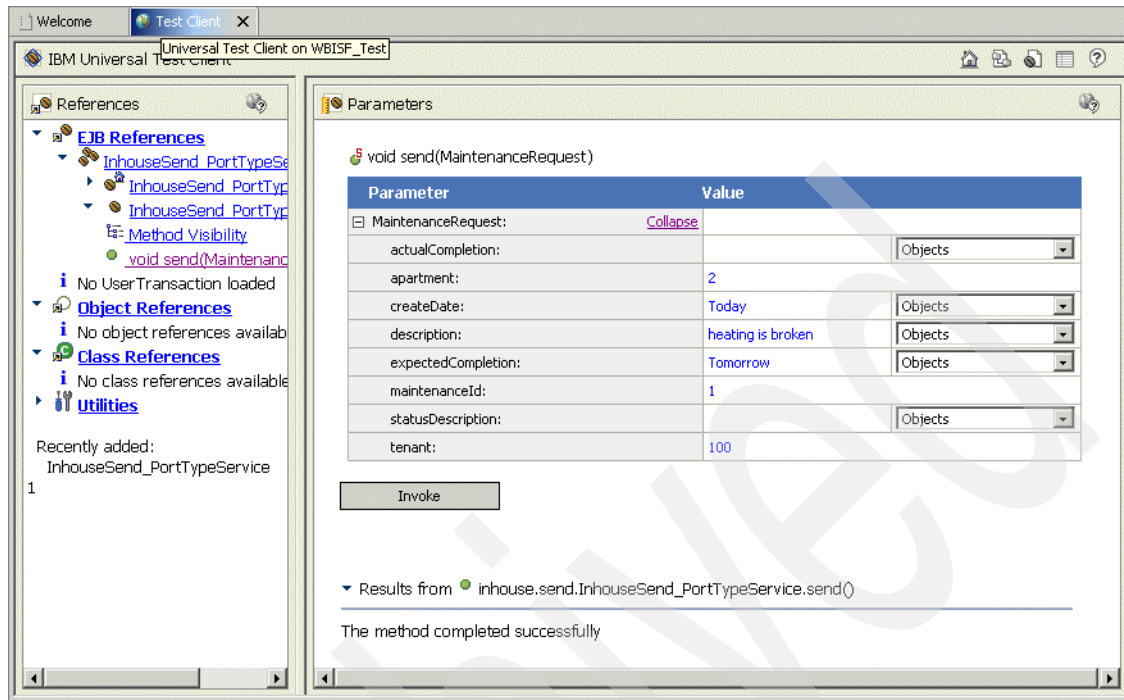


Figure 30-37 Provide input values to the send method

14. Use `rfhut11`, (located in the Additional Materials), to inspect the contents of the queue `inhouse.maintenance.request` on queue manager `wbisf.queue.manager`.

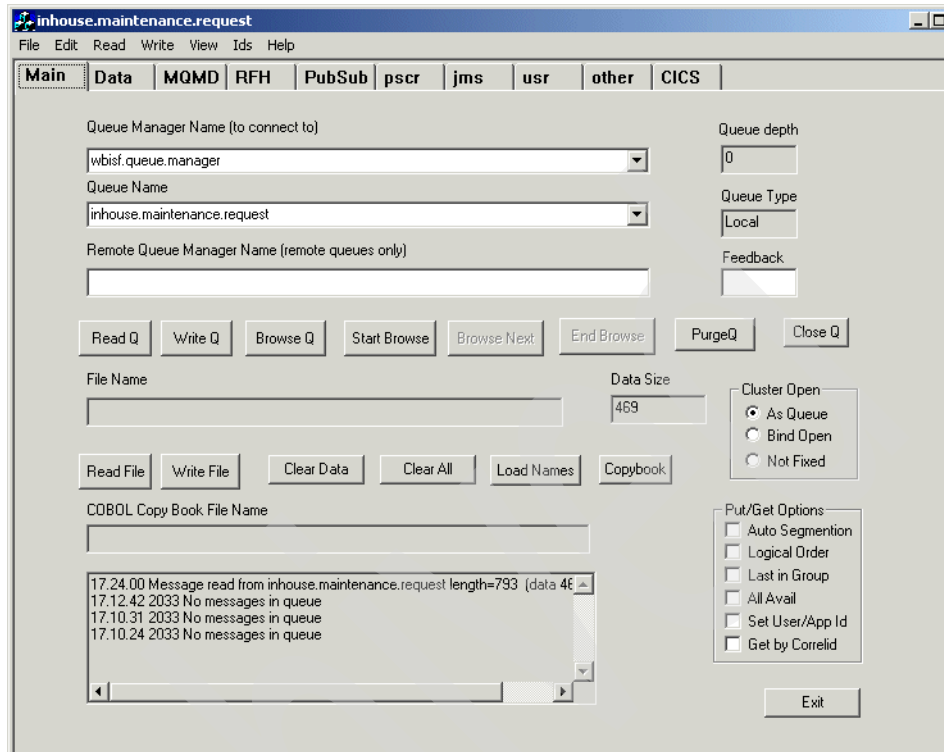


Figure 30-38 Use *rfhutil* to browse the message

Figure 30-39 shows the message data.

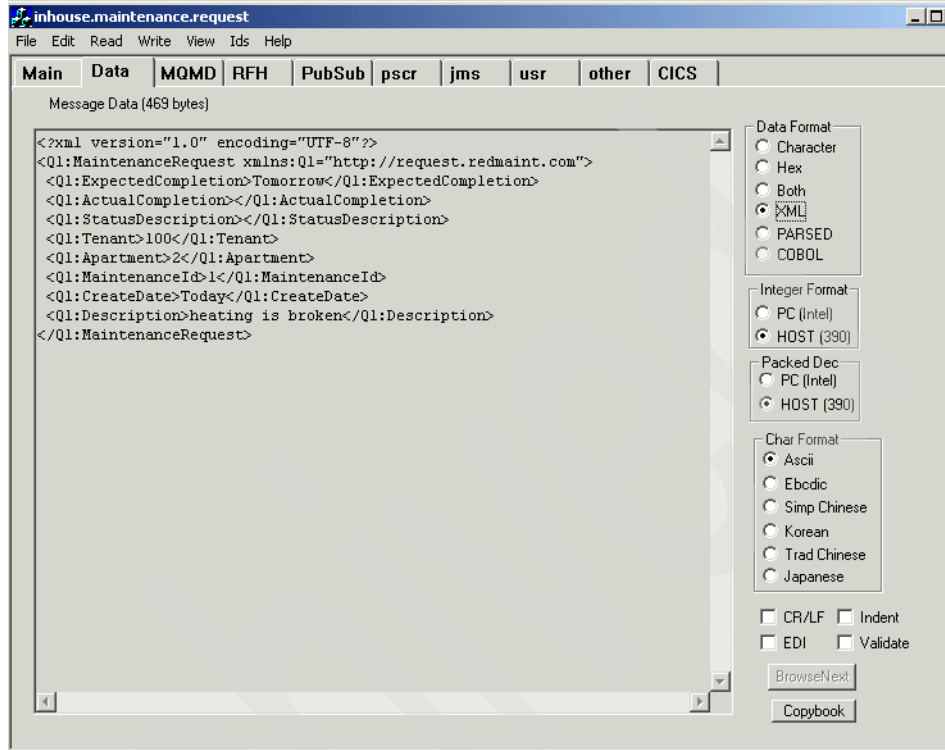


Figure 30-39 Message created by the InhouseSend service

30.2 Developing the adapter send update service

Similar to the InhouseSend service, you can now create a service for the RM2 connector that is configured on the Message Broker machine.

30.2.1 Creating artifacts and deploying adapter in System Manager

To create artifacts and deploy the adapter:

1. On the Message Broker machine, start System Manager.
2. Create a new ICL for the artifacts that you create for the WebSphere version of RMConnector. (We called ours RedMaintenanceForSF.)
3. Copy the RM_Maintenance business object from the RedMaintenance ICL to the new ICL.

4. Create a new Connector (RM2Connector) with the properties that are shown in the following figures. Remember to first change the Standard Property for Broker Type to WAS to ensure that the correct properties display.

Standard Properties		Connector-Specific Properties	Supported Business Objects	Trace/Log Files
	Property	Value		
1	AdminInQueue	REDCONTRACT.ADMININ		
2	AdminOutQueue	REDCONTRACT.ADMINOUT		
3	AgentTraceLevel	5		
4	ApplicationName	RM2Connector		
5	BrokerType	WAS		
6	CharacterEncoding	ascii7		
7	ContainerManagedEvents			
8	DeliveryQueue	REDCONTRACT.DELIVERY		
9	DeliveryTransport	JMS		
10	DuplicateEventElimination	false		
11	FaultQueue	REDCONTRACT.FAULT		
12	jms.FactoryClassName	CxCommon.Messaging.jms.JBMMQSeriesFactory		
13	jms.MessageBrokerName	REDBROKER		
14	jms.NumConcurrentRequests	10		
15	jms.Password	*****		
16	jms.UserName			
17	Locale	en_US		
18	MessageFileName	c:\IBM\WebSphereAdapters\connectors\vmessages\RMConnectorMessages		
19	PollEndTime	HH:MM		
20	PollFrequency	2000		
21	PollStartTime	HH:MM		
22	RepositoryDirectory	C:\IBM\WebSphereAdapters\repository		
23	RequestQueue	REDCONTRACT.REQUEST		
24	ResponseQueue	REDCONTRACT.RESPONSE		
25	RestartRetryCount	3		
26	RestartRetryInterval	1		
27	RFH2MessageDomain	mrm		
28	SynchronousRequestQueue	REDCONTRACT.SYNCHRONOUSREQUEST		
29	SynchronousRequestTimeout	0		
30	SynchronousResponseQueue	REDCONTRACT.SYNCHRONOUSRESPONSE		
31	WireFormat	CwXML		
32	WsrfSynchronousRequestTimeout	0		
33	XMLNamespaceFormat	long		

Figure 30-40 RM2Connector standard properties

Standard Properties				Connector-Specific Properties	Supported Business
	Property	Value	Encrypt		
1	UseDefaults	true	<input type="checkbox"/>		
2	MaxConnection	10	<input type="checkbox"/>		
3	CallbackBusObj		<input type="checkbox"/>		
4	CallbackCollaboration		<input type="checkbox"/>		
5	CallbackVerb		<input type="checkbox"/>		
6	ApplicationHost	student	<input type="checkbox"/>		
7	CallbackPrimary	false	<input type="checkbox"/>		
8	ApplicationRegisteredName	rm	<input type="checkbox"/>		

Figure 30-41 RM2Connector connector-specific properties

Standard Properties		Connector-Spec
	Business Object Name	
1	RM_Maintenance	
2		

Figure 30-42 RM2Connector supported business objects

Note that the Supported Business Objects contains only the business objects (for a WebSphere connector there is no concept of a Message Set ID or a map to accompany the business object).

5. Create a User Project of the type WAS Project (we named ours ForSF).
6. Add the business object and the connector from the new ICL to this project.

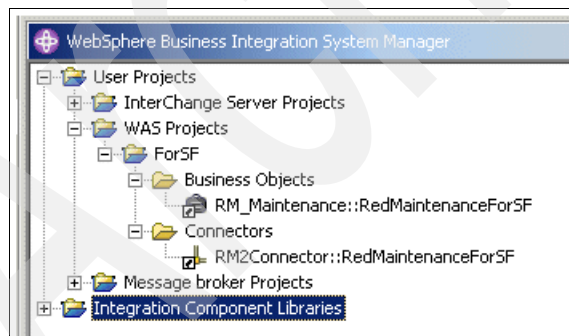


Figure 30-43 New User Project

7. Right-click this project and select **Deploy WAS Project**.
8. Verify that connector and business object are selected and click **Next**.

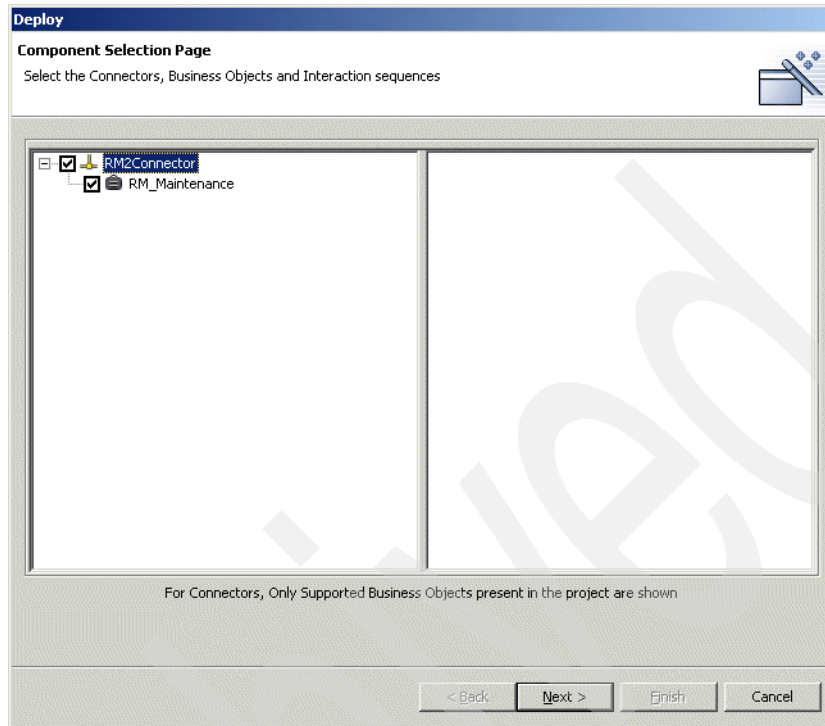


Figure 30-44 Select objects to deploy

9. Select the option to export to a directory and click **Browse**.
10. Point to a directory (we chose C:\Additional Materials\RM2) and click **Finish**.

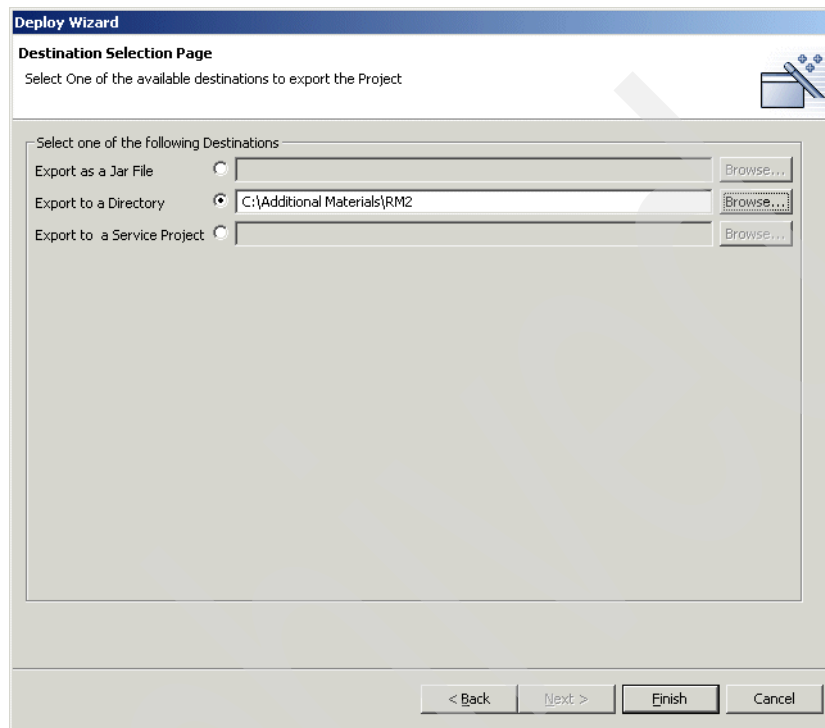


Figure 30-45 Deploy

30.2.2 Importing the deployed connector in a service project

To import the deployed connector:

1. Create a new package (for example, adapter.RM2Connector), as shown in Figure 30-46.

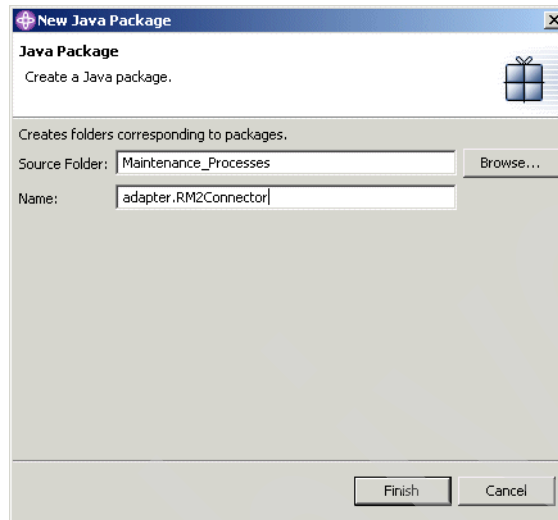


Figure 30-46 Create new package

2. On the Server Foundation machine, using Explorer, map the C: drive of the Message Broker system (\\student\C\$).
3. Locate the WSDL and schema files in the folder \\Additional Materials\RM2 on the network drive.
4. Import the WSDL files and schema file in the existing service project Maintenance_Processes and in the package adapter.RM2Connector.

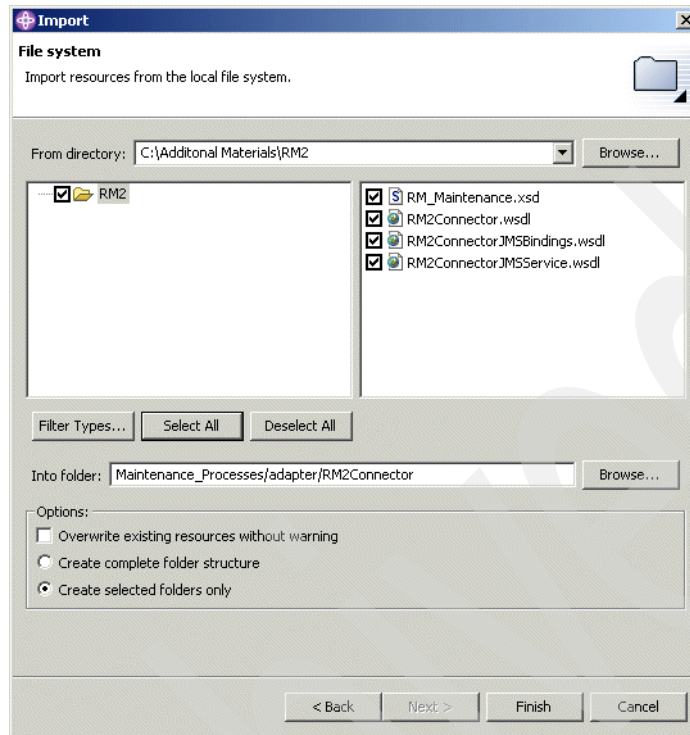


Figure 30-47 Import connector resources in service project

For this specific scenario, you need to make a slight change to the WSDL. When you want to invoke the adapter directly, no change is required. However, in our scenario, we want to add monitoring to the application. Therefore, we need ensure that the message broker intercepts the call to the adapter. The message broker can then generate monitoring data that can be used in WebSphere Business Integration Monitor. (We implement this monitoring logic in Part 8, “Looking after the run-time environment” on page 823.)

To meet the message requirements of the message broker, the message must have a correct RFH, including a message set, format, and type. However, in the WSDL, there is no information about message sets. To add message set information to the WSDL:

1. Open the WSDL `RM2ConnectorJMSBindings.wsdl` and select the source view.

The original `JMSType` information is shown in Figure 30-48 on page 677. Note that there is no message set information. However, there is a message domain, message type, and format. The message set should be added between the two slash (/) characters.

```
<jms:propertyValue name="JMSType" type="xsd:string"
value="mcd://mrm//RM_Maintenance?format=CwXML"/>
```

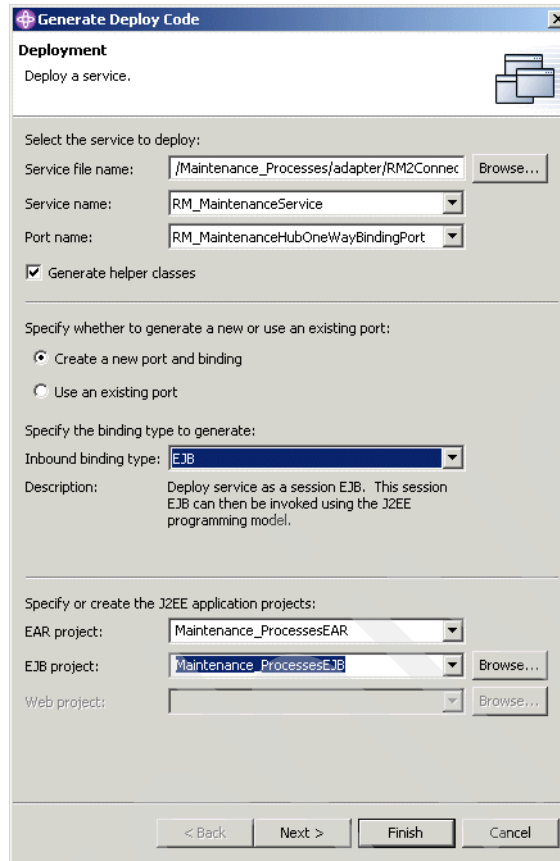
Figure 30-48 Generated message properties in the WSDL

2. Perform a find for the text `mrm//` and replace it with `mrm/RedTenant/`. Perform a change-all. Example 30-1 shows the correct JMSType information.

Example 30-1 Corrected message properties in the WSDL

```
<jms:propertyValue name="JMSType" type="xsd:string"
value="mcd://mrm/RedTenant/RM_Maintenance?format=CwXML"/>
```

3. Right-click the WSDL file called `RM2ConnectorJMSService.wsdl` and select **Enterprise Services** → **Generate Deploy Code**.
4. Select the port for Hub One Way, which means the port that is used by the broker (WebSphere Business Integration Server Foundation) sends a message to the adapter without waiting for a response.
5. Set the inbound protocol to EJB and click **Finish** so that you can invoke the adapter as though it was an EJB.



The image shows a 'Generate Deploy Code' dialog box with a 'Deployment' tab. The 'Service file name' is '/Maintenance_Processes/adapter/RM2Connec'. The 'Service name' is 'RM_MaintenanceService'. The 'Port name' is 'RM_MaintenanceHubOneWayBindingPort'. The 'Generate helper classes' checkbox is checked. The 'Specify whether to generate a new or use an existing port' section has 'Create a new port and binding' selected. The 'Specify the binding type to generate' section has 'Inbound binding type' set to 'EJB'. The 'Description' is 'Deploy service as a session EJB. This session EJB can then be invoked using the J2EE programming model.' The 'Specify or create the J2EE application projects' section has 'EAR project' set to 'Maintenance_ProcessesEAR', 'EJB project' set to 'Maintenance_ProcessesEJB', and 'Web project' is empty. The 'Finish' button is highlighted.

Generate Deploy Code

Deployment
Deploy a service.

Select the service to deploy:

Service file name: /Maintenance_Processes/adapter/RM2Connec **Browse...**

Service name: RM_MaintenanceService

Port name: RM_MaintenanceHubOneWayBindingPort

☒ Generate helper classes

Specify whether to generate a new or use an existing port:

☒ Create a new port and binding

☐ Use an existing port

Specify the binding type to generate:

Inbound binding type: EJB

Description: Deploy service as a session EJB. This session EJB can then be invoked using the J2EE programming model.

Specify or create the J2EE application projects:

EAR project: Maintenance_ProcessesEAR

EJB project: Maintenance_ProcessesEJB **Browse...**

Web project: **Browse...**

< Back Next > Finish Cancel

Figure 30-49 Generate deploy code for the adapter

6. Open the Deployment Descriptor for the EJB and correct the resources as shown in Figure 30-50 and Figure 30-51 on page 680.

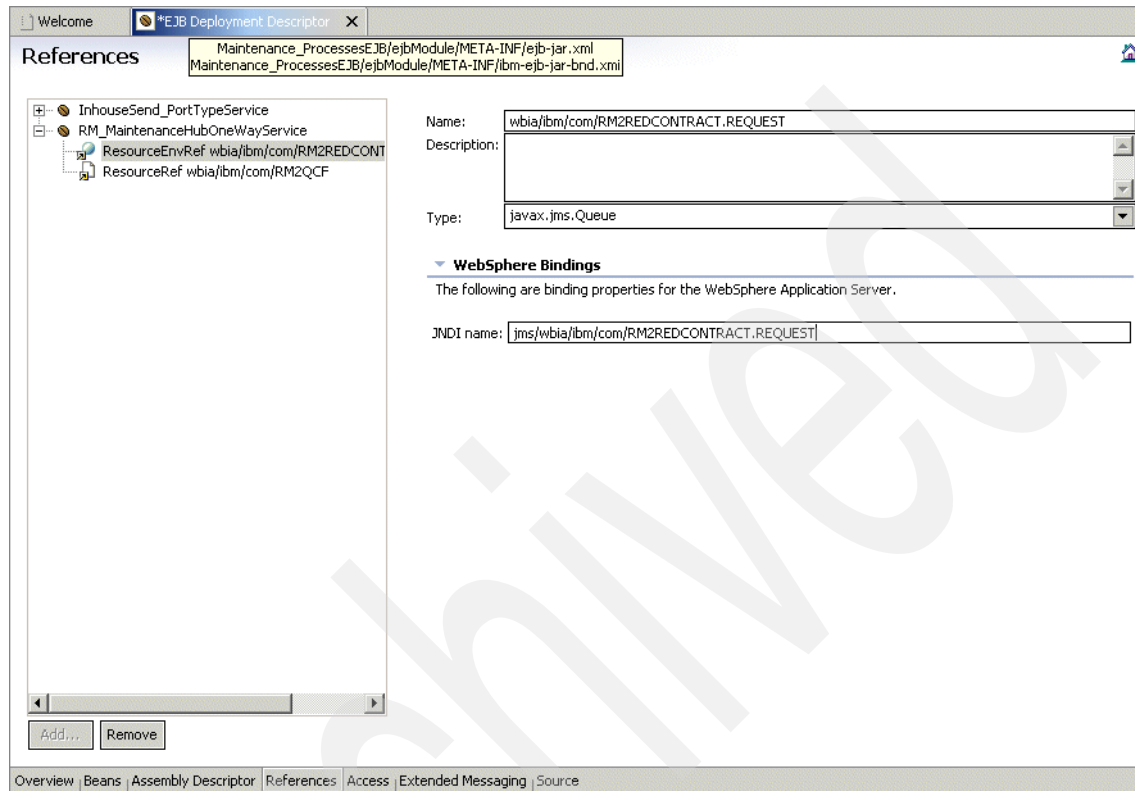


Figure 30-50 Update resource environment reference

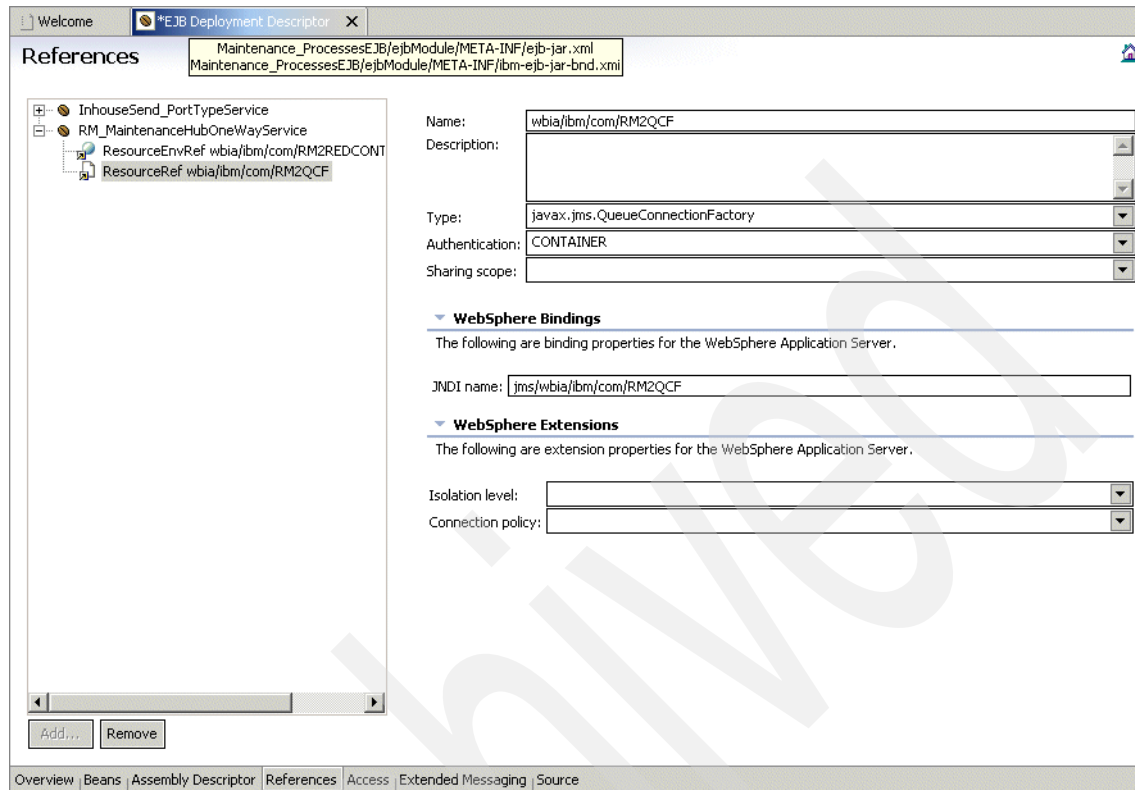


Figure 30-51 Update resource reference

7. Save and close the deployment descriptor.

30.2.3 Defining resources in the test server

The EJB for the adapter invocation uses two resources that need to be defined to WebSphere. The server will likely be in a state of *need to be restarted*, but you can ignore that for now. To define resources:

1. To avoid a possible problem during the creation of the resources, remove the project from the server. Right-click the server and select **Add and remove projects**.
2. In the section Configured projects, select the Maintenance_ProcessesEAR project and click **Remove**.
3. Restart the server.
4. Right-click the server WBISF_Test and select **Run administrative console**.

5. Alternatively, we can open Internet Explorer and go to <http://localhost:9090/admin>.
6. When the log-on prompt appears, provide db2admin as the user ID and its04you as the password.

Note: When we installed our application server, we enabled security, and its04you is our db2admin password.

7. In the left pane, select **Resources** → **WebSphere MQ JMS Provider**.
8. In the right pane, scroll down and click **WebSphere MQ Queue Connection Factories**. A list of existing queue connection factories is shown (Figure 30-52).

The screenshot displays the WebSphere Administrative Console interface. The top navigation bar includes links for Home, Save, Preferences, Logout, and Help. The left-hand navigation pane shows a tree structure under 'localhost', with 'Resources' expanded to show various providers, including 'WebSphere MQ JMS Provider'. The main content area is titled 'WebSphere MQ Queue Connection Factories' and contains a table listing existing factories. The table has columns for Name, JNDI Name, Description, and Category. A single entry is visible: 'wblsf.queue.manager' with JNDI Name 'jms/wblsf.queue.manager'. Below the table, there is a 'WebSphere Status' section and a 'WebSphere Configuration Problems' section, both showing zero issues.

Name	JNDI Name	Description	Category
wblsf.queue.manager	jms/wblsf.queue.manager		

Figure 30-52 List of queue connection factories

9. Click **New**.
10. Set the name of the queue connection factory to REDBROKER.
11. Set the JNDI name to the value used in Figure 30-51 on page 680.

The screenshot shows the WebSphere Administrative Console interface. The top navigation bar includes 'Home', 'Save', 'Preferences', 'Logout', and 'Help'. The left sidebar shows a tree view with 'localhost' expanded, containing 'Servers', 'Applications', and 'Resources'. Under 'Resources', various providers are listed, including 'JDBC Providers', 'Generic JMS Providers', 'WebSphere JMS Provider', 'WebSphere MQ JMS Provider', 'Mail Providers', 'Resource Environment Provider', 'URL Providers', 'Resource Adapters', 'Cache Instances', 'Extended Messaging Provider', 'Object Pools', 'Scheduler Configuration', 'Staff Plugin Provider', 'Work Manager', 'WebSphere Business Integration', and 'Common Event Infrastructure Provider'. The main content area is titled 'Configuration' and shows the 'General Properties' for a new resource. The properties are as follows:

Scope	* cells:localhost:nodes:localhost	<i>i</i> The scope of the resource. This value the configuration local configuration file.
Name	* REDBROKER	<i>i</i> The required display name for the resource.
JNDI Name	* jms/wbia/bm/com/RM2QCF	<i>i</i> The JNDI name for the resource.
Description		<i>i</i> An optional description of the resource.
Category		<i>i</i> An optional category which can be used to group the resource.
Component-managed Authentication Alias	(none)	<i>i</i> References authentication data for component-signon to the resource.
Container-managed Authentication Alias	(none)	<i>i</i> References authentication data for container-managed signon to the resource.
Mapping-Configuration Alias	(none)	<i>i</i> Select a suitable configuration from the J2EE configuration.

At the bottom, the 'WebSphere Status' section shows 'WebSphere Runtime Messages' with a summary: 'Total All Messages: 121', '3 new, 3 total', '0 new, 0 total', and '118 new, 118 total'. There is a 'Clear All' button next to the summary.

Figure 30-53 Create queue connection factory

12. Scroll down and set the MQ connection attributes, as shown in Figure 30-54 on page 683.

Note: We use an MQ client connection to talk to the broker queue manager. The broker queue manager intercepts the message to add monitoring, as we see in Part 8, “Looking after the run-time environment” on page 823. Normally, this is not needed.

13. Click **OK** to save the new resource.

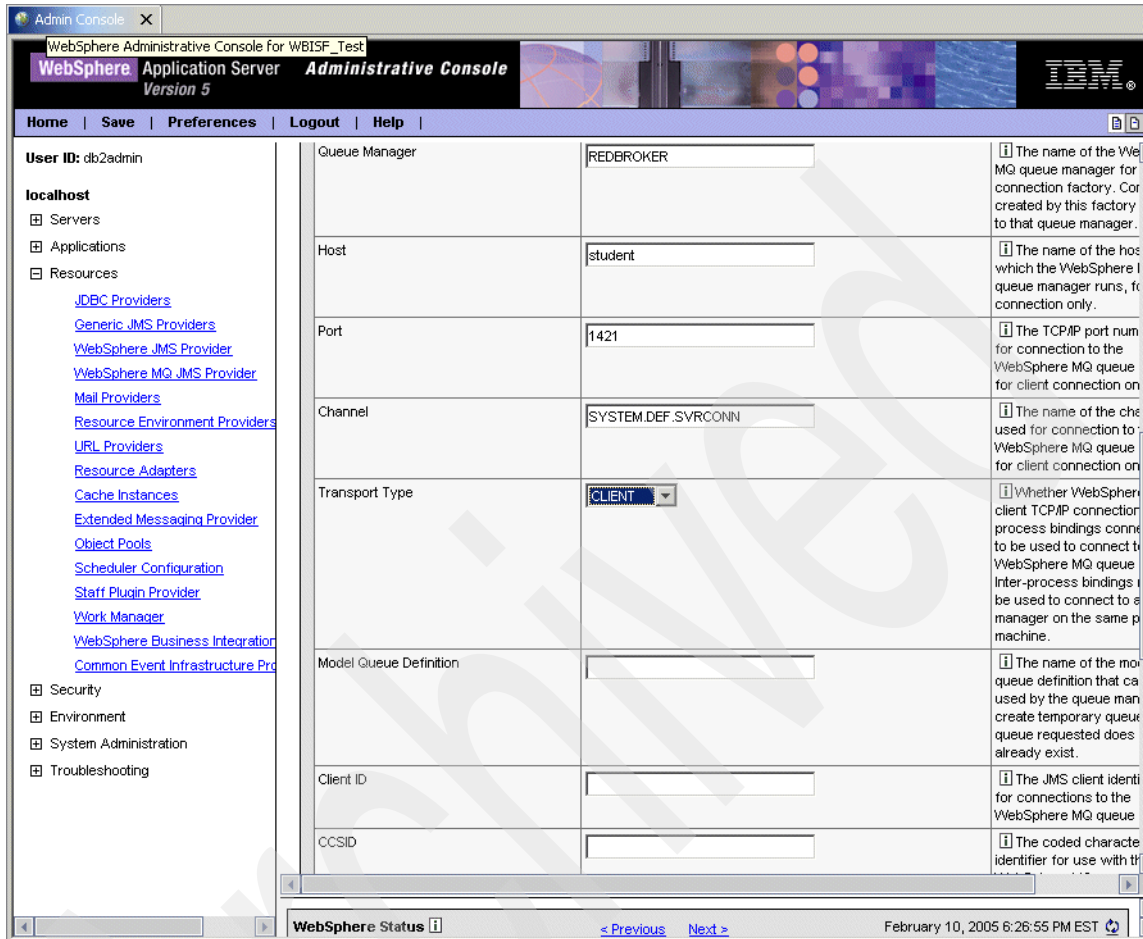


Figure 30-54 Create queue connection factory

Figure 30-55 on page 684 shows the updated list of queue connection factories.

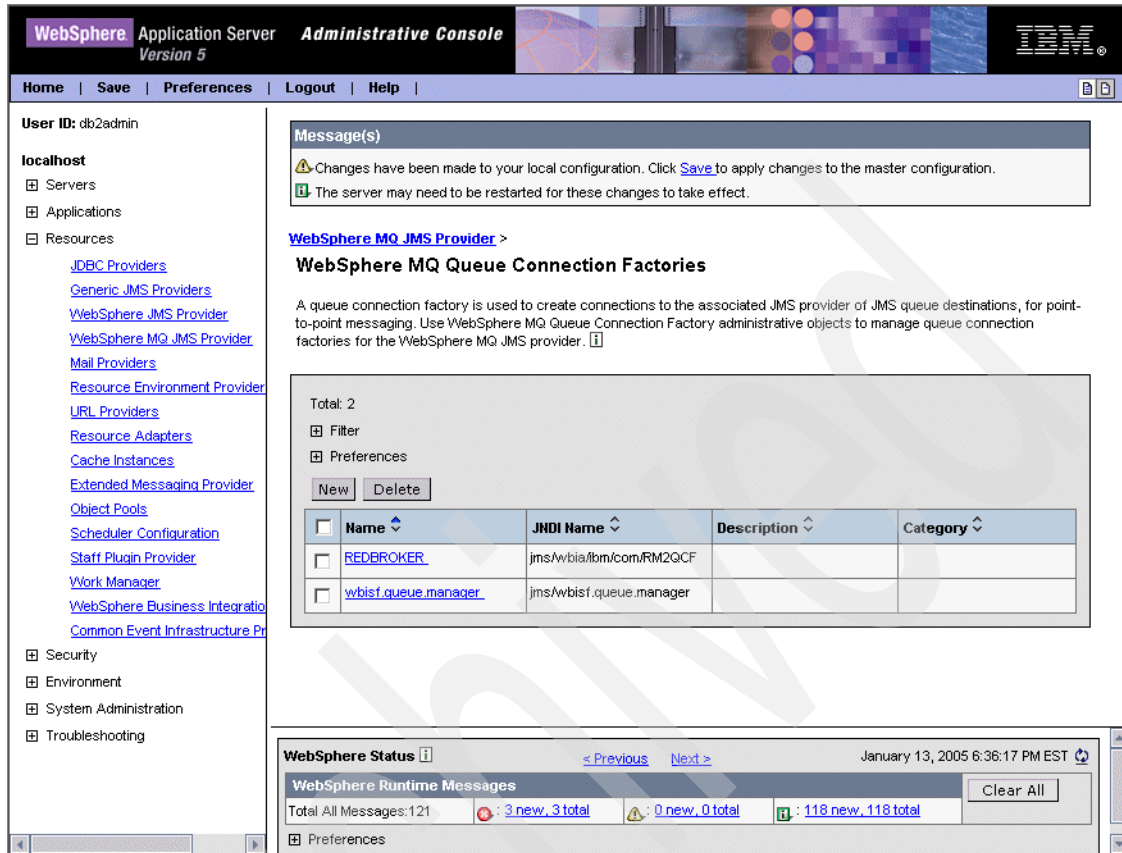


Figure 30-55 List of queue connection factories

- In the left pane, select **Resources** → **WebSphere MQ JMS Provider**.
- In the right pane, scroll down and click **WebSphere MQ Queue Destinations**. A list of existing queue destinations is shown (Figure 30-56 on page 685).
- Click **New**.

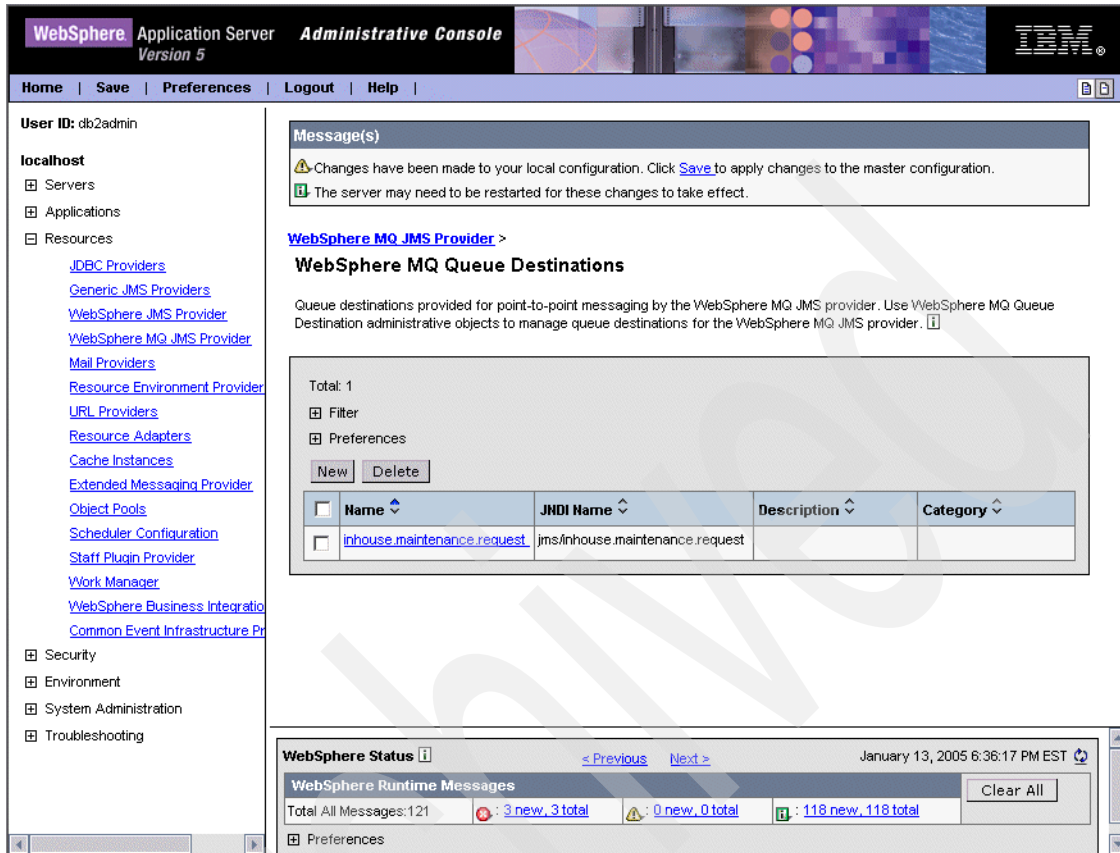


Figure 30-56 List of queue destinations

17. Set the name of the destination to RM2REDCONTRACT.REQUEST.
 18. Set the JNDI name to the value that is used in Figure 30-50 on page 679.
 19. Set the Expiry to UNLIMITED.
 20. Scroll down and set the attribute Base Queue Name to REDCONTRACT.EMITTER.
- Again, we use this queue for monitoring purposes. Normally, you can use the adapter request queue directly.
21. Click **OK**.

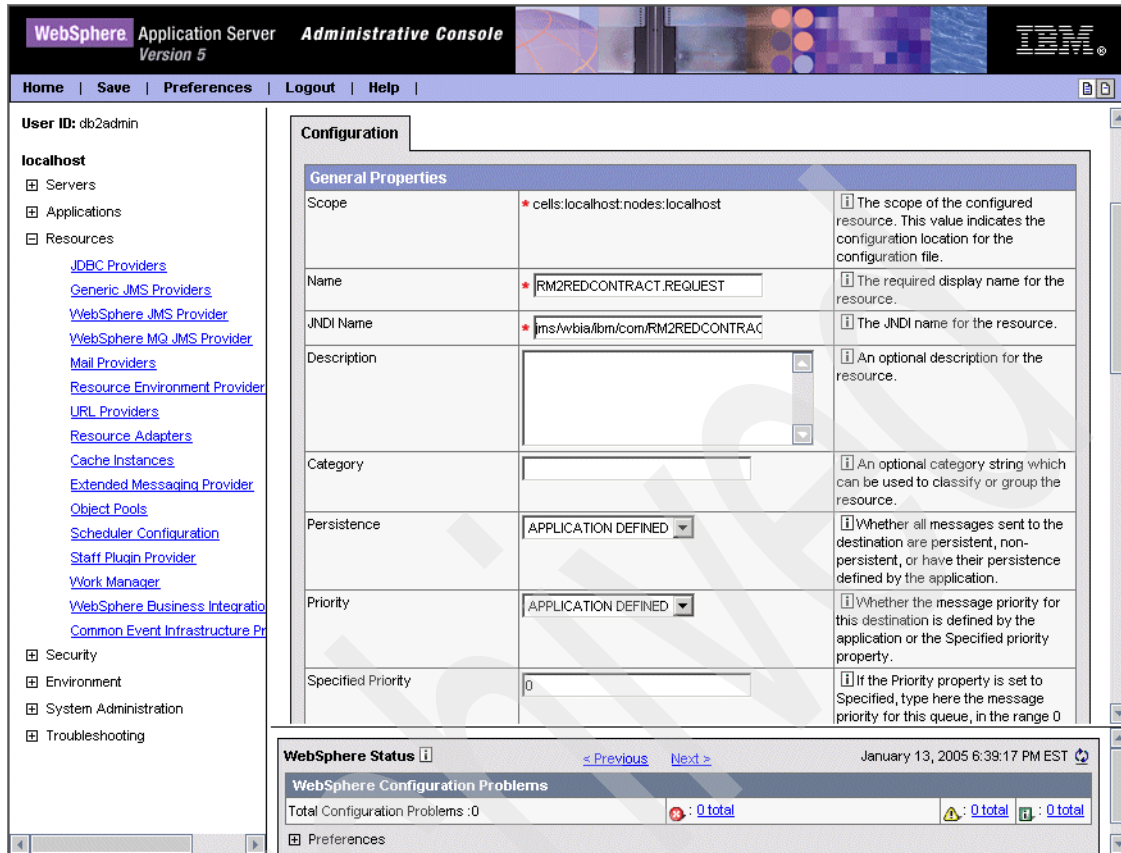


Figure 30-57 Define new queue destination

22. Figure 30-58 on page 687 shows the updated list of queue destinations. At the top of Figure 30-58 on page 687, we see a link to save the new resources to the server. Click the Save link.

WebSphere Application Server **Administrative Console** Version 5

Home | Save | Preferences | Logout | Help

User ID: db2admin

localhost

- Servers
- Applications
- Resources
 - JDBC Providers
 - Generic JMS Providers
 - WebSphere JMS Provider
 - WebSphere MQ JMS Provider
 - Mail Providers
 - Resource Environment Provider
 - URL Providers
 - Resource Adapters
 - Cache Instances
 - Extended Messaging Provider
 - Object Pools
 - Scheduler Configuration
 - Staff Plugin Provider
 - Work Manager
 - WebSphere Business Integration
 - Common Event Infrastructure Provider
- Security
- Environment
- System Administration
- Troubleshooting

Message(s)

Changes have been made to your local configuration. Click **Save** to apply changes to the master configuration.

The server may need to be restarted for these changes to take effect.

WebSphere MQ JMS Provider >

WebSphere MQ Queue Destinations

Queue destinations provided for point-to-point messaging by the WebSphere MQ JMS provider. Use WebSphere MQ Queue Destination administrative objects to manage queue destinations for the WebSphere MQ JMS provider.

Total: 2

Filter

Preferences

New Delete

<input type="checkbox"/>	Name	JNDI Name	Description	Category
<input type="checkbox"/>	RM2REDCONTRACT.REQUEST	jms/wbia/ibm/com/RM2REDCONTRACT.REQUEST		
<input type="checkbox"/>	inhouse.maintenance.request	jms/inhouse.maintenance.request		

WebSphere Status January 13, 2005 6:39:17 PM EST

WebSphere Configuration Problems

Total Configuration Problems: 0

0 total

0 total

0 total

Preferences

Figure 30-58 List of queue destinations

23. Click **Save** again (Figure 30-59 on page 688).

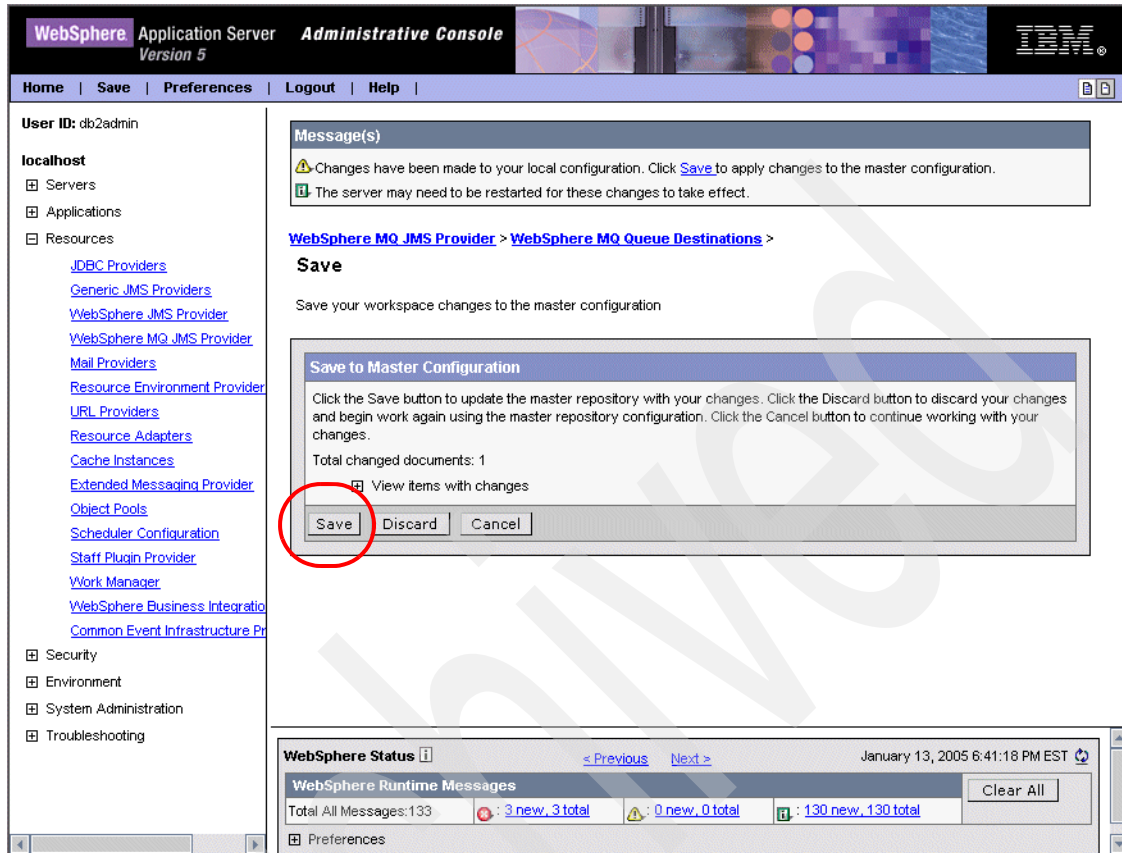


Figure 30-59 Save new resources to the server

24. Add the project to the server again.

25. Restart the server so that these new resources and the extended Maintenance_Processes project are picked up by the server.

26. During the restart of the server, review the information in the console to verify that the new resources are found and bound (Example 30-2).

Example 30-2 Binding JMS resources during server start

```
ResourceMgrIm I WSVR0049I: Binding wbisf.queue.manager as  
jms/wbisf.queue.manager  
ResourceMgrIm I WSVR0049I: Binding inhouse.maintenance.request as  
jms/inhouse.maintenance.request  
ResourceMgrIm I WSVR0049I: Binding REDBROKER as jms/wbia/ibm/com/RM2QCF  
ResourceMgrIm I WSVR0049I: Binding RM2REDCONTRACT.REQUEST as  
jms/wbia/ibm/com/RM2REDCONTRACT.REQUEST
```

30.2.4 Testing the adapter service using the test client

In a very similar way to the InhouseSend service, you can use the test client to test the adapter service. However, make sure that the following components are running before you begin:

- ▶ RMConnector is running
- ▶ The broker is running
- ▶ The RedMaintenance application is running

To test the adapter service:

1. Right-click the EJB and select **Run On Server**.
2. As before, create a new instance of the EJB (Figure 30-60 on page 690).

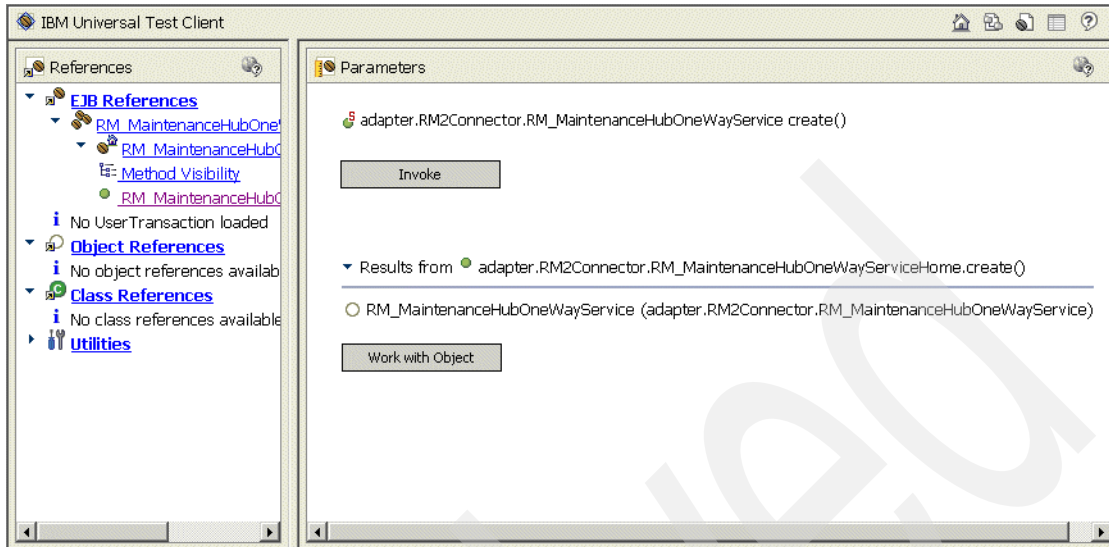


Figure 30-60 Create instance of adapter EJB

3. Locate the Update method of the new instance.
4. Provide appropriate parameters, as shown in Figure 30-61 on page 691.

Note: Dates are expressed in milliseconds since epoch (that is, since January 1, 1970). The adapter requires this specific format. See the value for expectedCompletion in Figure 30-61.

5. The value CxIgnore is a key word that instructs the adapter to ignore the value of this attribute.
6. Use P for the field status to indicate that this request is in progress.

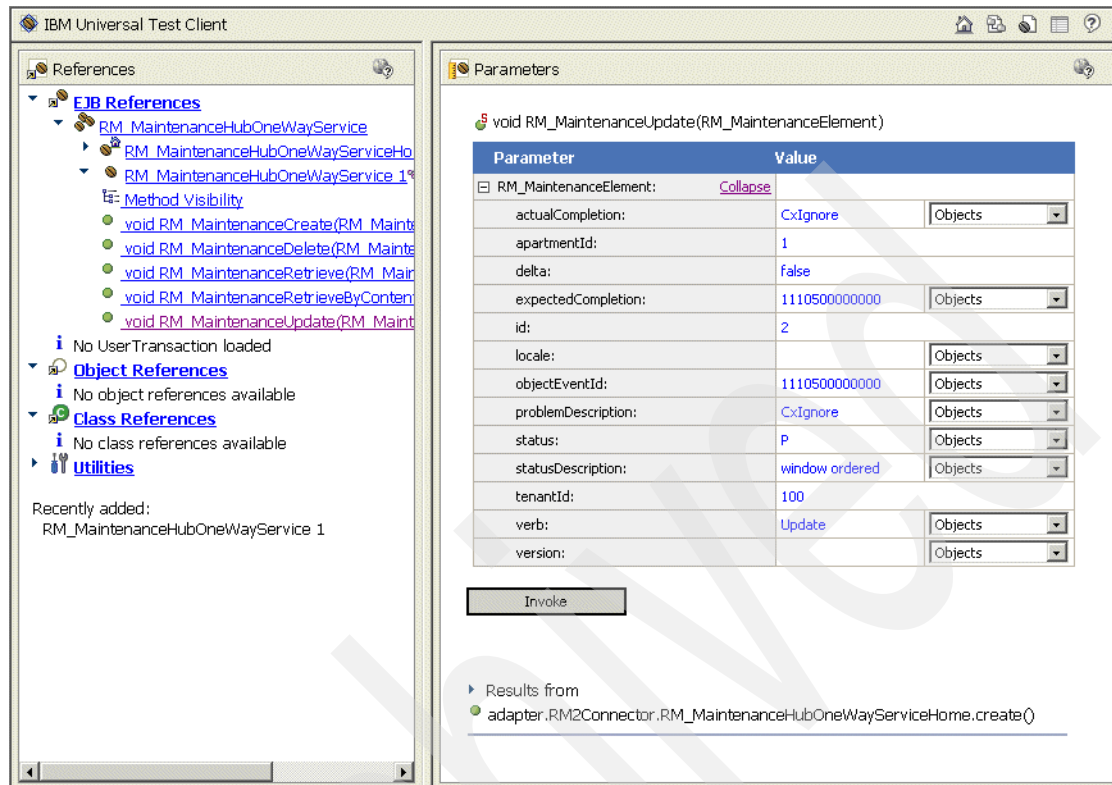


Figure 30-61 Provide parameters to the update method of the adapter

Figure 30-62 on page 692 shows the message that is generated by Server Foundation to invoke the adapter's Update function.

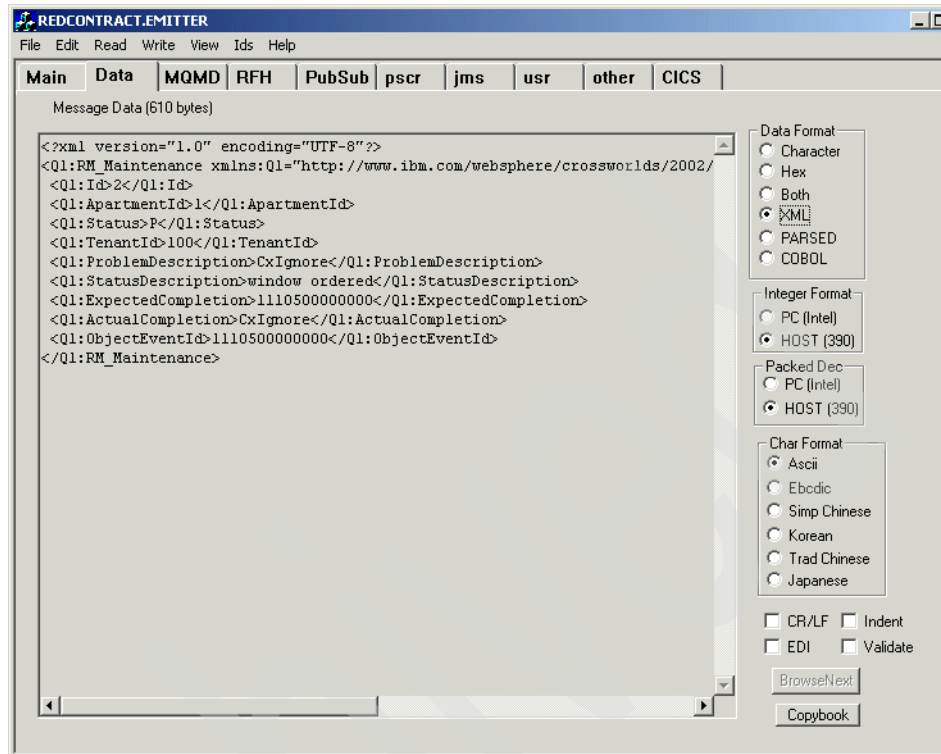


Figure 30-62 Message for RMConnector sent by Server Foundation

7. In the actual process flow, invoke the adapter also to indicate the completion of the maintenance request. Invoke the update method again, this time with values that are appropriate for a status of C (Completed).

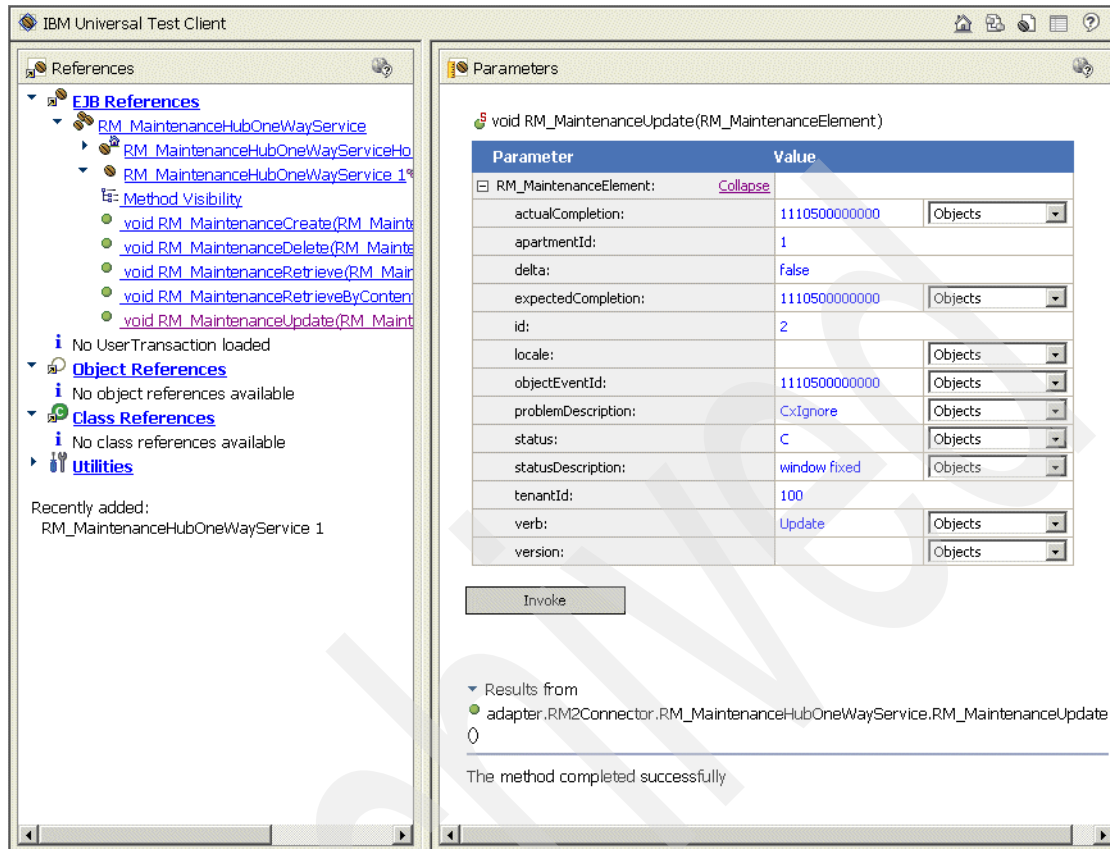


Figure 30-63 Provide parameters to the update method of the adapter

8. Inspect the database on the message broker machine. The status of the maintenance requests are stored in the table ITSO.MAINTENANCE.

Also, inspect the adapter log.

30.2.5 Developing the WSDL for the staff activity

Staff activity is also described in WSDL. To develop the WSDL for the staff activity:

1. Create a new empty service ReviewMaintenance, as shown in Figure 30-64.

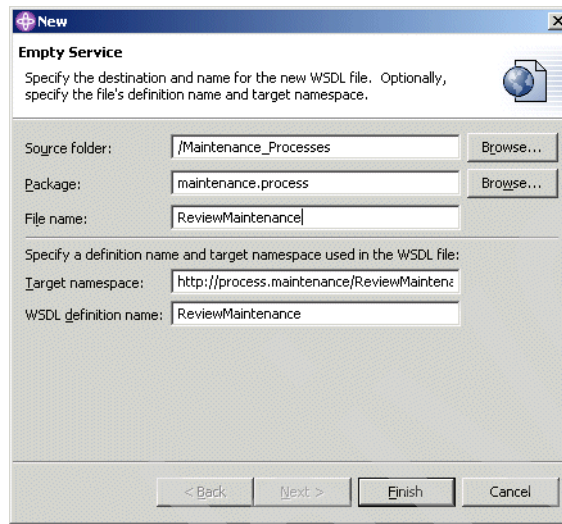


Figure 30-64 Creating new service

2. Use the WSDL editor to develop a single port type.
3. The source for the input message is MaintenanceRequest.xsd.
4. The source for the output message is ContractorRequest.xsd, which can be imported from the directory Additional Materials.

Figure 30-65 on page 695 shows the completed WSDL for the staff activity.

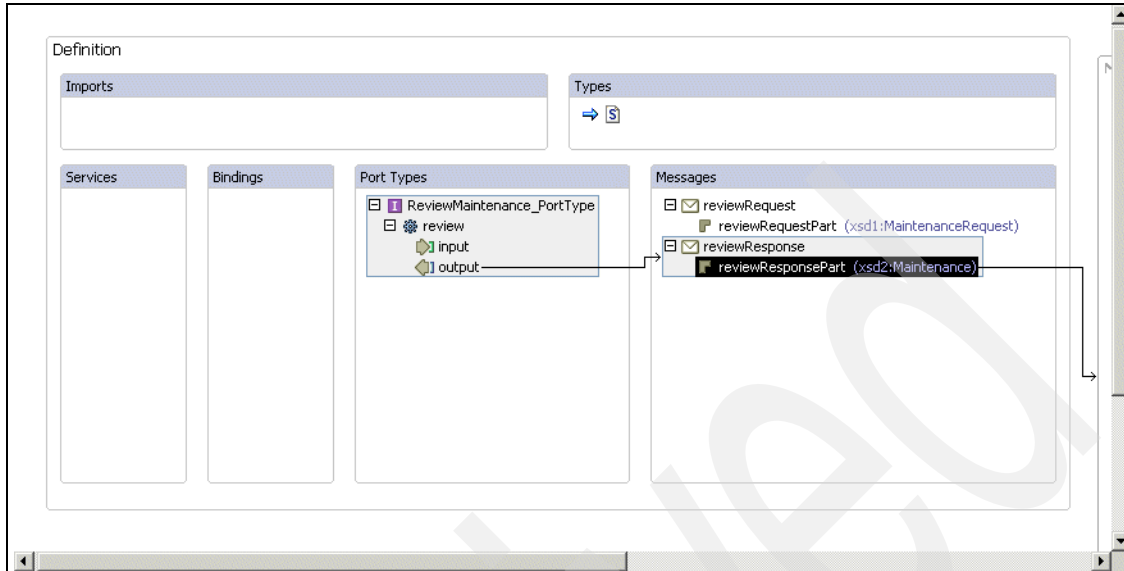


Figure 30-65 Completed WSDL for the staff activity

5. Add another operation, as shown in Figure 30-66 on page 696. This operation is executed when the inhouse technician has completed the maintenance request.

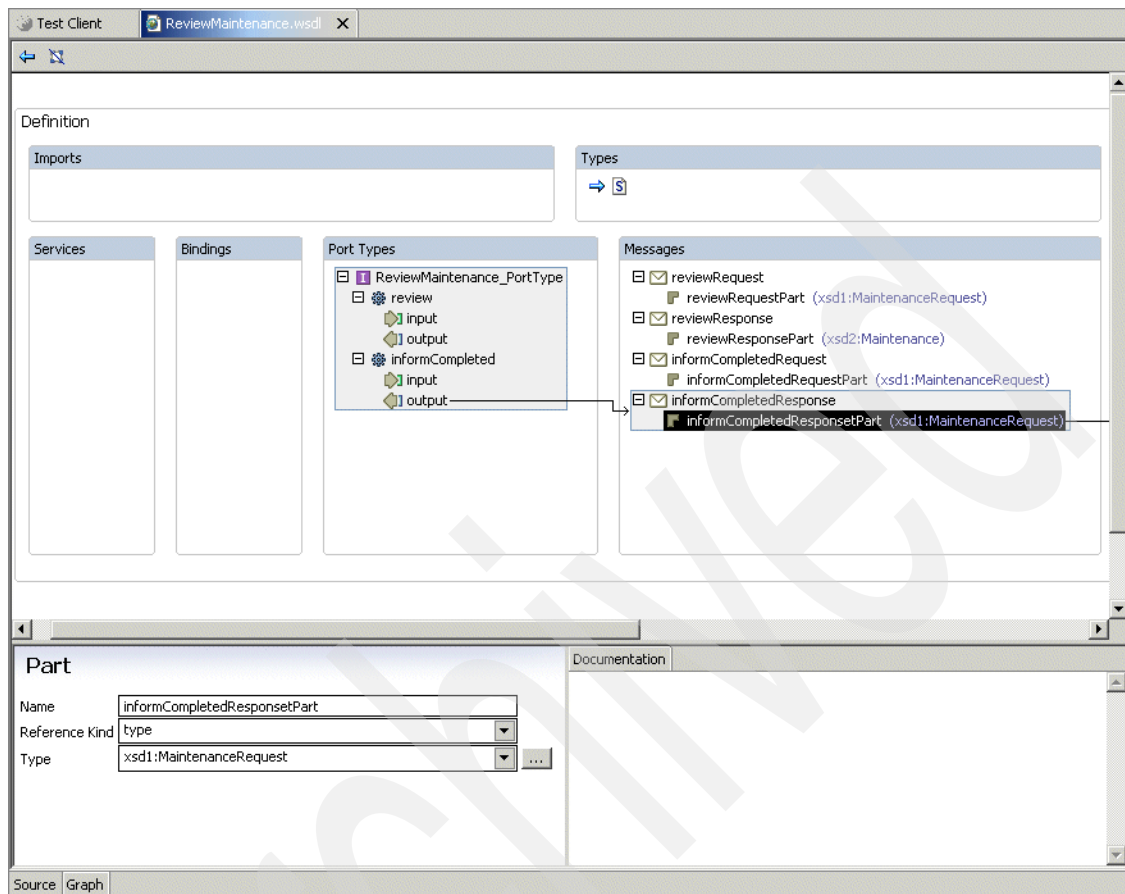


Figure 30-66 Completed WSDL for the staff activities

30.3 Developing a business process

We have now developed a number of services. This section describes how to add those services to a business process.

30.3.1 Creating a business process

To create a business process:

1. Open WebSphere Studio Application Developer in the business integration perspective.
2. Right-click the package maintenance.process and select **New** → **Business Process**.
3. Call the new process MaintenanceProcess and click **Next**.

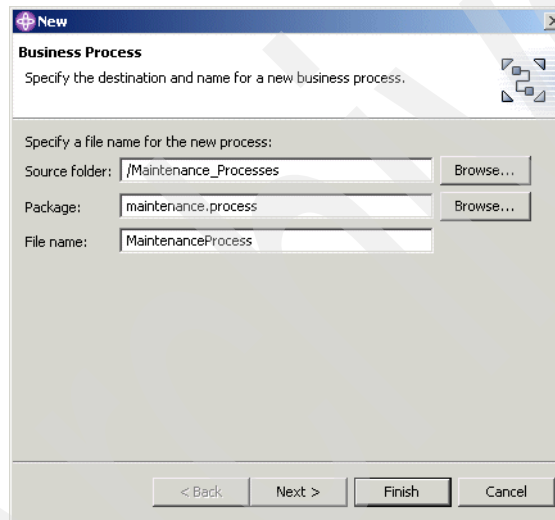


Figure 30-67 Create new business process

4. Select Flow-Based Process and click **Finish**.

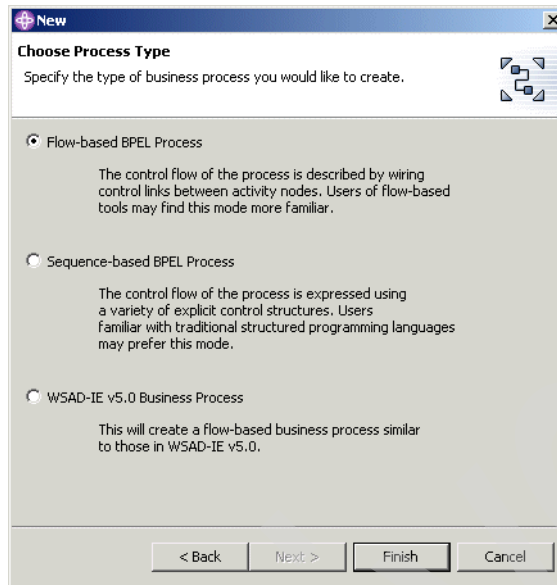


Figure 30-68 Select process type

The process editor opens.

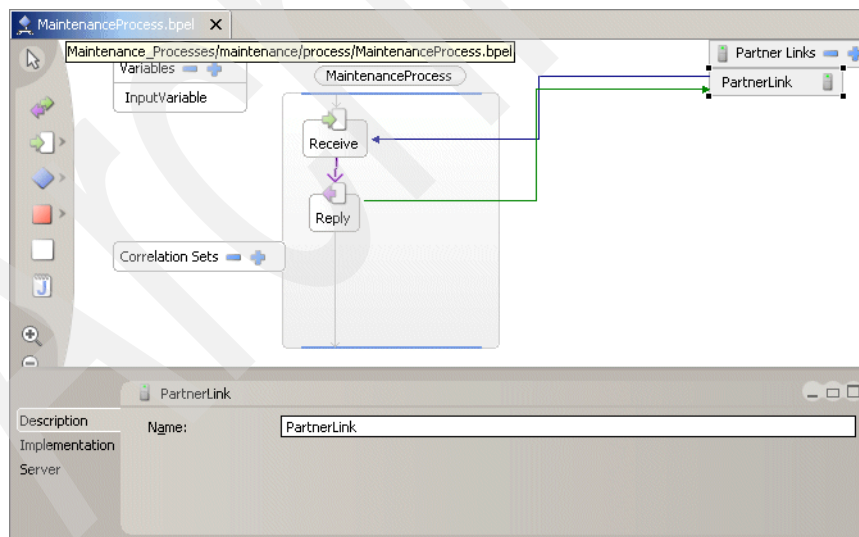


Figure 30-69 Process editor

30.3.2 Overview of the BPEL editor

Note: The following sections provide a brief overview of the BPEL editor. It is not necessary to do the tasks described. These sections are for reference only. If you want to go directly to the step-by-step instructions for building the first BPEL process, go to 30.3.6, “Developing the business process” on page 708.

The basic terms used by BPEL when describing a process are:

- ▶ Partners, which are the external users or services that interact with the process.
- ▶ Activities, which are the individual business tasks within the process that compose the larger business goal.
- ▶ Elements, which supplement activities and assist them in accomplishing their tasks. Elements are nested within the activities with which they interact.
- ▶ Variables, which store the messages that are passed between these activities and partners.
- ▶ Staff Assignment, which sends a task out to a human for interaction.
- ▶ Compensation, which returns the business process to a balanced state if something happens during execution to upset that balance.
- ▶ Fault handling, which identifies possible problems ahead of time and tells the process how to deal with them.
- ▶ Correlation sets, which identify tokens that allow two participants in a conversation to identify each other in subsequent communications.

30.3.3 Anatomy of the BPEL editor

The BPEL editor can be split up roughly into eight separate areas (labeled 1 to 8 in Figure 30-70 on page 700). Those areas are:

1. Palette, which are icons of the activities that can be dragged to the process area. Note some of the icons on the palette actually open drop-down menus when clicked, as shown in Figure 30-72 on page 703.
2. Variables, which are the defined variables for the process.
3. Process area, which is the section on the canvas to visually arrange the activities.
4. Action bar, which is a series of actions that are related to the currently selected activity.

5. Partner links, which are the external users or services that interact with this process.
6. Canvas, which is the area that contains all visual elements of the editor.
7. Details, which are situated below the canvas and allow the currently select activity to be configured. Reflects the parameters that are available for the selected activity.
8. Correlation sets, which show the defined correlation sets for the process.

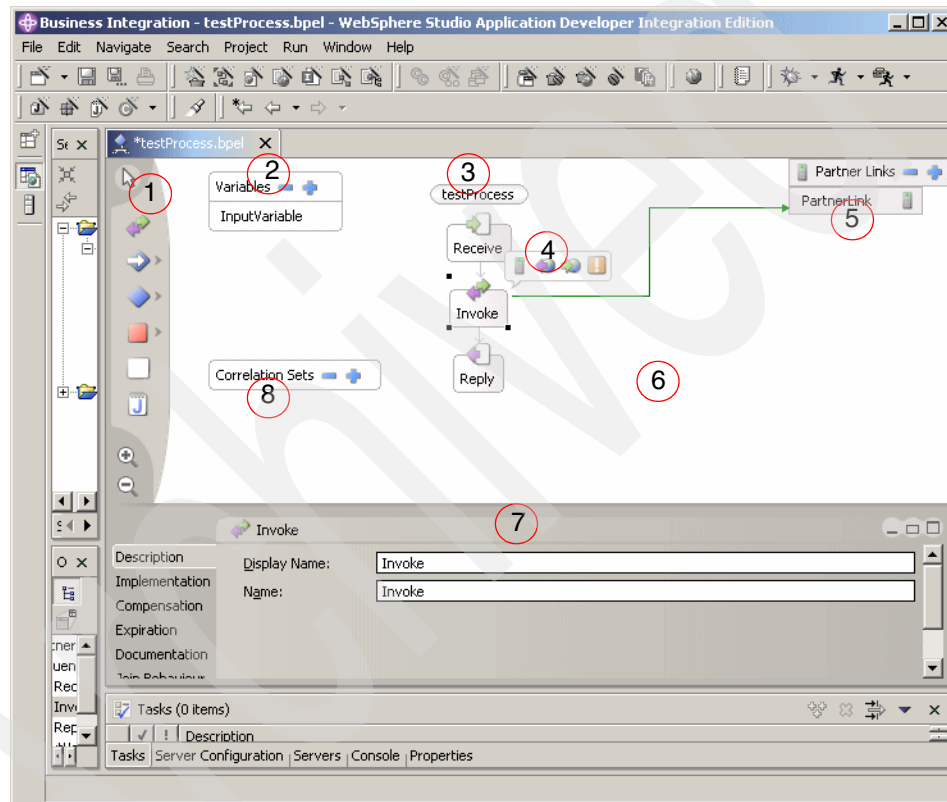


Figure 30-70 The BPEL editor

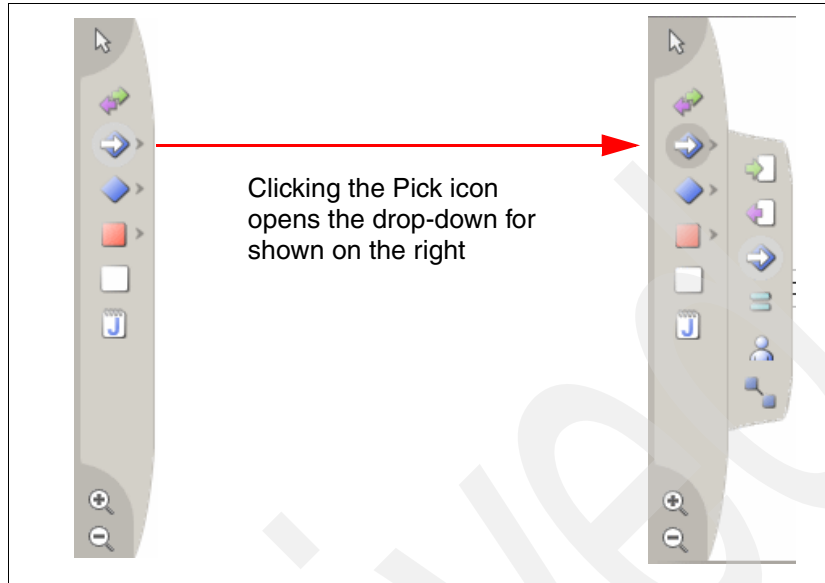
















Figure 30-71 Drop-down list from the palette

30.3.4 Activities and their icons

Table 30-1 provides an overview of the available BPEL4WS process activities. The icons in the table are what appear in the process area of the canvas. The icon you choose in the activities palette looks similar to the small icon in the top center of these icons.

Table 30-1 WebSphere Process Choreographer activities

Activity	Description
	The invoke activity performs an operation. The operation is defined by a partner link and can be synchronous or asynchronous.
	The receive activity waits for an external input to the process before continuing. The operation supported by the receive activity is defined by a partner link.
	The reply activity sends a message to the partner defined by a partner link. This is typically used in processes which need to return a message to the partner which instigated the process.

Activity	Description
 Pick	The pick activity waits for an incoming message and selects a path appropriate to the first message received. A time-based path can be configured to manage situations where no message is received. A partner link is associated with each message path.
 Staff	The staff activity delegates a task within the process to a human. The user interface in this case is either a custom application based on the process choreographer API or the web client provided that comes with WebSphere Business Integration Server Foundation.
 Transform	The transformer activity maps the contents of one or more message types to the contents of another.
 Assign	The assign activity copies information from one part of the process to another.
 Switch	The switch activity evaluates the conditions on a series of control paths and follows the first one which matches.
 While	The while activity repeats the activities which it contains as long as a condition is met.
 Wait	The wait activity stops the process until a point in time has occurred or a time interval has elapsed.
 Sequence	The sequence activity defines a serial control path within a process.
 Flow	The flow activity defines a potentially parallel control path within a process.
 Terminate	The terminate activity stops the process immediately without performing any compensation or fault handling. The behavior of this activity depends on the location within the process.
 Throw	The throw activity signals that an error has occurred. This is typically handled by a fault handler element associated with a higher level of process structure.


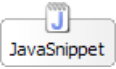
Activity	Description
	The empty element does nothing. It can be used as a placeholder during process design, and then changed to the appropriate activity when the process is implemented.
	Java code can be embedded into the process using the JavaSnippet activity. While it is possible to embed business logic into this type of activity it is not advisable, as it removes the clarity of the process modelling. Snippets are designed to perform lightweight utility activities such as data mapping.

Figure 30-72 shows the Invoke activity. You can see the small icon on the right, which you choose in the activity palette, is the same as the small upper center of the icon that appears on the process area.

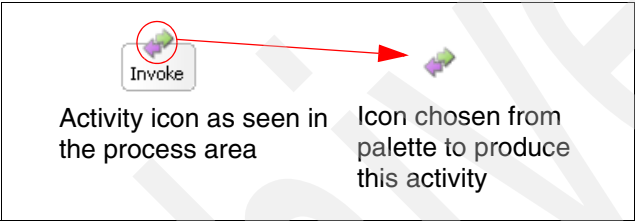


Figure 30-72 Palette icons

Tip: You can change the activity type by right-clicking the activity in the process editor and selecting **Change Type**. Use this facility with caution, however, because changing the type of a structured activity (such as sequence or flow) deletes the contents of that block.

30.3.5 Tasks

In the following sections, we show you how to complete some of the most common tasks to get you started.

Creating a new business process

To create a new business process:

1. Switch to the Business Integration Perspective.
2. Create a Service project by selecting **File** → **New** → **Service Project** from the main menu.
3. Select this newly created Service project and click **File** → **New** → **Business Process**.

4. In the New Business Process window, specify a package and a name for the new process, and click **Next**.
5. In the Choose Process Type window, choose from the following BPEL process options:

Flow-based	Use this setting if you want control dictated by control links.
Sequence-based	Use this setting if you want your process controlled by structured activities.
6. Click **Finish**.

These steps create a process and launch that process in a new process editor. A few files are generated that you can view in the Services view. The file with the BPEL extension contains the code that describes this process, and the WSDL file describes the service interface.

Creating a partner link

To create a partner link:

1. In the Partner Links area on the canvas, click the plus icon.
2. Give the partner link an appropriate name.
3. Click the Implementation tab in the Details area.

Figure 30-73 shows where you can find these fields and icons in the editor.

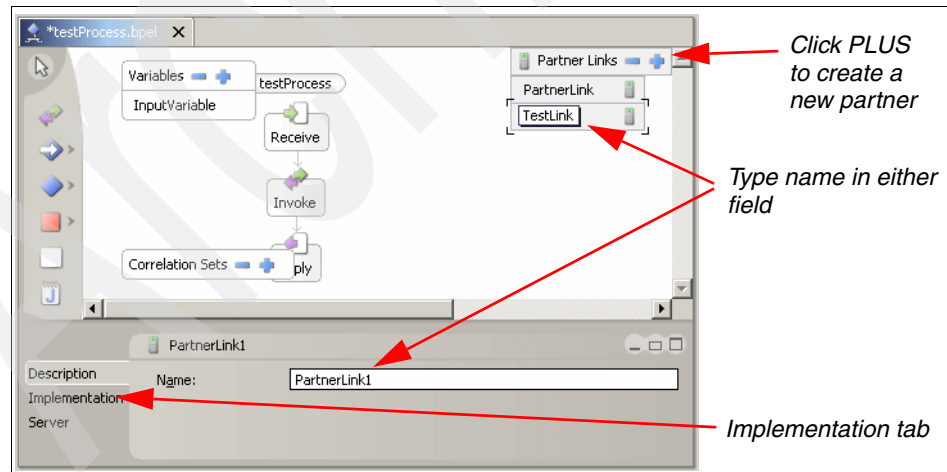


Figure 30-73 Create and name a new partner link

Figure 30-74 on page 705 shows the fields in the Implementation tab of the new partner link.

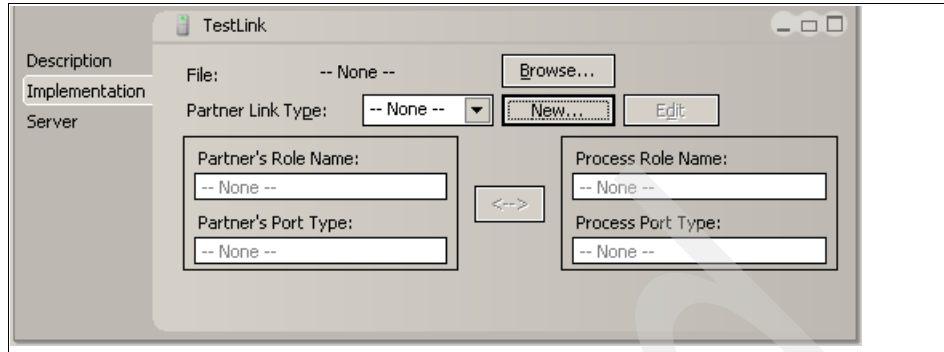


Figure 30-74 Implementation tab for a partner link

4. If you would like to reference a WSDL interface file (such as the one that was created automatically by the Process editor), click **Browse**.
5. To launch the New Partner Link Type window, either click **New**, or select a Partner Link Type from the drop-down list and click **Edit**.

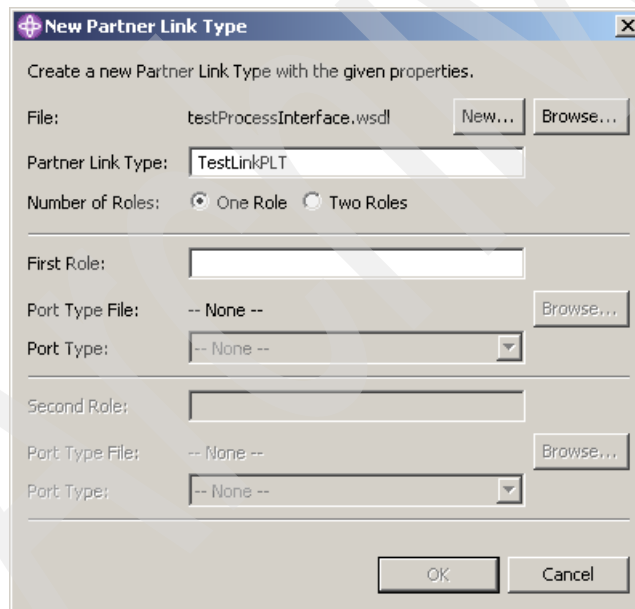


Figure 30-75 New partner link type window

6. Modify the fields in the New Partner Link Type window (Figure 30-75) as follows:

- a. If this is a new partner link, then you can choose to reference an existing WSDL interface file (click **Browse**), or create a new one (click **New**).
 - b. Select the number of roles that this partner link will expose.
 - c. Name the roles, browse to the WSDL interface file that defines the port type that you want to associate with the role, and select it.
7. Click **OK**.

Creating a variable

To create a variable:

1. In the Variables area on the canvas, click the plus symbol (+).
2. Give the variable an appropriate name.

Figure 30-76 shows where you can find these fields in the editor.

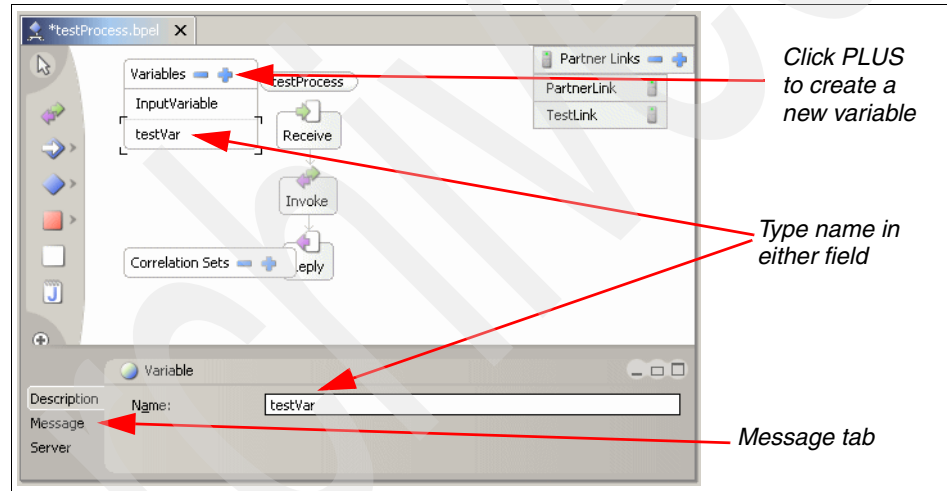


Figure 30-76 Create a new variable

3. Configure the variable as follows:
 - a. Click the Message tab in the Details area.
 - b. Browse to a WSDL file that has at least one message type and part defined.
 - c. Select an appropriate Message type and part.

Figure 30-77 on page 707 shows the message tab fields.

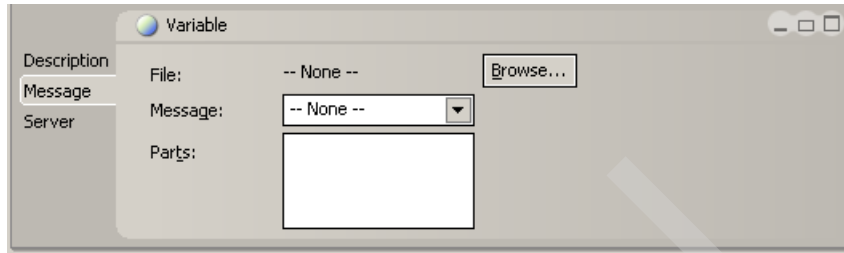


Figure 30-77 Message tab fields

Creating an activity

To use the Palette to add an activity:

1. In the palette, click an activity's icon.
2. Click a position in the process editor where you want the activity to appear.

To use the context menu to add an activity:

1. On the canvas, decide where you want to place your new activity, and select the existing activity that is directly below that position.
2. Press SHIFT+F10 or right-click to launch the context menu.
3. From the menu, select **Insert before** and select the desired activity.

Figure 30-78 on page 708 shows the Insert before selection in the context menu.

Note: The top of the context menu has been removed from the figure.

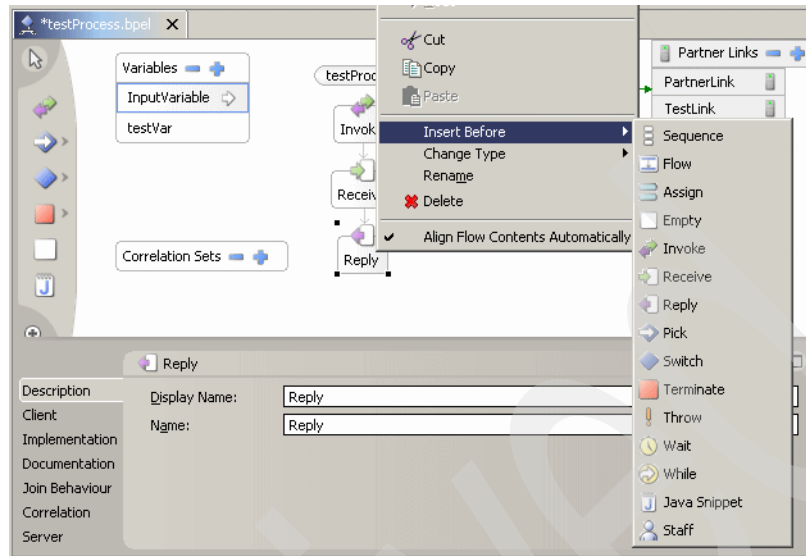


Figure 30-78 Add activity context menu

30.3.6 Developing the business process

When you created the business process, a default process interface was generated. You first customize this interface, following these steps:

1. Select the partnerlink in the process editor.
2. Select the section Implementation at the bottom of the editor.
3. Click the link to the process interface WSDL to open the WSDL editor.

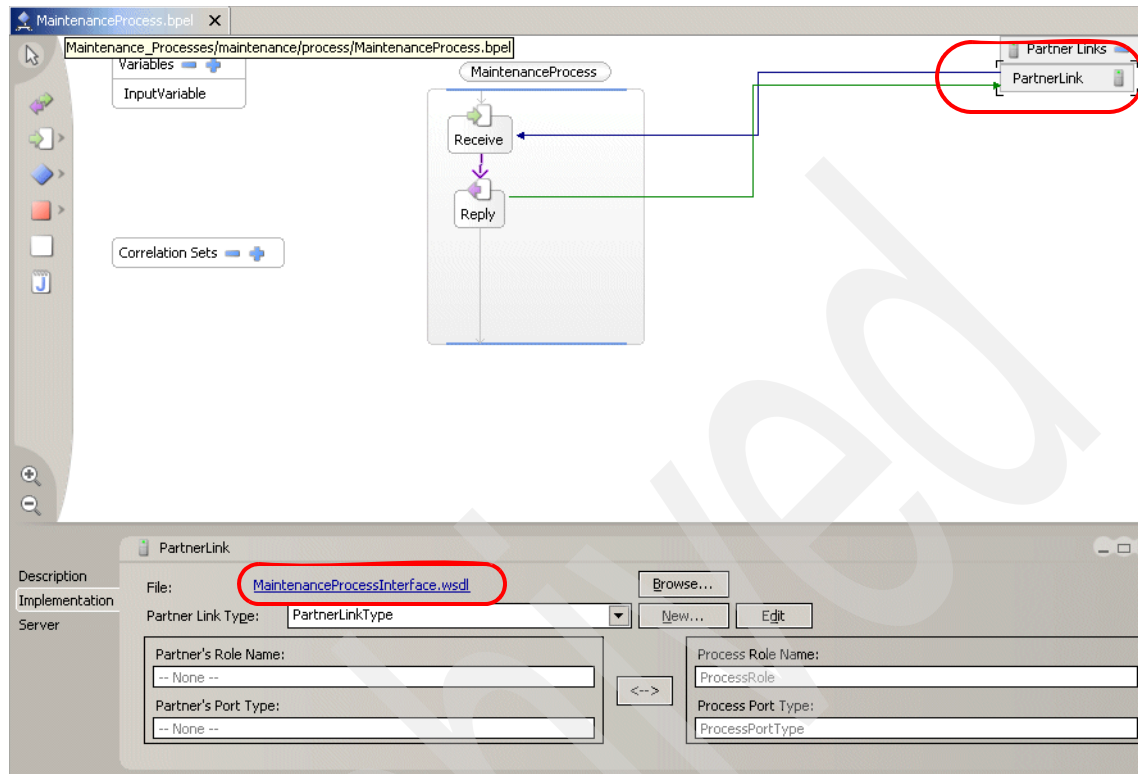


Figure 30-79 Edit the process interface

4. In the WSDL editor, correct the type of the input message. Set it to the schema definition MaintenanceRequest.xsd.

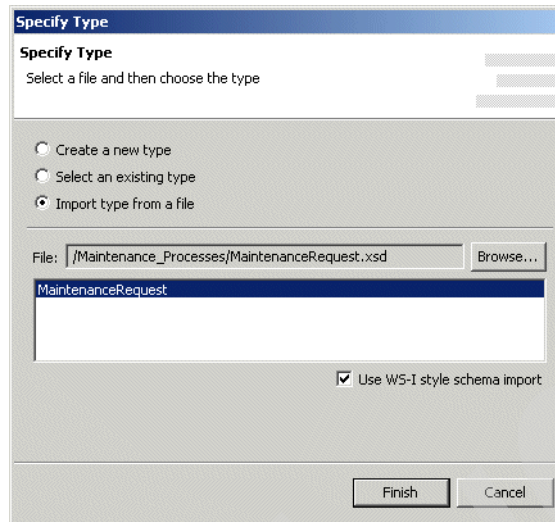


Figure 30-80 Update type of input message

5. In the existing interface, the output message had the same structure as the input message. Change this by creating a new message.

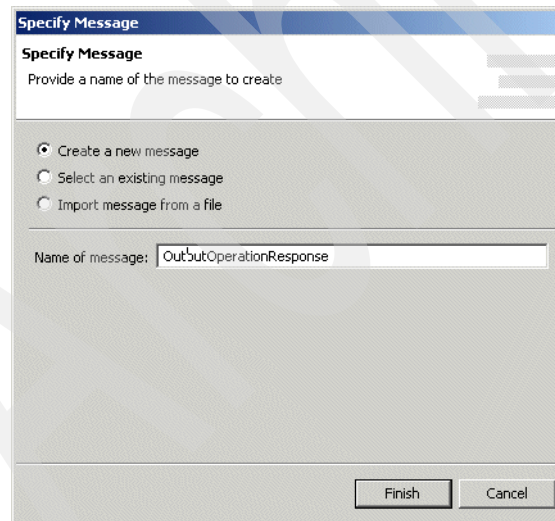


Figure 30-81 Update message of output of operation

6. Add a part (contents) to this new message. Figure 30-82 shows the changed WSDL for the process interface.
7. Save the WSDL file and close it.

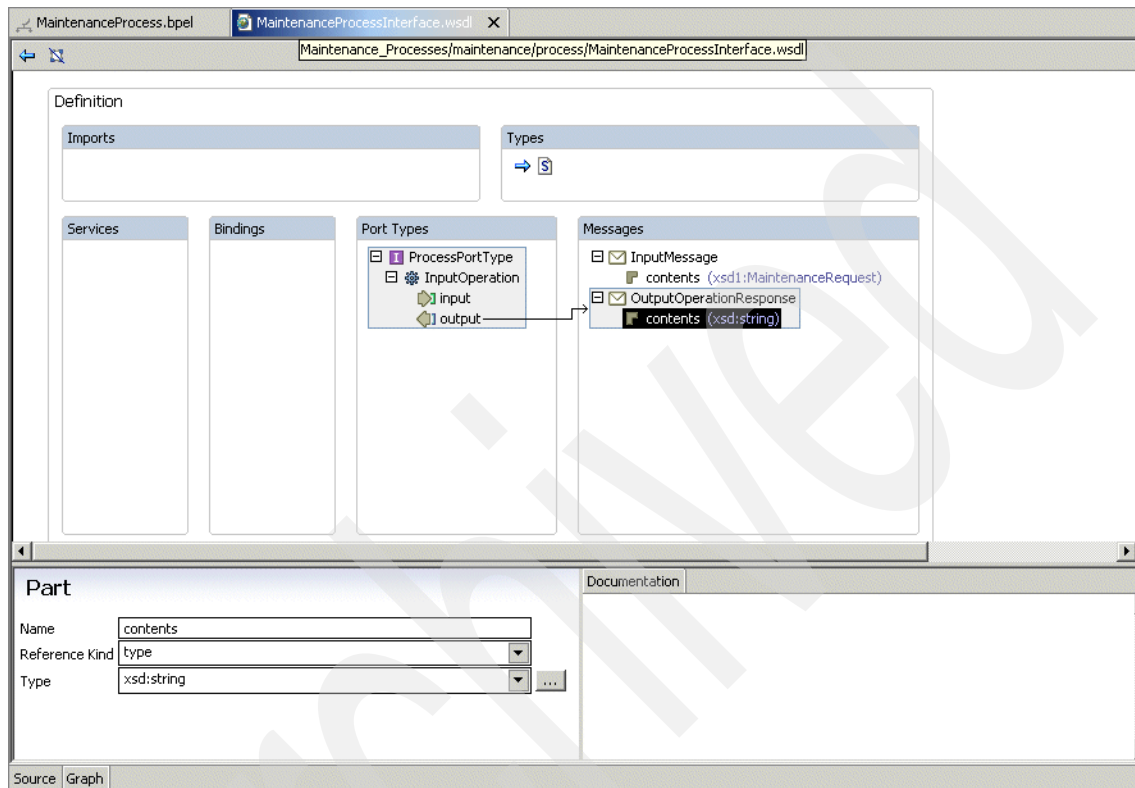


Figure 30-82 Updated process interface

The change to the interface causes an error in the process editor, as shown in Figure 30-83 on page 712.

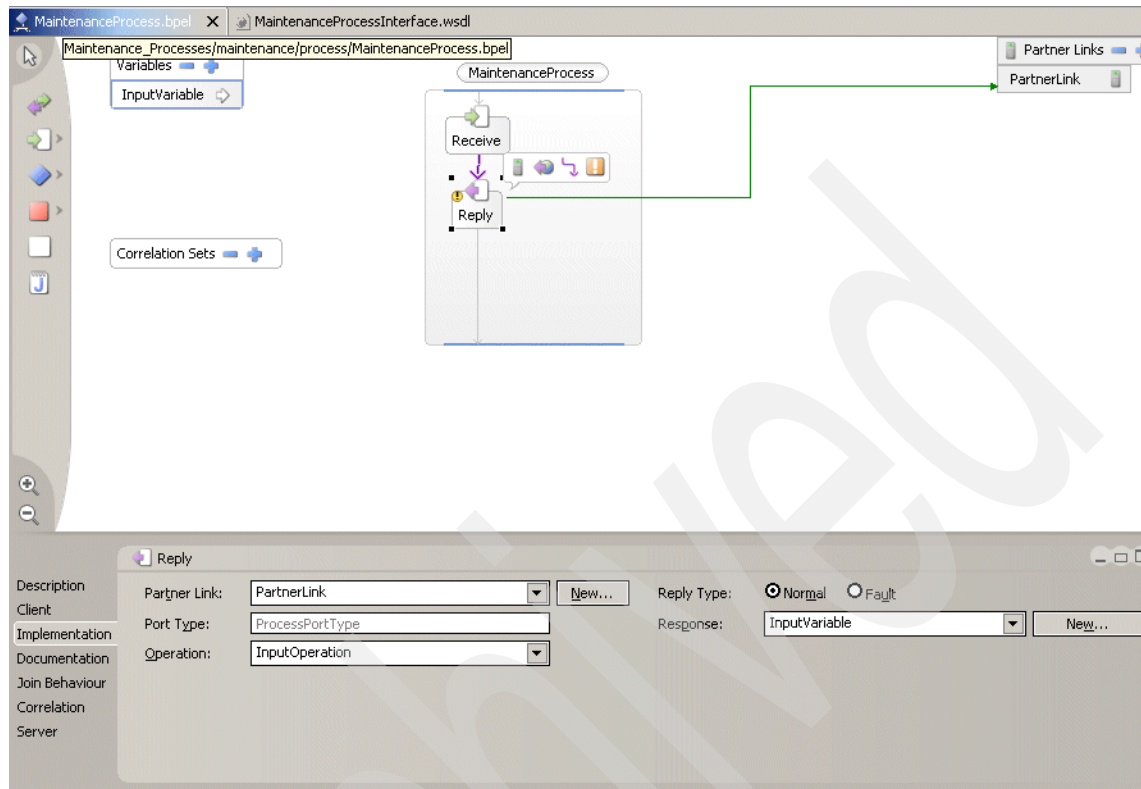


Figure 30-83 Reply activity needs to be corrected

8. To fix the error, select the Reply activity.
9. Select the Implementation tab if it is not selected.
10. Click **New** next to the response variable.
11. Give the new response variable a name and click **OK**.

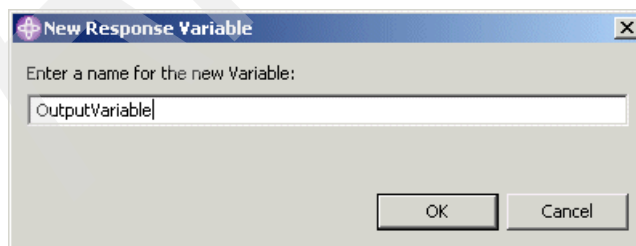


Figure 30-84 Create new response variable

12. Save the changes. The error should be fixed.

13. There might still be warnings in the task view. However, these will be corrected later.

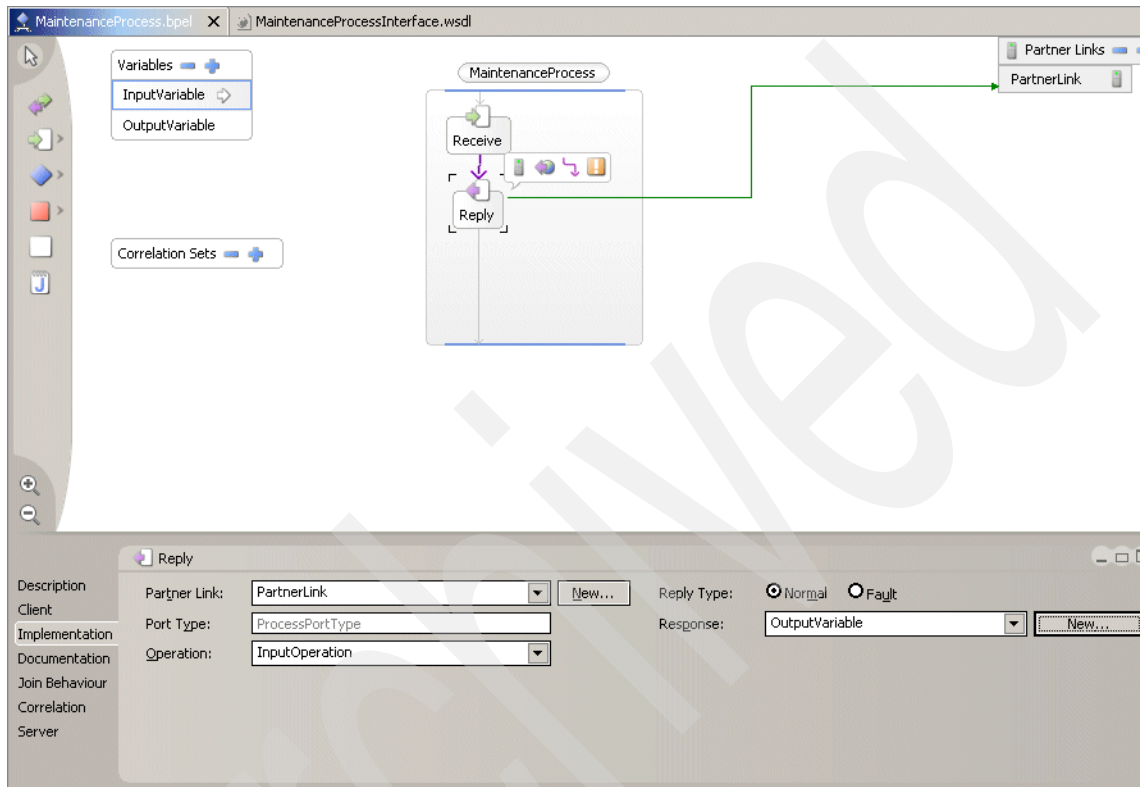


Figure 30-85 Corrected Reply activity

14. Add all the services to the process and connect them by dragging the ReviewMaintenance.wsdl to the process and dropping it on the canvas.

15. You are prompted to select the service. Click **OK**.

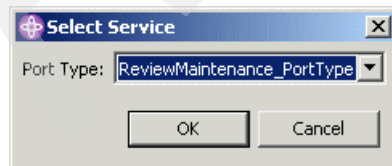


Figure 30-86 Select service

16. Copy the WSDL files for the InhouseSend service from the package inhouse.send to the package maintenance.process.
17. Drag the file InhouseSendService.wsdl to the process and drop it on the canvas.

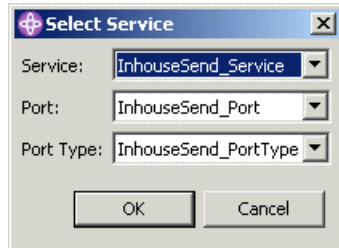


Figure 30-87 Select service

18. Copy the three WSDL files and the schema file for the connector service from the package adapter.RM2Connector to the package maintenance.process.
19. Add the adapter call, using the Service WSDL for the EJB binding. This WSDL is located in the project Maintenance_ProcessesEJB.

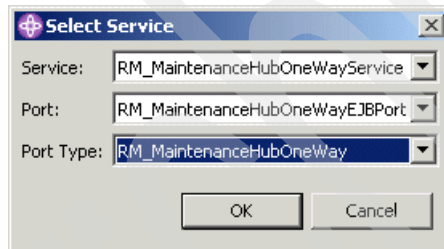


Figure 30-88 Select service

Figure 30-89 on page 715 shows the process editor with the added partners.

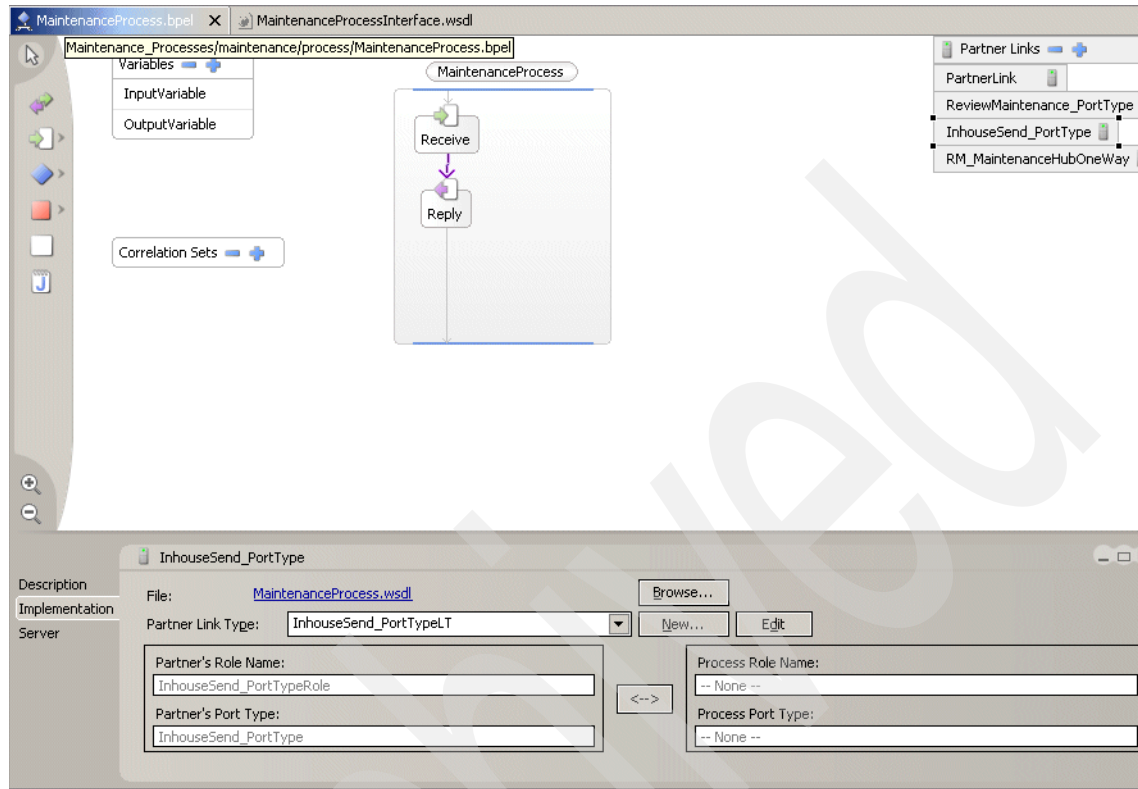


Figure 30-89 Partners added to the process editor

20. Open the Java class file `RM_MaintenanceHubOneWayServiceBean.java` and locate the method `RM_MaintenanceUpdate`. Add a line of code to set the verb to `Update`, as shown in Example 30-3. When we used the test client, we set the value of `Verb` by hand.

Example 30-3 Add code to the EJB

```
// user code begin {pre_execution}
argBodyPart.setVerb("Update");
// user code end
```

21. Save and close the Java class file.
22. Back in the BPEL editor, from the palette, add a staff activity.
23. While the staff activity is selected, click **Browse** at the bottom of the window.
24. Locate the file ReviewMaintenance.wsdl to link it to this activity.

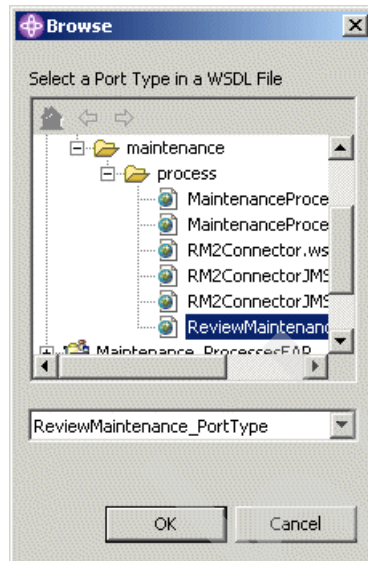


Figure 30-90 Select the interface of the staff activity

25. Set the request variable to the process input variable.
26. Click **New** to create a new variable to store the result of the review step.

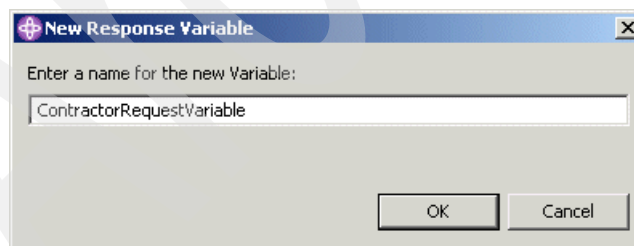


Figure 30-91 Create a new response variable

27. Connect the staff activity to the other activities.

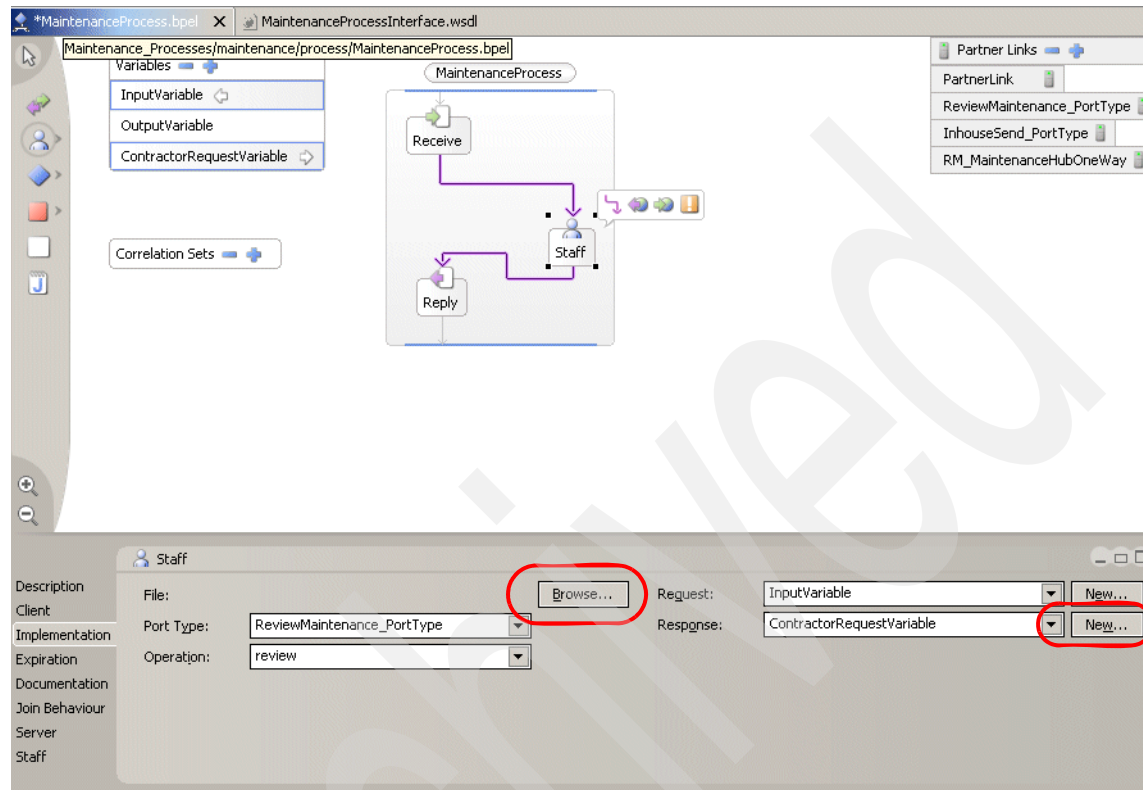


Figure 30-92 Details of the staff activity

28. Add an invoke activity to the flow.
29. Select the correct Partner Link from the drop-down box.
30. Create a new input variable, for example InhouseSendRequestVariable.
31. Connect the invoke activity to the other activities (Figure 30-93).

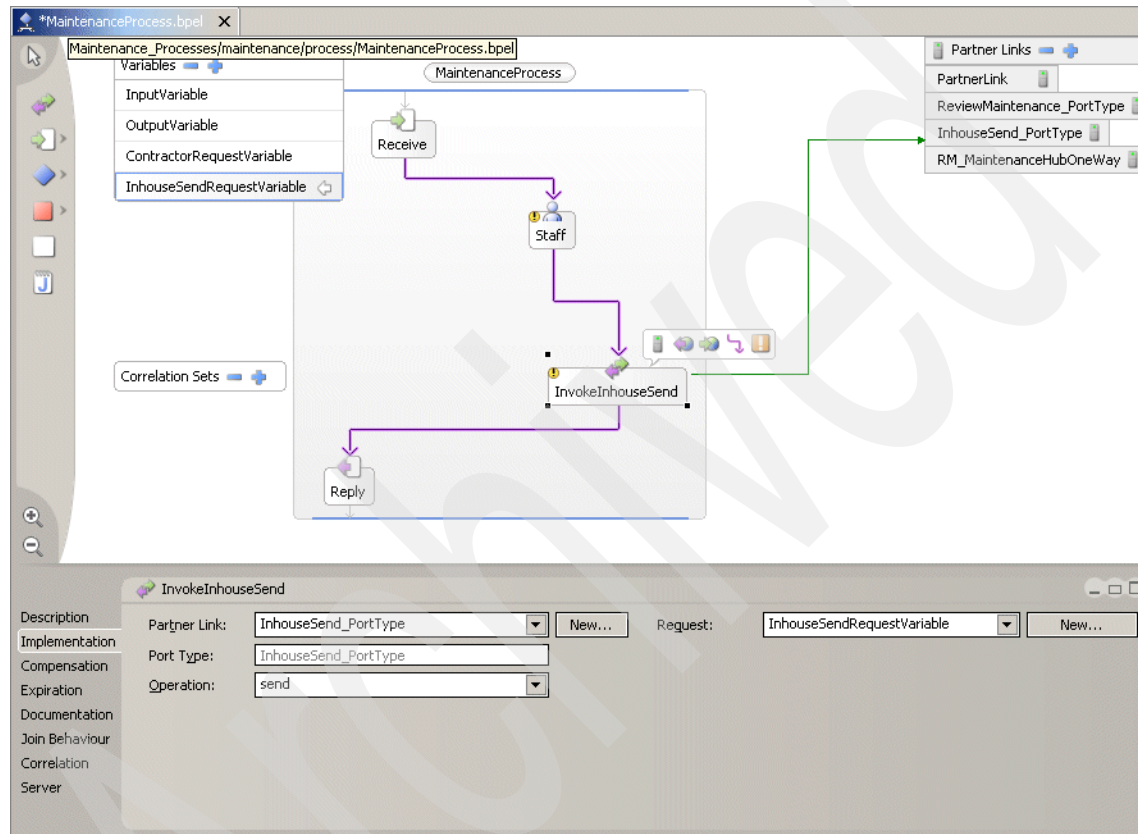


Figure 30-93 Process with two activities

32. Add another staff activity to the flow. This time, select the operation `informCompleted`.
33. Create a new response variable and set the variables as shown in Figure 30-94.

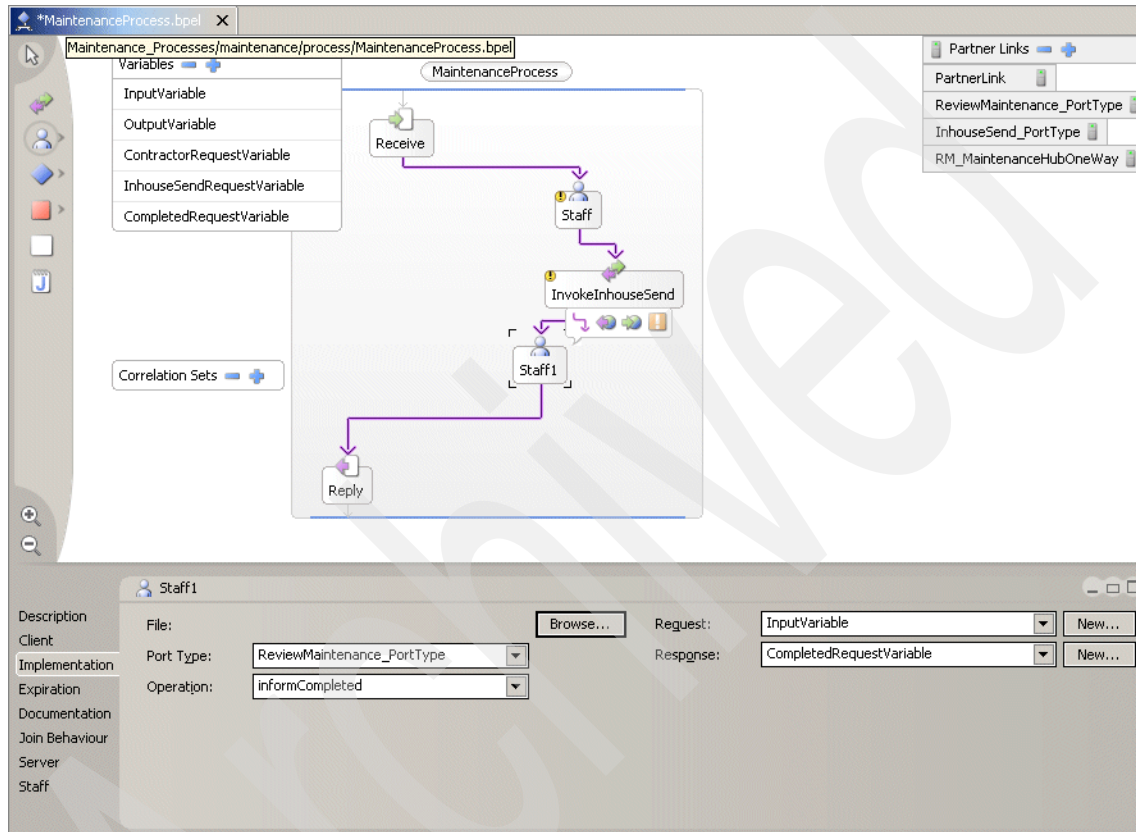


Figure 30-94 Process with two staff activities

34. The output of the second Staff activity is a complex type, while the process output is only a single string. To set the process output, add an Assign activity to the process flow.
35. While the Assign activity is selected, select the tab Implementation at the bottom.
36. In the drop-down boxes From and To, select Variable or Part.
37. As the From variable, select `CompletedRequestVariable`. Expand it so that we can select the element `StatusDescription`.

38. As the To variable, select OutputVariable. Expand it to locate the element contents.
39. Save the process.

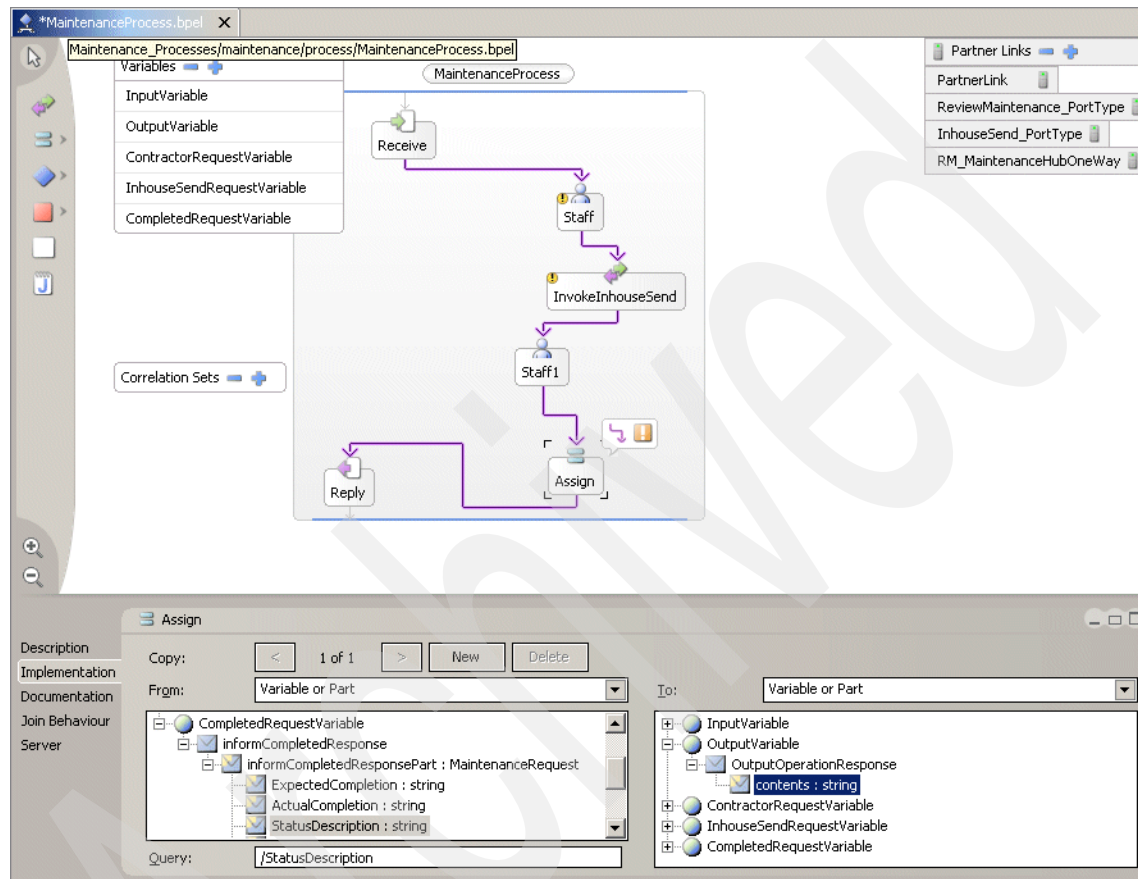


Figure 30-95 Assign activity

30.3.7 Correcting some errors

If you save the process, you see that there are several errors to resolve. To resolve these errors, follow these steps:

1. In the process editor, select the process symbol (top of flow). Then, select the section Server. Check the option that this is a long-running process. This action should again fix some issues.

Note: Processes that include staff activities are by definition long-running.

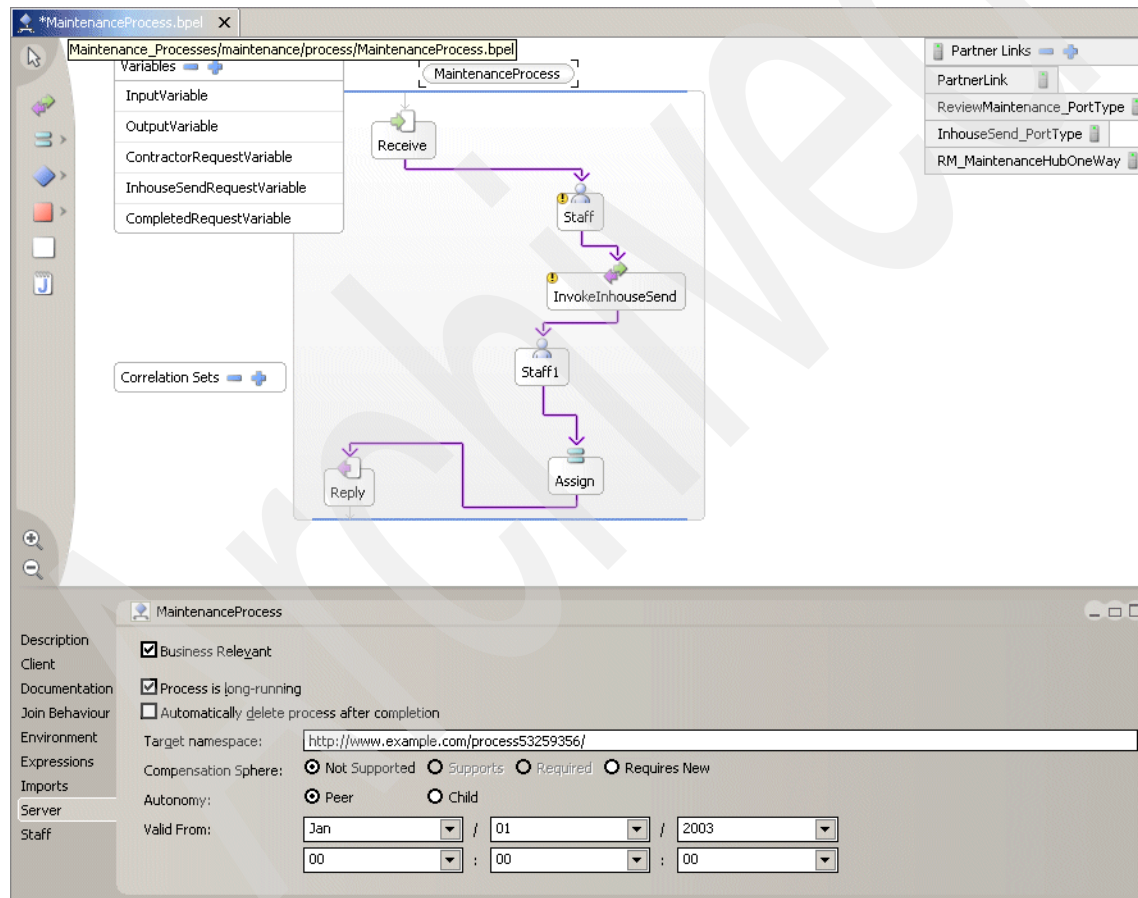


Figure 30-96 Make the process long-running

The staff activities are not really assigned to anybody. For our purposes, it is sufficient to assign them to everybody.

2. Select a staff activity, and click Staff in the bottom pane.
 3. Set the verb to Everybody.
- Assign the staff activities to Everybody for the second activity also.

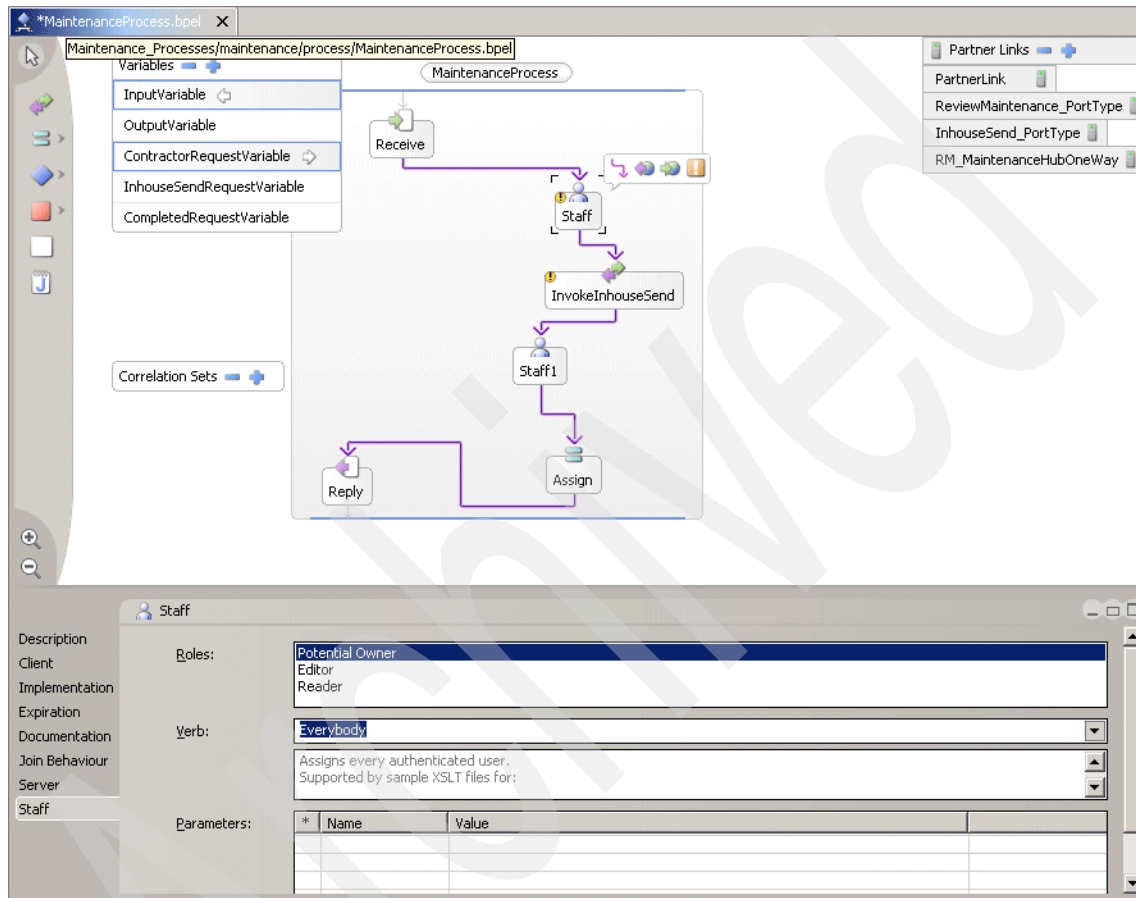


Figure 30-97 Assign activity to everybody

4. The BPEL process contains a partner link for the review process. We had added this WSDL file to assist in the modeling, but this link is not required anymore. Delete the partner link and save the process.

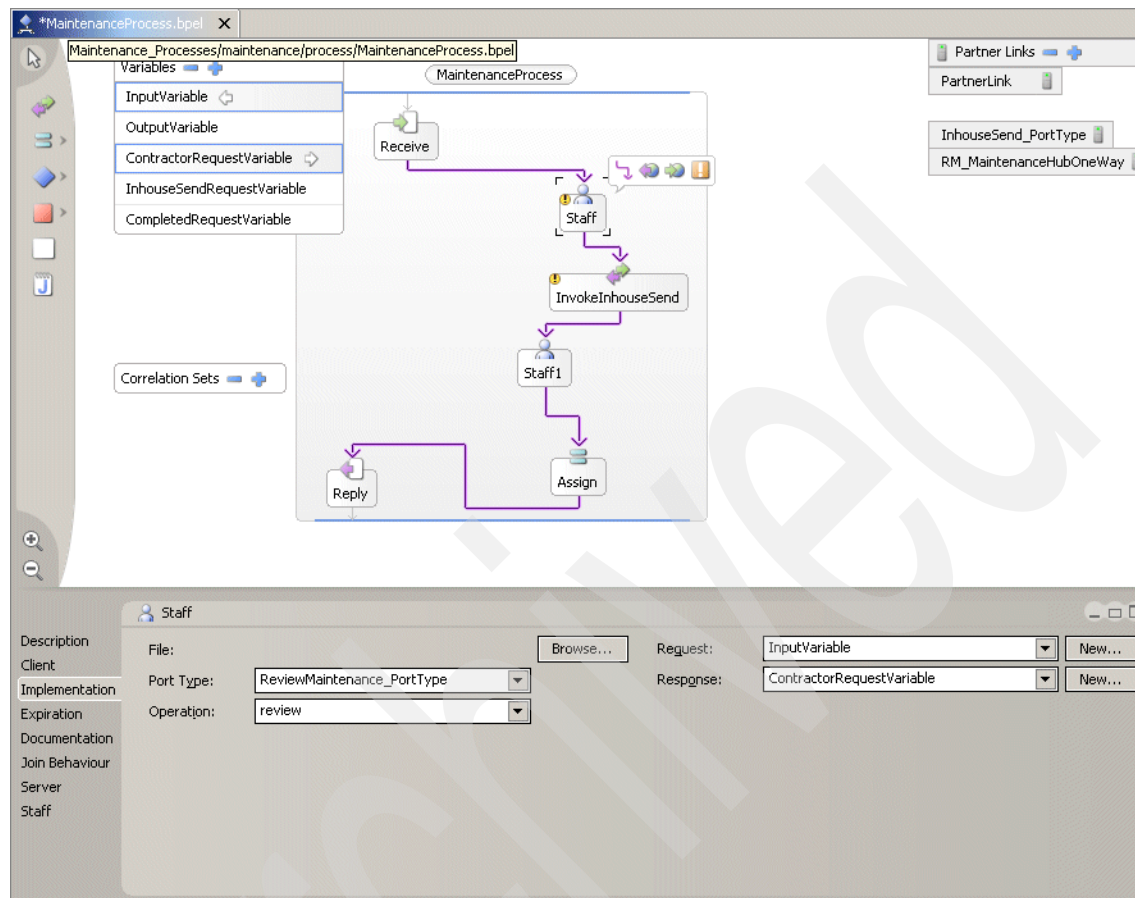


Figure 30-98 Partner link removed

- The input to the InvokeInhouseSend is retrieved from the variable InhouseSendRequestVariable. To populate this variable, use a transformer service. Add a transform activity to the flow, and name it PrepareInhouseSend. Connect the activity to the other activities, as shown in Figure 30-99.

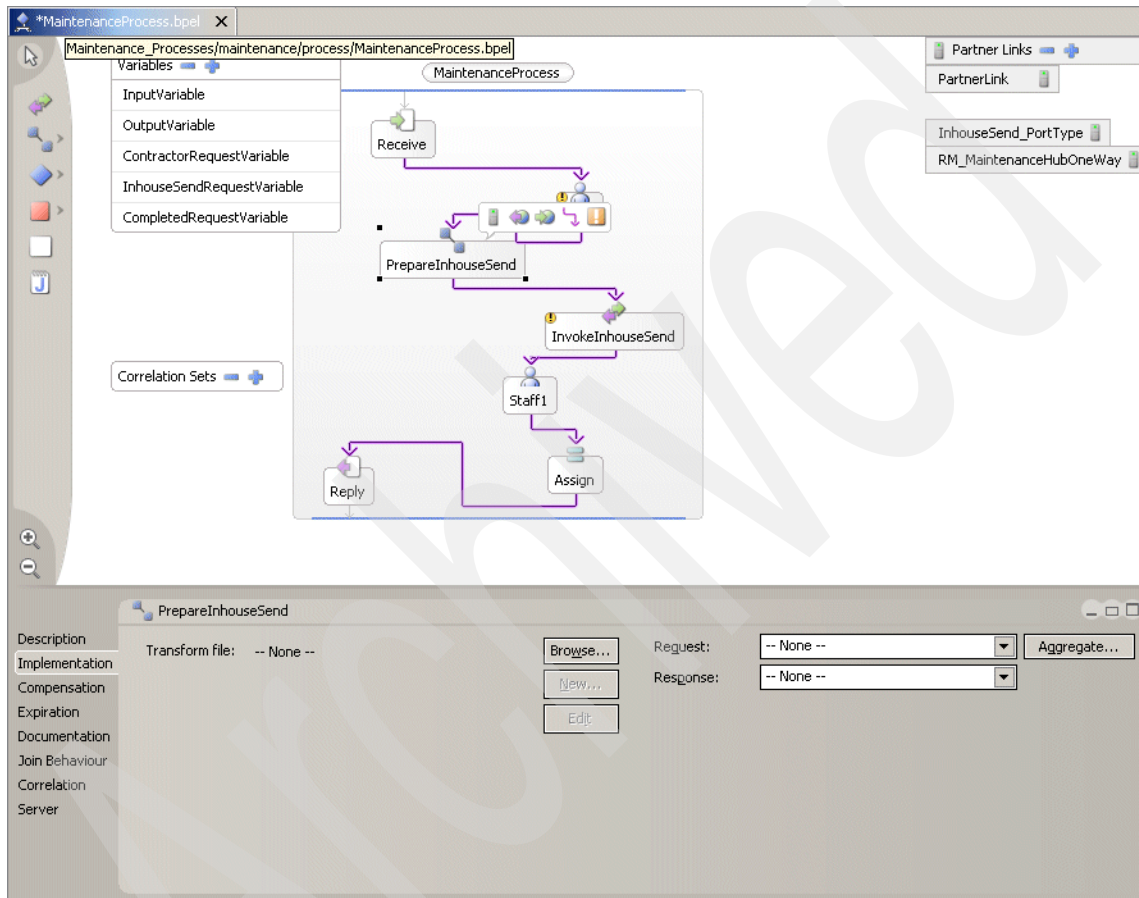


Figure 30-99 Add a transform activity

- The transform activity takes the output of the Staff activity as input, which is the variable ContractorRequestVariable. It creates as output the variable InhouseSendRequestVariable. Select these variables on the implementation tab of the transform service.

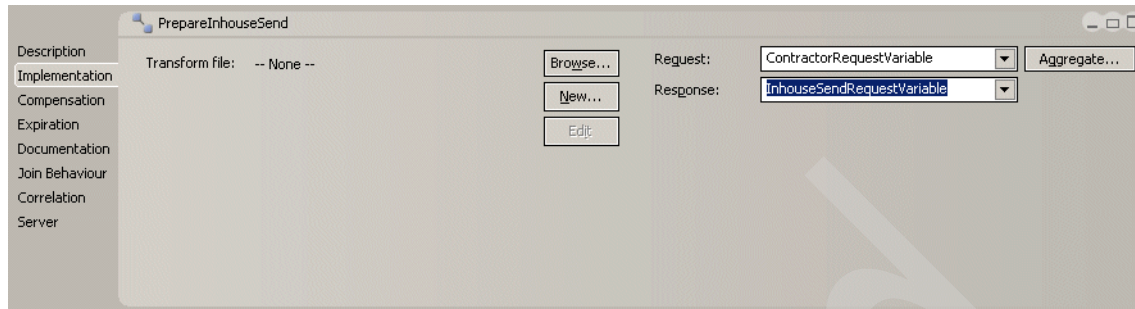


Figure 30-100 Set the variables for the transform service

7. When the variables are set, click **New**.
8. Provide suitable names for the link, service, port, and operation, as shown in Figure 30-101. Click **OK**.

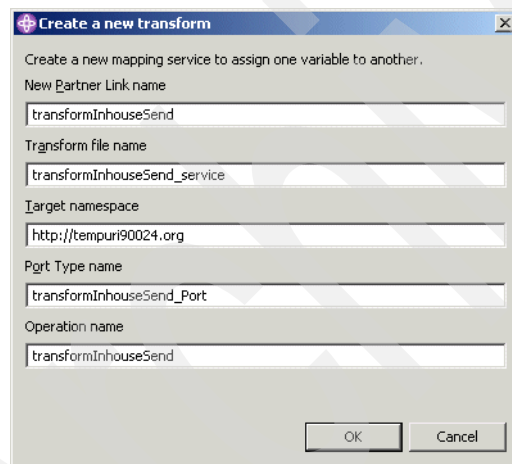


Figure 30-101 Create a new transform activity

9. The transformer editor opens, showing the input and output variables at the top. Expand both variables and drag-and-drop the corresponding elements from the left pane to the right pane. Transformation statements are added at the bottom on the editor.

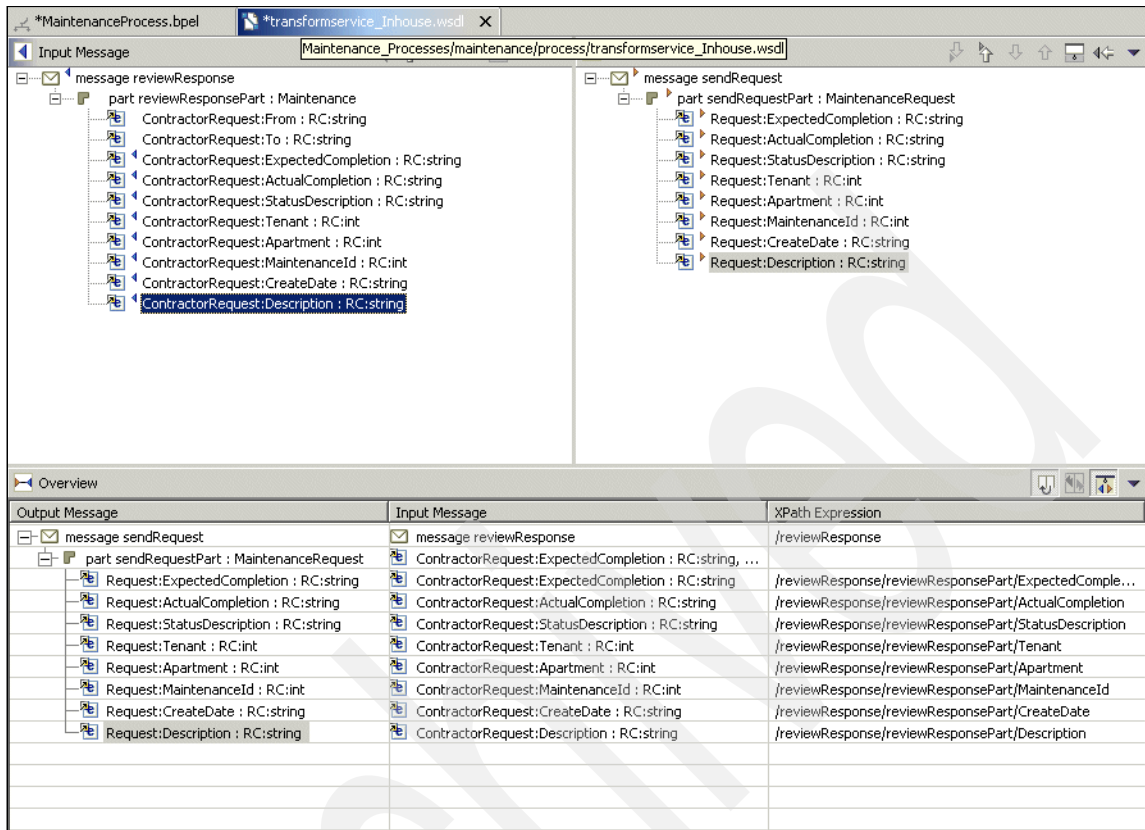


Figure 30-102 Create transformation mappings

10. Save the transformation WSDL and close it.

30.4 Deploying and testing

To deploy and test:

1. Close all open editors.
2. Stop first the Test server, if it is running.
3. Right-click the .bpel file and select **Enterprise Services** → **Generate Deploy Code**.

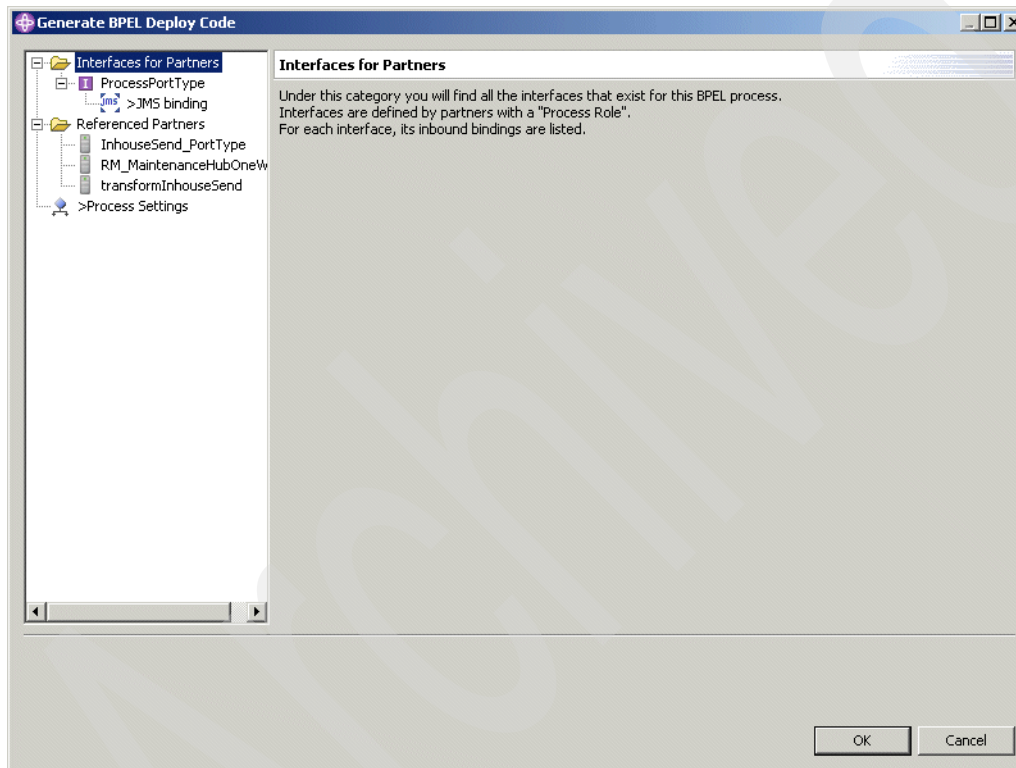


Figure 30-103 Generate deployment code

4. When the wizard starts, no additional changes are required. Click **OK**.
5. You might get a warning message about repair the server configuration. Click **OK**.
6. When the deployment is completed, switch to the server perspective.
7. Right-click the server in the pane server configuration and select **Create tables and data sources**.

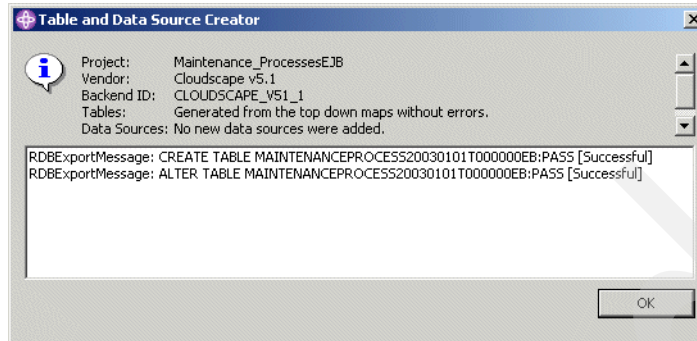


Figure 30-104 Tables created to support business process

8. When the table is created, restart the server.

30.5 Testing the process flow

To test the process flow:

1. When the server is restarted, right-click it and select **Launch Business Process Client**.
2. Log on as db2admin with password itso4you.
3. Select **My Templates**.

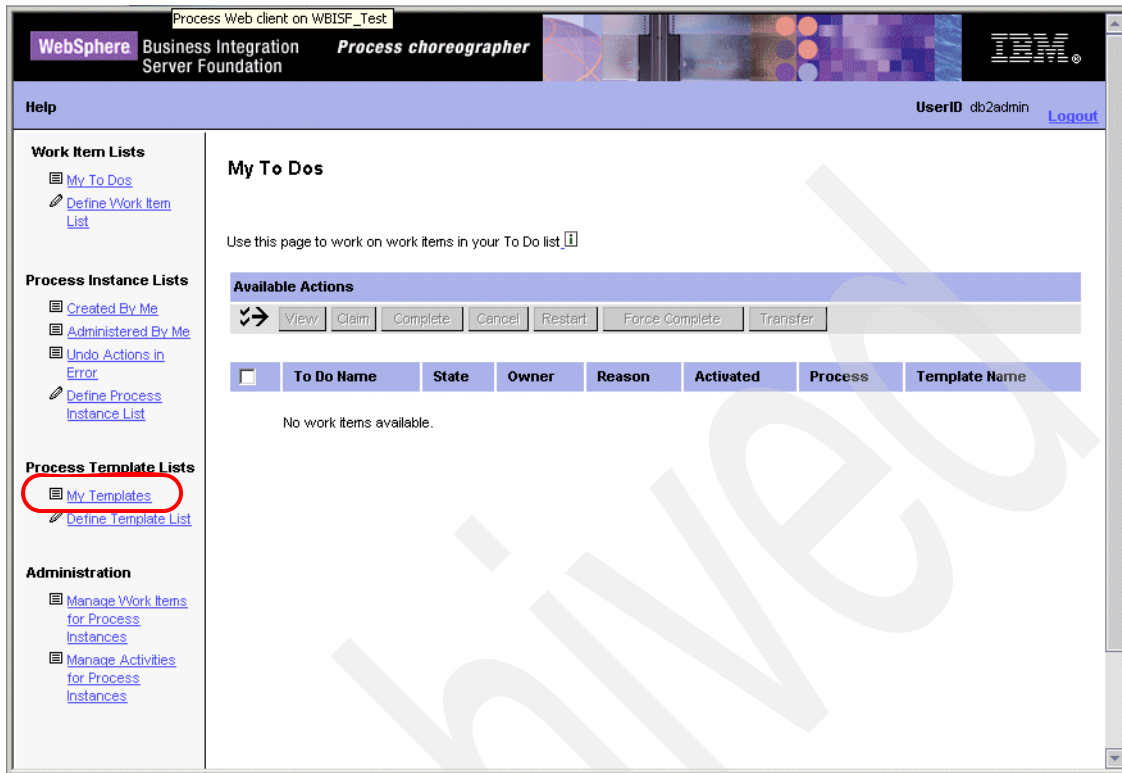


Figure 30-105 Home page of the business process Web client

4. Select the template MaintenanceProcess and click **Start Instance**.

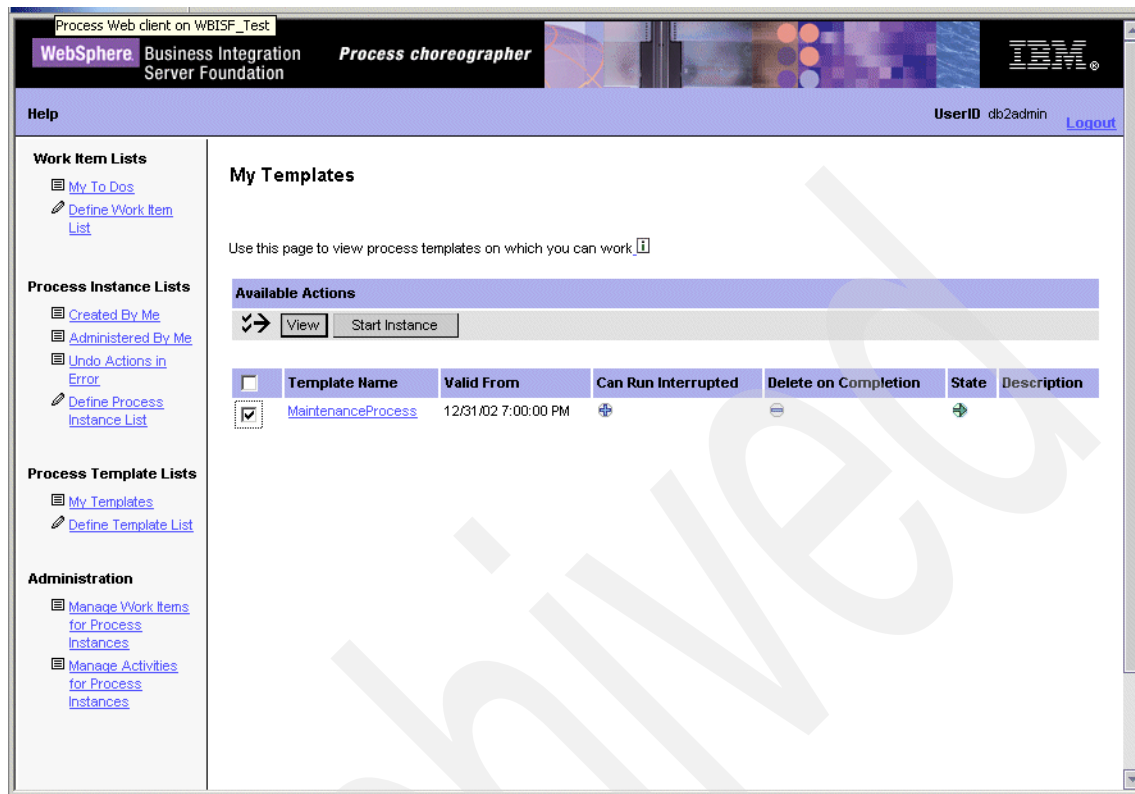


Figure 30-106 List of templates

5. Provide input parameters and click **Start Instance**.

WebSphere

Business Integration
Server Foundation

Process choreographer

IBM

Help

UserID db2admin [Logout](#)

Work Item Lists

[My To Dos](#)
[Define Work Item List](#)

Process Instance Lists

[Created By Me](#)
[Administered By Me](#)
[Undo Actions in Error](#)
[Define Process Instance List](#)

Process Template Lists

[My Templates](#)
[Define Template List](#)

Administration

[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Process Input Message

Use this page to change the input message before you complete the actions to start the business process [i](#)

Available Actions

Start Instance

Process Template Description

Documentation	-	Valid From	12/31/02 7:00:00 PM
Description	-	Delete on Completion	-
Template Name	MaintenanceProcess	Can Run Interrupted	+
Created	1/29/05 1:35:51 AM	Can Run Synchronously	-

Service

Name	Receive
Description	-
PortType	ProcessPortType
Operation	InputOperation

Process Instance Name

Process Instance Name

Figure 30-107 Provide input parameters to the process - part 1

[Undo Actions in Error](#)
[Define Process Instance List](#)
Process Template Lists
[My Templates](#)
[Define Template List](#)
Administration
[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Process Template Description

Documentation -

Valid From 12/31/02 7:00:00 PM

Delete on Completion

Can Run Interrupted

Can Run Synchronously

Description -

Template Name MaintenanceProcess

Created 1/29/05 1:35:51 AM

Service

Name Receive

Description -

PortType ProcessPortType

Operation InputOperation

Process Instance Name

Process Instance Name

Process Input Message

contents.actualCompletion (string)

contents.apartment 1 (int)

contents.createDate Today (string)

contents.description broken window (string)

contents.expectedCompletion Tomorrow (string)

contents.maintenanceId 2 (int)

contents.statusDescription (string)

contents.tenant 100 (int)

Figure 30-108 Provide input parameters to the process - part 2

- When the process is started, click **My To Dos**.

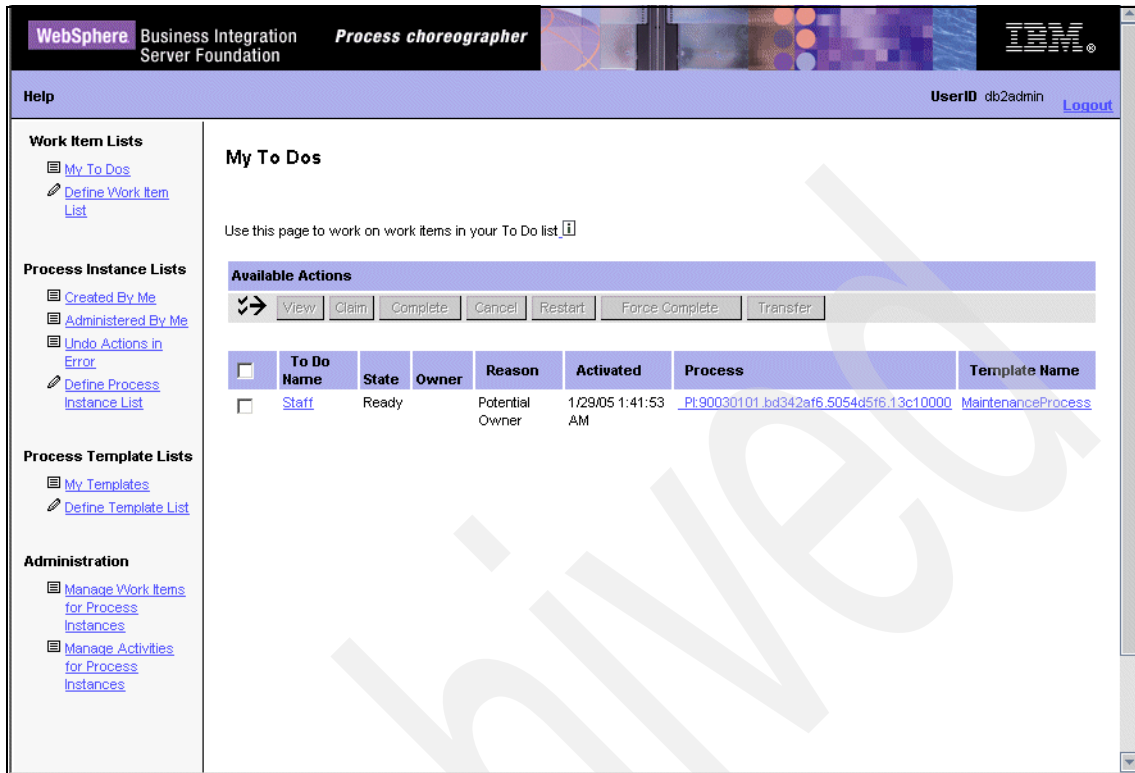


Figure 30-109 Activity in to do list

7. Select the activity and click **Claim** to claim the activity.
8. Select the claimed activity and click **View**.



Figure 30-110 Work with claimed activity

9. Provide values for the output message and click **Complete**.

Created By Me

Administered By Me

Undo Actions in Error

Define Process Instance List

Process Template Lists

My Templates

Define Template List

Administration

Manage Work Items for Process Instances

Manage Activities for Process Instances

Complete Save Changes Cancel

Process Context

Activity Name	Staff	Description	-
Template Name	MaintenanceProcess	Process Instance Name	_Pt:90030101.cc24c98f.5054d5f6.d8bc0000
State	Claimed	Administrators	db2admin
Reason	-		

[View more details about this activity.](#)
[View more details about this process.](#)

Activity Input Message

contents.actualCompletion	1
contents.apartment	Today
contents.createDate	broken window
contents.description	Tomorrow
contents.expectedCompletion	2
contents.maintenanceld	
contents.statusDescription	
contents.tenant	100

Activity Output Message

reviewResponsePart.actualCompletion	<input type="text"/>	(string)
reviewResponsePart.apartment	<input type="text" value="1"/>	(int)
reviewResponsePart.createDate	<input type="text" value="Today"/>	(string)
reviewResponsePart.description	<input type="text" value="broken window"/>	(string)
reviewResponsePart.expectedCompletion	<input type="text" value="Tomorrow"/>	(string)
reviewResponsePart.from	<input type="text" value="redmaint"/>	(string)
reviewResponsePart.maintenanceld	<input type="text" value="2"/>	(int)
reviewResponsePart.statusDescription	<input type="text" value="sent to inhouse technician"/>	(string)
reviewResponsePart.tenant	<input type="text" value="100"/>	(int)
reviewResponsePart.to	<input type="text" value="inhouse"/>	(string)

Figure 30-111 Provide output message for activity

10. Click **Manage Work Items for Process Instances**.

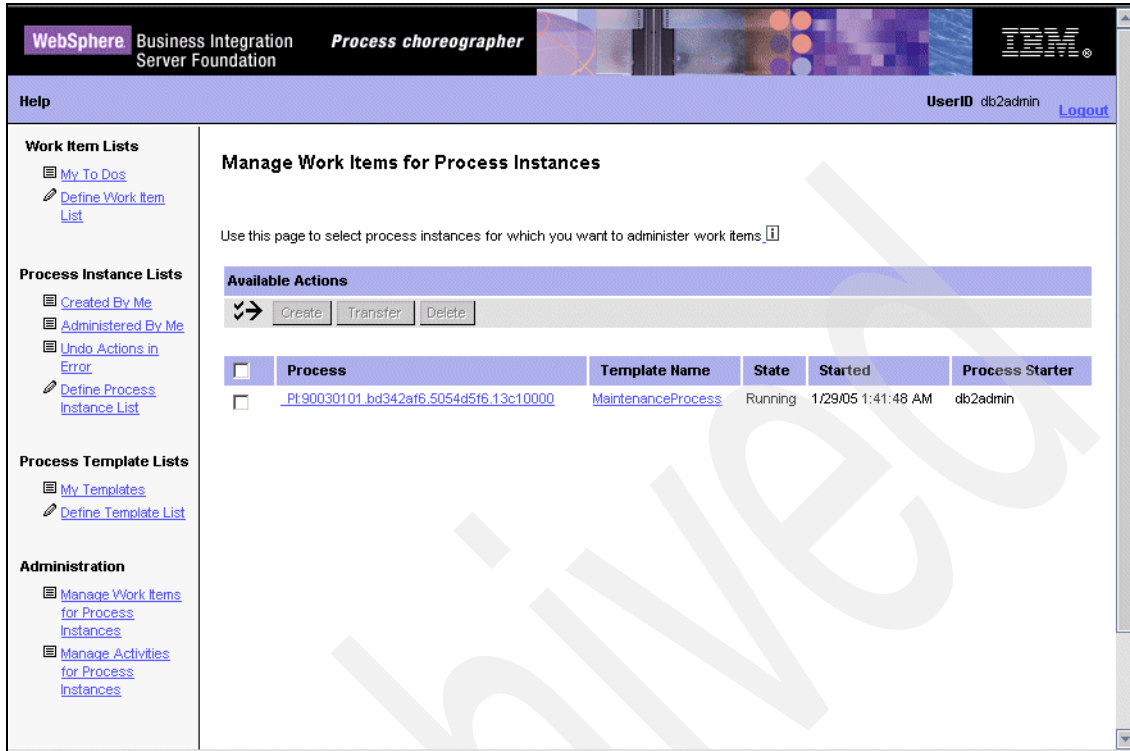


Figure 30-112 List of work items for process instances

11. Click the instance. The process is running and ready to execute activity Staff1.

[List](#)

Process Instance Lists

- Created By Me
- Administered By Me
- Undo Actions in Error
- Define Process Instance List

Process Template Lists

- My Templates
- Define Template List

Administration

- Manage Work Items for Process Instances
- Manage Activities for Process Instances

Use this page to display information about the process and, optionally, to act on the process

Available Actions

Compensate
Terminate
Delete
Monitor
Repair Compensation

Process Description

Process Instance Name
_Template Name
Description
Starter
Readers
Administrators

_Pt:90030101.bd342af6.5054d5f6.13c10000
MaintenanceProcess
db2admin
db2admin
db2admin

State
Running
Started
1/29/05 1:41:48 AM

Process Input Message

contents.actualCompletion
contents.apartment
contents.createDate
contents.description
contents.expectedCompletion
contents.maintenanceId
contents.statusDescription
contents.tenant

1
Today
broken window
Tomorrow
2
100

My To Dos

Name	State	Owner	Activated	Completed
Staff	Finished	db2admin	1/29/05 1:41:53 AM	1/29/05 1:50:18 AM
Staff1	Ready		1/29/05 1:50:20 AM	

Figure 30-113 Details of the state of a process instance

- Click **Monitor** to verify the status of activities. You should see that the activities Staff and InvokeInhouseSend are finished and that the activity Staff1 is ready.
- Return to My To Dos. The process should be ready for the second Staff activity.
- Claim this activity. Provide output values and click **Complete**.

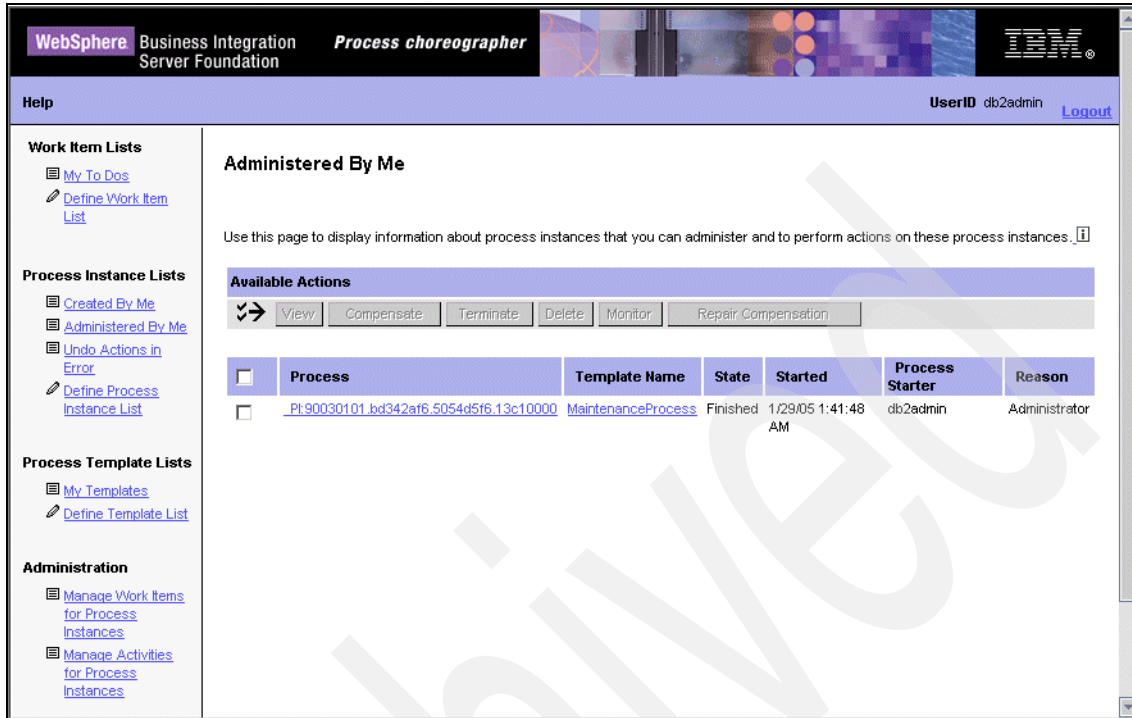


Figure 30-115 List of process instances

16. Select the instance and click **View** to look at the details.

Process Description

Process Instance Name	_PI:90030101.bd342af6.5054d5f6.13c10000	State	Finished
Template Name	MaintenanceProcess	Started	1/29/05 1:41:48 AM
Description			
Starter	db2admin		
Readers			
Administrators	db2admin		

Process Input Message

contents.actualCompletion	1
contents.apartment	Today
contents.createDate	broken window
contents.description	Tomorrow
contents.expectedCompletion	2
contents.maintenanceId	100
contents.statusDescription	
contents.tenant	

Process Output Message

contents	window fixed
----------	--------------

My To Dos

Name	State	Owner	Activated	Completed
Staff	Finished	db2admin	1/29/05 1:41:53 AM	1/29/05 1:50:18 AM
Staff1	Finished	db2admin	1/29/05 1:50:20 AM	1/29/05 2:00:21 AM

Figure 30-116 Details of a finished process

You have successfully validated the current process model.

30.6 Extending the business process

Now that you have validated the basic flow, you can add invocations of the RMConnector. To extend the business process:

1. Add a new invoke activity and name it InvokeAdapter_InProgress.
2. While this new activity is selected, select the Implementation tab in the lower pane.
3. Select the partner link type for the adapter.
4. Select the operation RM_MaintenanceUpdate.
5. Create a new request variable AdapterUpdateRequest.

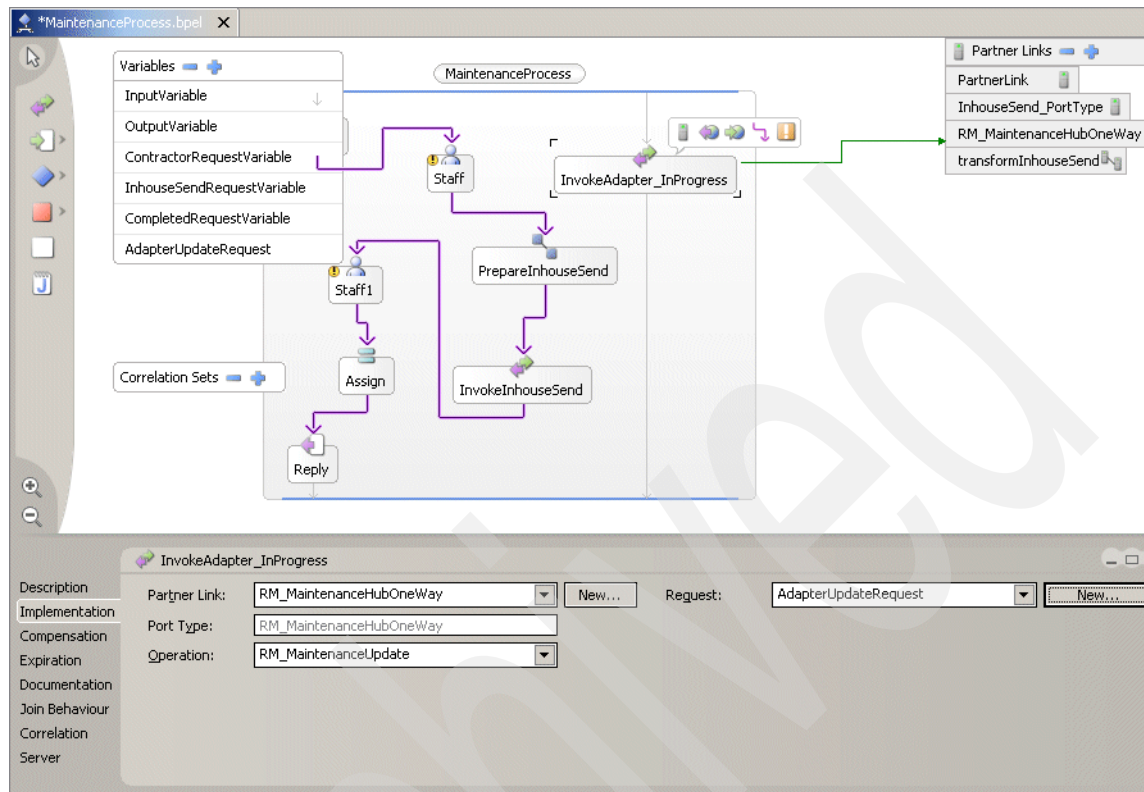
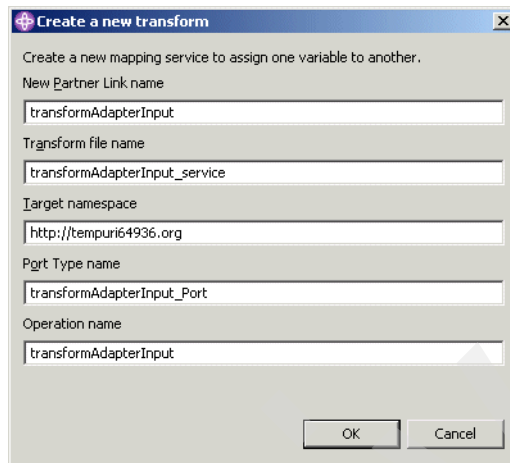


Figure 30-117 Add adapter service

6. The input to the adapter has a different structure than the process input or the staff output message. Therefore, we need to do some transformation using a transformer service. Add a transform activity and name this new activity PrepareAdapterInput.

8. Now that the variables are set, click **New** to create the transformation.
9. Provide suitable names for the partner link, service, port, and operation, and click **OK**.



Create a new transform

Create a new mapping service to assign one variable to another.

New Partner Link name
transformAdapterInput

Transform file name
transformAdapterInput_service

Target namespace
http://tempuri64936.org

Port Type name
transformAdapterInput_Port

Operation name
transformAdapterInput

OK Cancel

Figure 30-120 Create new transformer

10. The transformer editor opens. Drag-and-drop the corresponding elements from output to input to generate XPath statements that are executed at run-time (Figure 30-121 on page 744).
11. For the element `ObjectEventId`, use the input element `CreateDate`.
12. For the element `Status` in output, there is no corresponding element in the input. Assign it the correct value. Drag any element to the status field, so that it appears in the Overview pane. Click in the XPath cell for the element status. A little grey box appears in the cell. Click this grey box to open more advanced options for XPath.

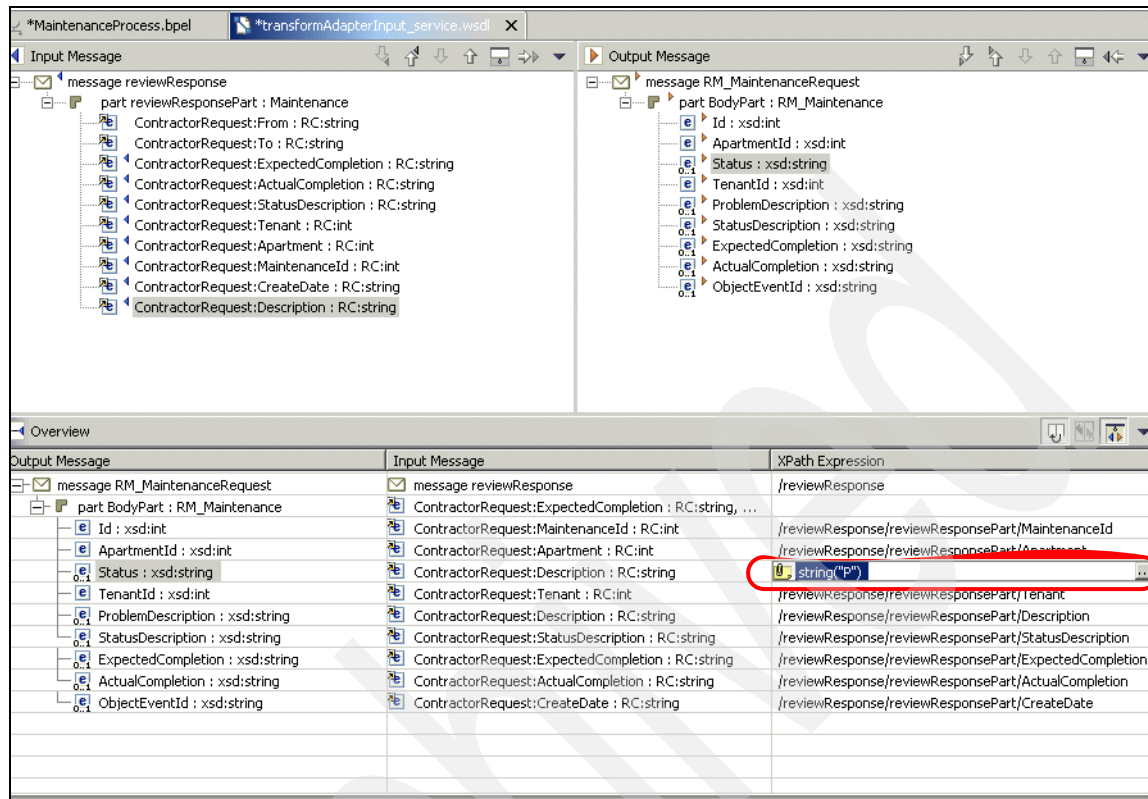


Figure 30-121 Mapping statements in the transformer service

13. After clicking the grey box, Figure 30-122 on page 745 appears. Select **Advanced** and click **Next**.

Transformation Details

Specify Transformation Details
Specify how to map between input and output messages.

☐ **Move**
Copy the value of the input message to the output message.

☐ **Extract**
Extract a part of the message value.
 Delimiter:
 Index:
 Example:

☐ **Join**
Combine inputs by appending a delimiter after them.
 Prefix:
 Mapping Inputs:

Inputs	Delimiter
/reviewResponse/reviewRequestPart/D...	

Example:
 /reviewResponse/reviewRequestPart/Description

☒ **Advanced**
Create an XPath Expression

< Back Next > Finish Cancel

Figure 30-122 Details of a mapping statement

14. Build XPath commands by select the `string()` function and add P as the argument. Click **Finish**.

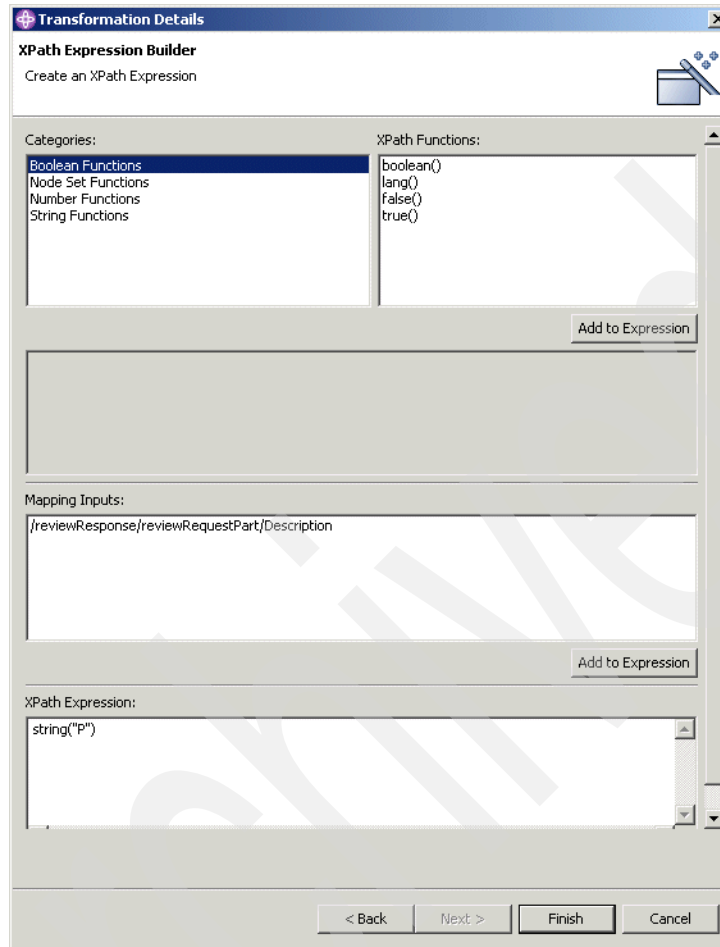


Figure 30-123 XPath expression builder

15. The elements `actualCompletion` and `problemDescription` should have a fixed value of `CxIgnore`. The actual completion is not yet known at this stage of the process flow, and the problem description should not be changed during the execution of the flow. Using the same technique that was demonstrated for the field `Status`, set the value of the elements `actualCompletion` and `problemDescription` to `CxIgnore`, as shown in Figure 30-124 on page 747.

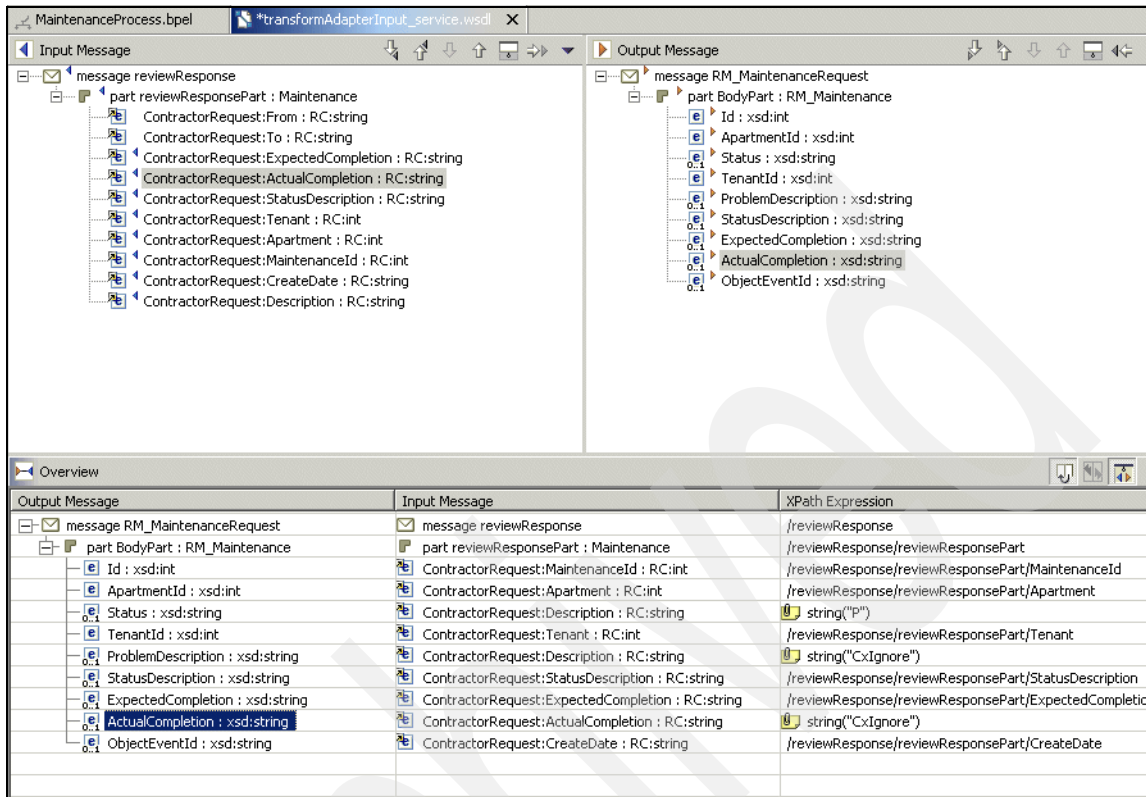


Figure 30-124 Completed mapping statements for the transformer

16. The transformer is completed. Using the same instructions, add another adapter invocation.

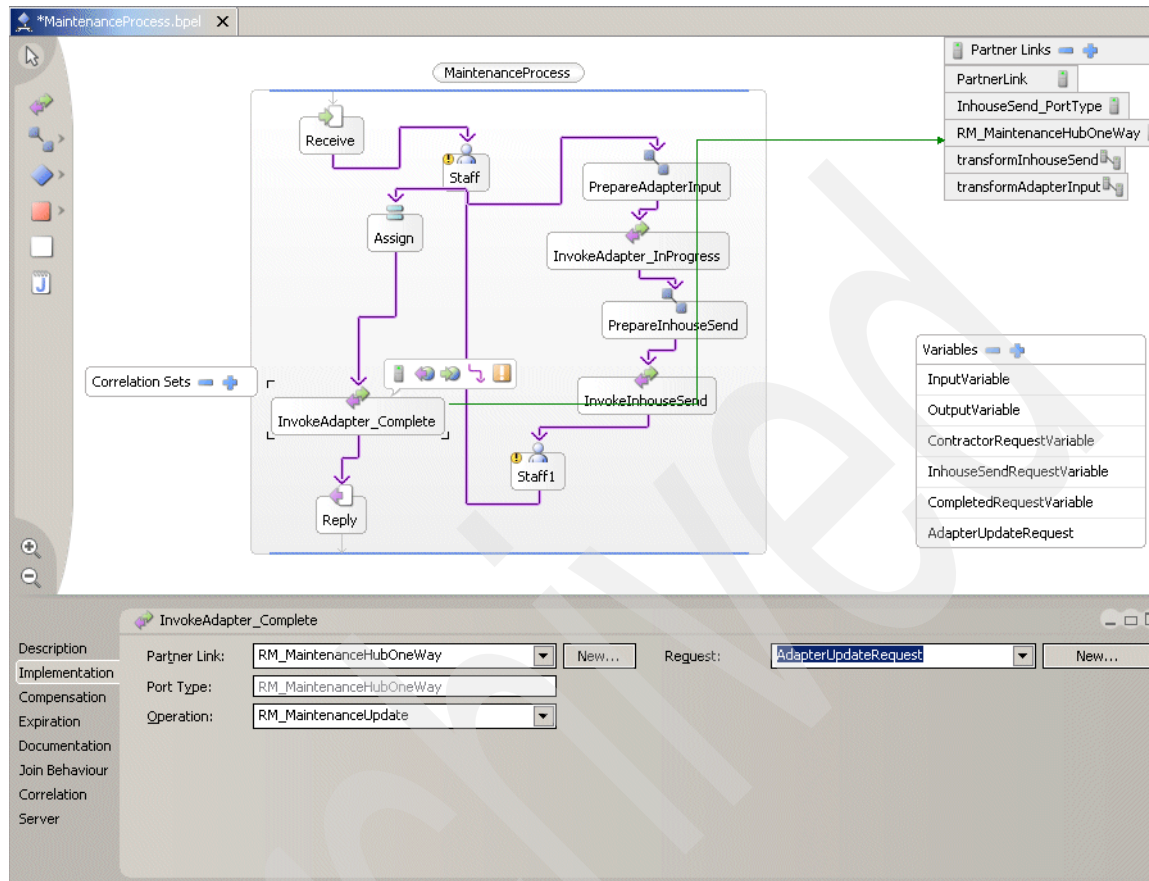


Figure 30-125 Add a second adapter invocation

17. Add a second transformer service. Use the existing variables `CompletedRequestVariable` and `AdapterInputVariable` (Figure 30-126 on page 749).

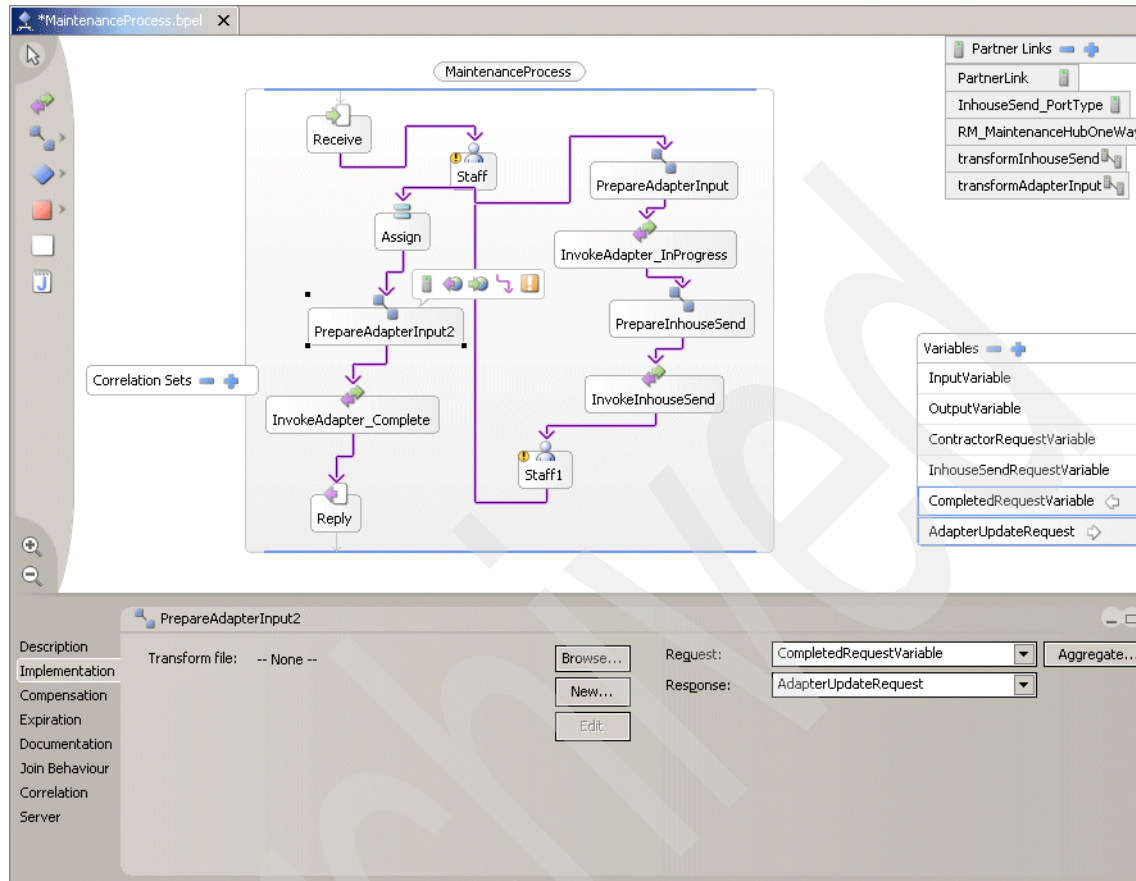
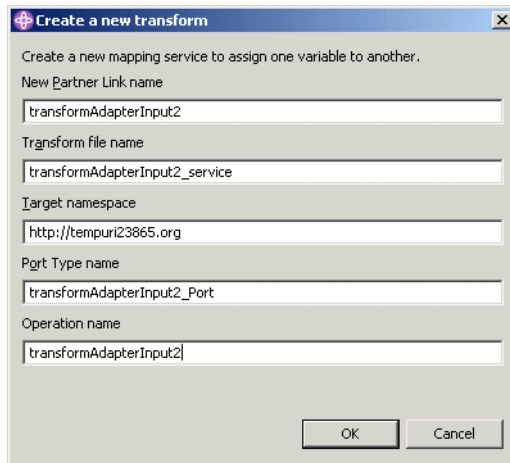


Figure 30-126 Add a second transformer

18. When the variables are defined, click **New** to create the second transformer service.
19. Provide appropriate names for the service, partner link, and operation.



The image shows a dialog box titled "Create a new transform" with a close button (X) in the top right corner. The dialog contains the following fields and text:

- Text: "Create a new mapping service to assign one variable to another."
- Field: "New Partner Link name" with the value "transformAdapterInput2".
- Field: "Transform file name" with the value "transformAdapterInput2_service".
- Field: "Target namespace" with the value "http://tempuri23865.org".
- Field: "Port Type name" with the value "transformAdapterInput2_Port".
- Field: "Operation name" with the value "transformAdapterInput2".
- Buttons: "OK" and "Cancel" at the bottom right.

Figure 30-127 Create new transformer

20. Do the same kind of mappings as before. For the status field, use an XPath statement that sets the status to C, meaning Completed. For the problem description, set it to CxIgnore.

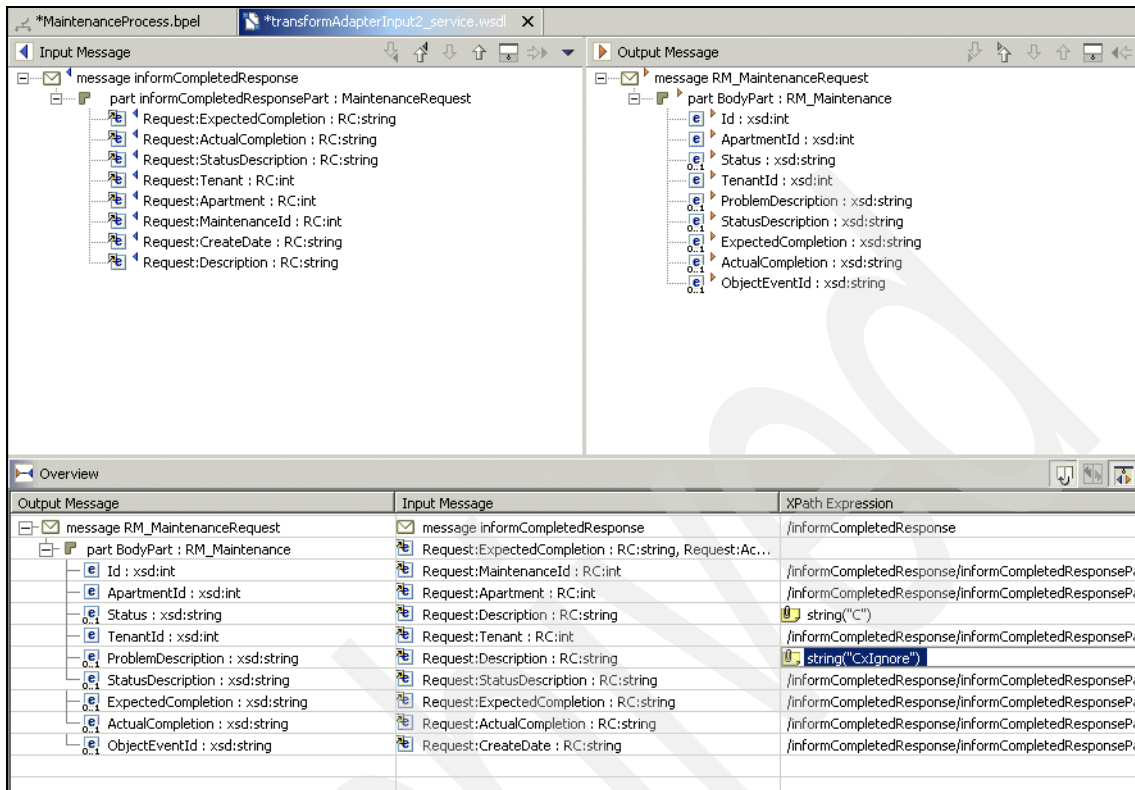


Figure 30-128 Mapping statements for the second transformer

21. The process model is now complete. Close all editors and generate deploy code.
22. Restart the server.
23. Perform similar testing steps as explained in 30.5, "Testing the process flow" on page 728.

24. Start a new process instance, as shown in Figure 30-129.

The screenshot shows the 'Process Web client' window. The left sidebar contains navigation links under three categories: 'Process Instance Lists' (Created By Me, Administered By Me, Undo Actions in Error, Define Process Instance List), 'Process Template Lists' (My Templates, Define Template List), and 'Administration' (Manage Work Items for Process Instances, Manage Activities for Process Instances). The main content area is titled 'Available Actions' and features a 'Start Instance' button. Below this, the 'Process Template Description' section shows details for the 'MaintenanceProcess' template, including its documentation, description, template name, creation date, valid from date, and various flags like 'Delete on Completion', 'Can Run Interrupted', and 'Can Run Synchronously'. The 'Service' section lists the name, description, port type, and operation. The 'Process Instance Name' section has a text input field. The 'Process Input Message' section contains a table of input fields with their names, values, and data types.

Process Input Message		
contents.actualCompletion		(string)
contents.apartment	1	(int)
contents.createDate	1110500000000	(string)
contents.description	leak in roof	(string)
contents.expectedCompletion	1110600000000	(string)
contents.maintenanceId	2	(int)
contents.statusDescription		(string)
contents.tenant	100	(int)

Figure 30-129 Provide input to the new process

25. After the process is started, return to My To Dos. Claim and complete the first staff activity, as shown in Figure 30-130.

The screenshot shows the 'Process Web client' window with the title 'Process Web client on WBISF_Test'. The left sidebar contains navigation links: 'Error', 'Define Process Instance List', 'Process Template Lists' (with sub-links 'My Templates' and 'Define Template List'), and 'Administration' (with sub-links 'Manage Work Items for Process Instances' and 'Manage Activities for Process Instances').

The main content area is titled 'Process Context' and displays the following information:

Activity Name	Template Name	State	Reason	Description	Process Instance Name	Administrators
Staff	MaintenanceProcess	Claimed	-	-	_PI:90030101_bd62750f.5054d5f6.ffe10000	db2admin

Below the process context, there are two links: 'View more details about this activity.' and 'View more details about this process.'.

The 'Activity Input Message' section shows the following data:

Property	Value
contents.actualCompletion	1
contents.apartment	1110500000000
contents.createDate	leak in roof
contents.description	1110600000000
contents.expectedCompletion	2
contents.maintenanceId	100
contents.statusDescription	
contents.tenant	

The 'Activity Output Message' section shows a list of properties with input fields and data types:

Property	Value	Type
reviewResponsePart.actualCompletion		(string)
reviewResponsePart.apartment	1	(int)
reviewResponsePart.createDate	1110500000000	(string)
reviewResponsePart.description	leak in roof	(string)
reviewResponsePart.expectedCompletion	1110600000000	(string)
reviewResponsePart.from	redmairnt	(string)
reviewResponsePart.maintenanceId	2	(int)
reviewResponsePart.statusDescription	sent to inhouse technician	(string)
reviewResponsePart.tenant	100	(int)
reviewResponsePart.to	inhouse	(string)

Figure 30-130 Complete the first staff activity

26.If the broker (or message flow) is not running, you should be able to catch the adapter update message, as shown in Figure 30-131. Note the P value for Status.

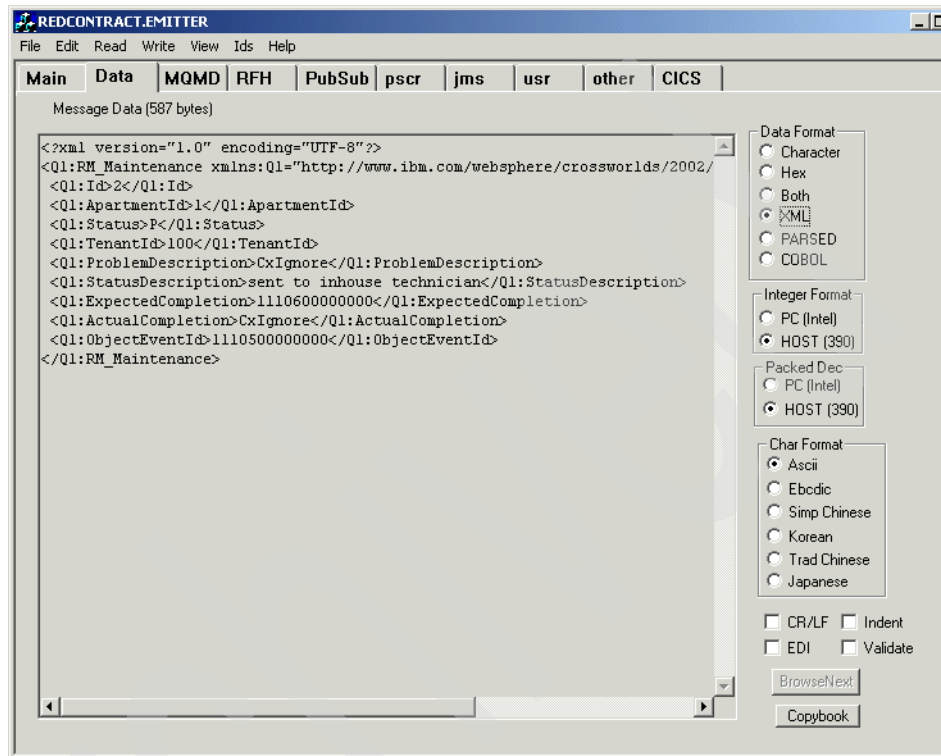


Figure 30-131 Adapter in-progress message

27. Return to My To Dos. You now see the second staff activity. Claim and complete it, as shown in Figure 30-132.

The screenshot shows the 'Process Web client' window with the 'Complete Actions' dialog open. The dialog has a left sidebar with navigation links and a main content area with several sections.

Process Context

Activity Name	Template Name	Description	Process Instance Name
Staff1	MaintenanceProcess	-	_Pt90030101_b062750f.5054d5f6.ffe10000
State	Claimed	Administrators	db2admin
Reason	-		

[View more details about this activity.](#)
[View more details about this process.](#)

Activity Input Message

contents.actualCompletion	1
contents.apartment	1110500000000
contents.createDate	leak in roof
contents.description	1110600000000
contents.expectedCompletion	2
contents.maintenanceId	
contents.statusDescription	
contents.tenant	100

Activity Output Message

informCompletedResponsePart.actualCompletion	1110600000000	(string)
informCompletedResponsePart.apartment	1	(int)
informCompletedResponsePart.createDate	1110500000000	(string)
informCompletedResponsePart.description	leak in roof	(string)
informCompletedResponsePart.expectedCompletion	1110600000000	(string)
informCompletedResponsePart.maintenanceId	2	(int)
informCompletedResponsePart.statusDescription	work completed	(string)
informCompletedResponsePart.tenant	100	(int)

Figure 30-132 Complete the second staff activity

28.If the broker (or message flow) is not running, you should be able to catch the adapter update message, as shown in Figure 30-133. Note the C value for Status.

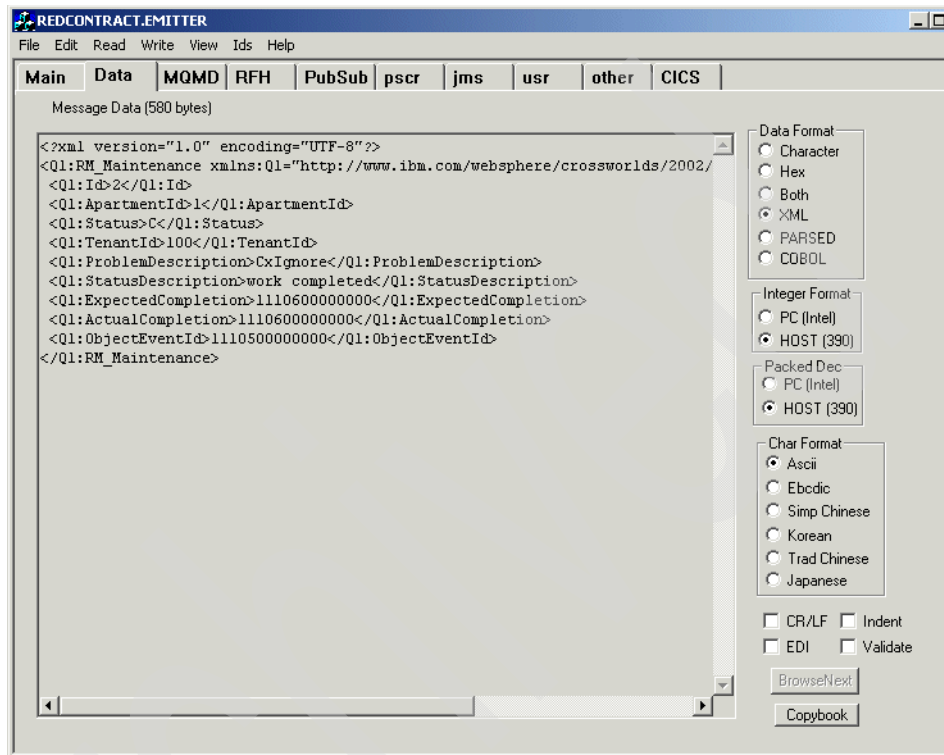


Figure 30-133 Adapter completed message

29.Return to the Process Web Client. Locate the instance by selecting **Process Instance Lists** → **Administered By Me**.

30. Click the process instance link to review the details, as shown in Figure 30-134. Notice the output of the process and the status.

Work Item Lists

[My To Dos](#)
[Define Work Item List](#)

Process Instance Lists

[Created By Me](#)
[Administered By Me](#)
[Undo Actions in Error](#)
[Define Process Instance List](#)

Process Template Lists

[My Templates](#)
[Define Template List](#)

Administration

[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Process Instance

Use this page to display information about the process and, optionally, to act on the process ¹

Available Actions

[Compensate](#)
[Terminate](#)
[Delete](#)
[Monitor](#)
[Repair Compensation](#)

Process Description

Process Instance Name

_PI:90030101.bd62750f.5054d5f6.f1e10000

State

Finished

Template Name

MaintenanceProcess

Started

1/29/05 2:32:21 AM

Description

Starter

db2admin

Readers

Administrators

db2admin

Process Input Message

contents.actualCompletion

1

contents.apartment

1110500000000

contents.createDate

leak in roof

contents.description

1110600000000

contents.expectedCompletion

2

contents.maintenanceId

100

contents.statusDescription

contents.tenant

Process Output Message

contents

work completed

My To Dos

Name	State	Owner	Activated	Completed
Staff	Finished	db2admin	1/29/05 2:32:25 AM	1/29/05 2:33:37 AM
Staff1	Finished	db2admin	1/29/05 2:33:43 AM	1/29/05 2:34:36 AM

Figure 30-134 Details of a finished process

31. Click Monitor that is shown in Figure 30-134 on page 757 to review the details about each activity as shown in Figure 30-135.

Process Web client X

Work Item Lists

- [My To Dos](#)
- [Define Work Item List](#)

Process Instance Lists

- [Created By Me](#)
- [Administered By Me](#)
- [Undo Actions in Error](#)
- [Define Process Instance List](#)

Process Template Lists

- [My Templates](#)
- [Define Template List](#)

Administration

- [Manage Work Items for Process Instances](#)
- [Manage Activities for Process Instances](#)

Process Instance Monitor

Use this page to view the status of the activities in a process instance [\[1\]](#)

Process Description

Process Instance Name	_Pt90030101.cfb62daf.5054d5f6.75b60000	State	Finished
Template Name	MaintenanceProcess	Started	2/1/05 3:56:58 PM
Description			
Starter	db2admin		
Readers			
Administrators	db2admin		

Activities

To Do Name	State	Activity Kind	Activated
Reply	Finished	Reply	2/1/05 3:59:22 PM
InvokeAdapter_Complete	Finished	Invoke	2/1/05 3:59:21 PM
PrepareAdapterInput2	Finished	Invoke	2/1/05 3:59:21 PM
Staff1	Finished	Staff	2/1/05 3:57:50 PM
InvokeInhouseSend	Finished	Invoke	2/1/05 3:57:50 PM
PrepareInhouseSend	Finished	Invoke	2/1/05 3:57:49 PM
InvokeAdapter_InProgress	Finished	Invoke	2/1/05 3:57:48 PM
PrepareAdapterInput	Finished	Invoke	2/1/05 3:57:45 PM
Staff	Finished	Staff	2/1/05 3:57:01 PM
Receive	Finished	Receive	2/1/05 3:56:59 PM

Figure 30-135 Process Instance Monitor for a completed process

You now have a fully working process model that includes the invocation of a JMS service, two invocations of an adapter service, two transformation services, and two staff activities.

Integrating and automating the process

This chapter provides instructions on integrating the WebSphere Business Integration Connect component in the process. It also describes how to automate the creation of a process instance. During our initial tests, we always use the Web client to create a process instance. However we want to achieve the creation of a process instance when the right message arrives on the right queue.

31.1 Developing the WebSphere Business Integration Connect send service

This section provides instructions on building a service that passes a message to WebSphere Business Integration Connect, so that it can be transmitted to RedContract. After testing this service, it is integrated in the process model. Based on the output of the first staff activity, the request is sent to the inhouse technician or to the contractor via WebSphere Business Integration Connect.

31.1.1 Define JNDI resources

WebSphere Business Integration Connect uses its own queue manager and two queues for receiving and sending documents. We first define these resources to WebSphere Application Server by following these steps:

1. Remove the Maintenance_ProcessesEAR project from the server.
2. Restart the server.
3. Open the Administrative Console.
4. Define the queue connection factory as follows:
 - Display name: partner_a.bcg.queue.manager
 - JNDI name: jms/partner_a.bcg.queue.manager
 - Queue Manager: partner_a.bcg.queue.manager
 - Bindings: Client
 - Channel Name: java.channel
 - Hostname: studentx
 - Port: 9999
5. Define the queue destination for sending as follows:
 - Display name: XML_OUT
 - JNDI name: jms/XML_OUT
 - Base queue name: XML_OUT
 - Expiry: UNLIMITED
6. Define the queue for receiving as follows:
 - Display name: XML_IN
 - JNDI name: jms/XML_IN
 - Base queue name: XML_IN
 - Expiry: UNLIMITED
7. Restart the server.

31.1.2 Building the send service

In the same way that we built the WSDL for the InhouseSend service, we build the WSDL to send a message to WebSphere Business Integration Connect by following these steps:

1. Right-click the project Maintenance_processes and select **New** → **Empty Service**.
2. Set the package name to wbic.send and the file name to WBICSend. Click **Finish**.

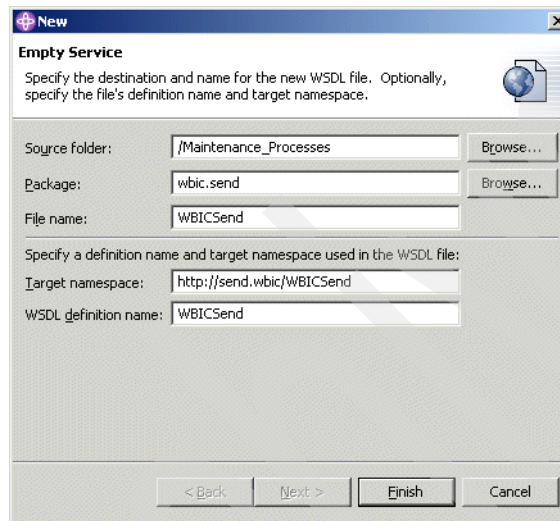


Figure 31-1 Create empty service

3. Add a port type WBICSend_Porttype.
4. Add an operation send to this port type.
5. Add an input and a message.
6. Set the type of the message to type Maintenance in the schema definition file ContractorRequest.xsd, as shown in Figure 31-2 on page 762.

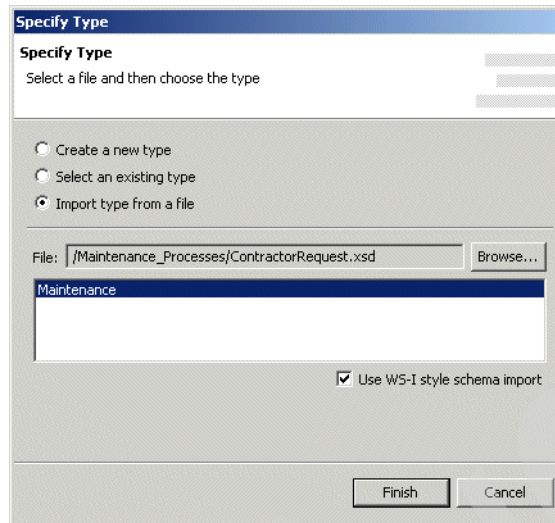


Figure 31-2 Set type of the input message

Figure 31-3 on page 763 shows the interface portion of the WSDL.

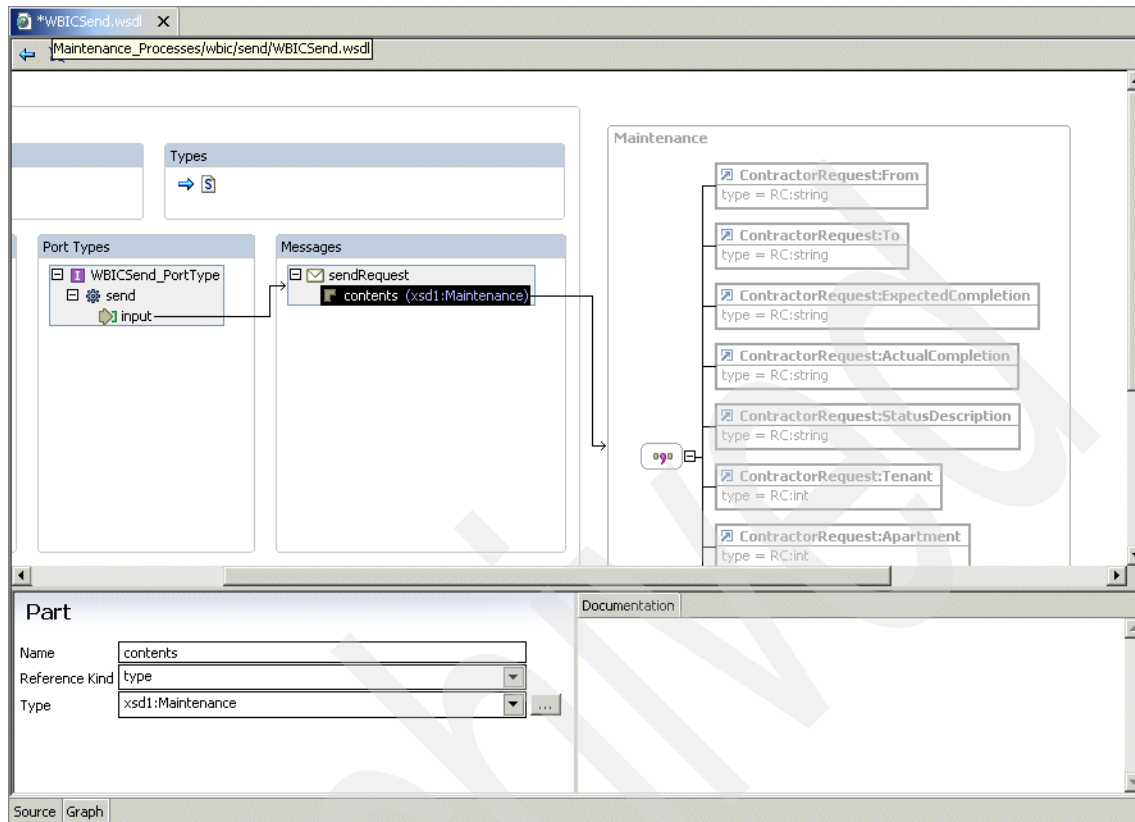


Figure 31-3 Interface portion of the WSDL

7. Save this WSDL file.
8. Similar to what was done for the InhouseSend service, create a new WSDL file, `WBICSendService.wsdl`.

9. Add import statements for the schema definition and for the abstract WSDL file, as shown in Figure 31-4.

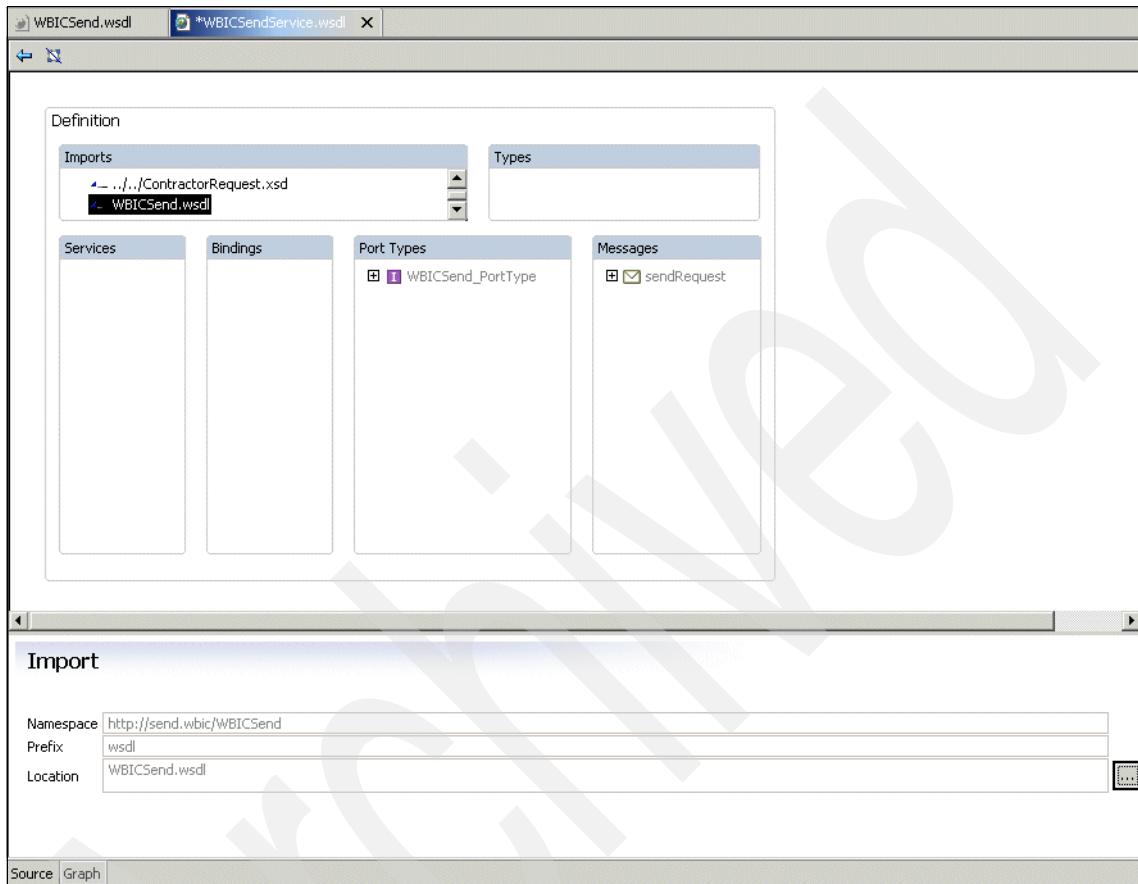


Figure 31-4 Import the interface and schema

10. Add a binding, `WBICSend_Binding`. Set the protocol to JMS and the message type to `TextMessage`.

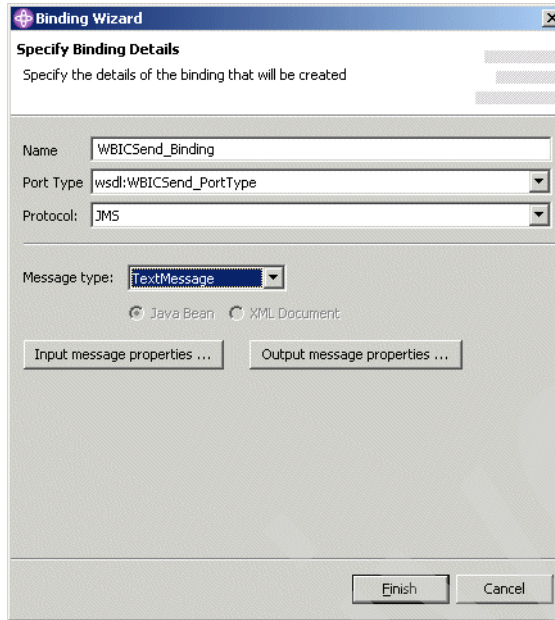


Figure 31-5 Create binding in the WSDL file

11. Add a service and a port to the WSDL. Set the protocol to JMS and set the JNDI names to the names that we created before.

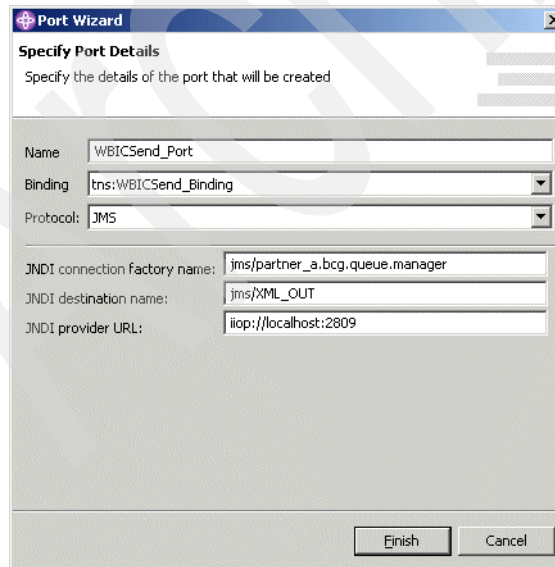


Figure 31-6 Create a port in the WSDL

Figure 31-7 shows the completed WSDL for the WBICSend service.

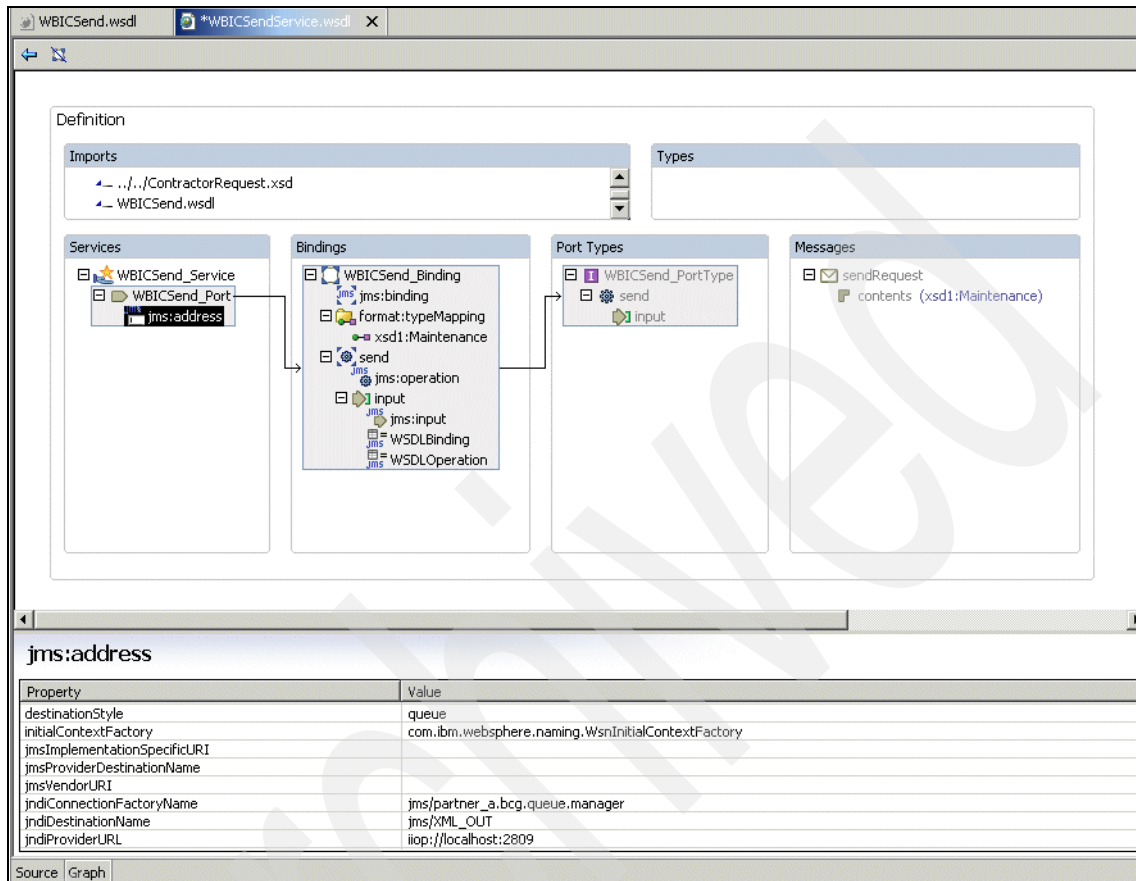


Figure 31-7 Completed WSDL for the WebSphere Business Integration Connect service

12. Right-click the WSDL file and select **Enterprise Services** → **Generate Deploy Code**.

13. Generate deploy code with EJB as the inbound protocol.

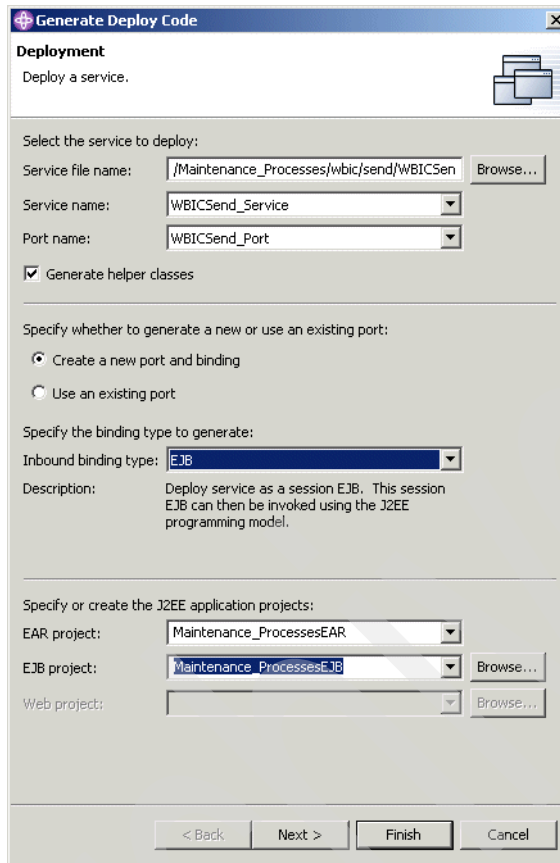


Figure 31-8 Generate deploy code for the WBICSend service

14. Locate the new EJB and open the EJB deployment descriptor editor. Select the tab references and correct the resource environment reference and the resource reference. The WebSphere JNDI names are:
 - jms/partner_a.bcg.queue.manager
 - jms/XML_OUT
15. Save and close the EJB deployment descriptor editor.
16. Add the project to the server.
17. Restart the server.
18. Use the universal test client to verify the WBICSend service.

Note: Make sure that the WebSphere Business Integration Connect servers are not running at this time.

19. Provide input to the send method, as shown in Figure 31-9.

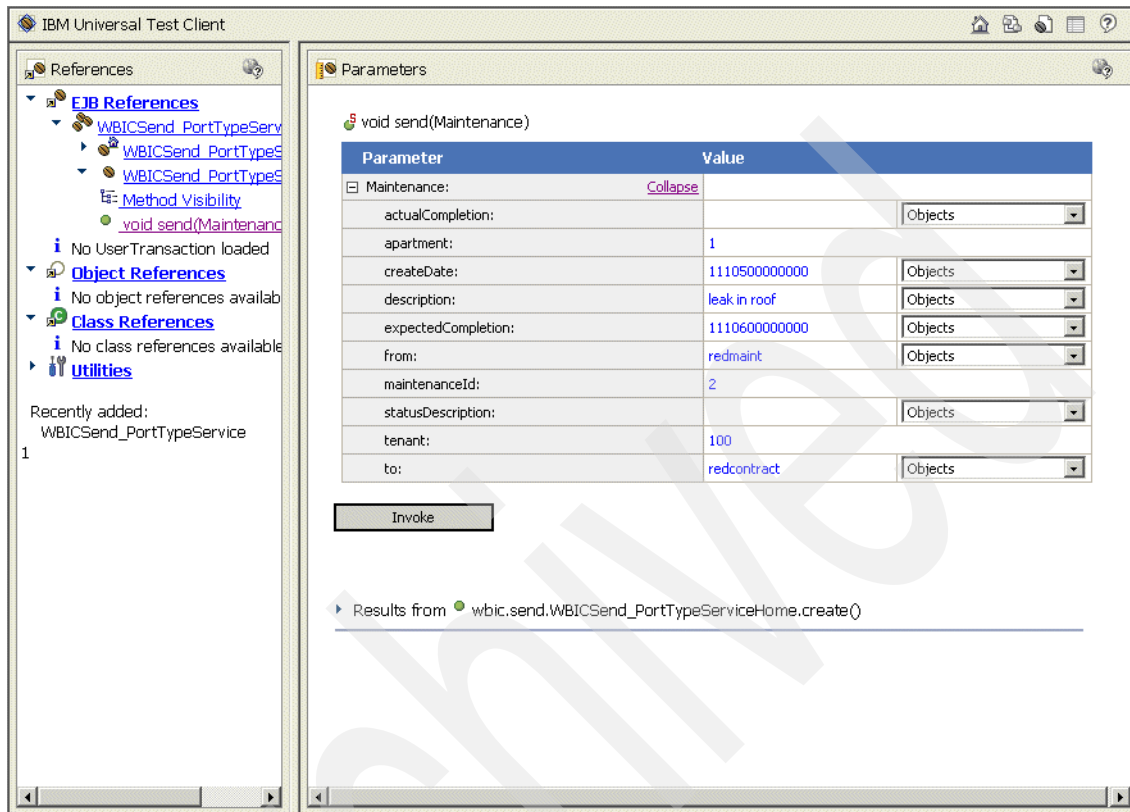


Figure 31-9 Using the Test Client to test the WBICSend service

20. Because WebSphere Business Integration Connect is not running, you should be able to browse the message on the queue using **rfhuti1**. Review the contents of the message (Figure 31-10).

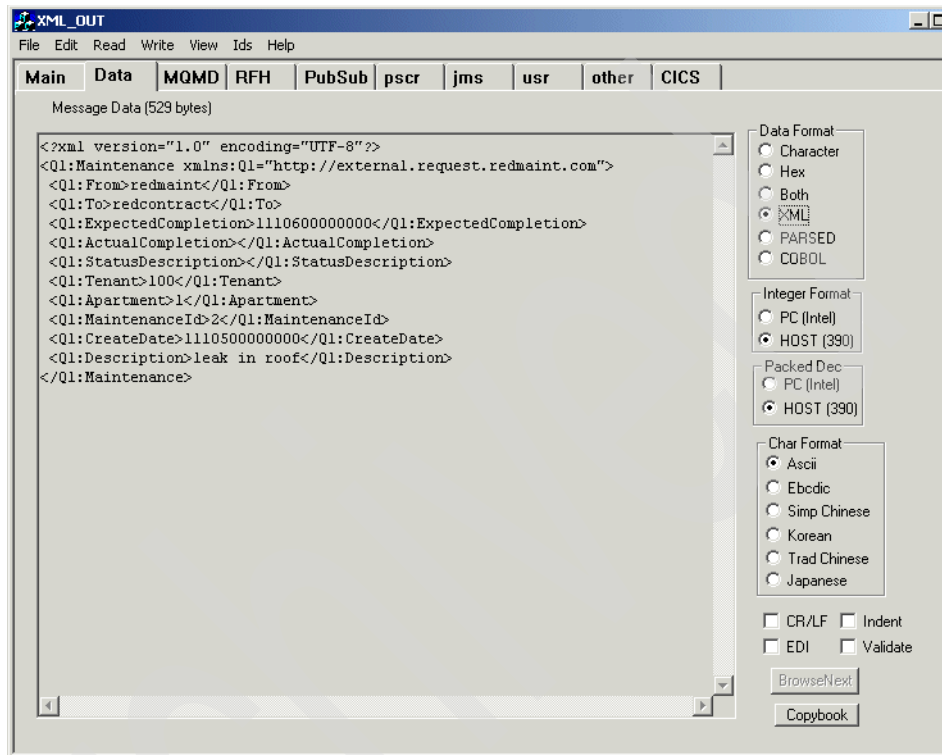


Figure 31-10 Message on the XML_OUT queue for WebSphere BI Connect

21. Start the WebSphere Business Integration Connect servers and also the WebSphere Business Integration Connect Express server.
22. Verify that the document is sent correctly and arrives at the RedContractor company. You can verify for example the contents of the folder Received of WebSphere Business Integration Connect Express. Or you can also use the viewers, such as the AS2 Viewer to validate the transmission.

Figure 31-11 shows the successful AS2 transaction.

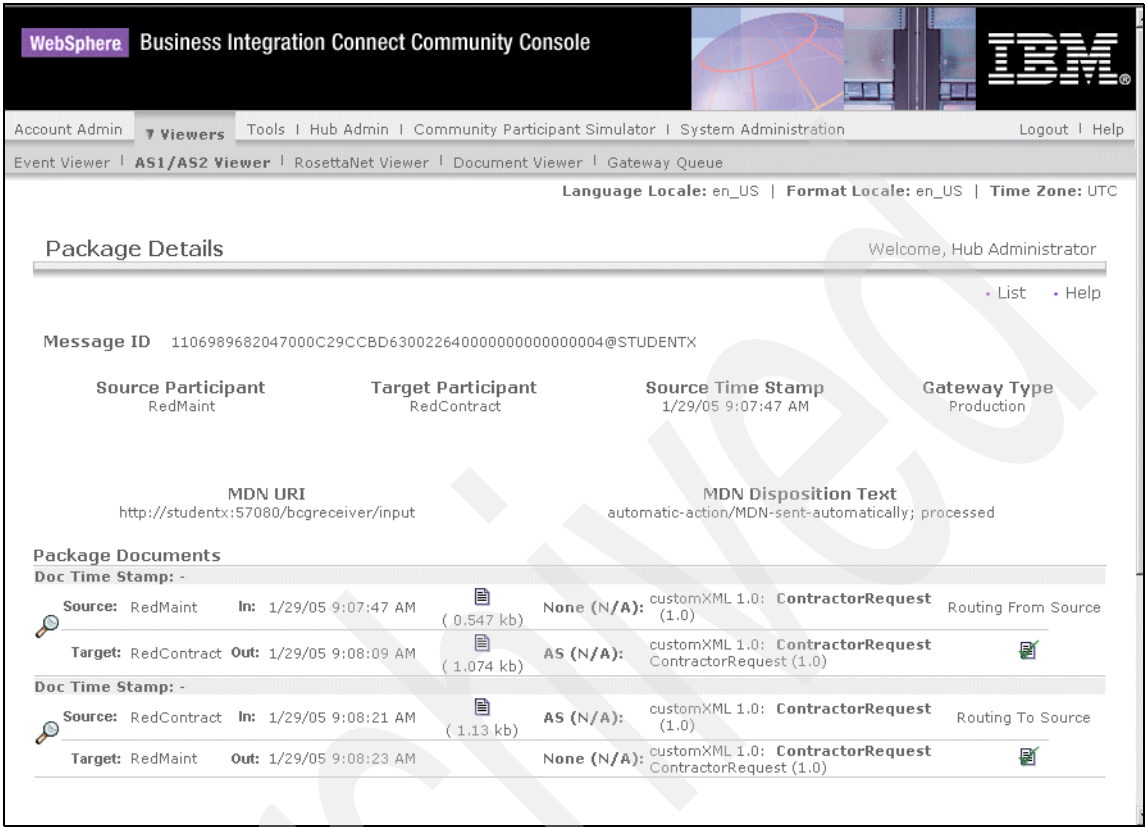


Figure 31-11 AS2 viewer

Figure 31-12 shows the actual document including the AS2 and HTTP headers.

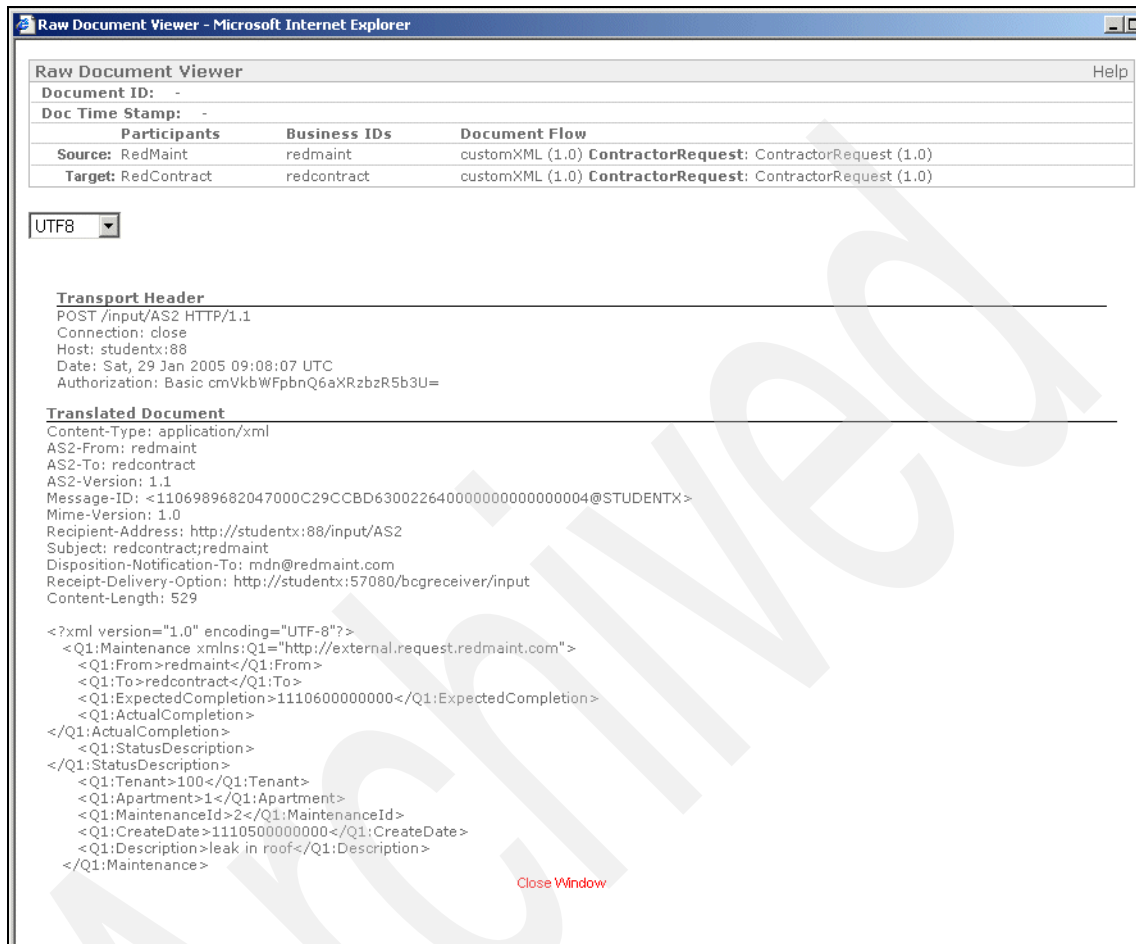


Figure 31-12 Raw document viewer

31.1.3 Adding the send service to the process model

Now that you have validated the WBICSend service, you can add it to the process model by following these steps:

1. Copy the WSDL files for the WBICSend service to the maintenance.processes package.
2. Open the process MaintenanceProcess in the BPEL editor.
3. Drag the WSDL WBICSendService.wsdl to the canvas.
4. Add a new invoke activity to the process and call it InvokeWBICSend.

5. On the Implementation tab, select the correct partner link.
6. Set the input variable to the new variable WBICSendRequestVariable

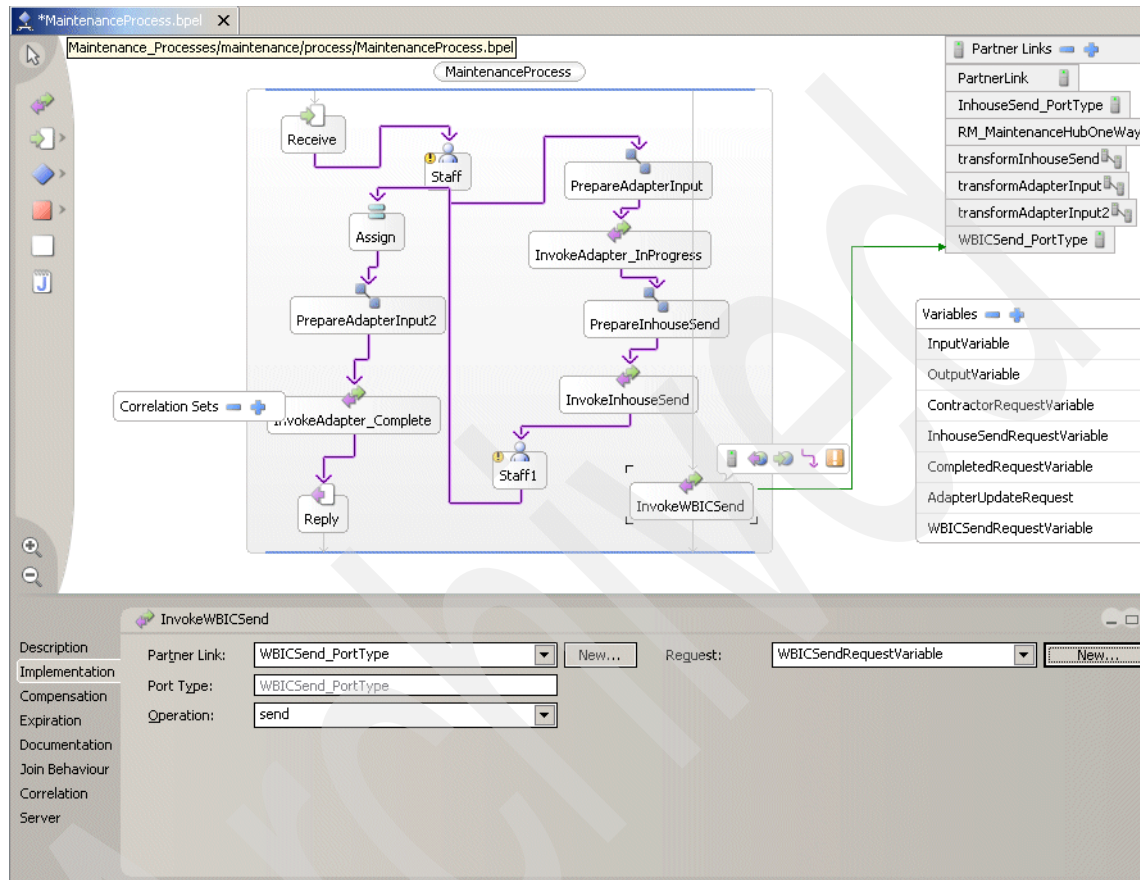


Figure 31-13 WBICSend service added to the process

7. Add a new Assign activity between the InvokeWBICSend and the Reply activities.
8. In the From section, select Fixed Value and type some informative text in the edit field.
9. In the To section, select Variable or Part. Select the contents of the process output variable.
10. Connect the invoke activity to the Assign activity and the Assign activity to the Reply node.

Note: When the document has been sent to WebSphere Business Integration Connect, we finish the process. When RedContract responds via WebSphere Business Integration Connect, we trigger a new process (developed in 31.3, “Building the receive process” on page 821) which invokes the adapter to indicate completion of the maintenance request.

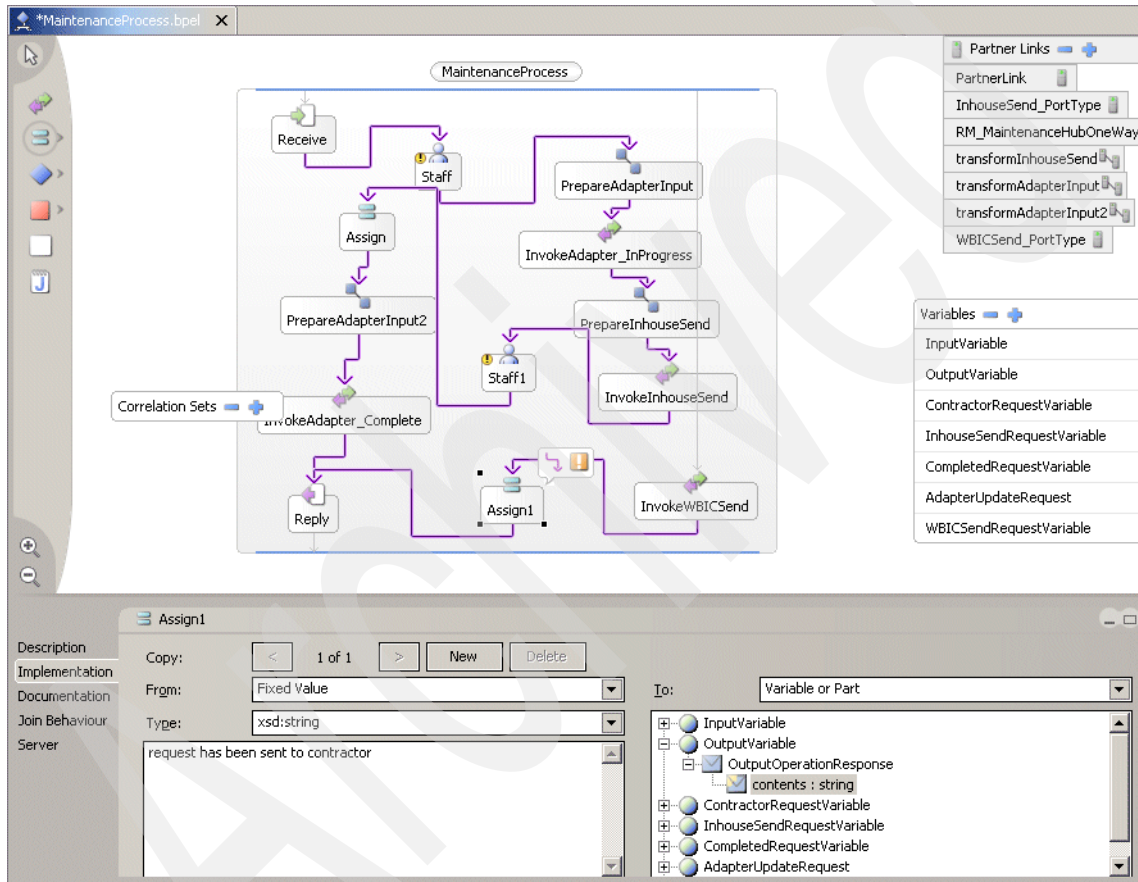


Figure 31-14 Assign activity added

11. You need a transform activity to prepare the variable that is used in the activity InvokeWBICSend. Call this new transformer PrepareWBICSend.
12. Add an extra connection from the InvokeAdapter_Progress to the PrepareWBICSend activity.

13. Set the input variable to ContractorRequestVariable and the output variable to WBICSendRequestVariable. Click **New** to create the transformation.

Note: We will soon add conditions to the two links leaving InvokeAdapter_Progress.

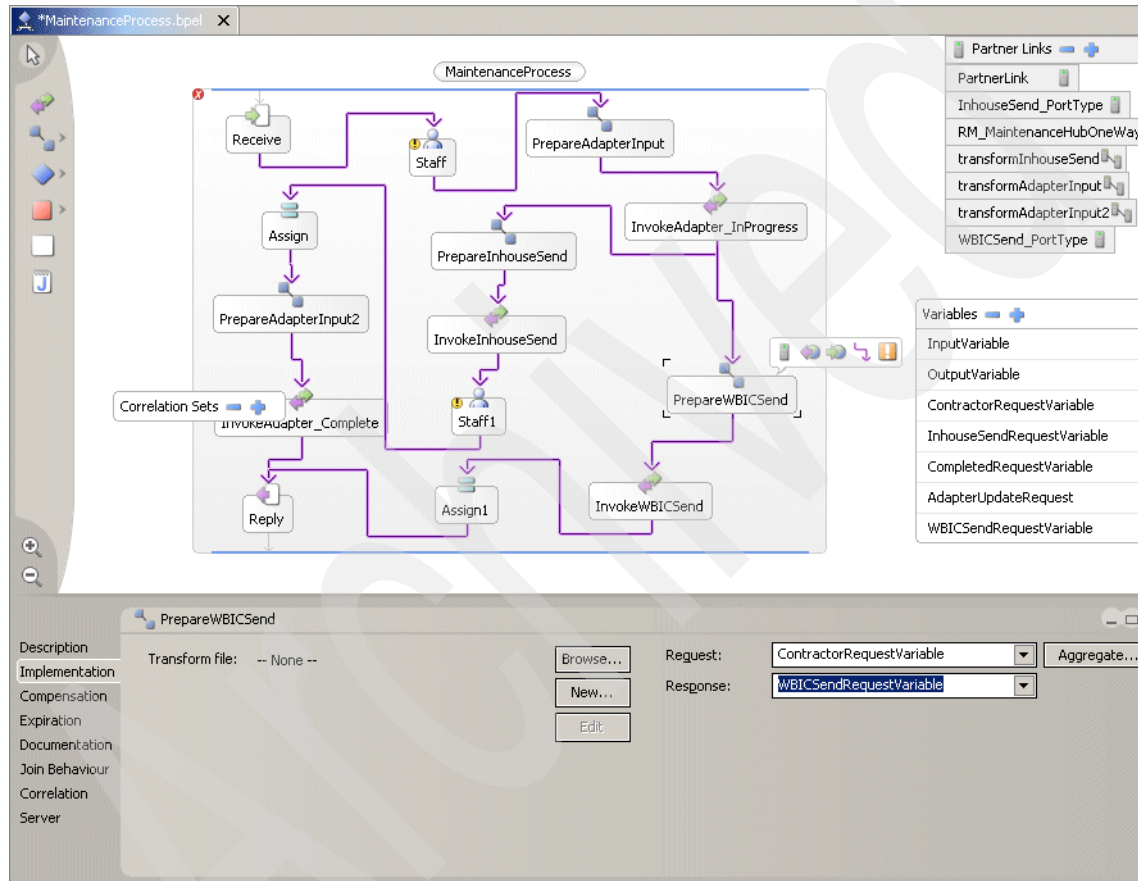


Figure 31-15 Add the transform activity PrepareWBICSend

14. Provide appropriate names for the link, service, port, and operation. Click **OK**.

Create a new transform

Create a new mapping service to assign one variable to another.

New Partner Link name
transformWBICSend

Transform file name
transformWBICSend_service

Target namespace
http://tempuri59512.org

Pgvt Type name
transformWBICSend_Port

Operation name
transformWBICSend

OK Cancel

Figure 31-16 Create a new transformer

15. The mappings are straight forward. Map each element in the input to the corresponding field in the output.

***MaintenanceProcess.bpel** ***transformWBICSend_service.wsdl**

Input Message

- message reviewResponse
 - part reviewResponsePart : Maintenance
 - ContractorRequest:From : RC:string
 - ContractorRequest:To : RC:string
 - ContractorRequest:ExpectedCompletion : RC:string
 - ContractorRequest:ActualCompletion : RC:string
 - ContractorRequest:StatusDescription : RC:string
 - ContractorRequest:Tenant : RC:int
 - ContractorRequest:Apartment : RC:int
 - ContractorRequest:MaintenanceId : RC:int
 - ContractorRequest:CreateDate : RC:string
 - ContractorRequest:Description : RC:string

Output Message

- message sendRequest
 - part contents : Maintenance
 - ContractorRequest:From : RC:string
 - ContractorRequest:To : RC:string
 - ContractorRequest:ExpectedCompletion : RC:string
 - ContractorRequest:ActualCompletion : RC:string
 - ContractorRequest:StatusDescription : RC:string
 - ContractorRequest:Tenant : RC:int
 - ContractorRequest:Apartment : RC:int
 - ContractorRequest:MaintenanceId : RC:int
 - ContractorRequest:CreateDate : RC:string
 - ContractorRequest:Description : RC:string

Overview

Output Message	Input Message	XPath Expression
message sendRequest	message reviewResponse	/reviewResponse
part contents : Maintenance	part reviewResponsePart : Maintenance	
ContractorRequest:From : RC:string	ContractorRequest:From : RC:string	/reviewResponse/reviewResponsePart/From
ContractorRequest:To : RC:string	ContractorRequest:To : RC:string	/reviewResponse/reviewResponsePart/To
ContractorRequest:ExpectedCompletion : RC:string	ContractorRequest:ExpectedCompletion : RC:string	/reviewResponse/reviewResponsePart/ExpectedCompletion
ContractorRequest:ActualCompletion : RC:string	ContractorRequest:ActualCompletion : RC:string	/reviewResponse/reviewResponsePart/ActualCompletion
ContractorRequest:StatusDescription : RC:string	ContractorRequest:StatusDescription : RC:string	/reviewResponse/reviewResponsePart/StatusDescription
ContractorRequest:Tenant : RC:int	ContractorRequest:Tenant : RC:int	/reviewResponse/reviewResponsePart/Tenant
ContractorRequest:Apartment : RC:int	ContractorRequest:Apartment : RC:int	/reviewResponse/reviewResponsePart/Apartment
ContractorRequest:MaintenanceId : RC:int	ContractorRequest:MaintenanceId : RC:int	/reviewResponse/reviewResponsePart/MaintenanceId
ContractorRequest:CreateDate : RC:string	ContractorRequest:CreateDate : RC:string	/reviewResponse/reviewResponsePart/CreateDate
ContractorRequest:Description : RC:string	ContractorRequest:Description : RC:string	/reviewResponse/reviewResponsePart/Description

Figure 31-17 Mapping statements of the transformer

16. Select the link between InvokeAdapter_InProgress and PrepareInhouseSend.
17. Select the Condition tab for this link and change the field Value to Expression. A small Java editor appears.
18. Write a few lines of code to obtain the response message of Staff activity that is stored in the variable ContractorRequestVariable. Get the message part of type Maintenance. Get the field To and compare with the string inhouse (Figure 31-18). Use Ctrl+Spacebar to help you build the right code.

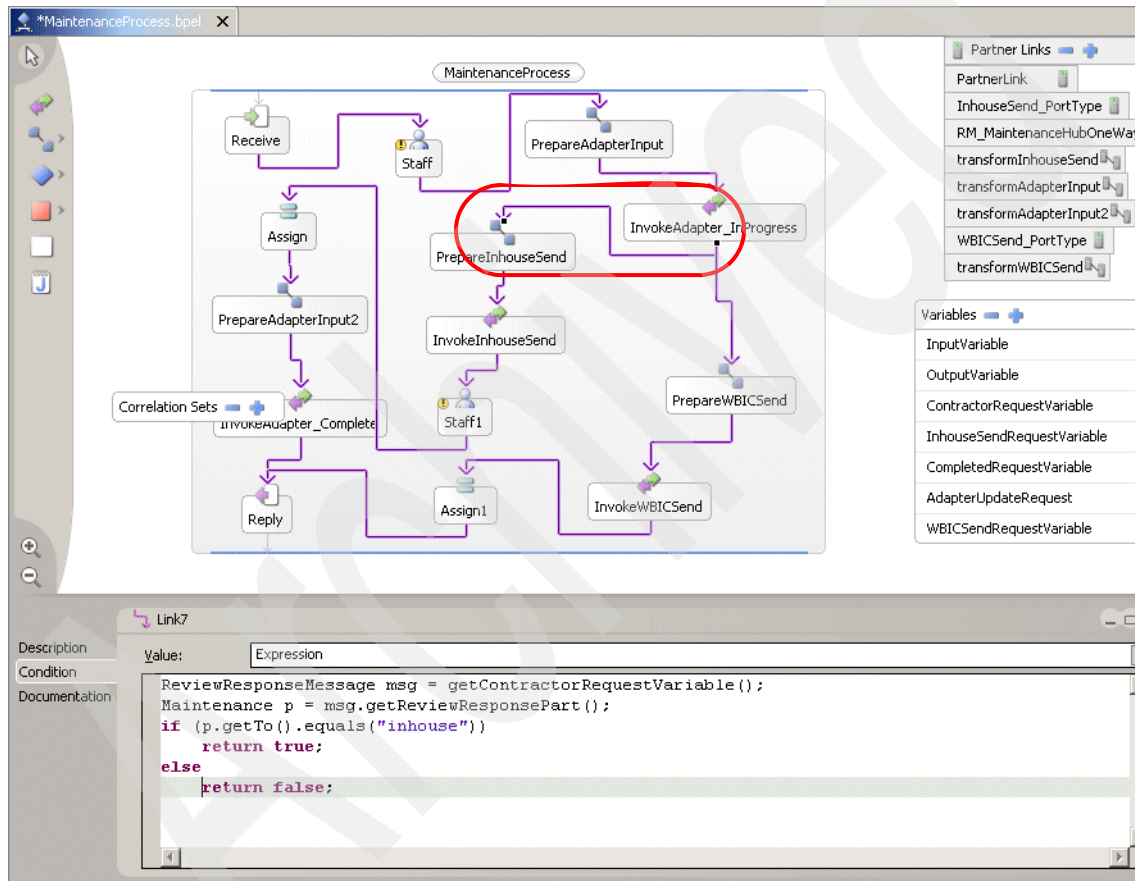


Figure 31-18 Add a condition expression

19. Select the link between InvokeAdapter_InProgress and PrepareWBICSend.
Select the tab Condition for this link and set its value to Otherwise.



Figure 31-19 Add a condition of Otherwise

20. The process model is now complete. Save all changes and close all editors.
21. Stop the server if it is running.
22. Generate deploy code.
23. Publish changes to the server.
24. Start the server.

31.1.4 Testing the extended business process

To test the extended business process:

1. Repeat the test process again using the Business Process Web Client.
2. Create a new instance, as shown in Figure 31-20.

The screenshot displays the Business Process Web Client interface. On the left, there are three main sections: 'Process Instance Lists' with links like 'Created By Me', 'Administered By Me', 'Undo Actions in Error', 'Define Process Instance List'; 'Process Template Lists' with 'My Templates' and 'Define Template List'; and 'Administration' with 'Manage Work Items for Process Instances' and 'Manage Activities for Process Instances'. The main area is titled 'Available Actions' and features a 'Start Instance' button. Below this, the 'Process Template Description' section shows details for 'MaintenanceProcess', including its valid from date (12/31/02 7:00:00 PM) and options like 'Delete on Completion', 'Can Run Interrupted', and 'Can Run Synchronously'. The 'Service' section lists 'Name: Receive', 'Description: -', 'PortType: ProcessPortType', and 'Operation: InputOperation'. A 'Process Instance Name' field is present. The 'Process Input Message' section contains a table of input parameters:

Parameter	Value	Type
contents.actualCompletion		(string)
contents.apartment	1	(int)
contents.createDate	1110500000000	(string)
contents.description	leak in roof	(string)
contents.expectedCompletion	1110600000000	(string)
contents.maintenanceId	2	(int)
contents.statusDescription		(string)
contents.tenant	100	(int)

Figure 31-20 Create a new process instance

3. In the first Staff activity, set the element To to redcontract, which is the AS2 identifier of RedContract.

Undo Actions in Error

Define Process Instance List

Process Template Lists

My Templates

Define Template List

Administration

Manage Work Items for Process Instances

Manage Activities for Process Instances

Process Context

Activity Name

Staff

Description

-

Template Name

MaintenanceProcess

Process Instance Name

_Pt:90030101.d016afcd.5054d5f6.18020000

State

Claimed

Administrators

db2admin

Reason

-

View more details about this activity.

View more details about this process.

Activity Input Message

contents.actualCompletion

1

contents.apartment

1110500000000

contents.createDate

leak in roof

contents.description

1110600000000

contents.expectedCompletion

2

contents.maintenanceId

contents.statusDescription

100

contents.tenant

Activity Output Message

reviewResponsePart.actualCompletion

(string)

reviewResponsePart.apartment

1

(int)

reviewResponsePart.createDate

1110500000000

(string)

reviewResponsePart.description

leak in roof

(string)

reviewResponsePart.expectedCompletion

1110600000000

(string)

reviewResponsePart.from

redmairt

(string)

reviewResponsePart.maintenanceId

2

(int)

reviewResponsePart.statusDescription

sent to contractor

(string)

reviewResponsePart.tenant

100

(int)

reviewResponsePart.to

redcontract

(string)

Figure 31-21 Send maintenance request to contractor

4. Select **Process Instance Lists** → **Administered By Me**. Notice that the process is already finished.

WebSphere

Business Integration
Server Foundation

Process choreographer

IBM

Help

UserID db2admin [Logout](#)

Work Item Lists

[My To Dos](#)
[Define Work Item List](#)

Process Instance Lists

[Created By Me](#)
[Administered By Me](#)
[Undo Actions in Error](#)
[Define Process Instance List](#)

Process Template Lists

[My Templates](#)
[Define Template List](#)

Administration

[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Administered By Me

Use this page to display information about process instances that you can administer and to perform actions on these process instances.

Available Actions

[View](#)
[Compensate](#)
[Terminate](#)
[Delete](#)
[Monitor](#)
[Repair Compensation](#)

	Process	Template Name	State	Started	Process Starter	Reason
<input checked="" type="checkbox"/>	PI:90030101_d016afcd.5054d516.18020000	MaintenanceProcess	Finished	2/1/05 5:42:23 PM	db2admin	Administrator

Figure 31-22 Process instance is finished

5. Select the finished process instance and click **Monitor**. Notice that certain activities have been skipped this time.

Work Item Lists

- [My To Dos](#)
- [Define Work Item List](#)

Process Instance Lists

- [Created By Me](#)
- [Administered By Me](#)
- [Undo Actions in Error](#)
- [Define Process Instance List](#)

Process Template Lists

- [My Templates](#)
- [Define Template List](#)

Administration

- [Manage Work Items for Process Instances](#)
- [Manage Activities for Process Instances](#)

Process Instance Monitor

Use this page to view the status of the activities in a process instance [1](#)

Process Description

Process Instance Name	_P1:90030101_d016afcd.5054d5f6.18020000	State	Finished
Template Name	MaintenanceProcess	Started	2/1/05 5:42:23 PM
Description			
Starter	db2admin		
Readers			
Administrators	db2admin		

Activities

To Do Name	State	Activity Kind	Activated
Reply	Finished	Reply	2/1/05 5:43:45 PM
InvokeWBICSend	Finished	Invoke	2/1/05 5:43:44 PM
PrepareWBICSend	Finished	Invoke	2/1/05 5:43:43 PM
PrepareAdapterInput2	Skipped	Invoke	2/1/05 5:43:43 PM
InvokeAdapter_Complete	Skipped	Invoke	2/1/05 5:43:43 PM
PrepareInhouseSend	Skipped	Invoke	2/1/05 5:43:43 PM
Staff1	Skipped	Staff	2/1/05 5:43:43 PM
InvokeInhouseSend	Skipped	Invoke	2/1/05 5:43:43 PM
InvokeAdapter_InProgress	Finished	Invoke	2/1/05 5:43:39 PM
PrepareAdapterInput	Finished	Invoke	2/1/05 5:43:36 PM
Staff	Finished	Staff	2/1/05 5:42:28 PM
Receive	Finished	Receive	2/1/05 5:42:24 PM

Figure 31-23 Monitoring a finished process

- Create another process instance. Use similar output as before.

- During the first Staff activity, set the element To to inhouse, as shown in Figure 31-24.

Process Context

Activity Name	Staff	Description	-
Template Name	MaintenanceProcess	Process Instance Name	_Pt90030101.d019b925.5054d5f6.18020039
State	Claimed	Administrators	db2admin
Reason	-		

[View more details about this activity.](#)
[View more details about this process.](#)

Activity Input Message

contents.actualCompletion	1
contents.apartment	1110500000000
contents.createDate	1110600000000
contents.description	leak in roof
contents.expectedCompletion	1110600000000
contents.maintenancel	2
contents.statusDescription	sent to inhouse technician
contents.tenant	100

Activity Output Message

reviewResponsePart.actualCompletion	<input type="text"/>	(string)
reviewResponsePart.apartment	<input type="text" value="1"/>	(int)
reviewResponsePart.createDate	<input type="text" value="1110500000000"/>	(string)
reviewResponsePart.description	<input type="text" value="leak in roof"/>	(string)
reviewResponsePart.expectedCompletion	<input type="text" value="1110600000000"/>	(string)
reviewResponsePart.from	<input type="text" value="redmaint"/>	(string)
reviewResponsePart.maintenancel	<input type="text" value="2"/>	(int)
reviewResponsePart.statusDescription	<input type="text" value="sent to inhouse technician"/>	(string)
reviewResponsePart.tenant	<input type="text" value="100"/>	(int)
reviewResponsePart.to	<input type="text" value="inhouse"/>	(string)

Figure 31-24 Send a request to the inhouse technician

- Select **Process Instance Lists** → **Administered By Me**. Notice that the process is not finished but is running.

WebSphere

Business Integration
Server Foundation

Process choreographer

IBM

Help

UserID db2adminLogout

Work Item Lists

My To Dos

Define Work Item List

Process Instance Lists

Created By Me

Administered By Me

Undo Actions in Error

Define Process Instance List

Process Template Lists

My Templates

Define Template List

Administration

Manage Work Items for Process Instances

Manage Activities for Process Instances

Administered By Me

Use this page to display information about process instances that you can administer and to perform actions on these process instances.

Available Actions

View

Compensate

Terminate

Delete

Monitor

Repair Compensation

	Process	Template Name	State	Started	Process Starter	Reason
<input type="checkbox"/>	Pt90030101_d016afcd.5054d5f6.18020000	MaintenanceProcess	Finished	2/1/05 5:42:23 PM	db2admin	Administrator
<input checked="" type="checkbox"/>	Pt90030101_d019b925.5054d5f6.18020039	MaintenanceProcess	Running	2/1/05 5:45:41 PM	db2admin	Administrator

Figure 31-25 Select the running process for monitoring

9. Select the running process and click **Monitor**. Notice that the WebSphere BI Connect activities have been skipped this time.

Work Item Lists

- My To Dos
- Define Work Item List

Process Instance Lists

- Created By Me
- Administered By Me
- Undo Actions in Error
- Define Process Instance List

Process Template Lists

- My Templates
- Define Template List

Administration

- Manage Work Items for Process Instances
- Manage Activities for Process Instances

Process Instance Monitor

Use this page to view the status of the activities in a process instance [\[1\]](#)

Process Description

Process Instance Name	_Pt:90030101.d019b925.5054d5f6.18020039	State	Running
Template Name	MaintenanceProcess	Started	2/1/05 5:45:41 PM
Description			
Starter	db2admin		
Readers			
Administrators	db2admin		

Activities

To Do Name	State	Activity Kind	Activated
Staff1	Ready	Staff	2/1/05 5:46:30 PM
InvokeInhouseSend	Finished	Invoke	2/1/05 5:46:30 PM
PrepareWBICSend	Skipped	Invoke	2/1/05 5:46:29 PM
InvokeWBICSend	Skipped	Invoke	2/1/05 5:46:29 PM
PrepareInhouseSend	Finished	Invoke	2/1/05 5:46:29 PM
InvokeAdapter_InProgress	Finished	Invoke	2/1/05 5:46:28 PM
PrepareAdapterInput	Finished	Invoke	2/1/05 5:46:28 PM
Staff	Finished	Staff	2/1/05 5:45:42 PM
Receive	Finished	Receive	2/1/05 5:45:41 PM
Reply	Inactive	Reply	

Figure 31-26 Monitoring a running process

10.Claim and complete the Staff1 activity to finish this process instance.

31.2 Developing a process start bean

In the overall scenario, it is the intention to instantiate the process when a new maintenance event is generated and not manually, as we have done so far. To achieve this, complete the following steps:

1. Generate an EJB that is called by an MDB upon the arrival of a message.
2. Generate the MDB for the incoming message from the adapter.
3. Generate a proxy for the process. The proxy is called by the EJB to instantiate the process.
4. Adapt the process to facilitate automatic start-up.

784 WebSphere Business Integration Adapters

The flow is as follows:

- ▶ Adapter puts message on queue.
- ▶ MDB gets triggered upon the arrival of this message.
- ▶ MDB calls EJB to handle message.
- ▶ EJB calls process proxy.
- ▶ Process proxy instantiates the process.

31.2.1 Generating the EJB that is called by an MDB

To generate the EJB:

1. Create an EJB project. Select **File** → **New** → **Other**.
2. Select EJB in left pane and EJB Project in right pane. Click **Next**.

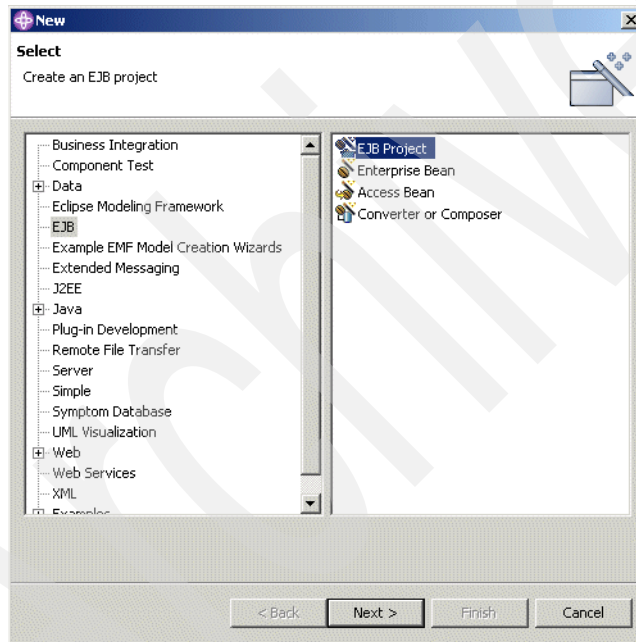


Figure 31-27 Create EJB project

3. Accept to create an EJB 2.0 project and click **Next**.
4. Provide a name for the EJB project, for example Process_StartEJB.
5. For the EAR project, click **New**. A new window appears. Name the EAR project Process_StartEJBEAR. Click **Finish** (Figure 31-29 on page 787) to return to Figure 31-28.
6. Click **Finish** in Figure 31-28 as well.

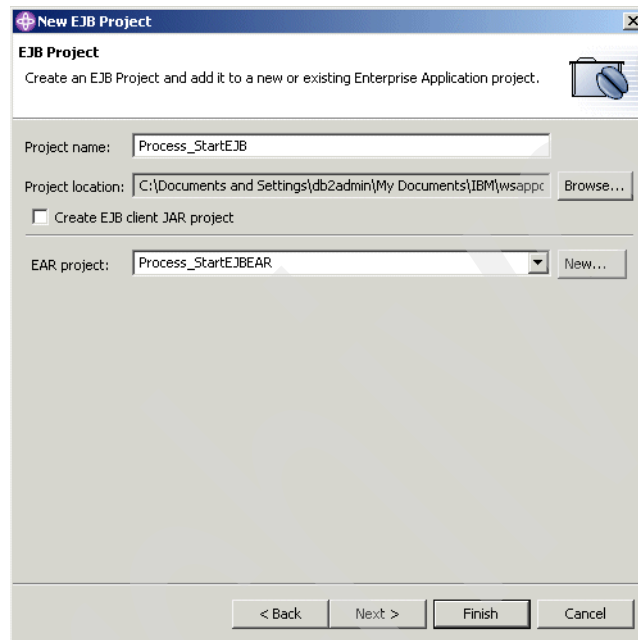


Figure 31-28 Create EJB project

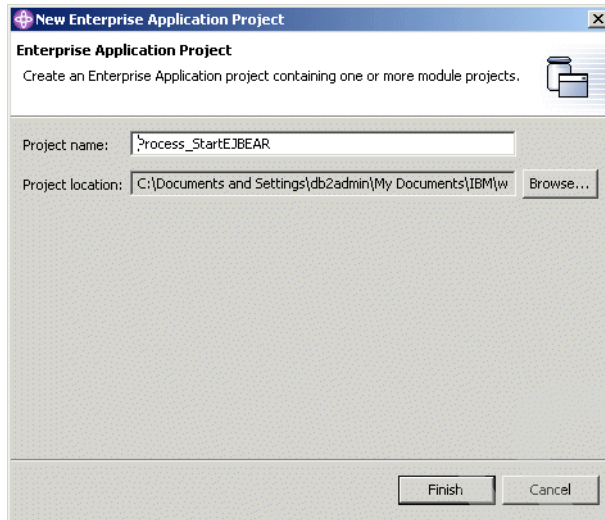


Figure 31-29 Create EAR project

7. WebSphere Studio Application Developer offers the option to switch perspective. Click **No**.

The EJB that we want to build is based on the WSDL of the RM2 connector. The connector sends a message, according to a certain operation in the WSDL. The EJB needs to do something for each operation in the WSDL.

8. Right-click the WSDL RM2Connector.wsdl and select **New** → **Build from service**.
9. Select EJB Service Skeleton and click **Next**.

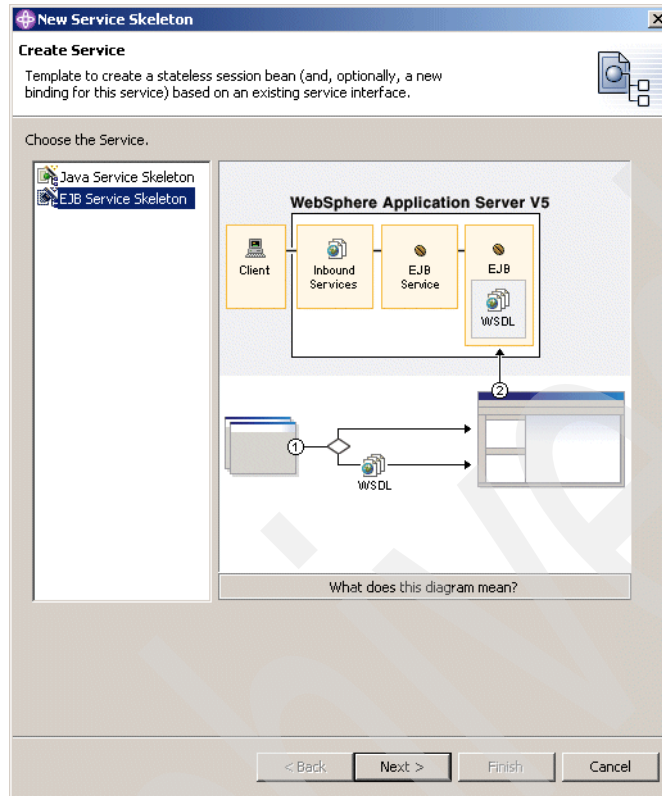


Figure 31-30 Create EJB based on existing WSDL

10. Select the option to create a new port and binding and click **Next**.

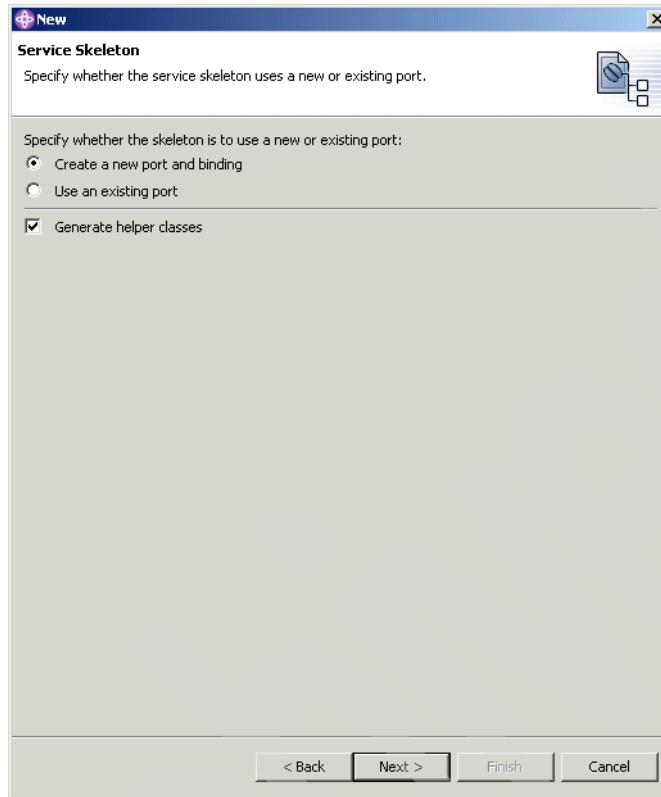


Figure 31-31 Generate new WSDL for a new port and binding

11. Make sure that Agent Delivery is selected as port type.
12. Update the source folder, as shown in Figure 31-32 on page 790. Click **Next**.

New

Service Skeleton

Create a service skeleton.

Select the service interface for the new skeleton:

WSDL file: /Maintenance_Processes/maintenance/process/RM2Cc **Browse...**

Port type name: RM_MaintenanceAgentDelivery

Specify the port location and its name:

Source folder: /Process_StartEJB/ejbModule **Browse...**

Package: maintenance.process **Browse...**

File name: RM_MaintenanceAgentDeliveryEJBService.wsdl **Browse...**

Service name: RM_MaintenanceAgentDeliveryEJBService **Browse...**

Port name: RM_MaintenanceAgentDeliveryEJBPort

Specify the binding location and its name:

Source folder: /Process_StartEJB/ejbModule

Package: maintenance.process

File name: RM_MaintenanceAgentDeliveryEJBBinding.wsdl **Browse...**

Binding name: RM_MaintenanceAgentDeliveryEJBBinding

< Back Next > Finish Cancel

Figure 31-32 Create skeleton - step 1

13. Select the EJB project that was created earlier and click **Finish**.

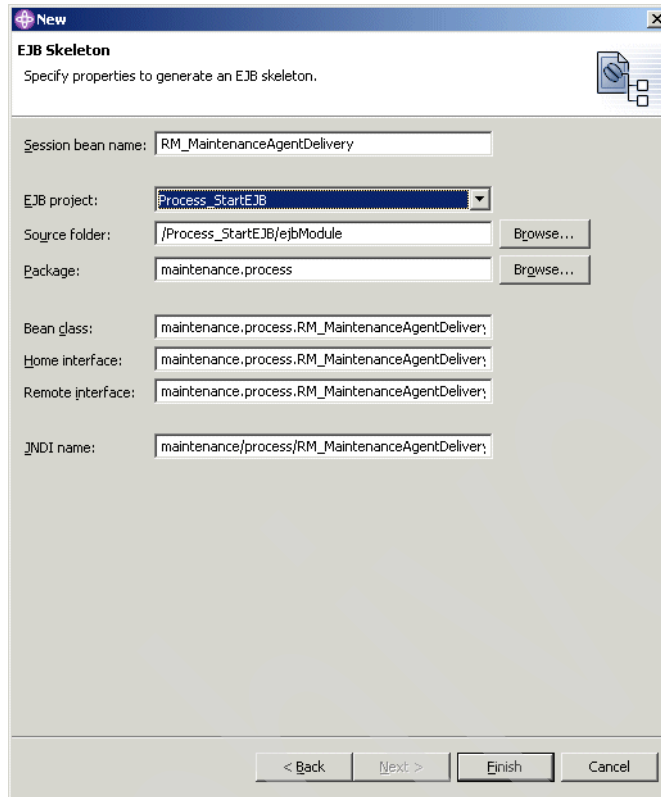


Figure 31-33 Create skeleton - step 2

14. As before, the project has an error due to a missing library. Open the properties of the project Process_Start_EJB to correct the error.

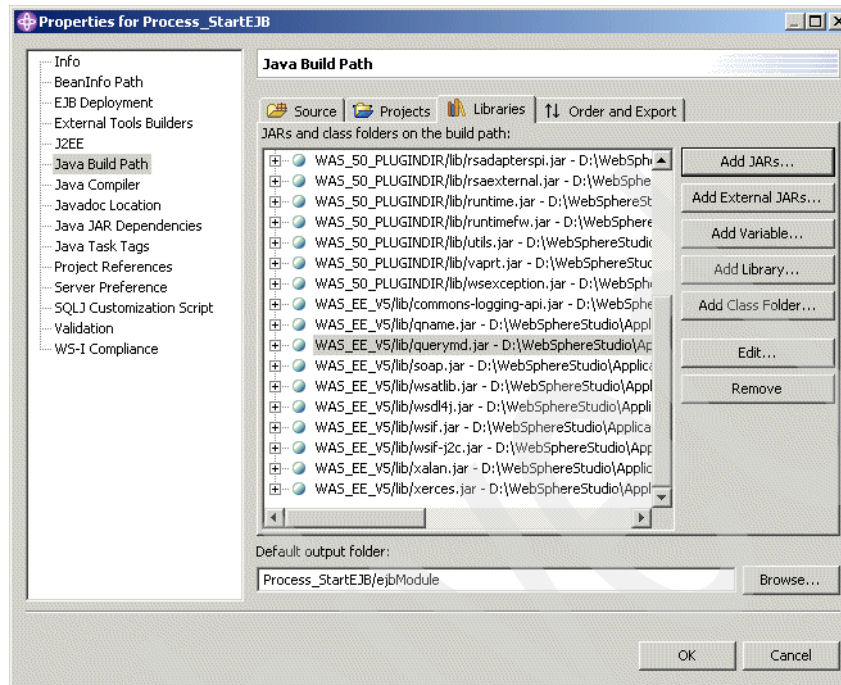


Figure 31-34 Correct build path

15. Open the source file `RM_MaintenanceAgentDeliveryBean.java`, where you find methods for each verb that the business object `RM_Maintenance` supports.
16. Locate the `RM_MaintenanceCreate` method and add a simple print statement in the user code section. We will update this method later to instantiate the process.

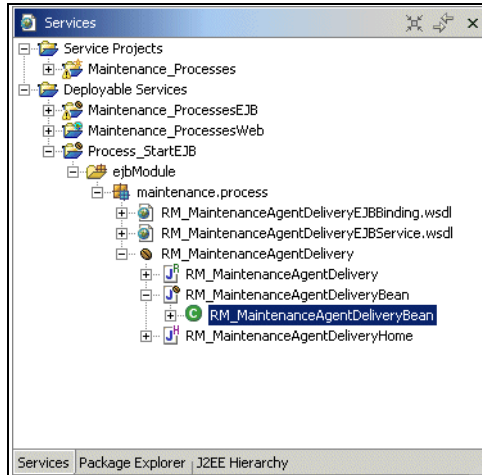


Figure 31-35

17. Update the RM_MaintenanceUpdate method in the same way.

Example 31-1 Implementation of the create method

```
public void RM_MaintenanceCreate(
    com
        .ibm
        .www
        .websphere
        .crossworlds
        ._2002
        .BOSchema
        .RM_Maintenance
        .RM_MaintenanceElement argBodyPart) {
    // user code begin {method_content}
    System.out.println("<<<< Message arrived for Create >>>>");
    return;
    // user code end
}
```

18. Right-click the EJB in the Services pane and select **Generate Deployment Code**.

31.2.2 Generating the MDB

A message-driven bean (MDB) is a special kind of bean that gets called by the JMS services in WebSphere Application Server when a message arrives in a queue. Usually, the MDB does not do any business logic. It calls another bean to process the actual message data.

In our scenario, we have the WSDL for the incoming message, and we have WSDL for the EJB that needs to process the message. The generation of the MDB uses both sets of WSDL files. To generate the MDB:

1. Locate the file `RM_MaintenanceAgentDeliveryEJBService.wsdl` in the deployable service project `Process_StartEJB`. Right-click this WSDL file and select **Enterprise Services** → **Generate Deploy Code**.
2. Select the option to use an existing port for the inbound message. Click **Next**.

Generate Deploy Code

Deployment
Deploy a service.

Select the service to deploy:

Service file name:

Service name:

Port name:

☐ Generate helper classes

Specify whether to generate a new or use an existing port:

☐ Create a new port and binding

☒ Use an existing port

Specify the binding type to generate:

Inbound binding type:

Description: Deploy service as a Web service. This Web service can then be invoked using the Web service programming model.

Specify or create the J2EE application projects:

EAR project:

EJB project:

Web project:

< Back Next > Finish Cancel

Figure 31-36 Generate deploy code for an existing port

3. Point to the WSDL of the existing port, which is the WSDL of the EJB. Click **Browse**.
4. Select the correct WSDL file (Figure 31-37) and click **OK**.

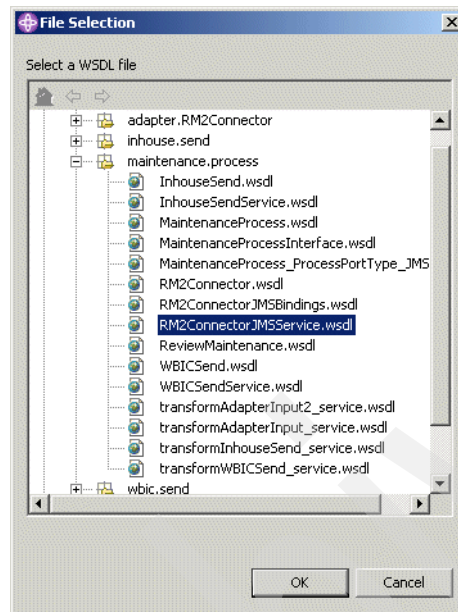


Figure 31-37 Select the inbound WSDL

5. The next window shows an error. Select the port that matches with the port in the EJB WSDL. Select the Agent Delivery port.

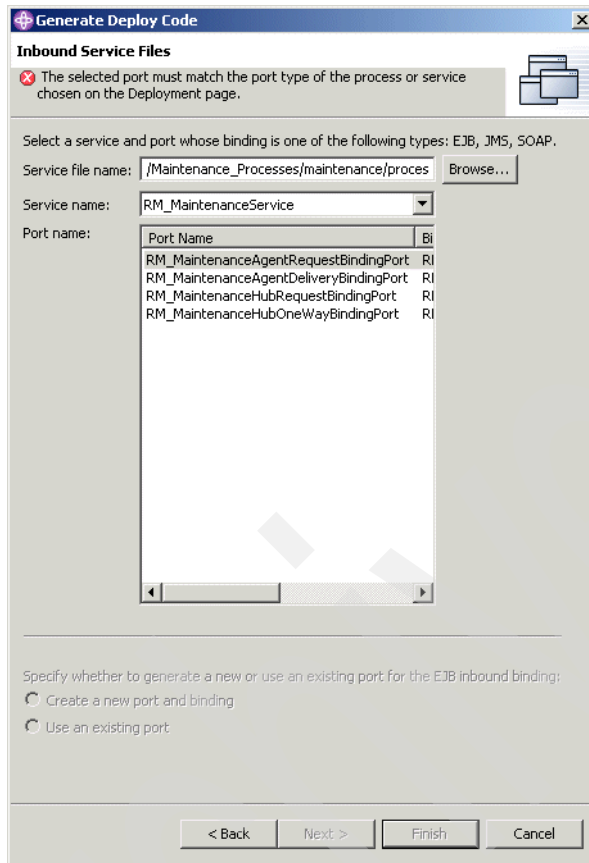


Figure 31-38 Select correct port

- Click **Finish** in the next window (Figure 31-39 on page 797).

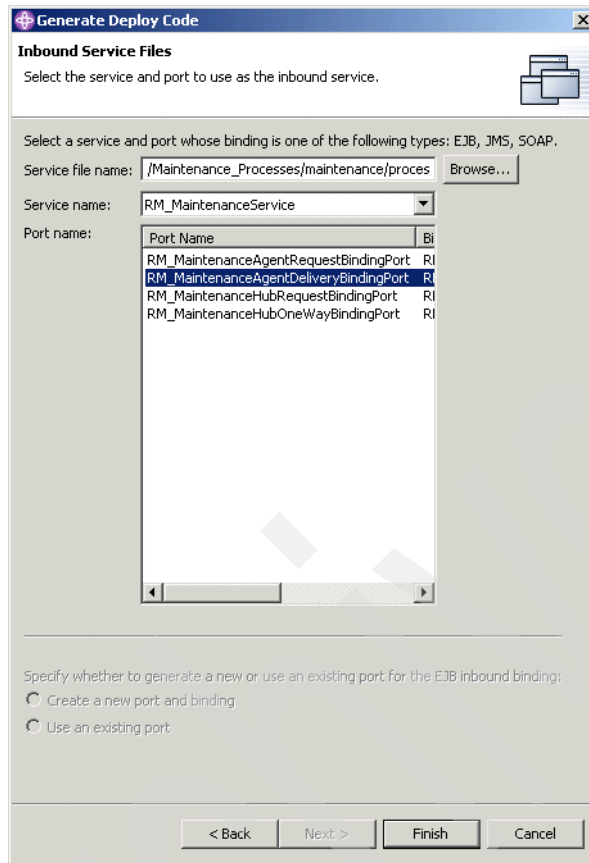


Figure 31-39 MDB generation

7. A new project Process_StartEJBEBJ has been created. Correct again the library error.
8. An MDB uses a number of resources, for which you need to change the JNDI names. Locate the MDB and open the EJB deployment descriptor.

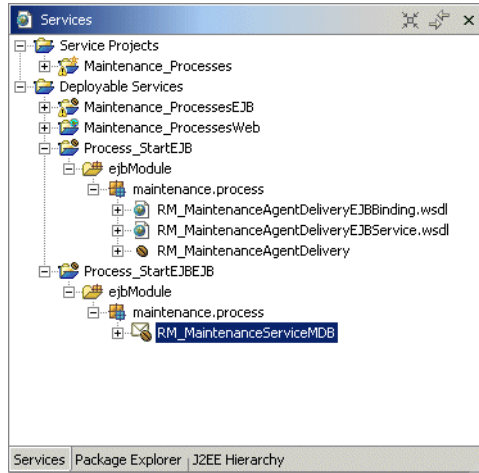


Figure 31-40 MDB in EJB project

9. In the EJB deployment descriptor editor, select the Beans tab.
10. Select the MDB itself and change the value of the listener port (Figure 31-41 on page 799).

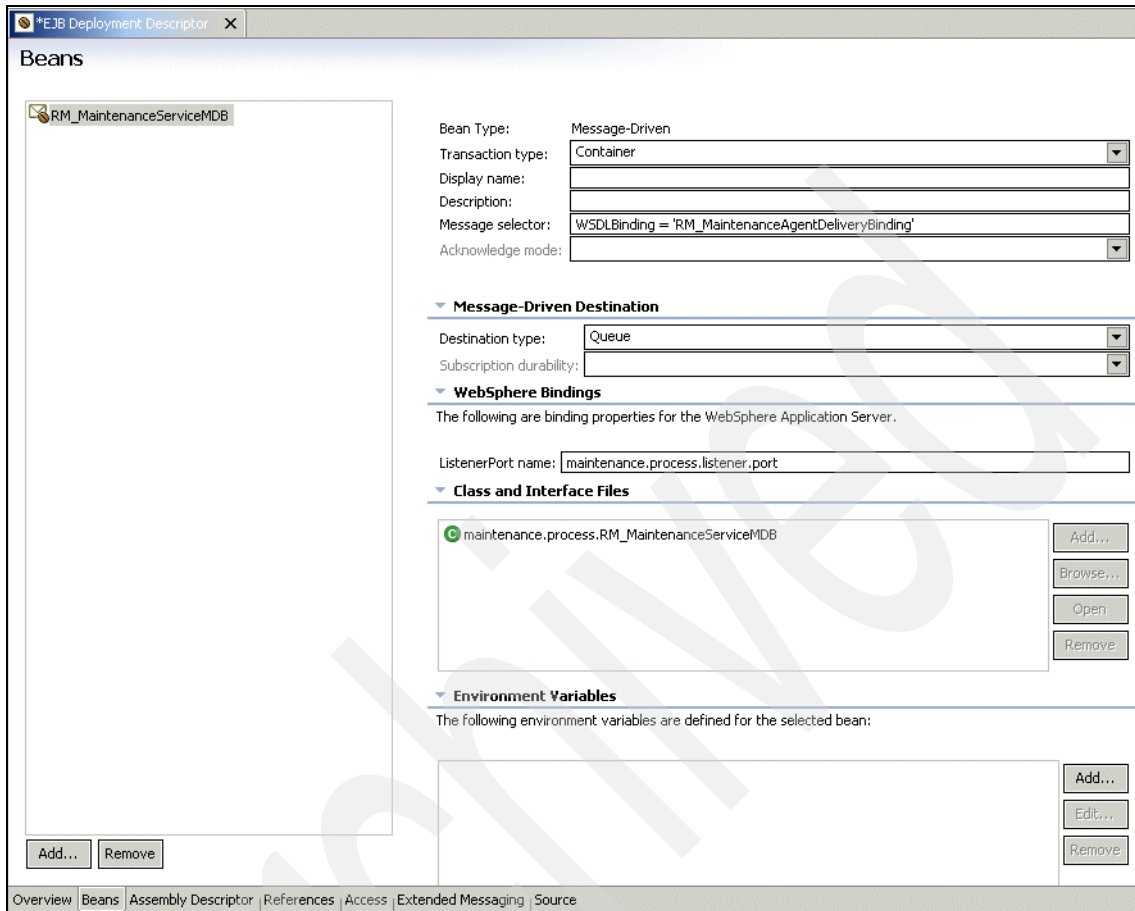


Figure 31-41 Change the listener port of the MDB

11. Select the References tab.
12. Select the resource environment reference and change the JNDI name.

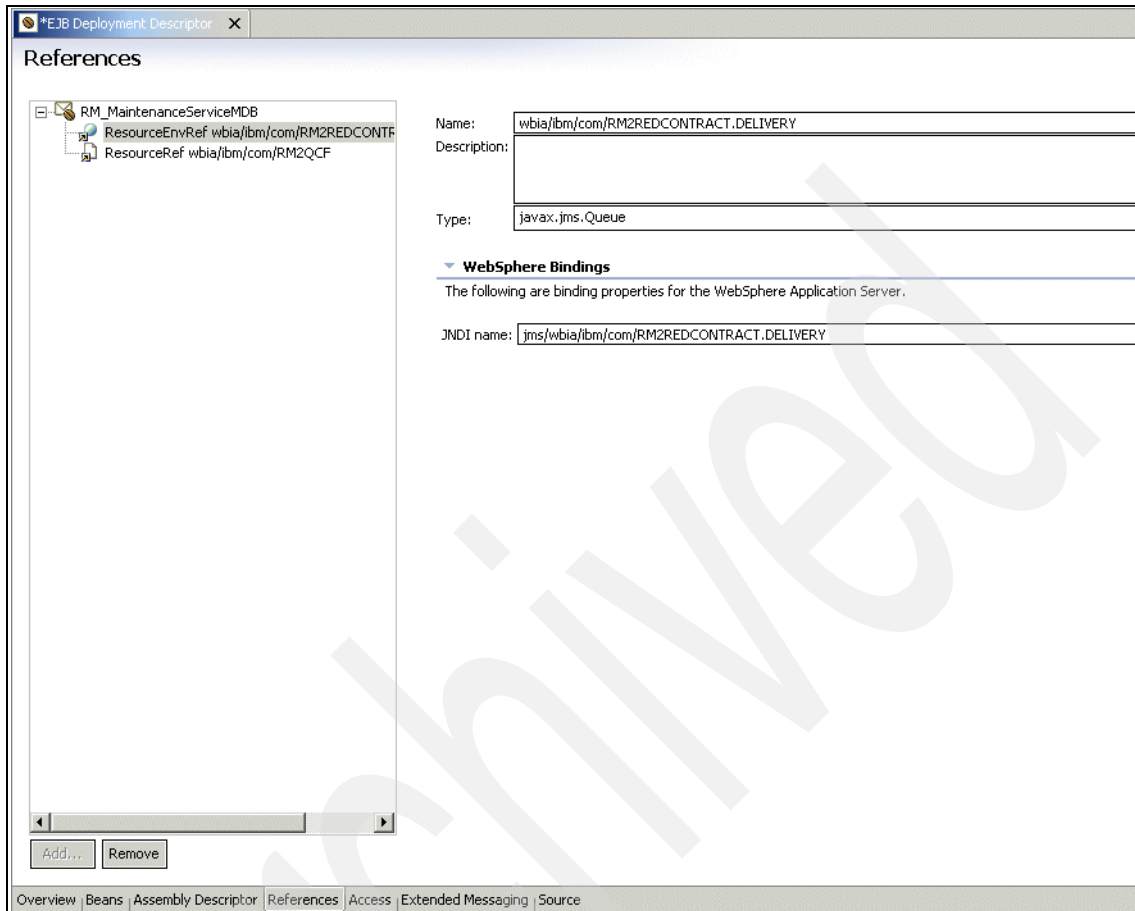


Figure 31-42 Change the JNDI name for the queue destination

13. Select the resource reference and change the JNDI name of the queue connection factory.

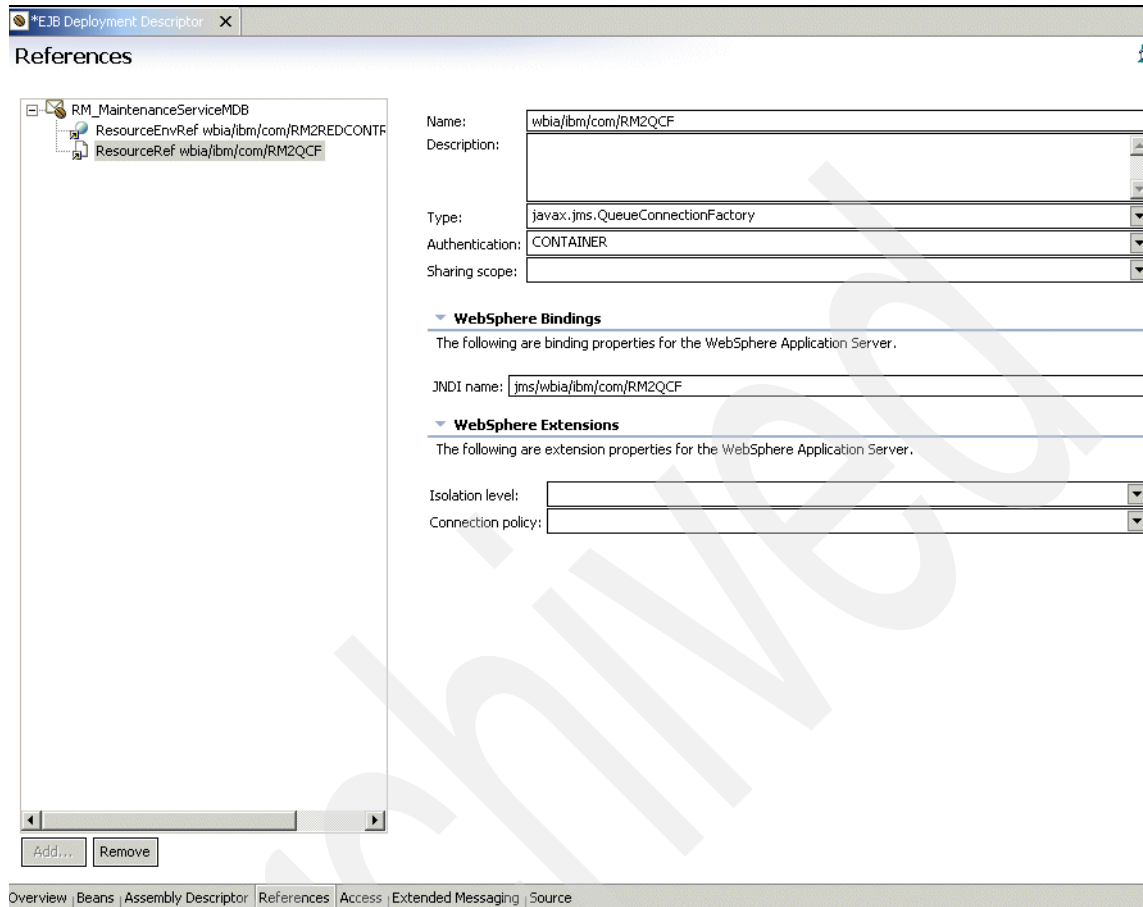


Figure 31-43 Change the JNDI name for the queue connection factory

14. Save the changes and close the EJB deployment descriptor editor.

31.2.3 Defining resources

Before you can test the MDB and the EJB, you need to define the listener port and one extra queue destination by following these steps:

1. Log on to Administrative Console (Figure 31-44 on page 802).
2. Select **Servers** → **Application Servers**.
3. Select server1 in the right pane.
4. Scroll down and click **Message Listener Service**.
5. Select **Listener Ports**.

6. Click **New**.
7. Use the following values:
 - Display Name: maintenance.process.listener.port
 - Destination JNDI Name: jms/wbia/ibm/com/RM2REDCONTRACT.DELIVERY
 - Connection Factory JNDI Name: jms/wbia/ibm/com/RM2QCF
8. Click **OK**.

WebSphere Application Server Administrative Console Version 5

Home | Save | Preferences | Logout | Help

User ID: db2admin

localhost

- Servers
 - Application Servers
 - Applications
 - Resources
 - Security
 - Environment
 - System Administration
 - Troubleshooting

Application Servers > server1 > Message Listener Service > Listener Ports > New

Listener ports for Message Driven Beans to listen upon for messages. Each port specifies the JMS Connection Factory and JMS Destination that an MDB, deployed against that port, will listen upon.

Runtime Configuration

General Properties		
Name	* maintenance.process.listener.port	i Name of the listener port
Initial State	* Started	i The execution state requested when the server is first started.
Description		i A description of the listener port, for administrative purposes
Connection factory JNDI name	* jms/wbia/ibm/com/RM2QCF	i The JNDI name for the JMS connection factory to be used by the listener port; for example, jms/connFactory1.
Destination JNDI name	* /com/RM2REDCONTRACT.DELIVERY	i The JNDI name for the destination to be used by the listener port; for example, jms/destn1.
Maximum sessions	1	i The maximum number of concurrent JMS server sessions used by a listener to process messages, in the range 1 through 2147483647.
Maximum retries	0	i The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the

WebSphere Status February 1, 2005 7:28:20 PM EST

WebSphere Runtime Messages

Total All Messages: 105 : 0 new, 0 total : 0 new, 0 total : 105 new, 105 total

Clear All

Preferences

Figure 31-44 Create a listener port

9. Create a new queue destination as follows:
 - Display Name: RM2REDCONTRACT.DELIVERY
 - JNDI Name: jms/wbia/ibm/com/RM2REDCONTRACT.DELIVERY
 - Base Queue Name: REDCONTRACT.DELIVERY

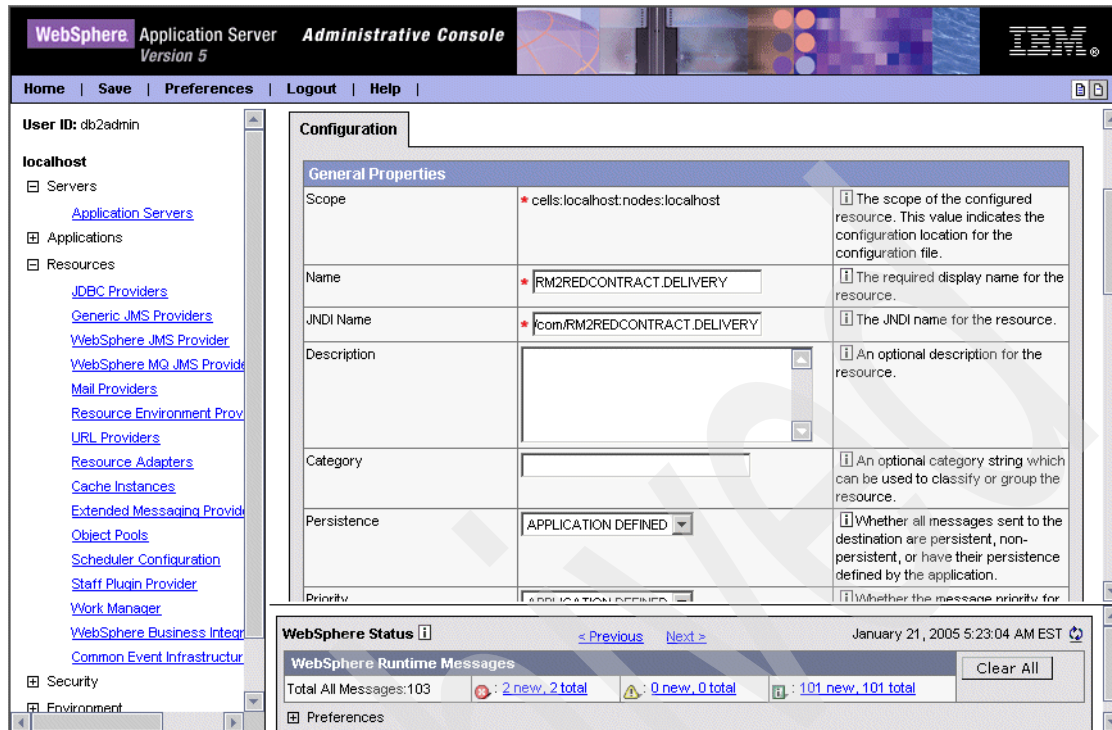


Figure 31-45 New queue destination

10. Save the changes to the master configuration of the Test server using the link at the top of the page.

31.2.4 Testing the MDB and EJB

To test the MDB and the EJB, follow these steps:

1. Right-click the Test server and select **Add and Remove Projects**.
2. Add the project Process_StartEJBEBEAR to the server and click **Finish**.

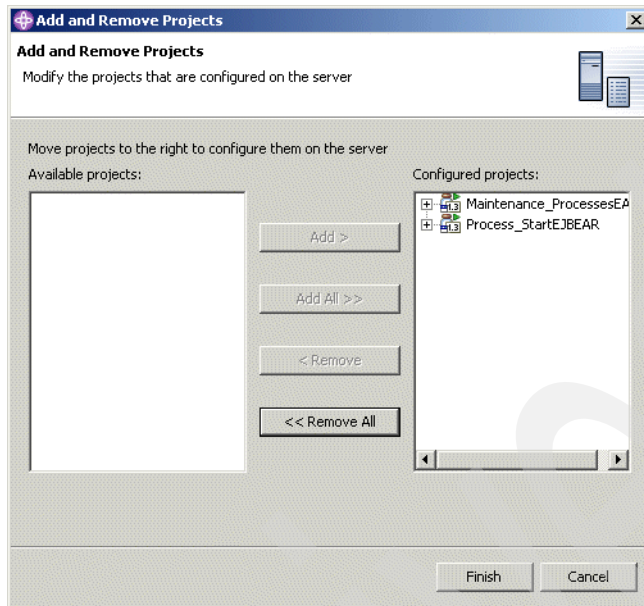


Figure 31-46 Add project to server

3. Stop the Test server.
4. Publish the changes to the server.
5. Start the server.
6. When the server has restarted, verify the console output. You should see a message about the start-up of the MDB, as shown in Example 31-2.

Example 31-2 MDB successfully started

MDB Listener maintenance.process.listener.port started successfully for
JMSDestination jms/wbia/ibm/com/RM2REDCONTRACT.DELIVERY

7. As an additional test, open WebSphere MQ Explorer on the Message Broker machine and verify MQ queue properties of the REDCONTRACT.DELIVERY queue. On the Statistics tab, you should see an Open Input Count of 1, which corresponds to the MDB.

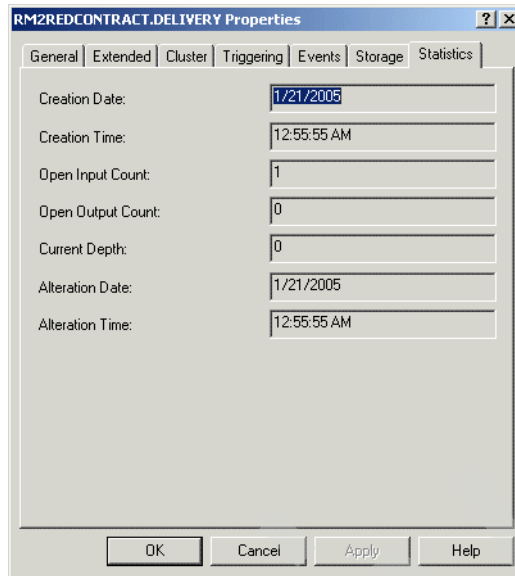


Figure 31-47 Queue properties

8. Start `rfhutil` on the Message Broker system. Read the `RM_Maintenance.Create` file, which is available in Additional Materials in the `RedTenantTestMessages` folder.

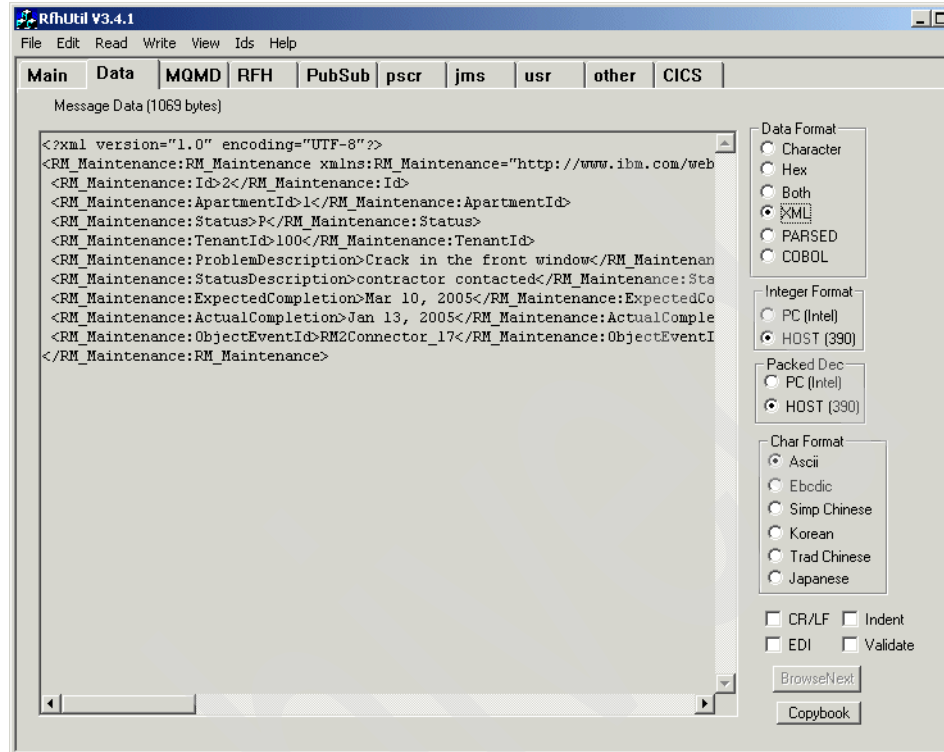


Figure 31-48 Connector event message

9. Select the **usr** tab in **rfhutil**. This tab lists user properties of the JMS message. Compare this value with the message selector value in the EJB deployment descriptor (see Figure 31-41 on page 799). The MDB only processes messages for which the WSDL binding matches.

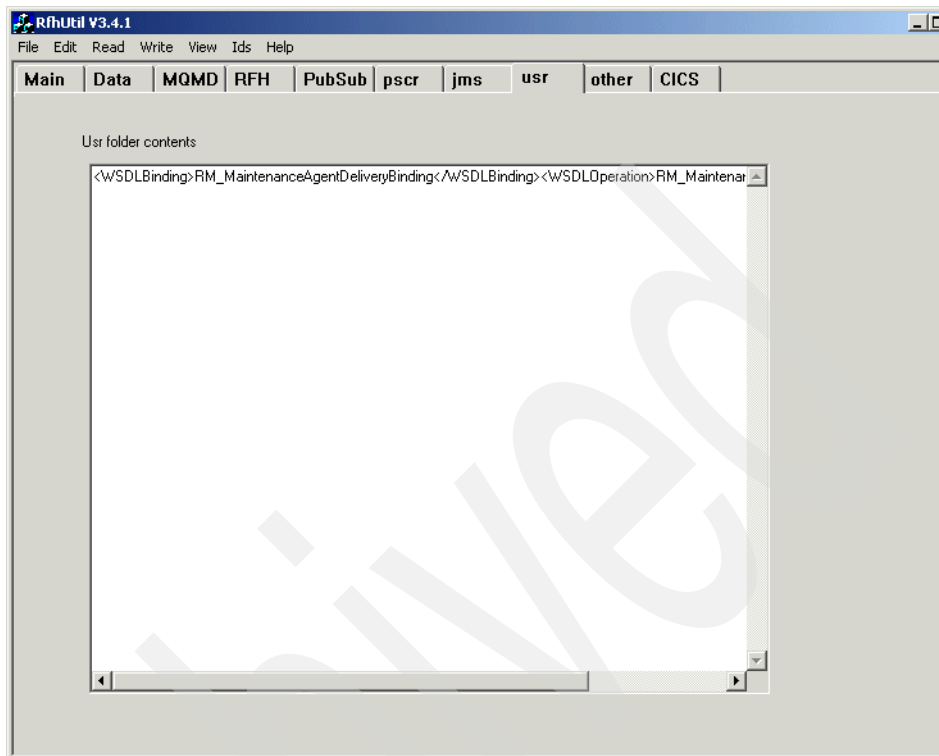


Figure 31-49 WSDL binding in the usr folder

10. Back on the Main tab in **rfhut i1**, click **Write Q**.

11. Return to the console of the Test server. If all went well, you should see the output of the print statement in the EJB, as shown in Example 31-3.

Example 31-3 Output that is generated by the EJB

SystemOut	0 <<<< Message arrived for Update >>>>
-----------	--

31.2.5 Generating proxy for process

Now that you have tested the EJB and MDB, you generate a proxy for the process. The EJB can then instantiate this proxy and call its operation to create and start a process instance. To generate a proxy:

1. In the package maintenance.process, right-click the WSDL file MaintenanceProcess_ProcessPortType_JMS.wsdl and select **Enterprise Services** → **Generate Service Proxy**.

Because it is a long-running business process, you can only use the WSIF framework.

2. Click **Next**.

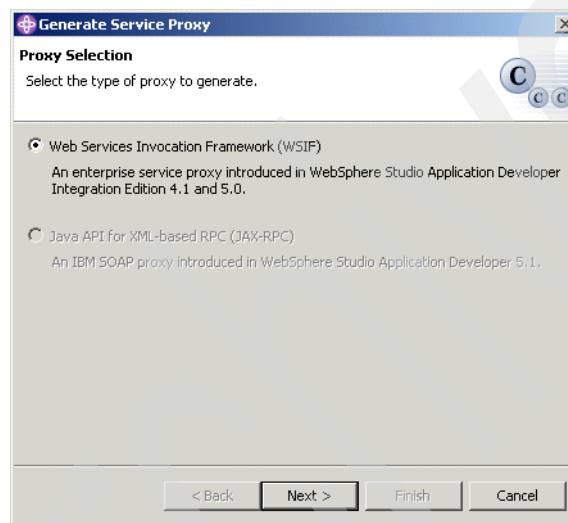


Figure 31-50 Generate service proxy - step 1

3. Review the selections and click **Next**.

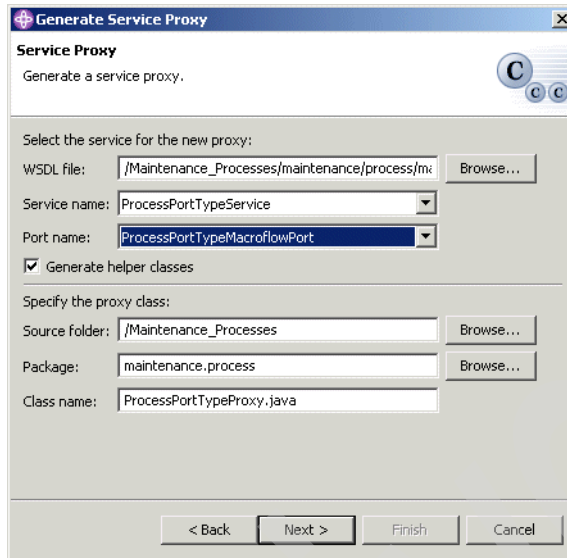


Figure 31-51 Generate service proxy - step 1

4. Select the port type and click **Finish**.

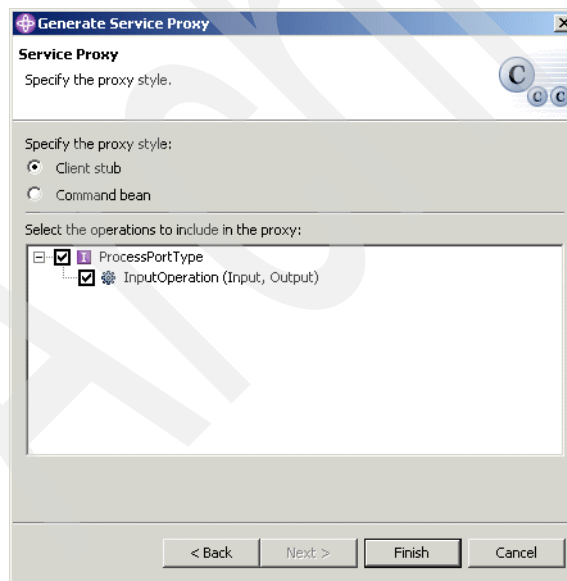


Figure 31-52 Generate service proxy - step 3

While you could execute and test this proxy, this is not very practical at the moment. The process has one operation with an input and output. As a result, the generated proxy expects a response message before completing. Given that the process is long-running, the proxy execution would be blocked until the process instance is finished.

There are two alternatives to solve this issue:

- ▶ Use the business process engine API, which is what the Web client does.
- ▶ Adapt the process model slightly, which is discussed in the next section.

31.2.6 Adapting the process model

The base issue is that the reply activity is only executed at the very end of the process and that the process can be paused for a significant amount of time due to its nature. We change the process so that its flow becomes as follows:

- ▶ Receive: the process input
- ▶ Assign: set the process output to *process started successfully*
- ▶ Reply: send this process output to the caller
- ▶ Staff: continue with the rest of the flow

Optionally, the actual process result can then be passed back to the caller via a JMS message.

To adapt the process model:

1. Open the process model in the BPEL editor.
2. Remove the link between InvokeAdapter_Complete and Reply.
3. Remove the link between InvokeWBICSend and Assign1.
4. Remove the Assign activity.
5. Reconnect the Receive activity to Assign1.
6. Connect Reply to Staff.
7. Update Assign1 activity to set an appropriate output text, as shown in Figure 31-53 on page 811.

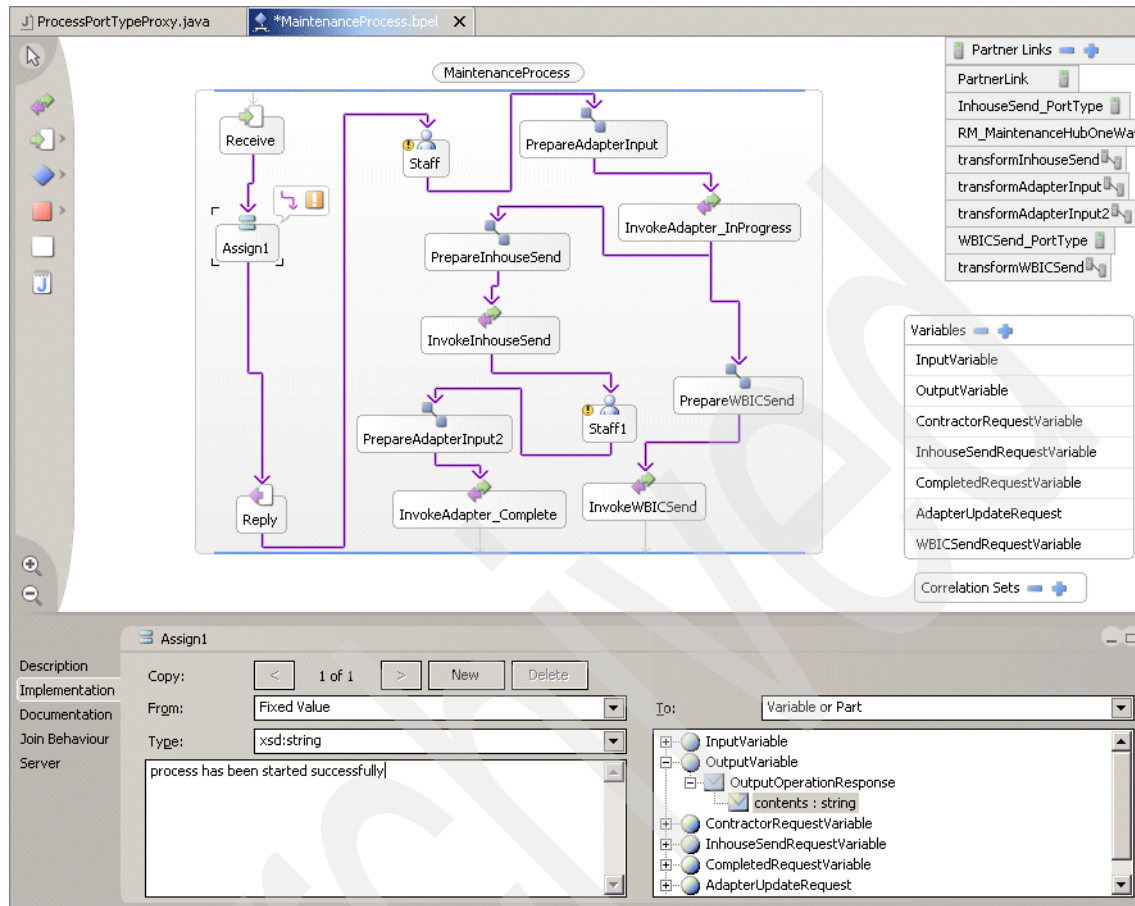


Figure 31-53 Updated process model

8. Process instances are no longer created by a user in the Web client. To make sure that you can administer process instances that are fired off asynchronously, update the Staff settings for the process itself. Select the maintenance.process symbol at the top of the model and select the Staff tab at the bottom.
9. Change the verb to Everybody.

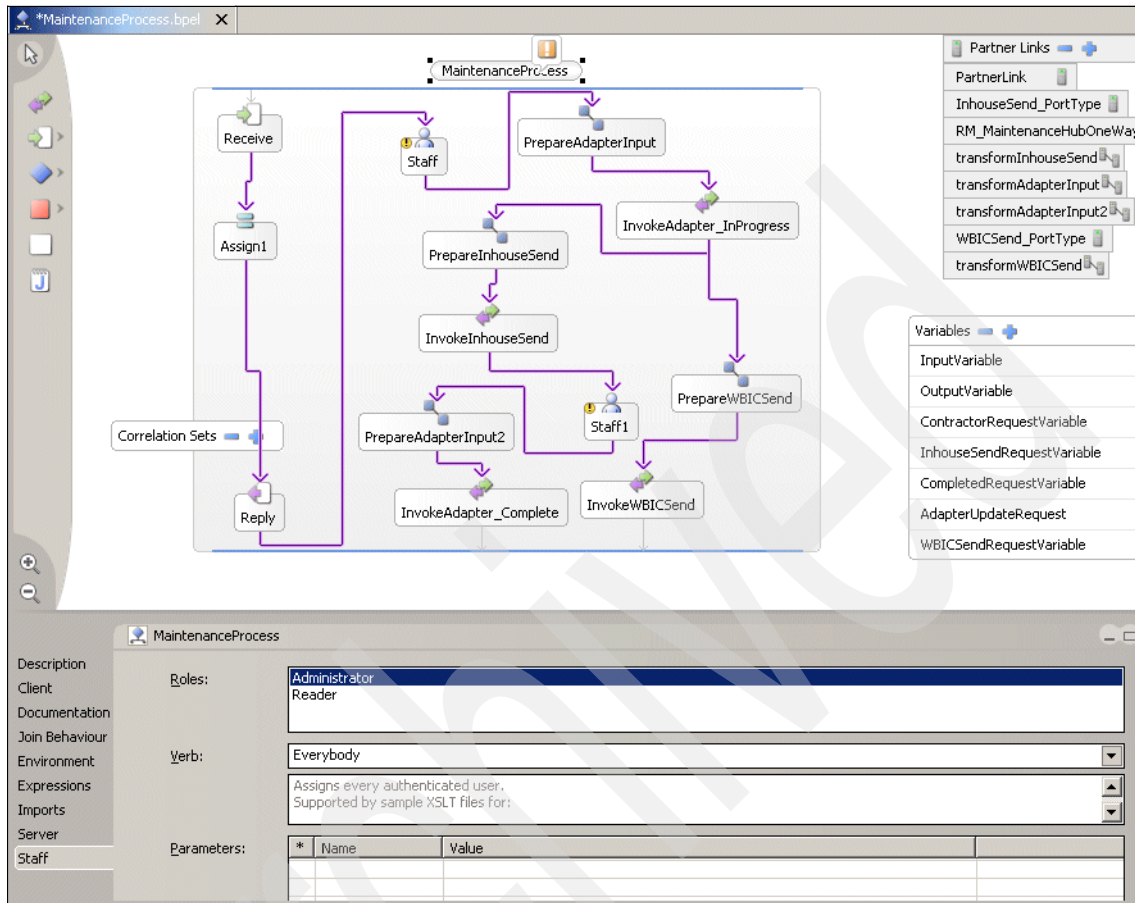


Figure 31-54 Set Staff properties for process

10. Save the new process model.

31.2.7 Testing the proxy and the adapted process model

You can now test the proxy and the adapted process model. The generated proxy contains all the required code, except for populating the input structure of the process, which is MaintenanceRequest. To test the proxy and the adapted process model:

1. Open the Java file ProcessPortTypeProxy.java.
2. Locate the main() method in this class.

3. Add a few lines of code to create an input structure and to set the appropriate fields. You also have to add an import statement. Example 31-4 shows a sample from our scenario.

Example 31-4 Updating the proxy

```
import com.redmaint.request.MaintenanceRequest;

public static void main(String[] args) {

    try {

        ProcessPortTypeProxy aProxy = new ProcessPortTypeProxy();

        // user code begin {proxy_method_calls}
        MaintenanceRequest rq = new MaintenanceRequest();
        rq.setApartment(1);
        rq.setCreateDate("Today");
        rq.setTenant(100);
        rq.setDescription("leak in roof");
        rq.setMaintenanceId(2);
        rq.setExpectedCompletion("Tomorrow");

        String resp = aProxy.InputOperation(rq);
        System.out.println("Output: " + resp);
        // user code end

    } catch (Exception e) {

        // user code begin {exception_handling}
        e.printStackTrace();
        // user code end

    }

}
```

4. Stop the server if it is started.
5. Generate deploy code for the updated process.
6. Regenerate the service proxy for the process. Your changes is not lost as long as you have placed them between the user code begin and user code end comments.
7. Publish the changes.
8. Restart the server.
9. When the server is started, launch the Test client.
10. Select the Bean Page.
11. Click **Utilities**.

12. Click **Load Class**.

13. Enter the name of the proxy class and click **Load**.

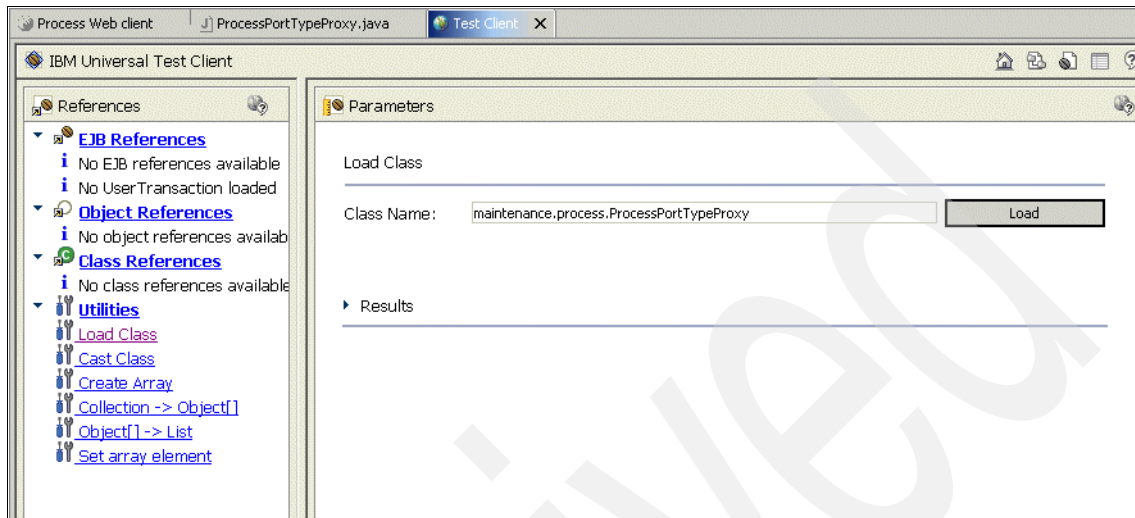


Figure 31-55 Load the proxy class in the Test client

14. When the class is loaded and an instance is created, click **Work with Object**.

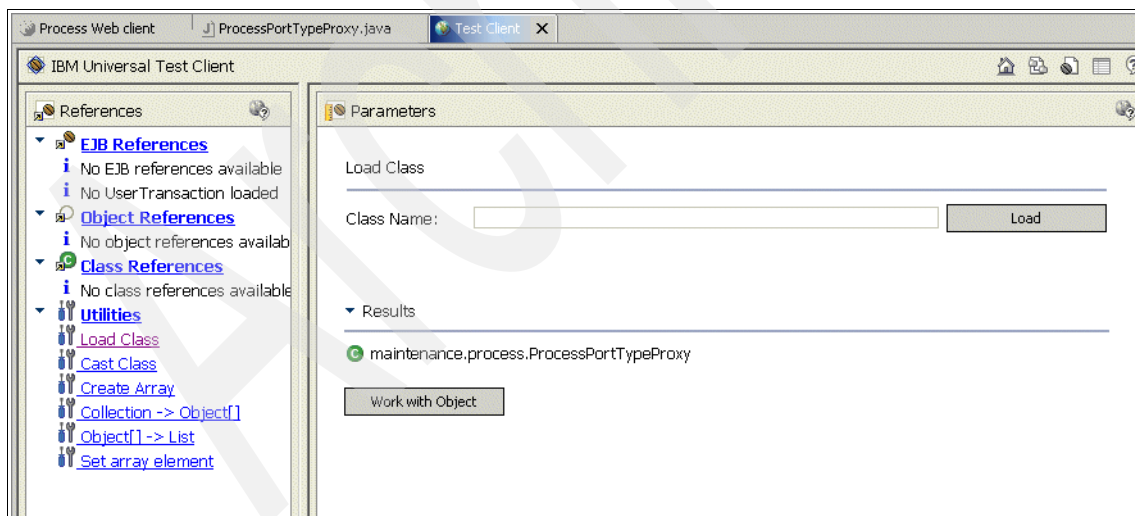


Figure 31-56 Add class instance to list of references

15. Expand the class references on the left to locate the main method of the proxy class.

16. Click **Invoke**.

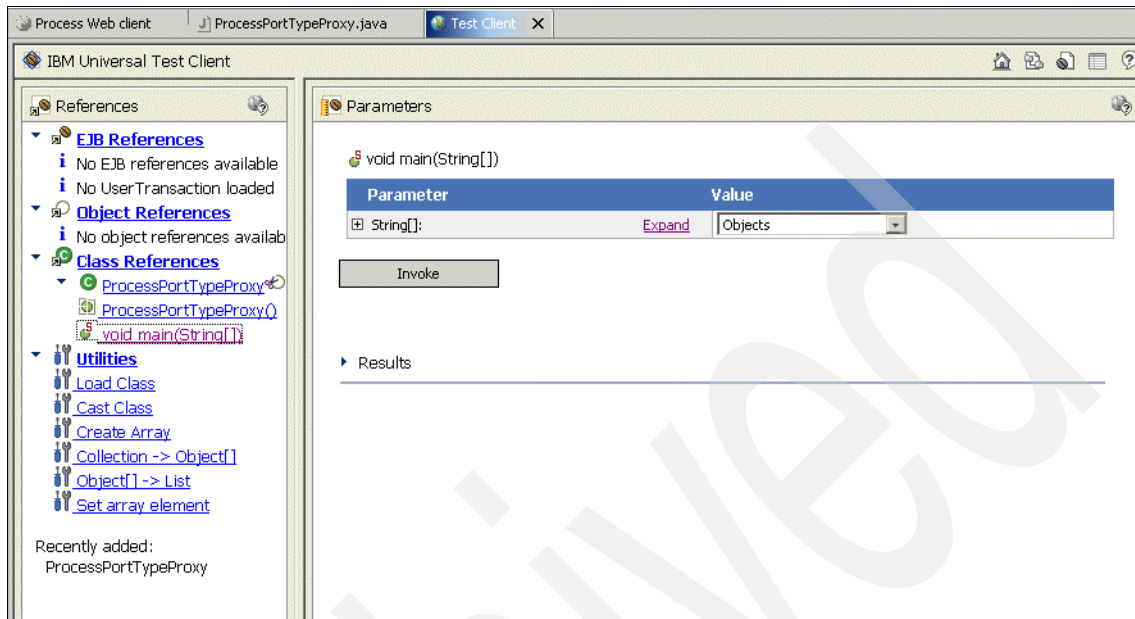


Figure 31-57 Invoke the main method of the proxy class

17. No output is returned by the proxy. However, there are a few locations where we can verify what happened. The quickest test is to look at the console and verify if the print statement has happened or if the console is filled with exceptions. Hopefully, you see the following:

```
SystemOut    0 Output: process has been started successfully
```

This message means that the instance was created successfully. Thus, we should have an activity in our To Do list.

18. Log on to the Process Web Client and verify your To Do list.

Process Web client X

Test Client

WebSphere Business Integration Server Foundation

Process choreographer

IBM

Help

UserID db2admin Logout

Work Item Lists

My To Dos

Define Work Item List

Process Instance Lists

Created By Me

Administered By Me

Undo Actions in Error

Define Process Instance List

Process Template Lists

My Templates

Define Template List

Administration

Manage Work Items for Process Instances

Manage Activities for Process Instances

My To Dos

Use this page to work on work items in your To Do list

Available Actions

View

Claim

Complete

Cancel

Restart

Force Complete

Transfer

	To Do Name	State	Owner	Reason	Activated	Process	Template Name
<input type="checkbox"/>	Staff	Ready	Potential Owner	2/1/05 11:54:47 PM	Pl90030101.d16b9228.5054d5f6.1f210000	MaintenanceProcess	

Figure 31-58 Activity in To Do list

19. Click the process link to verify the details.

Note: The user ID that has created the process is not db2admin this time. Note also that the process output is available while the process instance is still running.

Work Item Lists

[My To Dos](#)
[Define Work Item List](#)

Process Instance Lists

[Created By Me](#)
[Administered By Me](#)
[Undo Actions in Error](#)
[Define Process Instance List](#)

Process Template Lists

[My Templates](#)
[Define Template List](#)

Administration

[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Process Instance

Use this page to display information about the process and, optionally, to act on the process [\[i\]](#)

Available Actions

[Compensate](#)
[Terminate](#)
[Delete](#)
[Monitor](#)
[Repair Compensation](#)

Process Description

Process Instance Name

Template Name

Description

Starter

Readers

Administrators

_Pt:90030101.d16b9228.5054d5f6.1f210000
State
Running

Started
2/1/05 11:54:43 PM

admin

Everybody

Everybody

Process Input Message

contents.actualCompletion

contents.apartment

contents.createDate

contents.description

contents.expectedCompletion

contents.maintenanceId

contents.statusDescription

contents.tenant

1

Today

leak in roof

Tomorrow

2

100

Process Output Message

contents

process has been started successfully

My To Dos

Name	State	Owner	Activated	Completed
Staff	Ready		2/1/05 11:54:47 PM	

Figure 31-59 Details of process instance

20. Click Monitor to look at finished activities.

Chapter 31. Integrating and automating the process 817

WebSphere

Business Integration
Server Foundation

Process choreographer

IBM

Help

UserID db2admin [Logout](#)

Work Item Lists

[My To Dos](#)
[Define Work Item List](#)

Process Instance Lists

[Created By Me](#)
[Administered By Me](#)
[Undo Actions in Error](#)
[Define Process Instance List](#)

Process Template Lists

[My Templates](#)
[Define Template List](#)

Administration

[Manage Work Items for Process Instances](#)
[Manage Activities for Process Instances](#)

Process Instance Monitor

Use this page to view the status of the activities in a process instance [i](#)

Process Description

Process Instance Name	_PI:90030101.d16b9228.5054d5f6.1f210000	State	Running
Template Name	MaintenanceProcess	Started	2/1/05 11:54:43 PM
Description			
Starter	admin		
Readers	Everybody		
Administrators	Everybody		

Activities

To Do Name	State	Activity Kind	Activated
Staff	Ready	Staff	2/1/05 11:54:47 PM
Reply	Finished	Reply	2/1/05 11:54:47 PM
Receive	Finished	Receive	2/1/05 11:54:44 PM

Figure 31-60 Monitoring a process instance started by the proxy

31.2.8 Adapting the EJB

Now that the proxy is tested, you can make the final change. You need to copy the data that is part of the JMS message from the adapter into the structure MaintenanceRequest that is part of the process interface. Setting the contents of this structure is something you did already in the proxy. So, you can copy that code. You just need to parse the incoming message.

1. Open the Java class RM_MaintenanceAgentDeliveryBean.java, which is the class that contains a method per supported verb of the business object RM_Maintenance.
2. Copy the lines of code in the try-catch block in Figure 31-55 on page 814 to the Create method.
3. Replace the constant values with the appropriate getters on the variable argBodyPart.
4. Save the changes

Example 31-5 shows a sample from our scenario.

Example 31-5 Updated Create method

```
public void RM_MaintenanceCreate(
    com
        .ibm
        .www
        .websphere
        .crossworlds
        ._2002
        .BOSchema
        .RM_Maintenance
        .RM_MaintenanceElement argBodyPart) {
    // user code begin {method_content}
    System.out.println("<<<< Message arrived for Create >>>>");
    String resp;
    try {
        ProcessPortTypeProxy aProxy = new ProcessPortTypeProxy();

        // user code begin {proxy_method_calls}
        MaintenanceRequest rq = new MaintenanceRequest();
        rq.setApartment(argBodyPart.getApartmentId());
        rq.setTenant(argBodyPart.getTenantId());
        rq.setDescription(argBodyPart.getProblemDescription());
        rq.setMaintenanceId(argBodyPart.getId());
        rq.setExpectedCompletion(argBodyPart.getExpectedCompletion());

        resp = aProxy.InputOperation(rq);
        System.out.println("Output: " + resp);
    } catch (WSIFException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return;
    // user code end
}
```

31.2.9 Deploying and testing the changes

You are ready to test the last piece of the solution. Follow these steps:

1. Stop the Test server.
2. Publish the changes to the server.
3. Start the server.

4. Use `rfhutil` to write a message on the delivery queue. Use the sample file `RedContract_Delivery_Create` in the Additional Materials Directory.
5. Inspect the console of the Test server. You should see lines similar to Example 31-6.

Example 31-6 Output of the EJB on the console

```
SystemOut      0 <<<< Message arrived for Create >>>>
SystemOut      0 Output: process has been started successfully
```

6. Log on to the Process Web client, and verify your To Do list. There should be a new entry.
7. Click the process link to inspect the details of the process instance, shown in Figure 31-61. Notice the different data, which is this loaded from the file.

Work Item Lists

- [My To Dos](#)
- [Define Work Item List](#)

Process Instance Lists

- [Created By Me](#)
- [Administered By Me](#)
- [Undo Actions in Error](#)
- [Define Process Instance List](#)

Process Template Lists

- [My Templates](#)
- [Define Template List](#)

Administration

- [Manage Work Items for Process Instances](#)
- [Manage Activities for Process Instances](#)

Process Instance

Use this page to display information about the process and, optionally, to act on the process [\[i\]](#)

Available Actions

[Compensate](#) [Terminate](#) [Delete](#) [Monitor](#) [Repair Compensation](#)

Process Description

Process Instance Name	_Pl:90030101.d18e6ce3.5054d5f6.79770000	State	Running
Template Name	MaintenanceProcess	Started	2/2/05 12:32:47 AM
Description			
Starter	admin		
Readers	Everybody		
Administrators	Everybody		

Process Input Message

contents.actualCompletion	
contents.apartment	1
contents.createDate	
contents.description	Crack in the front window
contents.expectedCompletion	Mar 10, 2005
contents.maintenancelid	2
contents.statusDescription	
contents.tenant	100

Process Output Message

[contents](#) process has been started successfully

My To Dos

Name	State	Owner	Activated	Completed
Staff	Ready		2/2/05 12:32:51 AM	

Figure 31-61 Details of a process instance

8. Verify that all connectors are running, that the message flows are running, and that the back-end application is running.
9. Use the Maintenance Web application to create a new maintenance request. Do you get a process instance in the Test server?

31.3 Building the receive process

The full process is not yet finished. When the maintenance reviewer has sent a request to the contractor, the contractor eventually sends a response back to RedMaint. This response might be a progress update or a message indicating that the request has been completed. These messages arrive on the queue XML_IN. Somebody needs to pick them up and pass them on to the back-end application.

We have limited the instructions to a minimum in this section. You have performed all the tasks that are required to complete this section before.

31.3.1 Creating the service WBICReceive

Similar to the WBICSend service, you need to create a WBICReceive service. Use the exact same steps as in 31.1.2, “Building the send service” on page 761. The only difference is the queue name, XML_IN. You also need to define the queue destination in JNDI.

31.3.2 Building the EJB to process the incoming message

The next step is to create an EJB to process the incoming response from RedContract. As before, build this EJB based on the WSDL of the WBICReceive service. For assistance with this step, refer to 31.2.1, “Generating the EJB that is called by an MDB” on page 785. You should use a different EJB project.

31.3.3 Building the message-driven bean

Similar to 31.2.2, “Generating the MDB” on page 794, create an MDB that is based on the WSDL of the EJB and the WSDL of the WBICReceive service. Do not forget to correct the JNDI names in the deployment descriptor and the listener port. Correct also the listener port and define this listener port to the Test server.

31.3.4 Generating proxy for adapter

This time, the EJB does not need to invoke a process. It is sufficient to invoke the adapter. Generate a service proxy for the adapter. Use the same test procedure as explained in 31.2.7, “Testing the proxy and the adapted process model” on page 812.

31.3.5 Updating EJB to call adapter proxy

When the proxy has been tested, copy the appropriate lines of code in the EJB. Populate the RM_Maintanance business object based on the incoming message from the RedContract company, and invoke the update method of the adapter.

31.3.6 Deploying and testing

Add the new project to the server. Use the sample ContractorResponse.xml to build a response message that matches the data in the database. Send this response from WebSphere Business Integration Connect Express to WebSphere Business Integration Connect. Watch this response being picked up by the MDB that invokes the EJB who invokes the adapter service proxy. The service proxy writes the message to the queue of the adapter.



Part 8

Looking after the run-time environment



Gathering data from the run-time

Now that we have successfully deployed the integration solution, this chapter explores a method of gathering statistics and run-time data for the adapter environment.

32.1 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 that are 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 that is 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 that is provided to it. Thus, you can establish what activities are being run and where any potential bottlenecks in the business process might 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 that are 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.

The WebSphere Business Integration Monitor 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 run-time.
- ▶ 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.

Figure 32-1 illustrates these components.

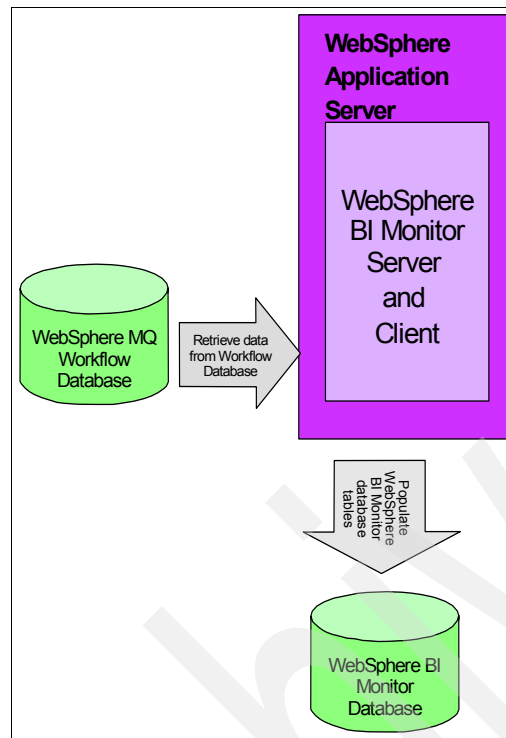


Figure 32-1 High level architecture: WebSphere Business Integration Monitor

These components can be accessed by both Monitor Portlets or by a traditional J2EE client and allow you 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 that 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 run-time 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. When 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 common frame 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.

When 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 install WebSphere Business Integration Monitor V4.2.4 including FixPack 3. We also install WebSphere Business Integration Modeler V5.1.1 to show our business process which was built for integration of 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.

32.2 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 that we created for our RedTenant process, do the following:

1. Open the WebSphere Business Integration Modeler by selecting **File** → **Import**, as shown in Figure 32-2.

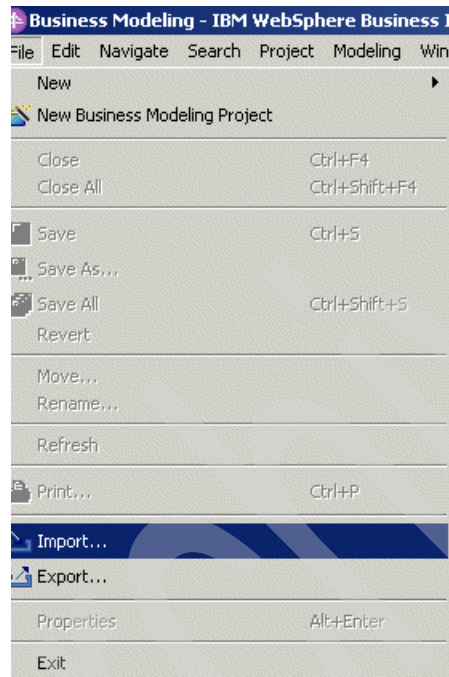


Figure 32-2 Import

2. Select **WebSphere Business Integration Modeler Import**, as shown in Figure 32-3 on page 831.

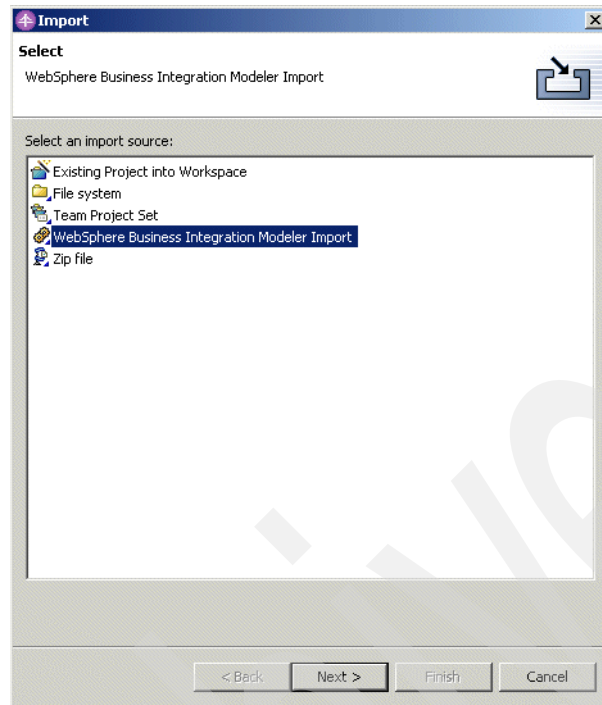


Figure 32-3 Select import type

3. Select **WebSphere Business Integration Modeler v4.2.4 (org)** because we are importing an organization file exported from V4.2.4 (Figure 32-4 on page 832).

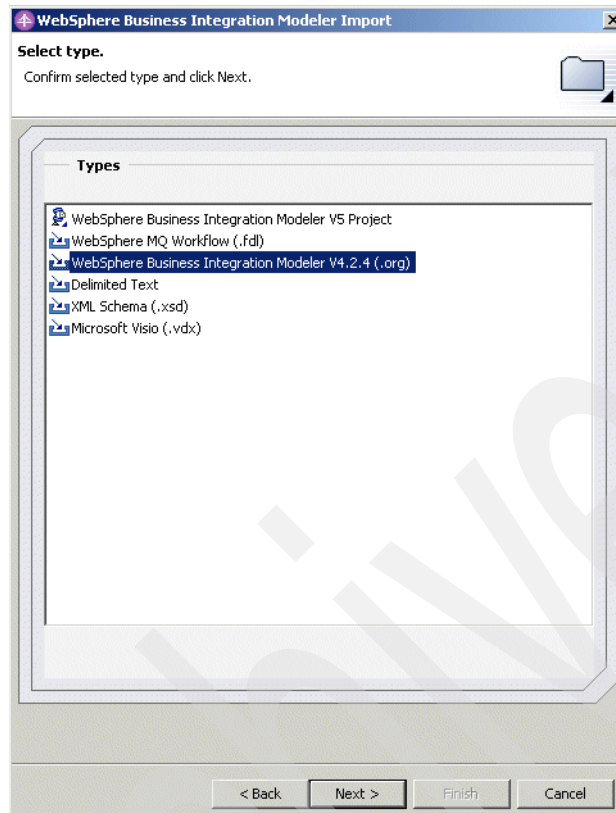


Figure 32-4 Select import source

4. 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 (Figure 32-5 on page 833).
5. Click **Finish**.

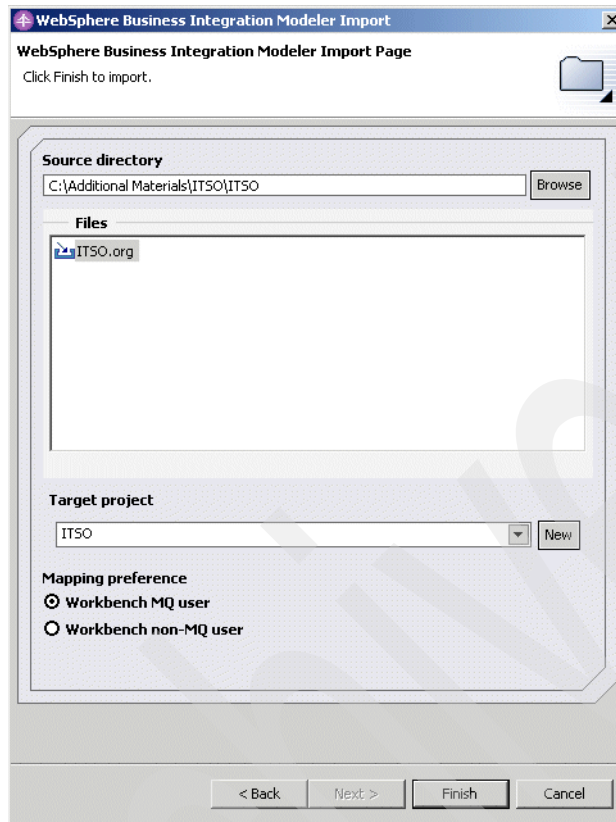


Figure 32-5 Select location

6. Select to **create a new project**. Name it ITSO when requested.
7. The new project is created for the import (Figure 32-6).

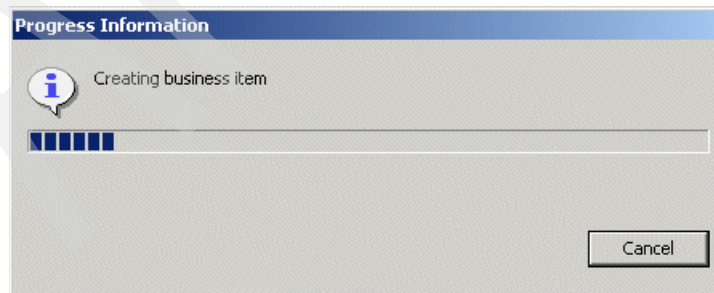


Figure 32-6 Creating project

8. When the project is created successfully (Figure 32-7), the import of the org file continues.

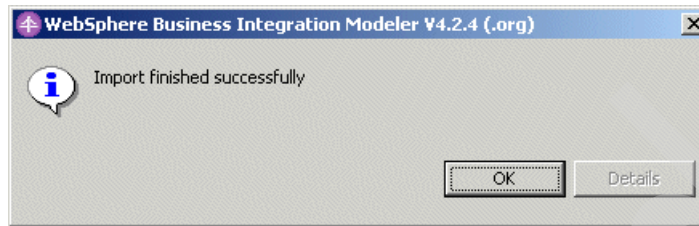


Figure 32-7 Successful completion

9. Figure 32-8 shows that we have a Business Item named RedTenant. This is the data structure that we populate as part of our process. You can also see a process named RedTenant.

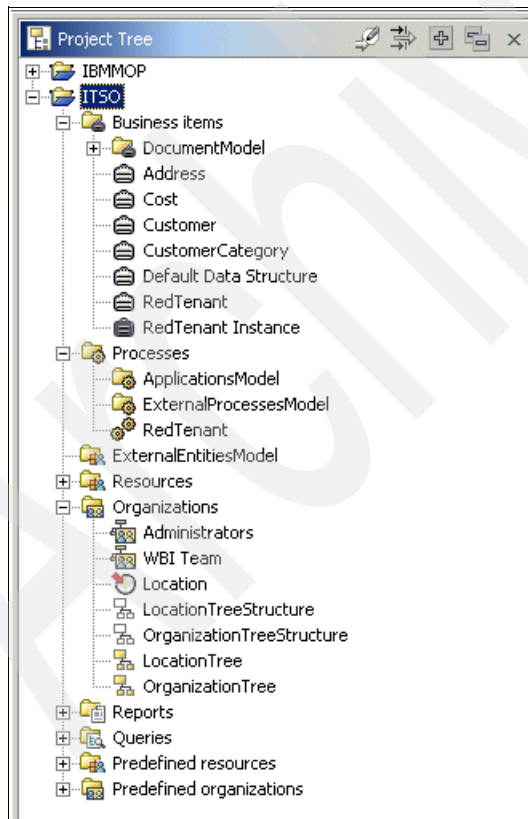


Figure 32-8 New project

10. Double-click **RedTenant** to open it.

Figure 32-9 shows that our process definition consists of two activities.

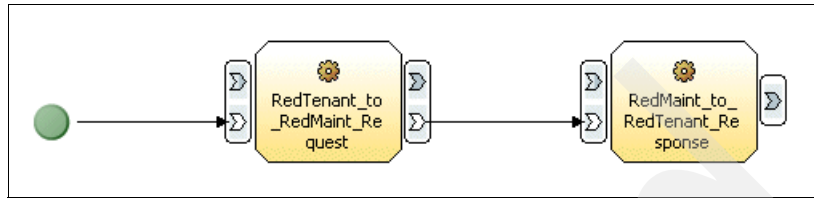


Figure 32-9 Process diagram

Figure 32-10, Figure 32-11 on page 836, and Figure 32-12 on page 837 shows elements as they were defined in the original V4.2.4 of the Modeler to illustrate the components that we created specifically for our process and feeding the Monitor from the Message Broker.

Figure 32-10 shows that our process consists of two tasks: RedTenant_to_RedMaint_Request and RedMaint_to_RedTenant_Response, which correspond to the name of our two primary Message Flows.

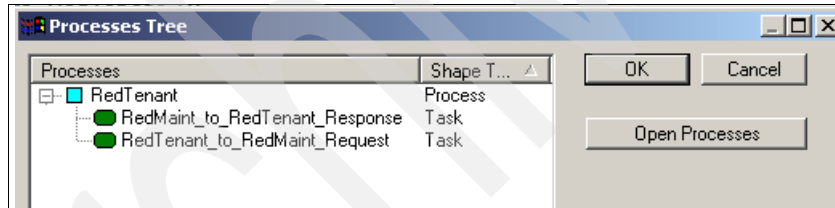
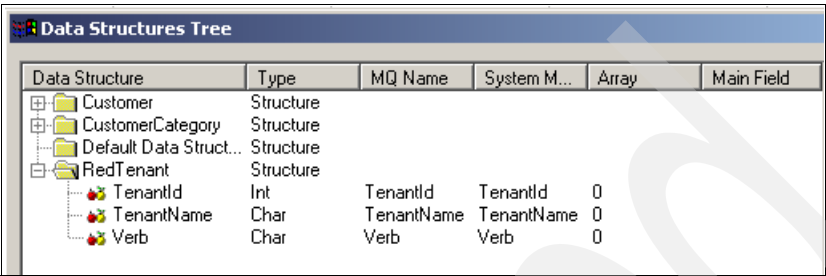


Figure 32-10 Process tree

Figure 32-11 shows the RedTenant data structure we populate as part of the process, which is the data that we feed to the Monitor along with our process and task information. The structure corresponds to a message definition that we will use in the Message Broker.



The screenshot shows a window titled "Data Structures Tree". It contains a tree view on the left and a table on the right. The tree view shows a hierarchy: "Customer" (Structure), "CustomerCategory" (Structure), "Default Data Struct..." (Structure), and "RedTenant" (Structure). Under "RedTenant", there are three items: "TenantId" (Int), "TenantName" (Char), and "Verb" (Char). The table on the right has columns: "Data Structure", "Type", "MQ Name", "System M...", "Array", and "Main Field". The rows correspond to the items in the tree view.

Data Structure	Type	MQ Name	System M...	Array	Main Field
Customer	Structure				
CustomerCategory	Structure				
Default Data Struct...	Structure				
RedTenant	Structure				
TenantId	Int	TenantId	TenantId	0	
TenantName	Char	TenantName	TenantName	0	
Verb	Char	Verb	Verb	0	

Figure 32-11 Data structure

Figure 32-12 on page 837 shows the business measures that we 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 collected for analysis by the Monitor. So, for our purposes, these measures are sufficient to illustrate the logistics.

Figure 32-12 Business measures

The file we 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.

32.3 Setting up the Monitor environment

You need to set up the Monitor database schema prior to import the business process. This setup must always be performed by the administrator prior to any use of the Monitor.

To set up the Monitor environment:

1. Ensure that the Monitor Web application is running.
2. Enter the Monitor Administration login screen, which is normally as follows:

`http://<host>/monitor/admin/index.cmd`

Address http://localhost/monitor/admin/setup.cmd

WebSphere Business Integration Monitor

Monitor Database Setup

Enter your login user name and password, and then click Create Database.

User Id:

Password:

Time Zone:
(GMT-05:00) Eastern Standard Time (America/New_York) ▼

Figure 32-13 Admin login

3. You can see whether the database schema is set up. In our example, shown in Figure 32-13, the database schema has not been set up because there is a button to create the database.
4. Log in as the administrator. The method of authentication varies depending on how you have installed the Monitor. In our case, we use the default, which is to use the workflow administrator user and password (admin/password).
5. Click **Create Database**.
6. When the database create has completed successfully, you arrive at the main administration screen, as shown in Figure 32-14 on page 839.

From here, you perform a some basic set up to get you going.

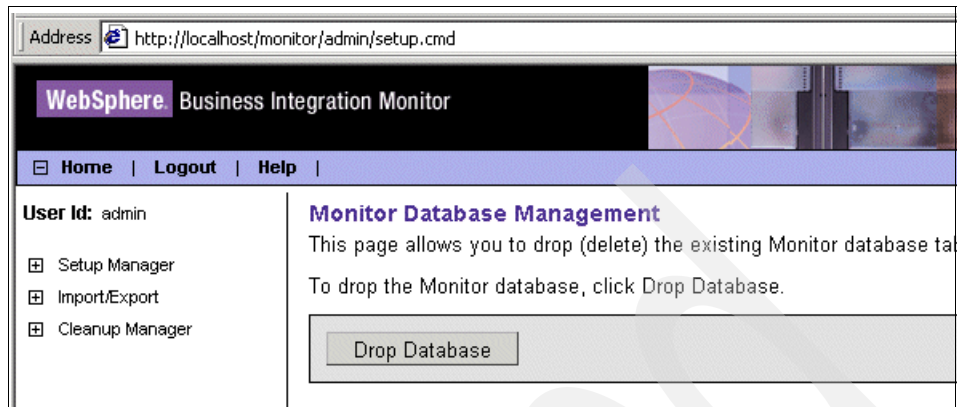


Figure 32-14 Main administration screen

7. On the left, select **Setup Manager** → **Event Queue** (Figure 32-15 on page 840).

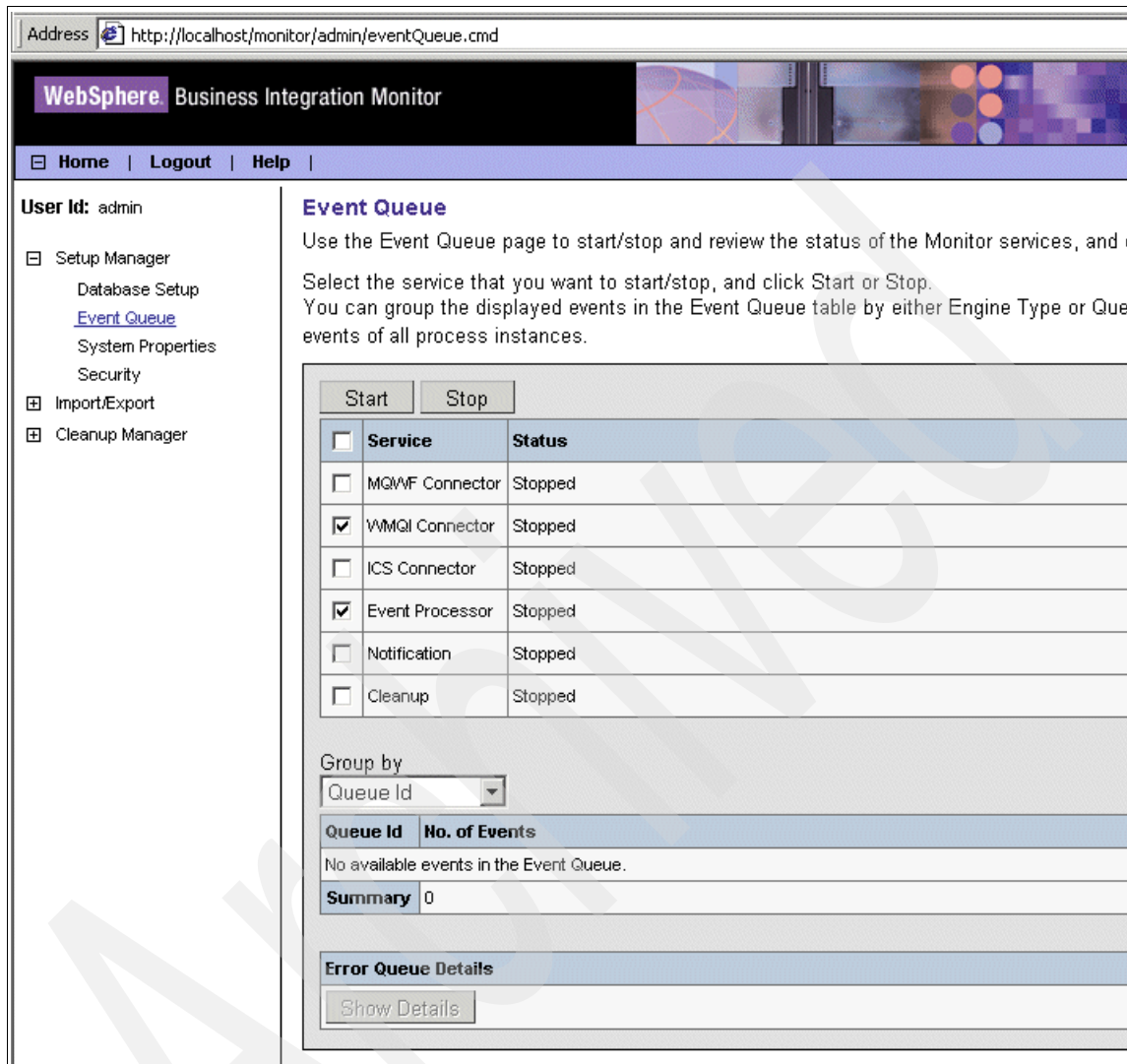


Figure 32-15 Event queue management

- For this exercise, start the WMQI Connector and the Event Processor by selecting these and clicking **Start**.
- Select the **System Properties** (Figure 32-16 on page 841).

Address <http://localhost/monitor/admin/systemProperties.cmd>

WebSphere Business Integration Monitor

Home | Logout | Help

User Id: admin

- Setup Manager
 - Database Setup
 - Event Queue
 - [System Properties](#)
 - Security
- Import/Export
- Cleanup Manager

System Properties

Enter or modify the Monitor client and server properties. Monitor Service new values are effective once the system is restarted.

Notification

☐ **Email Notification**

SMTP host

SMTP sender

☐ **Automatic Corrective Action**

Process name

User Id

☐ Password

☐ Credential

Monitor Client

Refresh rate sec

Monitor Service (The new settings will be in action after restarting the Monitor server.)

Cleanup agent refresh rate min

Time-based notification refresh rate min

Minimum event processor wait msec

Maximum event processor wait msec

Events count

Event Processors Count

OK

Figure 32-16 System properties

- Review the system properties and modify any if required. For this exercise, the defaults are sufficient.
- Import our business process and business measures by selecting **Import / Export** → **Import Organization**.
- Browse to the RedTenant.xml file in the Additional Materials ITSO folder, as shown in Figure 32-17 on page 842.
- Select **Import**.

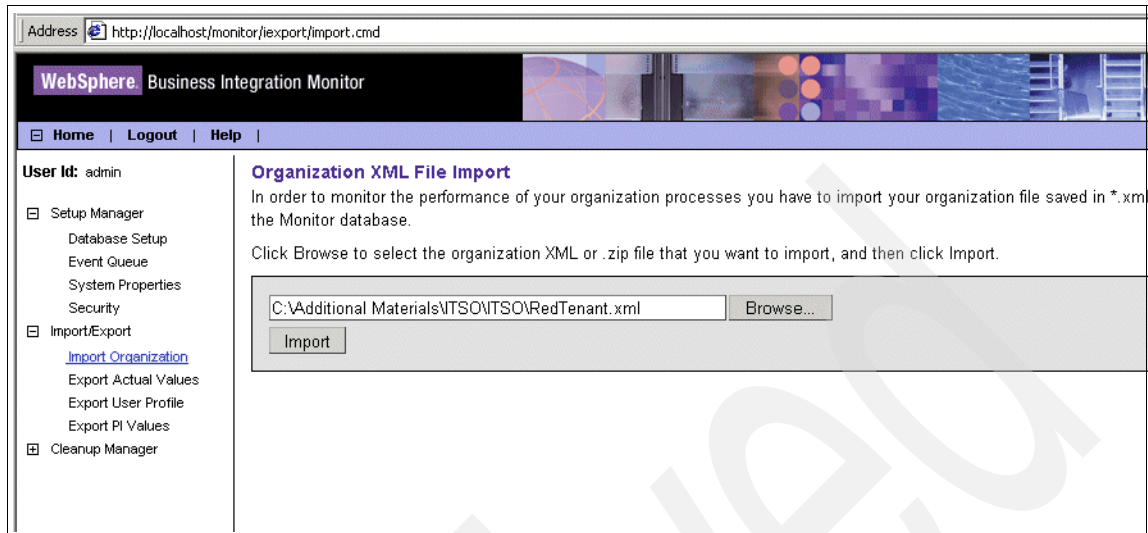


Figure 32-17 Import file

- When the import has completed successfully, verify that the process was imported correctly, as shown in Figure 32-18.

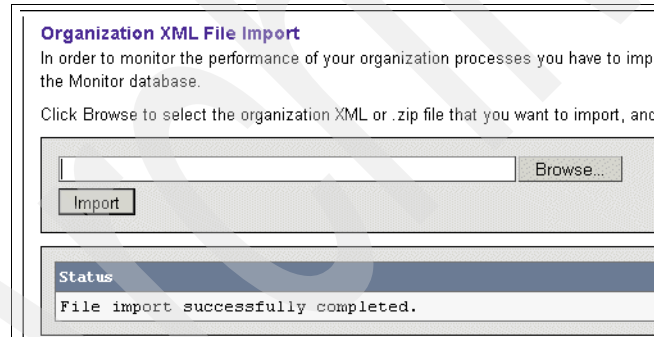


Figure 32-18 Import complete

- Navigate back to the Cleanup Manager, as shown in Figure 32-19 on page 843.

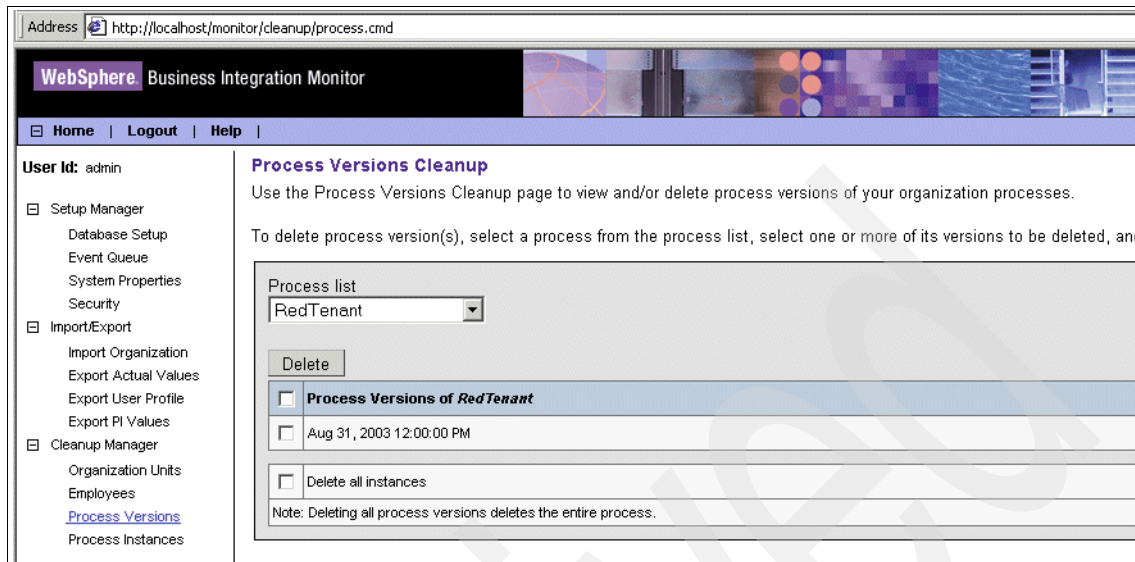


Figure 32-19 Cleanup manager

16. Select **Process Version**.

17. Use the drop-down list of Process Lists to verify that the RedTenant process is in the database.

You now have the basics in place. From here, you can make the required modifications to the Message Flows to enable data to be sent to the Monitor.

32.4 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.

Typically, to integrate WebSphere Business Integration Message Broker with the Monitor, you must work with the WebSphere Business Integration Modeler. The Modeler is used to construct the high-level view of the business process. This high-level view includes the data structures that the business process uses. When 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.

For the Monitor to integrate with WebSphere Business Integration Message Broker, it needs a common frame of reference, which is supplied by the export files from Modeler. In this instance, the XSD file that relates to the message structures for business processes needs to be exported from Modeler and into WebSphere Business Integration Message Broker. When this is complete, data sent from the Emitter Node can then be interpreted by the Monitor in a meaningful way. In our example, we installed the Emitter Node into the Message Broker Toolkit according to the instructions.

Note: The message set that we use and the subflows for the request and response flows are included in the Additional Materials.

The EmitterMessages Message Set Project contains the message for the data that we will emit. 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. This is discussed in more detail when we see the ESQL in Example 32-1 on page 847.

To set up the Message Broker environment:

1. Import the Message Set into your Broker Workspace.
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, which is the default name of the Monitor database (Figure 32-20 on page 845).

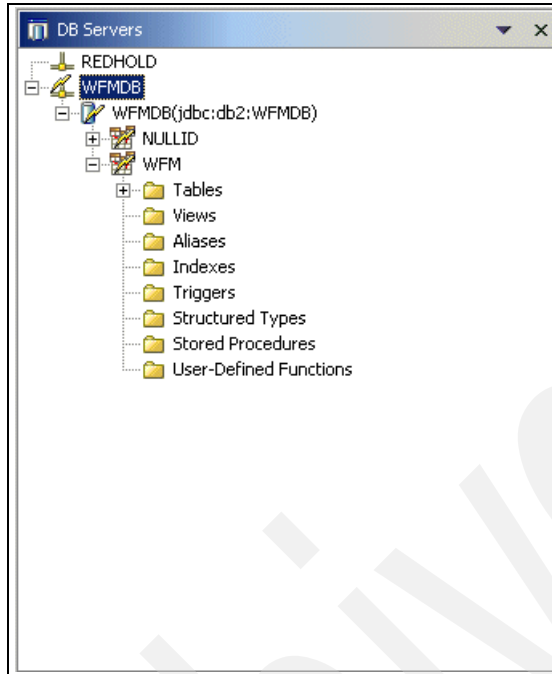


Figure 32-20 Data source WFMDB

4. Copy this data source to your RedMaintenanceFlows project.
5. EmitterFlows contains the two subflows that we 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 32-21) to open it in the Message Flow Editor.

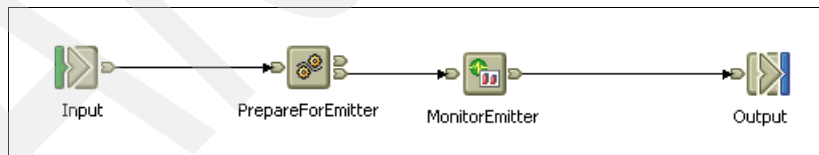


Figure 32-21 MonitorEmitter

7. Open the Compute node properties, as shown in Figure 32-22.

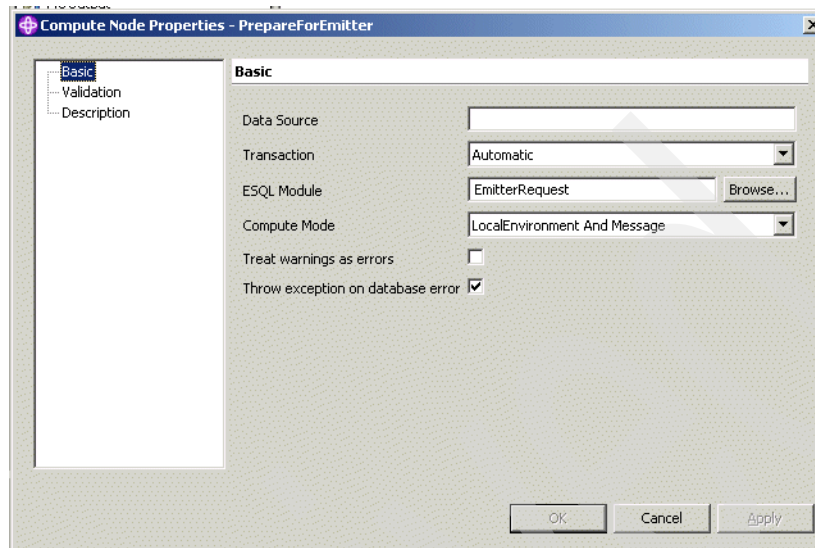


Figure 32-22 Compute node

8. The Compute Mode must be set to LocalEnvironment and Message.

9. The ESQL Module is EmitterRequest.

10. Open the ESQL that is shown in Example 32-1 on page 847.

The ESQL performs the following:

- Declares namespaces for our business object messages.
- Sets the Emitter data for each of the different type of messages that pass through.
- Passes the Input message to the output message. (In our scenario, we do want to modify the incoming message in any way, so we merely pass it on.)
- Sets the Emitter values in the Environment.
- Sets the MEmitter Task value, which is the process identifier in the Monitor.

Important: The MEmitter Task value *must* be unique for each message that passes through this flow. If the Monitor receives a duplicate task, it merely ignores it.

However, because we want to record the full round trip, we need to ensure that this value is correlated with an identical value on the response trip. We can ensure that the value is identical by using the ObjectEventId, which is unique for every new business object that is sent by the JMSAdapter.

- Sets the MEmitter business model solution name (BM_SOLUTIONNAME) to the name of our process as it is known to the Monitor, RedTenant.
- Sets 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 32-19 on page 843.
- Sets up the data to be emitted, which depends entirely on what is available at the time. We complete what we can here, and we complete the rest of the data when it is available in the return message. In the response flow, see Example 32-2 on page 852.

Example 32-1 EmitterRequest

```
CREATE COMPUTE MODULE EmitterRequest
```

```
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        -- Red Tenant Web Application
        DECLARE Web_tenant NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant';
        DECLARE Web_tenant_tenant NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant';
        DECLARE Web_newMaintenance NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance';
        DECLARE Web_newMaintenance_newMaintenance NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_newMaintenance_newMaintenance';

        DECLARE RM_Tenant NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';
        DECLARE RM_Maintenance NAMESPACE
        'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance';

        -- Set Emitter Data for a TENANT MESSAGE TYPE from Web Application
        -- *****

        IF InputRoot.Properties.MessageType = 'Web_tenant' THEN
            SET OutputRoot = InputRoot;
```

```

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.TenantId =
InputRoot.MRM.Web_tenant:ROOT.Web_tenant_tenant:Web_tenant_tenant:Web_tenant_tenant:name;
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;
set Environment.MEmitter.System.TenantName =
InputRoot.MRM.Web_newMaintenance:ROOT.ns:Web_newMaintenance_newMaintenance.ns:tenantId;
--
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;

```

```

        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;

```

11. Open the Properties of the MonitorEmitter node, as shown in Figure 32-23.

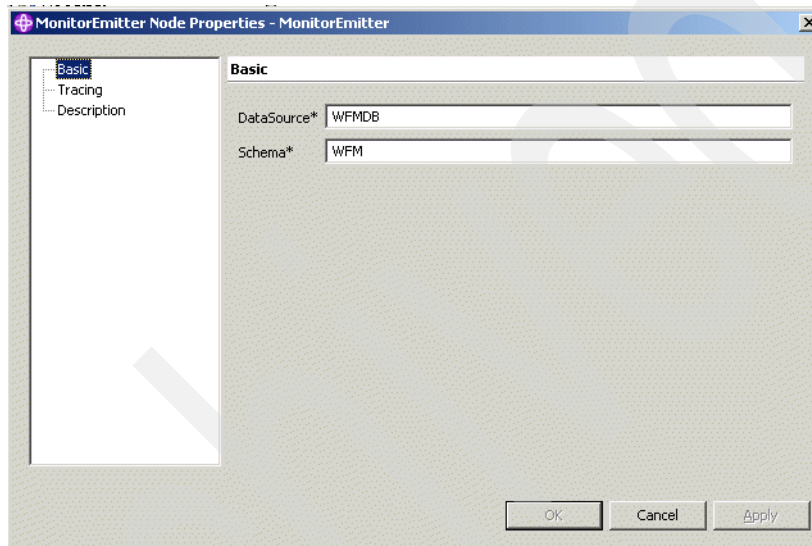


Figure 32-23 MonitorEmitter properties

- 12. The Datasource is the WFMDB data source
- 13. The Schema is WFM, the default schema for the Monitor at installation.
- 14. Select the **Tracing** properties.

15. We have chosen to create traces of the Emitter flows. Chose the c:\RedTenantTraces directory as the target for the traces (Figure 32-24).

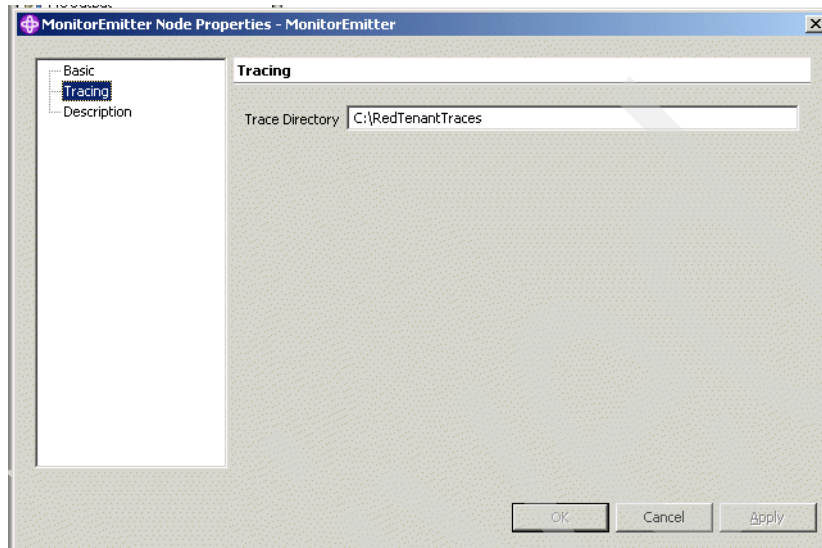


Figure 32-24 Trace properties

16. Select the EmitterResponse flow, as shown in Figure 32-25.

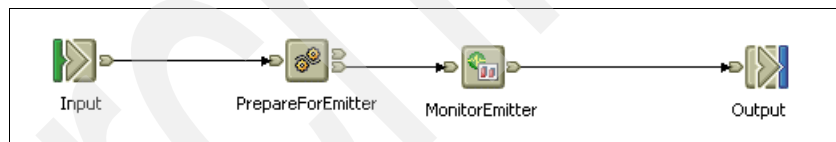


Figure 32-25 Emitter response

17. We set the Basic properties of the Compute node as before, this time using the EmitterResponse ESQL Module (Figure 32-26 on page 852).

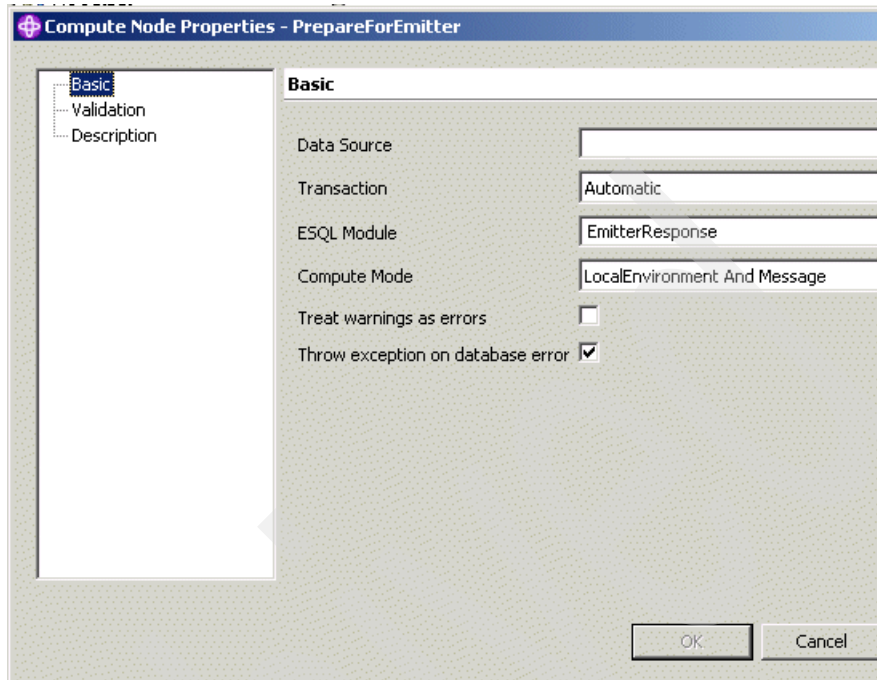


Figure 32-26 Compute node

18. Complete the values of the Customer message to be emitted with the return values from the back-end application (Example 32-2). It is important that you also set the Emitter task to the ObjectEventId to tie the request flow and the response flow together for the business process in the Monitor.

Example 32-2 EmitterResponse

```
CREATE COMPUTE MODULE EmitterResponse

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
  -- Red Maintenance Back-end Application
  DECLARE RM_Tenant NAMESPACE
  'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant';
  DECLARE RM_Maintenance NAMESPACE
  'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance';
  DECLARE RM_Apartment NAMESPACE
  'http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment';

  -- 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.
20. To modify the request and response flows to include the call to the Emitter subflows, 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 32-27 on page 855.
23. Save the Message Flow.

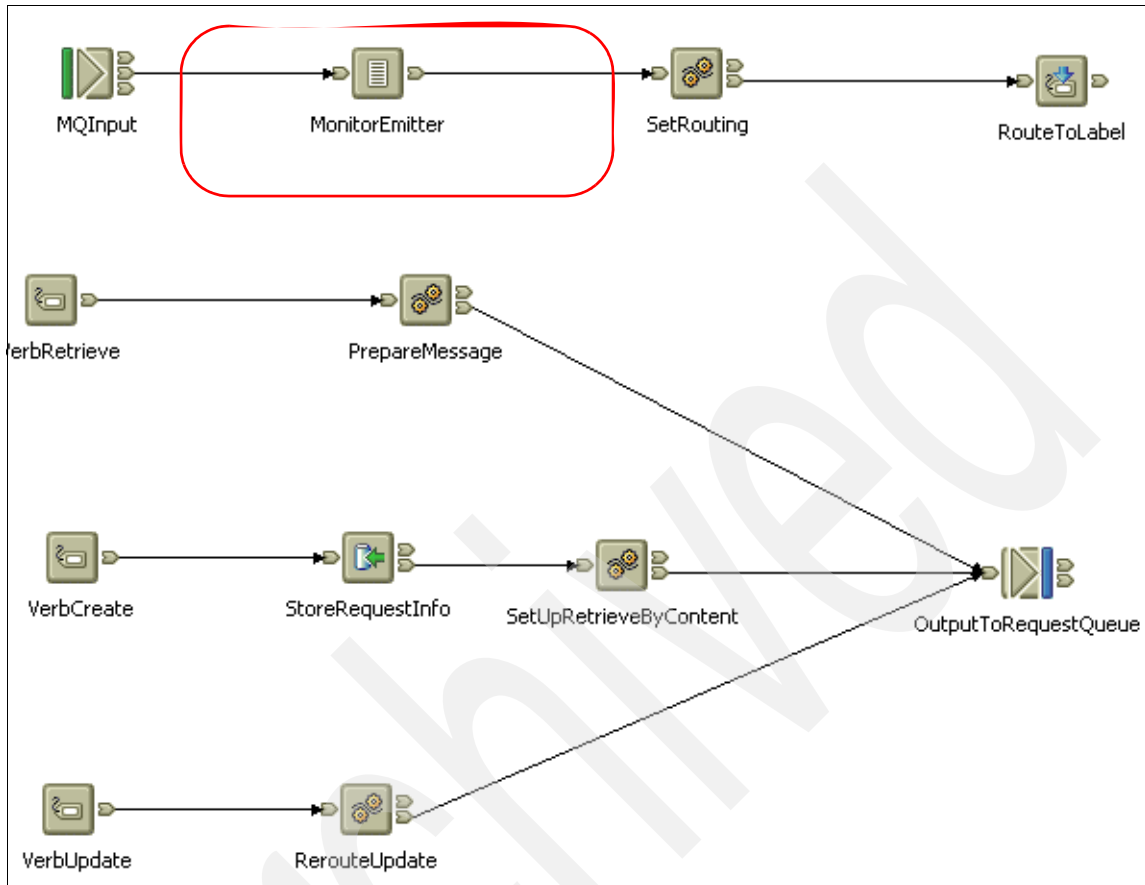


Figure 32-27 RedTenant_to_RedMaint_Request

24. Open the RedMaint_To_RedTenant_Response Message Flow.
25. Insert the MonitorEmitterResponse subflow between the MQInput and the SetRouting Compute node.
26. Connect the nodes as shown in Figure 32-28 on page 856.
27. Save the Message Flow.

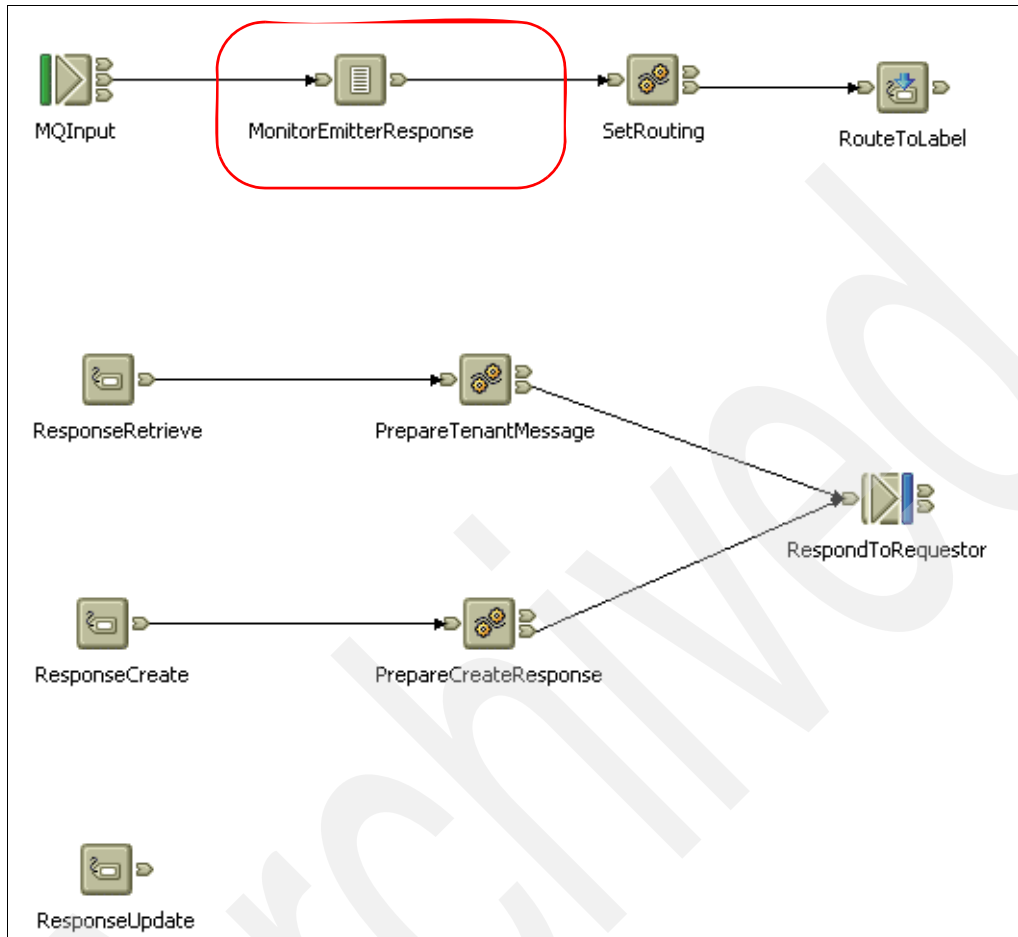


Figure 32-28 RedMaint_to_RedTenant_Response

As 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 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, because we have sent the response back through this flow, our emitter completes the trip and merely throws the response away, as opposed to the Retrieve and Create where we respond to the requestor.

32.5 Using the Monitor data

You now need to populate the Monitor database with some data. To do this, use the solution thus far to send retrieves, creates, and updates to populate event data in the monitor.

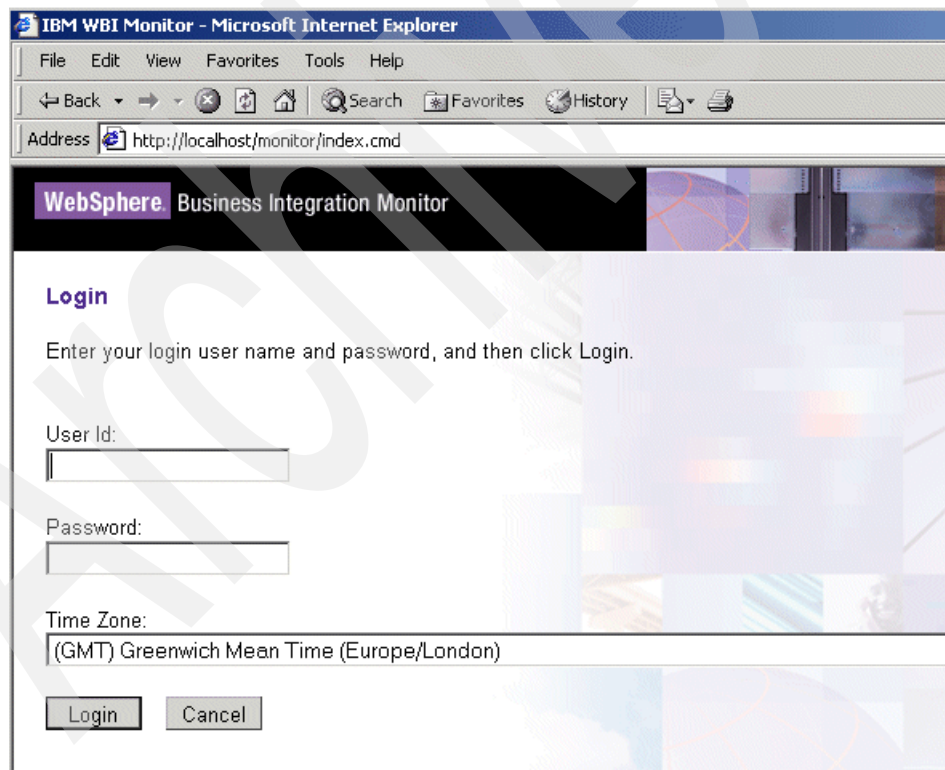
32.5.1 Workflow Dashboard

The Workflow Dashboard allows you to track processes, work items, and business performance measures in real time.

To start the Workflow Dashboard:

1. Ensure that the Monitor is running.
2. Enter the Web address for the WebSphere Business Integration Monitor:

`http://localhost/monitor/index.cmd`



The screenshot shows a Microsoft Internet Explorer browser window titled "IBM WBI Monitor - Microsoft Internet Explorer". The address bar displays "http://localhost/monitor/index.cmd". The page content includes the "WebSphere Business Integration Monitor" header. Below the header is a "Login" section with the instruction "Enter your login user name and password, and then click Login." There are three input fields: "User Id:" with a text box, "Password:" with a text box, and "Time Zone:" with a dropdown menu showing "(GMT) Greenwich Mean Time (Europe/London)". At the bottom of the login section are two buttons: "Login" and "Cancel".

Figure 32-29 Monitor login

3. Log on using administrator privileges and your password.
4. Select **Workflow Dashboard**. Look first at the events that you are sending to ensure that they are making it through to the Monitor.

Workflow Dashboard

Use the Workflow Dashboard to view business measures values for the process instances currently running in your organization processes.

Select a saved view or create a new one.

Saved views

Running

Configure the process instances table by selecting the business process you want to monitor, the business measures you want to view, and the filter that specifies the set of process instances to be displayed in the table.

Process

RedTenant

Business Measures

Set Filter

Specify the process instances (items) you want to see on this page.

Sample

% Items 1 to 3 of 3

Go

Preview

View

Export All

Export Page

Item	TenantId	TenantName	Verb	Starting Time	Elapsed Duration	Status	Process Diagram
1	100	---	Update	Jan 9, 2005 6:43:59 AM	8 d, 12 h, 5 m, 4 s	Running	+
2	100	---	Update	Jan 9, 2005 7:07:08 AM	8 d, 11 h, 41 m, 55 s	Running	+
3	100	---	Retrieve	Jan 11, 2005 2:28:14 PM	6 d, 4 h, 20 m, 49 s	Running	+

Figure 32-30 Workflow Dashboard

5. Use the drop-down list to find our process, RedTenant.
6. Select **Business Measures** to determine which business measures you want to see. See Figure 32-31 on page 859.

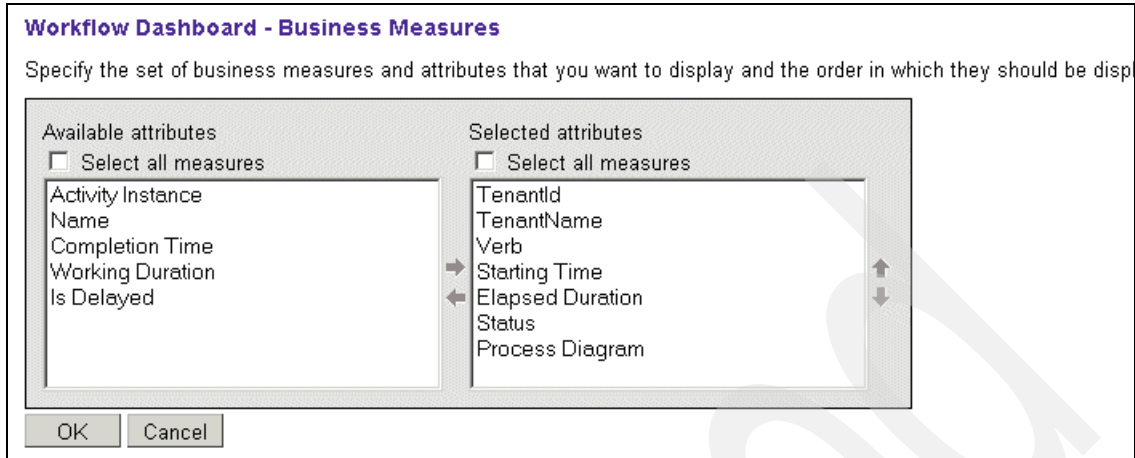


Figure 32-31 Select business measures

7. From this window, select:
 - TenantId, TenantName, and Verb from our emitter message
 - Starting Time
 - Elapsed Time
 - Status
 - Process Diagram

You can select business measures by 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. Set the filter for the activities that you want to see. Click **Set Filter**.

Figure 32-32 and Figure 32-33 show only processes that did not complete successfully. That is, we put a filter on the status and selected only running, terminating, and so forth.

However, it is a good idea to begin by seeing everything. With that in mind, select all statuses by selecting the values and then adding them to the Filter expression.

Set Filter

Use the Filter page to specify the process instances you want to display in the Process Instances table. You can filter by one or more of the available business measures so that only the process instances that have specific business measures values will be displayed.

Filter by
Status

Ready
Running

Add

Filter

You can set the filter by editing (typing/pasting) text, and/or appending constraints. Click Validate to make sure that it is correct, and click OK to save and use the filter that has been validated.

Status = {Running, Suspended, Terminated, Suspending, Terminating}

Figure 32-32 Filter for incomplete

Use the Filter page to specify the process instances you want to display in the Process Instances table. You can filter by one or more of the available business measures so that only the process instances that have specific business measures values will be displayed.

Filter by
Status

Ready
Running

Add

Filter

You can set the filter by editing (typing/pasting) text, and/or appending constraints. Click Validate to make sure that it is correct, and click OK to save and use the filter that has been validated.

Status = {Ready, Running, Suspended, Completed, Terminated, Suspending, Terminating}

OK Cancel Validate Clear AND OR NOT ()

Figure 32-33 Filter on status

11. Click Save View As icon and give the view a name. We selected All, as shown in Figure 32-34.

Workflow Dashboard

Use the Workflow Dashboard to view business measures values for the process instances currently running.

Select a saved view or create a new one.

Saved views
































All                               

Figure 32-34 Save view

12. Click **Go** to see the number of instances that are available for viewing.

13. Set the number to view all that are available.

14. Click **View**. See Figure 32-35.









Export Page						
Item	Starting Time	TenantId	TenantName	Verb	Process Diagram	Elapsed Duration
1	Jan 18, 2005 8:42:29 AM	102	Redbook Reader	Retrieve		0 s
2	Jan 18, 2005 11:29:53 AM	102	Redbook Reader	Create		1 s
3	Jan 18, 2005 8:44:38 AM	102	Redbook Reader	Retrieve		0 s
4	Jan 18, 2005 8:44:46 AM	102	Redbook Reader	Create		3 s
5	Jan 18, 2005 8:27:07 AM	100	Lee Gavin	Retrieve		11 s
6	Jan 18, 2005 8:42:09 AM	100	Lee Gavin	Retrieve		0 s
7	Jan 18, 2005 11:27:48 AM	102	Redbook Reader	Retrieve		0 s
8	Jan 18, 2005 8:42:21 AM	101	ITSO Resident	Retrieve		0 s

Figure 32-35 Selected process instances

15. The list of process instances that are available displays, based on the filter with the Business Measures that you selected.
16. If you click the Process Diagram icon, you see the graphical representation of the process model, as shown in Figure 32-36. Note this is a V4.2.4 representation.

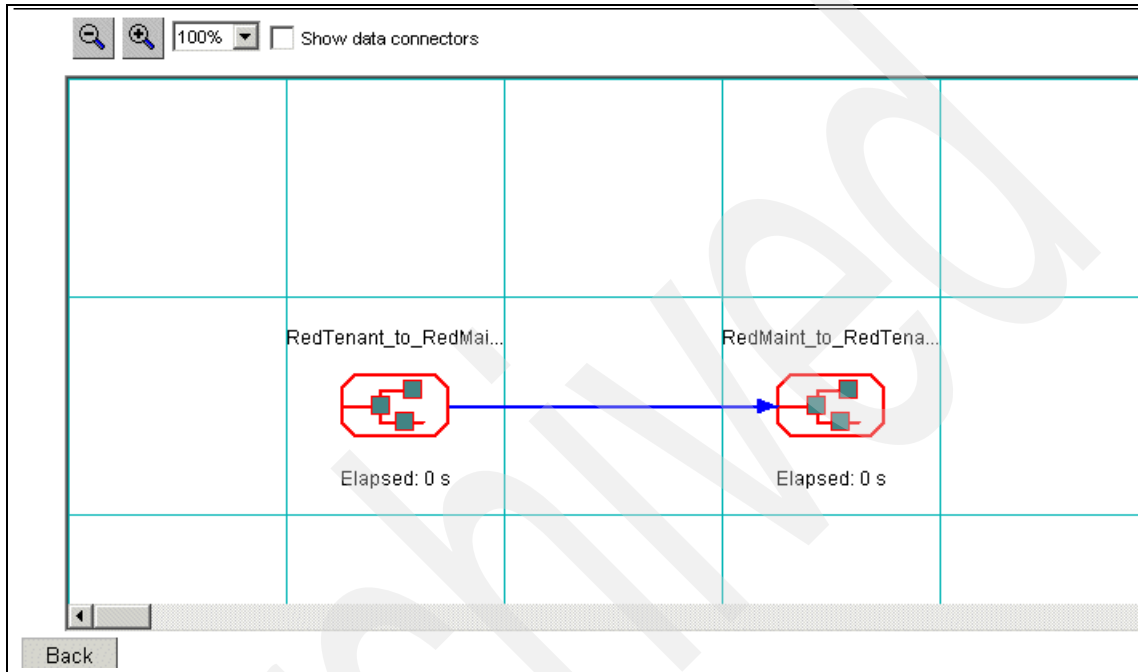


Figure 32-36 Process diagram

17. Run some more processes to check the real-time updates from the Emitter.

32.5.2 Business Dashboard

To start the Business Dashboard:

1. Select **Configuration View** in the Business Dashboard (Figure 32-37).
2. Using the drop-down list, select the **RedTenant** process.
3. Select an analysis type. We chose **Basic Analysis**.
4. Click **Business Measures**.

Business Dashboard

The Business Dashboard allows you to create several analysis reports applied to historical business measures for a given process, organization unit/user.

Select a saved view or create a new one.

Saved views
Basic

Select the business process, and the organization unit/user where you want to analyze historical business measures data. Select the business measure and the type of the analysis to be performed.

Process: RedTenant Analysis type: Basic Analysis Business Measures

Breakdown attribute: Verb Weighting factor: 0.18

Organization unit/user: [ITSO]

Specify the interval within which to perform the analysis, and the time-scale base unit (frequency) you want to use for displaying the report's data.

Figure 32-37 Create view

5. Select a business measure as the basis for the analysis. We chose **Number of new items**, as shown in Figure 32-38.

Business Dashboard

Select the business measures that you want to display in the analysis report, and click Back.

<input type="checkbox"/>	Fn	Business measures
<input checked="" type="checkbox"/>	sum	Number of new items
<input type="checkbox"/>	sum	Number of outstanding items
<input type="checkbox"/>	sum	Number of resolved items
<input type="checkbox"/>	sum	Number of carried over items
<input type="checkbox"/>	avg	Elapsed duration
<input type="checkbox"/>	avg	Working duration

Back

Figure 32-38 Select business measure

6. Click **Back**.
7. Select a breakdown attribute. We chose Verb because we want to see an analysis of historical data that is based on the number of new items per day, by verb.

Specify the interval within which to perform the analysis, and the time-scale base unit (frequency) you want to use for displaying the report's data.

Period from To Frequency

How would you like to view your report? Select a report type.

Report type

Figure 32-39 Select date range

8. Move down the screen (Figure 32-39). Select an interval for reporting. We chose YTD so that 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 the newly saved view.
13. Click **View** to run the query.

When you have fed your Monitor over a period of time, you see a report similar to that shown in Figure 32-40 on page 865. This particular report is a snapshot of our testing the emitter over a few days.

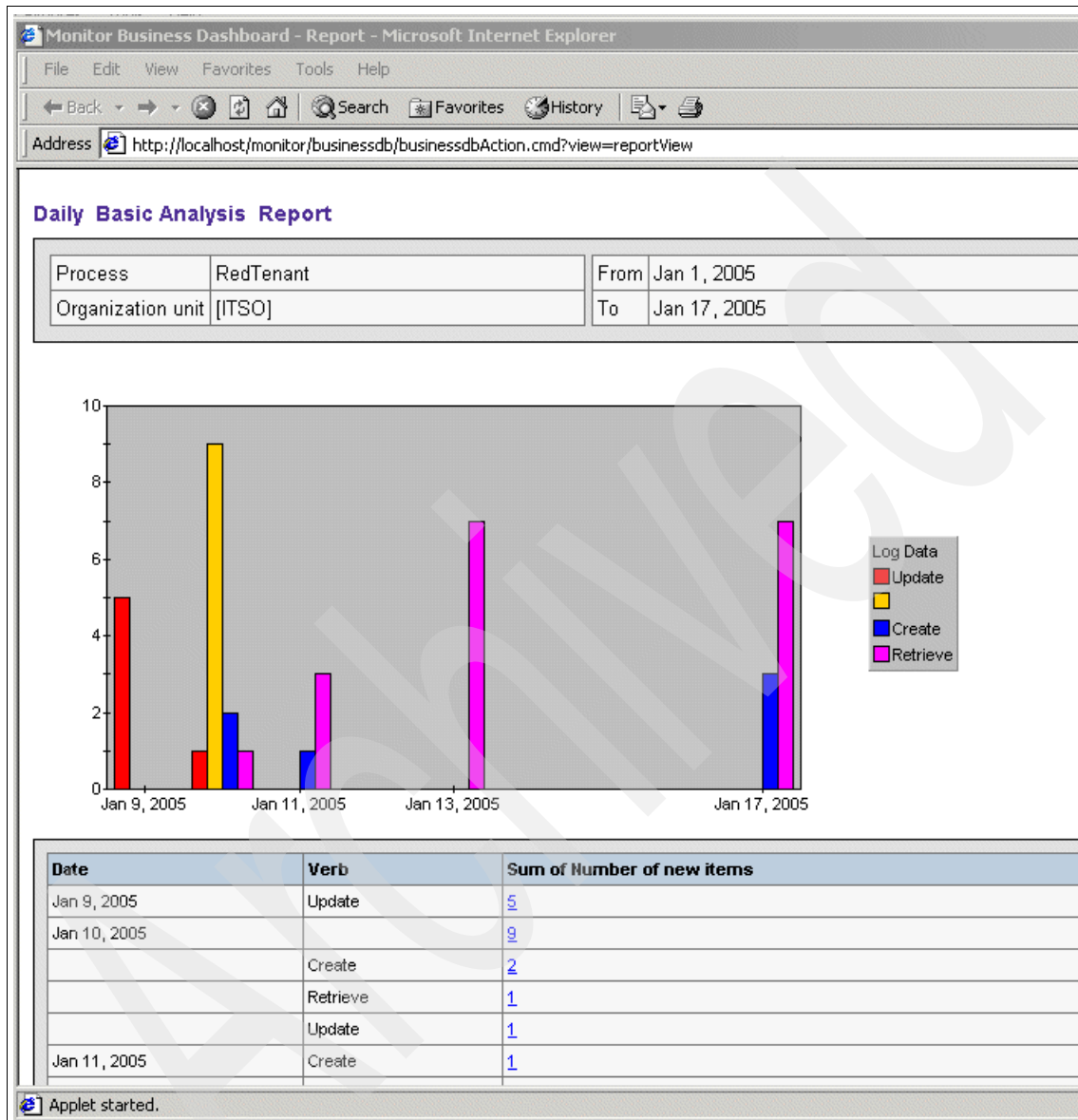


Figure 32-40 Daily Basic Analysis Report

As before, you can drill down for a more detailed analysis (Figure 32-41). You can set filters or export the data as a CSV file to be used in reporting packages, spreadsheets and so forth.































Items <input type="text" value="1"/> to <input type="text" value="10"/> of 10 <input type="button" value="Go"/> <input type="button" value="Export Page"/> <input type="button" value="Export All"/> <input type="button" value="Business Measures"/> <input type="button" value="Set Filter"/> <input type="button" value="Back"/>										
Item	Activity Instance	Process Diagram	Status	Starting Time	Completion Time	Working Duration	Elapsed Duration	Is Delayed	T	
1			Completed	Jan 9, 2005 7:01:20 AM	Jan 9, 2005 7:01:20 AM	0 s	---		100	
2			Completed	Jan 9, 2005 7:05:28 AM	Jan 9, 2005 7:06:38 AM	0 s	1 m, 9 s		100	
3			Completed	Jan 9, 2005 7:13:21 AM	Jan 9, 2005 7:14:37 AM	0 s	1 m, 15 s		100	
4			Completed	Jan 10, 2005 1:54:57 PM	Jan 10, 2005 1:56:10 PM	0 s	1 m, 13 s		100	
5			Completed	Jan 10, 2005 6:59:30 PM	Jan 10, 2005 6:59:31 PM	0 s	1 s		100	
6			Completed	Jan 10, 2005 7:00:19 PM	Jan 10, 2005 7:00:23 PM	1 s	3 s		100	
7			Completed	Jan 10, 2005 7:04:11 PM	Jan 10, 2005 7:04:13 PM	0 s	1 s		100	
8			Completed	Jan 11, 2005 4:15:02 PM	Jan 11, 2005 4:15:23 PM	10 s	20 s		100	
9			Completed	Jan 11, 2005 4:16:19 PM	Jan 11, 2005 4:16:21 PM	1 s	2 s		100	
10			Completed	Jan 11, 2005 4:18:32 PM	Jan 11, 2005 4:18:41 PM	0 s	8 s		100	

Figure 32-41 Report details

This is not a particularly elegant use of the Monitor. 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. However, 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.

This appendix contains samples of the traces that you should expect to see from your unit testing for the message flows.

Example: A-1 *InputMessageFromFrontEnd.txt*

```
(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'0000000000000000000000000000000000000000000000000000000000000000'
  (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'414d512052454442524f4b45522020207f33654120001b01'
(0x03000000):CorrelId          = X'000000000000000000000000000000000000000000000000'
(0x03000000):BackoutCount       = 0
(0x03000000):ReplyToQ          = ' '
(0x03000000):ReplyToQMgr       = 'REDBROKER '
(0x03000000):UserIdentifier     = 'db2admin '
(0x03000000):AccountingToken    = X'16010515000000a837d66532621f2a07e53b2be9030000000000000000000000b'
(0x03000000):ApplIdentityData  = ' '
(0x03000000):PutApplType       = 11
(0x03000000):PutApplName       = 'ereAdapters\jre\bin\java.exe'
(0x03000000):PutDate           = DATE '2004-10-08'
(0x03000000):PutTime           = GMTTIME '14:40:47.290'
(0x03000000):ApplOriginData    = ' '
(0x03000000):GroupId           = X'000000000000000000000000000000000000000000000000'
(0x03000000):MsgSeqNumber      = 1
(0x03000000):Offset            = 0
(0x03000000):MsgFlags          = 0
(0x03000000):OriginalLength    = -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): = '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'
  )
)
)
(0x0100001B):MRM = (
  (0x0300000B):version = '3.0.0'
  (0x0300000B):verb =
'Retrieve'
  (0x0300000B):locale = 'en_US'
  (0x0300000B):delta = FALSE
  (0x0300000B)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 = (
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:Web_tenant_tenant = (
      (0x0300000B):version
= '3.0.0'
      (0x0300000B):verb
= ''
      (0x0300000B):locale
= 'en_US'
      (0x0300000B):delta
= FALSE
      (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:schemaLocation =
'http://www.ibm.com/RedTenant_tenant.xsd'
      (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:name
= '100'
    )
  )
  (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:ObjectEventId =
'JMSConnector_ID:414d512052454454454e414e542020209e33654120000504'
)
)

```

RedTenantDelivery_Retrieve.txt

Example: A-2 RedTenantDelivery_Retrieve.txt

```
**** START OF MESSAGE TREE ****
*** VERB IDENTIFIED AS RETRIEVE ***
(
  (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'0000000000000000000000000000000000000000000000000000000000000000'
    (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'414d512052454442524f4b45522020207f33654120001b01'
    (0x03000000):CorrelId        = X'0000000000000000000000000000000000000000000000000000000000000000'
    (0x03000000):BackoutCount     = 0
    (0x03000000):ReplyToQ        = ' '
    (0x03000000):ReplyToQMgr     = 'REDBROKER '
    (0x03000000):UserIdentifier   = 'db2admin '
    (0x03000000):AccountingToken  = X'16010515000000a837d66532621f2a07e53b2be9030000000000000000000000'
    (0x03000000):ApplIdentityData = ' '
    (0x03000000):PutApplType     = 11
    (0x03000000):PutApplName     = 'ereAdapters\jre\bin\java.exe'
    (0x03000000):PutDate         = DATE '2004-10-08'
    (0x03000000):PutTime         = GMTTIME '14:40:47.290'
    (0x03000000):ApplOriginData  = ' '
    (0x03000000):GroupId         = X'0000000000000000000000000000000000000000000000000000000000000000'
    (0x03000000):MsgSeqNumber    = 1
    (0x03000000):Offset          = 0
    (0x03000000):MsgFlags        = 0
    (0x03000000):OriginalLength  = -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): = '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
  (0x0300000B)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      = (
(0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:Web_tenant_tenant = (

```

```

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

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:schemaLocation =
'http://www.ibm.com/RedTenant_tenant.xsd'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:name
= '100'
    )
)
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:ObjectEventId =
'JMSConnector_ID:414d512052454454454e414e542020209e33654120000504'
    )
)

```

RedMaintRequest.txt

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'0000000000000000000000000000000000000000000000000000000000000000'
    (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'414d512052454442524f4b45522020207f33654120001b01'
(0x03000000):CorrelId          = X'000000000000000000000000000000000000000000000000'
(0x03000000):BackoutCount      = 0
(0x03000000):ReplyToQ          = ' '
(0x03000000):ReplyToQMgr       = 'REDBROKER'
(0x03000000):UserIdentifier     = 'db2admin'
(0x03000000):AccountingToken    = X'16010515000000a837d66532621f2a07e53b2be90300000000000000000000b'
(0x03000000):ApplIdentityData  = ' '
(0x03000000):PutApplType       = 11
(0x03000000):PutApplName       = 'ereAdapters\jre\bin\java.exe'
(0x03000000):PutDate           = DATE '2004-10-08'
(0x03000000):PutTime           = GMTTIME '14:40:47.290'
(0x03000000):ApplOriginData    = ' '
(0x03000000):GroupId           = X'000000000000000000000000000000000000000000000000'
(0x03000000):MsgSeqNumber      = 1
(0x03000000):Offset            = 0
(0x03000000):MsgFlags          = 0
(0x03000000):OriginalLength    = -1
)
(0x01000000):MQRFH2            = (
  (0x03000000):Version          = 2
  (0x03000000):Format           = 'MQSTR'
  (0x03000000):Encoding         = 273
  (0x03000000):CodedCharSetId   = 1208
  (0x03000000):Flags            = 0
  (0x03000000):NameValueCCSID   = 1208
  (0x01000000):mcd              = (
    (0x01000000):Msdl           = (
      (0x02000000): = 'mrm'
    )
    (0x01000000):Set            = (
      (0x02000000): = 'RedTenant'
    )
  )
  (0x01000000):Type             = (
    (0x02000000): = 'RM_Tenant'
  )
  (0x01000000):Fmt              = (
    (0x02000000): = 'CwXML'
  )
)
)
(0x01000000):jms                = (
  (0x01000000):Dst              = (
    (0x02000000): = 'queue:///WEBTENANT.DELIVERY'
  )
  (0x01000000):Tms              = (
    (0x02000000): = '1097246447297'
  )
  (0x01000000):Dlv              = (
    (0x02000000): = '2'
  )
  (0x01000000):Rto              = (

```

```

        (0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'
    )
)
(0x01000000):usr          = (
    (0x01000000):WSDLBinding = (
        (0x02000000): = 'Web_tenantAgentDeliveryBinding'
    )
    (0x01000000):WSDLOperation = (
        (0x02000000): = 'Web_tenantRetrieve'
    )
)
)
)
(0x0100001B):MRM          = (
    (0x0300000B):version      = '3.0.0'
    (0x0300000B):delta        = FALSE
    (0x0300000B):verb         = 'Retrieve'
    (0x0300000B):locale       = 'en_US'
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:Id = '100'
)
)
)

```

ResponseFromBackEnd_WhatIsTheVerb.txt

Example: A-4 ResponseFromBackEnd_WhatIsTheVerb.txt

```

*** What is the verb on the the response message? ***
'Retrieve'
'ResponseRetrieve'

```

ResponseMessageFromBackEnd.txt

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'
    )
)

```



```

    )
  )
  (0x01000000):jms      = (
    (0x01000000):Dst = (
      (0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'
    )
    (0x01000000):Rto = (
      (0x02000000): = 'queue:///REDMAINT.DELIVERYQUEUE'
    )
    (0x01000000):Tms = (
      (0x02000000): = '1097254333719'
    )
    (0x01000000):Pri = (
      (0x02000000): = '0'
    )
    (0x01000000):Dlv = (
      (0x02000000): = '2'
    )
  )
  (0x01000000):usr      = (
    (0x01000000):Description = (
      (0x01000000):WSDLBinding = (
        (0x02000000): = 'Web_tenantAgentDeliveryBinding'
      )
      (0x01000000):Status      = (
        (0x02000000): = '1'
      )
      (0x01000000):WSDLOperation = (
        (0x02000000): = 'Web_tenantRetrieve'
      )
      (0x01000000):RequestType = (
        (0x02000000): = 'Response'
      )
    )
  )
  (0x01000021):MRM      = (
    (0x0300000B):version      = '3.0.0'
    (0x0300000B):verb        =
'Retrieve'
    (0x0300000B):locale      = 'en_US'
    (0x0300000B):delta       = FALSE
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:Id = 100
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:Name = 'Lee
Gavin'
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:ApartmentId = 1
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:EMail =
'leegavin@uk.ibm.com'
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Tenant:RM_Apartment = (
      (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/B0Schema/RM_Apartment:RM_Apartment = (
        (0x0300000B):version      =
'3.0.0'
        (0x0300000B):verb        =
'Retrieve'

```

```

        (0x0300000B):locale                                     =
'en_US'
        (0x0300000B):delta                                     =
FALSE
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:Id       =
1
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:ApartmentNumber =
'135'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:AddressLine1 =
'IBM Hursley Park'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:AddressLine2 =
'Winchester'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:AddressLine3 =
'Hampshire'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:AddressLine4 =
'United Kingdom'
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Apartment:PostCode    =
'SO21 2JN'
    )
)
(0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Tenant:RM_Maintenance = (
    (0x0300000B):size                                           = 7
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
        (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:Status
    = 'C'
        (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'
    )
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
        (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
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
= 1
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
= 'A'
        (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 =
'Crack in the front window'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:StatusDescription =
'Glass delivered, size incorrect. Re-ordered.'

(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'
    )
        (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
        (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
= 3
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
= 1
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
= 'A'
        (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 =
'High speed internet not working'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:StatusDescription =
'Cable company contacted, modem to be replaced.'

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

```



```

        (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
= 10
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
= 1
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
= 'p'
        (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 =
'Test for a new maintenance request'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:StatusDescription =
'Part on order'

(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'
    )
        (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
        (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
= 11
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
= 1
        (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
= 'p'
        (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 =
'Test for a new maintenance request'

(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:StatusDescription =
'Lee's Test - Number Only'

(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'

```

```

    )
    (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
      (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
    = 12
      (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
    = 1
      (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
    = 'A'
      (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 =
    'Test for a new maintenance request'

    (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'
  )
  (0x01000013)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:RM_Maintenance = (
    (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
  = 13
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:ApartmentId
  = 1
    (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/RM_Maintenance:Status
  = 'A'
    (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 =
  'Test for a new maintenance request'

  (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'
)
)
)
)
)

```

Example: A-6 *ResponseToBeSentToFrontEndAdapter.txt*

```
(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'414d512052454442524f4b45522020207f33654120002502'
  (0x03000000):ReplyProtocol   = 'MQ'
  (0x03000000):Topic           = NULL
)
(0x01000000):MQMD = (
  (0x03000000):SourceQueue      = 'REDMAINT.DELIVERYQUEUE'
  (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         = 0
  (0x03000000):Persistence     = 1
  (0x03000000):MsgId            = X'414d512052454442524f4b45522020207f33654120001e05'
  (0x03000000):CorrelId         = X'414d512052454442524f4b45522020207f33654120002502'
  (0x03000000):BackoutCount     = 0
  (0x03000000):ReplyToQ        = ' '
  (0x03000000):ReplyToQMgr     = ' '
  (0x03000000):UserIdentifier   = 'db2admin '
  (0x03000000):AccountingToken  = X'16010515000000a837d66532621f2a07e53b2be9030000000000000000000b'
  (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'0000000000000000000000000000000000000000000000000000000000000000'
  (0x03000000):MsgSeqNumber     = 1
  (0x03000000):Offset          = 0
  (0x03000000):MsgFlags        = 0
  (0x03000000):OriginalLength  = -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): = '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 = (
    (0x02000000): = 'Web_tenantAgentDeliveryBinding'
  )
  (0x01000000):Status      = (
    (0x02000000): = '1'
  )
  (0x01000000):WSDLOperation = (
    (0x02000000): = 'Web_tenantRetrieve'
  )
  (0x01000000):RequestType = (
    (0x02000000): = 'Response'
  )
)
)
)

```

```

(0x01000021):MRM          = (
  (0x03000000):version      = '3.0.0'
  (0x03000000):delta        = FALSE
  (0x0300000B):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"'
  (0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant:ROOT      = (
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:Web_tenant_tenant = (
(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:schemaLocation =
'"http://www.ibm.com/RedTenant_tenant.xsd"'
  (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:id
= 100
  (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:name
= 'Lee Gavin'
  (0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:email
= 'leegavin@uk.ibm.com'
  (0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_tenant:apartments
= (
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartments:Web_tenant_apartm
ents = (
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartments:apartment = (
  (0x03000000):size
= '1'
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment:Web_tenant_apartme
nt = (
  (0x03000000)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'
(0x0300000B)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment:postcode      =
'S021 2JN'
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_apartment:maintenances = (
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenances:Web_tenant_maint
enances = (
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenances:maintenance = (
  (0x03000000):size
= '7'
(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:Web_tenant_maint
enance = (

```

```

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:id          =
'1'

(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:Web_tenant_maint
enance = (

(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'

(0x03000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:description =
'Crack in the front window Glass delivered, size incorrect. Re-ordered. Expected completion date: 0'
)

(0x01000000)http://www.ibm.com/websphere/crossworlds/2002/BOSchema/Web_tenant_maintenance:Web_tenant_maint
enance = (

(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:Web_tenant_maint
enance = (

(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:Web_tenant_maint
enance = (

```


Additional material

This appendix describes how to download from the Internet the additional material that is referred to in this book.

Locating the Web material

The Web material that is associated with this redbook 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 **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 redbook includes the following files:

<i>File name</i>	<i>Description</i>
ESQL.zip	Zipped ESQL for Message Flows
ITSO.zip	Zipped org. files
JMSConnector.zip	JMS Connector configuration
Meta Objects.zip	Meta objects for JMS Connector
ODA.zip	Custom ODA for back-end application
queues.txt	List of queues
REDHOLD.zip	Database definitions
RedMaintenance.zip	Front-end schema
RedMaintenanceApplication.zip	Back-end application
RedTenant.ear.zip	Front-end application
RedTenant.zip	Front-end schema
RedTenantTestMessages.zip	Test data
RedTenantTraces.zip	Sample trace files
ReposJarFiles.zip	Repos jar files
rfhutil.exe	rfhutil
RM.zip	RM Adapter
SupportPac™ IB01.zip	Monitor Emitter SupportPac
WBIC and WBISF.zip	Test files for contractor component

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	20 MB minimum
Operating System:	Windows
Processor:	1600 MHz
Memory:	1.5 GB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation named Additional Materials and unzip the contents of the Web material zipped files into 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 redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 889. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *B2B Solutions using WebSphere BI Connect Version 4.2.2*, SG24-6355
- ▶ *EDI Solutions using WebSphere BI Connect Version 4.2.2*, SG24-6355
- ▶ *WebSphere Business Integration for SAP*, SG24-6354
- ▶ *WebSphere Business Integration Message Broker Basics*, SG24-7090
- ▶ *Migration to WebSphere Business Integration Message Broker V5*, SG24-6995

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

Index

A

- access
 - the archive store 108
 - the event store 101
- activities and their icons 701
- adapt process model 810
- adapt the EJB 818
- adapter
 - configure for startup in remote environments 161
- adapter design
 - identifying the application-specific business objects 5
 - identifying the directionality of the connector 5
 - investigating communication across operating systems 7
 - investigating the application data interaction interface 6
 - investigating the event management and notification mechanism 7
 - metadata-driven or not 42
 - understanding the application 5
- Adapter Framework 190
- adapter initialization
 - accessing a subscription manager 117
 - verifying the connection 117
- adapter polling
 - accessing the archive store 108
- adapter shutdown
 - abnormal termination 147
 - normal termination 147
- adapter startup
 - configuring for remote environments 161
- adapter startup in remote environments
 - configure the 161
- add the send service to the process model 771
- agent and a broker
 - setting up the communication between remote 155
- agent connectivity
 - configuration steps for the remote 156
- AgentDelivery
 - bean 792, 818

- EJB service WDSL 794
- agentInit() method 35
- application-specific information (ASI) 263, 350, 390
- archive store
 - accessing 108
 - create 107
- archiveEvent() method 103
- archiving events 113, 131
 - configuring a connector for 108

B

- back-end application 197, 310, 315, 326, 330, 376, 414, 444, 481, 521, 830
 - gui activity window 388
 - required business objects 379
- basic logic for the poll method 112
- BPEL
 - tasks 703
- BPEL editor 699
- branching on the active verb 260
- broker archive (BAR) file 487, 560
- build
 - EJB to process the incoming message 821
 - message-driven bean 821
- build the send service 761
- Business Dashboard 828, 863
- business measure 826, 828
- business object 335, 345, 379, 399, 412, 451, 469, 847
 - accessing to the 262
 - application-specific 14
 - application-specific information 264
 - attribute 351
 - Business Object Designer 346, 381
 - cardinality 16
 - child metaobject 359
 - child object 344, 391, 393
 - child objects 352
 - create new with wizard 382
 - data 197, 310, 522
 - data value 480
 - definition 343, 399, 415, 451–452, 463
 - deploy to Message Broker 452

- design 389
 - extracting attribute values 265
 - field 536
 - flat business object 15
 - format 197, 310, 374, 472
 - generate definitions 387
 - generic business objects 15
 - handling 439
 - hierachical business object 16
 - key attributes 344
 - message object (bomo) 432
 - name 94, 344, 387, 415, 463
 - processing flat 70
 - processing hierarchical 72
 - repository 454
 - request 359
 - root element 350
 - saving attribute values 266
 - schema 463
 - schema namespace information 350
 - schema representations 358
 - setting the verb 135
 - structure 16
 - subscription to 92
 - support 365, 414
 - wizard 382
 - wrapper business objects 18
 - Business Object Designer 346, 381
 - business object handler
 - accessing the attributes 263
 - accessing the business object 262
 - accessing the business object attributes 263
 - attribute application-specific information 263
 - branching on active verb 260
 - configuring a retrieve to ignore missing child objects 64
 - create the class for a customer handler 76
 - determining whether to process an attribute 265
 - extracting attribute application-specific information 264
 - extracting attribute values from a business object 265
 - outcome status for update verb processing 68
 - retrieve child objects 63
 - saving attribute values in a business object 266
 - standard processing for create verb 60
 - verb action 59
 - business object name 94
 - business process 826
 - high level view 828
 - message structures 829
 - potential bottlenecks 826
 - WSIF 808
- ## C
- check
 - connector version 252
 - for subscription to event 121
 - response message received 443
 - child object
 - business object name 344
 - choosing an ODA content protocol 210
 - cleanupResources() method 103
 - communication between remote agent and a broker
 - native WebSphere MQ 156
 - communication between remote agent and broker 155
 - Community Manager Profile
 - redmaint 594
 - Community Participant Profile
 - redcontract 601
 - compile an ODA 221
 - completing the processing of an event 130
 - configuration
 - remote agent connectivity for connector 156
 - configuration properties 330, 356, 397
 - connector-specific 80
 - standard 80
 - configure
 - adapter startup in remote environments 161
 - community manager profile 593
 - community participant profile 601
 - connector for archiving 108
 - resources for WebSphere MQ JMS provider 317
 - retrieve to ignore missing child objects 64
 - routes for mqipt1 159
 - routes for mqipt2 160
 - startup in a unix environment 153
 - startup in the windows environment 152
 - startup in unix environment 153
 - startup in windows environment 152
 - steps for the remote agent connectivity 156
 - configure routes
 - mqipt1
 - 159
 - mqipt2

- 160
 - configuring the community manager's profile 593
 - configuring the community participant's profile 601
 - connector
 - add supported business objects 414
 - configuration for remote agent connectivity 156
 - configure for archiving 108
 - configure for startup in remote environments 161
 - connector archive store 108
 - connector configuration properties
 - standard 80
 - connector configurator 362, 399, 414, 514
 - connector definition 397, 414
 - connector for archiving
 - configuring a 108
 - connector framework
 - sending the business object 125
 - connector startup
 - Business Integration Message Broker 34
 - checking the connector version 36
 - establishing a connection 35
 - recovering in-progress events 36
 - WebSphere Application Server 34
 - WebSphere InterChange Server 34
 - connector startup configuring for a windows environment 152
 - connectors
 - metadata-driven 52
 - partially metadata-driven 53
 - that do not use metadata 55
 - connectors with JMS event stores
 - enabling the feature for 137
 - connectors with non-JMS event stores
 - enabling the feature for 140
 - connector-specific class 408
 - connector-specific configuration properties 80
 - connector-specific property 358
 - container-managed events 137
 - content of an event store 93
 - content verb
 - handling the retrieve by 65
 - correcting some errors 721
 - create
 - archive store 107
 - artifacts and deploy adapter in system manager 670
 - business objects with the ODA 336
 - business process 697
 - class for a custom business object handler 76
 - class for custom business object handler 76
 - community manager 577
 - community participant 582
 - document flow interaction 583
 - event 444
 - gateway 624
 - interaction 591
 - JMS
 - context 623
 - JMS connector 355
 - JMS gateway 624
 - JMS target 626
 - metaobjects 360
 - new business process 703
 - new XML format 587
 - participant connection 605
 - partner link 704
 - RedTenant application server 316
 - startup script for connector 406
 - startup scripts 224
 - target 626
 - targets 574
 - the service WBICReceive 821
 - variable 706
 - create community manager 577
 - create community participant 582
 - create document flow interaction 583
 - create interaction 591
 - create operation
 - implement 267
 - create targets 574
 - create the class for the custom business object handler 76
 - create the JMS target 626
 - create the participant connection 605
 - create verb
 - handling the 60
 - standard processing for 60
 - createHandler() method 43
 - custom adapter 197, 310, 355, 397, 448
 - custom business object handler
 - create the class for the 76
- ## D
- data handler
 - actual configuration information 359
 - connector metaobjects 359

- XML 351
- database table 530, 828
- data-handler metaobject 359
- define
 - b2b capabilities 599, 604
 - connector configuration 399
 - connector configuration properties 401
 - gateway 602
 - gateways 595
 - JNDI resources 760
 - Listener Ports 801
 - Message Listener Service 801
 - message set 452
 - MQ resources for JMS 621
 - resources in the test server 680
- define MQ resources 621
- defining the connector 399
- delete verb
 - handling 69
 - standard processing for a 69
- delete verb processing
 - outcome status for 70
- deleteEvent() method 103
- deliveryqueue 367, 421
- deliverytransport 85
- deploy
 - changes to Unit Test server 819
 - new service 662
 - RedTenant application 323
 - WBICReceiver service 822
- designing application-specific business objects 201
- determining whether to process an attribute 265
- develop
 - business process 708
 - inhousesend service - step 1 634
 - inhousesend service - step 2 642
 - WSDL for staff activity 694
- development environment
 - set up 204
- double click 356, 414, 492, 835
- doVerbFor() method 50, 57, 59, 62, 74
- doVerbForCustom method 76
- doVerbForCustom() method 76
- drivers
 - business 4
 - technical 4
- drop-down box 511
- drop-down list 346, 419
 - new business object 346

- duplicate event elimination 139
- during mapping
 - setting verb 136

E

- editor panes 640
- editor views 640
- effect on event polling 139, 141
- EJB
 - adapt for JMS message 818
 - add Java code 715
 - as an implementation of a service 655
 - as an inbound protocol 766
 - create instance 689
 - Deployment Descriptor 660, 679, 767, 797
 - depayable service 658
 - for JMS send 655
 - from WSDL 787
 - generate 656
 - generate deployment code 793
 - generate from WSDL 655
 - instantiate the proxy 808
 - invoke 667
 - JNDI names 767
 - listener port 801
 - MDB in EJB project 798
 - output 807
 - proxy called by an EJB 784
 - resources 680
 - run on server 689
 - service skeleton 787
 - set JNDI name 656
 - test 801, 803
 - that is called by an MDB 784–785
 - Universal Test Client as client of 663
 - update to call adapter proxy 822
 - WSDL 795
- elements pane 641
- elimination
 - duplicate event 139
- e-mail 99
- enabling JMS 622
- enabling the feature for connectors with JMS event stores 137
- enabling the feature for connectors with non-JMS event stores 140
- end module 478, 537, 560, 850
- environment

- configure startup in unix 153
 - configure startup in windows 152
 - set up development 204
 - set up test 217
- environments
 - configure the adapter startup in remote 161
- esql 469, 519, 560, 844
- esql editor 511
 - eclipse toolkit 511
- establishing a connection to the application 245
- event
 - archiving the 131
- event distribution 115
- event elimination
 - duplicate 139
- event identifier 94
- event inbox 97
- event object 105
 - the 105
- event polling
 - accessing the archive store 108
 - basic logic for the poll method 112
 - effect on 139, 141
 - event distribution 115
 - standard behavior 110
- event priority 114
- event processing
 - accessing the archive store 108
 - accessing the event store 101
 - archiving events 113
 - check event message sent 445
 - handling successful 130
 - handling unsuccessful 131
 - polling for events 112
- event record
 - setting the verb in the 135
- event status 95
- event store
 - accessing the 101
 - content of an 93
 - contents 93
 - implement
 - an 96
- event store object 102
 - the 102
- event store resources
 - releasing 133
- event stores
 - enabling the feature for connectors with JMS
 - 137
 - enabling the feature for connectors with non-JMS 140
- event table 98
- event verb 95
- events
 - archiving 113
 - checking for subscription 121
 - completing the processing of an event 130
 - container managed 137
 - getting the business object name
 - verb and key 120
 - processing events by event priority 114
 - retrieve application data 123
 - retrieve event records 118
 - with subscriptions 123
 - without subscriptions 123
- examining the application API 199
- examining the application environment 198
- executeCollaboration() method 143
- explore the WSDL editor 639
- exporting the ODA 221
- extending the ODA base class 206
- extracting attribute application-specific information 264
- extracting attribute values from a business object 265
- extracting business object application-specific information 263

F

- feature for connectors with JMS event stores
 - enabling the 137
- feature for connectors with non-JMS event stores
 - enabling the 140
- fetchEvents() method 102, 119
- file
 - flat 101
- flat business objects
 - processing 70
- flat file 101
- front-end application 197, 309, 311, 327, 335, 457, 492, 519
 - browser window 492
 - home screen 555
 - new maintenance orders 519
 - tenant ID 327
- function main 536, 560, 847

G

- gateway
 - parameters for JMS 624
- generate
 - business object definitions 212
 - EJB that is called by an MDB 785
 - MDB 794
 - proxy for adapter 822
 - proxy for process 808
 - source nodes 211
- generate deploy code for the JMS service 655
- generateBinFiles() method 31
- generateBoDefs() method 29
- getAgentProperties() method 28
- getAllAgentProperties() method 28
- getAllConfigProperties() method 83
- getAllConnectorAgentProperties() method 81
- getBO() method 45, 102, 124
- getBOName() method 45–46
- getBusObjName() method 105
- getByteArrayFromBO() method 45
- getCardinality() method 83
- getChildPropsWithPrefix() method 84
- getChildPropValue() method 84
- getConfigProp() method 81
- getConfigProperty() method 35
- getConnectorBOHandlerForBO() method 37–38, 51, 57
- getConnectorID() method 106
- getEncryptionFlag() method 83
- getEventID() method 105
- getEventStore() method 104–105, 118
- getEventTimeStamp() method 106
- getHierarchicalConfigProp() method 83
- getHierChildProp() method 84
- getHierChildProps() method 84
- getIDValues() method 106
- getKeyDelimiter() method 106
- getMetaData() method 28
- getName() method 83
- getNextEvent() method 102, 119
- getObjectCount() method 75
- getPriority() method 106
- getPropType() method 83
- getStatus() method 106
- getStreamFromBO() method 45
- getStringFromBO() method 45
- getStringValues() method 84
- getTerminate() method 103

- getTreeNodes() method 29
- getVerb() method 59, 105, 126
- getVersion() method 36
- global element 460
- gotApplEvent() method 116, 126–127
- gui log 332, 446
- gui utility 459

H

- handler
 - create the class for the custom business object 76
- handling delete verb 69
- handling successful event processing 130
 - events
 - handling successful event processing 130
- handling the create verb 60
- handling the delete verb 69
- handling the retrieve by content verb 65
- handling the retrieve verb 62
- handling the update verb 66
- handling unsuccessful event processing 131
 - events
 - handling unsuccessful processing 131
- hasChildren() method 84
- hasValue() method 84
- hierachical business object 16
- hierachical properties 82
- hierarchical business objects
 - processing 72
- HTML 826
- http/https 158
- hubadmin profile 573
- HubOneWay
 - service bean 715

I

- IBM JDK 1.3.1 172
- IBM WebSphere
 - Application Server 315
 - Business Integration system 397
- ICS broker 404
- identical value 847
- identifier
 - event 94
- ignore missing child objects
 - configuring a retrieve to 64
- implement

- our create operation 267
 - our retrieve operation 269
- an event store 96
- create operation 267
- methods of the CWConnectorEventStore 279
- our verb operation 267
- the create operation 267
- the doVerbForCustom() method 76
- the retrieve operation 269
- import deployed connector in service project 675
- import schema definitions and create message definitions 454
- import statement 460
- imported files 641
- inbox
 - event 97
- information
 - extracting attribute application-specific 264
 - extracting business object application-specific 263
- init() method 28
- initAndValidateAttributes() method 60
- initializing ODA metadata 208
- initializing the configuration-property array 207
- initializing the ODA start 209
- integer 1 478, 560
- integration broker 356, 398, 414, 455, 826
- integration development 414
 - early stages 414
- InterChange Server
 - communicate 399
 - repository 399
- Interchange Server 164
- invoking the connector 409
- isMultipleCard() method 74
- isObjectType() method 74
- isSubscribed() method 116, 121

J

JMS

- change context 623
- configure JNDI 622
- create
 - context 623
- create gateway 624
- create target 626
- using JMSAdmin 623

- validate configuration 628
- JMS (all broker types) 86
- JMS connector 327, 335, 469, 552
 - event queue 327
 - request queue 496
- JMS event stores
 - enabling the feature for connectors with 137
- JMSConnector 355, 484, 554
- JNDI name 318, 365

L

- Label node 523, 559

M

- maintenance record 376, 394, 444, 492, 521
 - select one 376
- maintenance request 196, 308, 315, 438, 517, 521, 879
 - ID 437
 - Part 884
- managed events
 - container 137
- manager or queue (MQ) 413
- mapping
 - setting verb during 136
- mb 165
- MDB
 - arrival of message 785
 - call EJB 784
 - generate 794
 - generate EJB to call 785
 - generate service proxy 808
 - JNDI names 797
 - listener port 799, 801
 - startup, console message 804
 - test 803
 - WSDL 794
 - WSDL binding 806
- Message Broker 335, 393, 399, 414, 451, 557, 826, 829
 - message aggregation function 534
 - message flows 448
 - process data 836
 - version 5 486
- message definition file 460
- message flow 469, 519, 521, 557, 828, 835, 867
- message set 358, 415, 452, 471, 480, 535, 844
 - CwXML wire format 463

- for business objects 452
- message tree 471
 - for adapter messages 471, 867
- metadata
 - connectors that do not use 55
- metadata-driven connectors 52
- method 29
 - agentInit() 35, 39
 - archiveEvent() 103
 - cleanupResources() 103
 - createHandler() 43
 - DataHandler() 44
 - deleteEvent() 103
 - doVerbFor 76
 - doVerbFor() 50, 57, 59–60, 62, 74
 - doVerbForCustom() 76
 - executeCollaboration 143
 - fetchEvents() 102
 - fetchEvents() method 119
 - generateBinFiles() 31
 - generateBoDefs() 29
 - getAgentProperties() 28
 - getAllAgentProperties() 28
 - getAllConfigProperties() 83
 - getAllConnectorAgentProperties() 81
 - getBO 124
 - getBO() 45, 102
 - getBOName() 45–46
 - getBusObjName() 105
 - getByteArrayFromBO() 45
 - getCardinality() 83
 - getChildPropsWithPrefix() 84
 - getChildPropValue() 84
 - getConfigProp() 81
 - getConfigProperty() 35
 - getConnectorBOHandlerForBO() 37–39, 51, 57
 - getConnectorID() 106
 - getEncryptionFlag() 83
 - getEventID() 105
 - getEventStore() 104–105, 118
 - getEventTimeStamp() 106
 - getHierarchicalConfigProp() 83
 - getHierChildProp() 84
 - getHierChildProps() 84
 - getIDValues() 106
 - getKeyDelimiter() 106
 - getMetaData() 28
 - getName() 83
 - getNextEvent() 102
 - getNextEvent() method 119
 - getObjectCount() 75
 - getPriority() 106
 - getPropType() 83
 - getStatus() 106
 - getStreamFromBO() 45
 - getStringFromBO() 45
 - getStringValues() 84
 - getTerminate() 103
 - getTreeNodes() 29
 - getVerb() 59, 105, 126
 - getVersion() 36, 39
 - gotAppEvent() 116, 126–127
 - hasChildren() 84
 - hasValue() 84
 - init() 28
 - isMultipleCard() 74
 - isObjectType() 74
 - isSubscribed() 116, 121
 - pollForEvents() 104, 109, 112, 114–115, 128
 - recoverInProgressEvents() 36, 102
 - resubmitArchivedEvents() 102
 - RetrieveByContent() 65
 - setEncryptionFlag() 83
 - setEventStatus() 102
 - setEventsToProcess() 102
 - setTerminate() 103
 - setVerb() 126
 - terminate() 39
 - updateEventStatus() 102, 119, 122
- mime type 359
- missing child objects
 - configuring a retrieve to ignore 64
- modifying business objects for use 343
- moving into the connector directory 408
- mqinput 474, 541, 854
- mqoutput node 483, 546
- mqrfh2.fiel d 482
- multithreaded connectors
 - connectors
 - multithreaded 114

N

- name
 - business object 94
- namespace declaration 479
- naming conventions for the ODA 206
- naming the ODA 206

- native WebSphere MQ 156
- non-JMS event stores
 - enabling the feature for connectors with 140
- notifications 828

O

- object
 - accessing to the business 262
 - extracting attribute values from a business 265
 - saving attribute values in a business 266
 - setting the verb in the business 135
 - subscription to business 92
 - the event 105
 - the event store 102
- object application-specific information
 - extracting business 263
- object handler
 - create the class for the custom business 76
- object key 95
- object key for connector retrieve 95
- object name
 - business 94
- ObjectEventId 532
 - duplicate events 141
 - generation of 94
 - in request message flow 480
 - request message flow 532
 - response message flow 503
 - support for duplicate events 141
 - update processing 743
 - what is 94, 125
- objects
 - configuring a retrieve to ignore missing child 64
 - processing flat business 70
 - processing hierarchical business 72
 - retrieve child 63
- obtain the active verb 257
- obtain the handle to the ODKUtility object 207
- ODA 335, 379
 - compiling 221
 - completing the business objects 232, 389
 - configuration properties 27
 - generating content 29
 - naming the 206
 - process flows 26
 - saving content 31
 - selecting and confirming source data 29
- ODA setting 353, 385

- open profile of redcontract 601
- open profile of redmaint 594
- operation
 - implement
 - our verb 267
- operation implement
 - our retrieve 269
 - our create 267
- org file 832
- outcome status
 - Create verb processing 61
 - Delete verb processing 70
 - Retrieve verb processing 64
 - RetrieveByContent processing 66
 - Update verb processing 68
- outcome status for delete verb processing 70
- outcome status for retrieve verb processing 64
- outcome status for RetrieveByContent processing 66
- outcome status for update verb processing 68

P

- parameters 144
- partially metadata-driven connectors 53
- participant connection
 - update gateway 625, 627
- perform the verb action 59
- pollForEvents() method 104, 109, 112, 114–115, 128
- polling
 - effect on event 139, 141
- prepare test data for processing
 - (rm_maintenance.create) 436
- prepare test data for request processing
 - (rm_tenant.retrieve) 419
- preparing the connector's directory 405
- preparing the environment 316
- process an attribute
 - determining whether to 265
- Processing 114
- processing
 - outcome status for delete verb 70
 - outcome status for retrieve verb 64
 - outcome status for RetrieveByContent 66
- processing events by event priority 114
- processing flat business objects 70
- processing for a delete verb

- standard 69
- processing for an update verb
 - standard 67
- processing for update verb
 - standard 62
- processing hierarchical business objects 72
- properties
 - connector-specific configuration 80
 - ODA configuration 27
 - of a JMS gateway 624
 - of a JMS target 626
 - standard connector configuration 80
- providing access to generated business object definitions 215

Q

- queue connection
 - factory 317, 362
 - factory name 318
- queue manager 312, 357, 413
 - actual name 413
 - name 413
 - redbroker 372, 422
 - redtenant 362, 374, 512

R

- record
 - setting the verb in the event 135
- recoverInProgressEvents() method 36, 102
- Redbooks Web site 889
 - Contact us xx
- RedMaint
 - request queue 428
 - response queue 433
- RedMaint.delivery queue 403
- RedMaint.synchronousresponse queue 536
- RedMaintenance
 - Integration Component Library 381, 457
- RedTenant
 - application 308
 - create the application server 316
 - installing the application 315
 - unit test with application 367
- redtenant application 315
- releasing event store resources 133
- remote adapter agent and a broker
 - setting up the communication 155
- remote agent and a broker

- setting up the communication between 155
- remote agent connectivity
 - configuration steps for the 156
- remote environments
 - configure the adapter startup in 161
- repositorydirectory 85
- resources
 - releasing event store 133
- response flow 519, 557, 561, 844–845
- response message 311, 403, 432, 491, 548, 561, 874
 - test connector log 438
- resubmitArchivedEvents() method 102
- retrieve
 - application data 123
 - by content 65
 - child objects 63
 - child properties 84
 - connector configuration properties 241
 - connector-property value 84
 - event records 118
 - ignore missing child objects 64
 - implement 269
 - string values 84
 - top-level connector-property object 83
 - verb 62
 - verb processing 64
- retrieve application data 123
- retrieve child objects 63
- retrieve event records 118
- retrieve operation
 - implement 269
- RetrieveByContent 66
- RetrieveByContent() method 65
- RFH2 header 365, 374, 543–544
- RM_Maintenance.Create 437, 443
- RM_Tenant business object 394, 430
- RM_Tenant.Retrieve 428, 430, 440
- RM2Connector
 - adapter service 634, 670
 - bindings WSDL 676
 - create 671
 - deploy to WebSphere Integration Broker 674
 - import to service project 675
 - import WSDL 675
 - properties 671
 - service WSDL 677
 - supported business objects 672
 - WSDL 676–677, 714, 787

- RMConnector 412, 483, 539
 - configuration file 404
 - naming conventions 398
 - queues 402
 - shortcut 410
 - startup script 407
 - supported business objects 414
 - test for startup 412
 - test interaction with business objects 414
 - troubleshooting startup 412
 - use Visual Test Connector 416
- RouteToLabel 474

S

- saving attribute values in a business object 266
- schema file
 - WebSphere Business Integration Modeler 829
- schema location 349, 460, 503, 551
- scripts
 - create startup 224
- seconds 460
- seconds number 460
- select file 345, 382, 452, 471
- select import
 - source 832
 - type 831
- send request business object
 - RM_Maintenance.Create 443
 - RM_Tenant.Retrieve 440
- send response to request
 - RM_Tenant.Retrieve 430
- send test data
 - RM_Maintenance.Create 437
 - RM_Tenant.Retrieve 428
- set up development environment 204
- set up test environment 217
- setEncryptionFlag() method 83
- setEventStatus() method 102
- setEventsToProcess() method 102
- setTerminate() method 103
- setting the verb
 - during mapping 136
 - in the business object 135
 - in the event record 135
- setting the verb in the business object 135
- setting the verb in the event record 135
- setting up the communication between remote agent and a broker 155

- setVerb() method 126
- single-valued simple properties 81
- stages of development 193
- standard connector configuration properties 80
- standard processing
 - create verb 60
 - delete verb 69
 - retrieve verb 62
 - update verb 62, 67
- start connector 366
- start the test connector 416
- startup in remote environments
 - configure the adapter 161
- startup in unix environment
 - configure 153
- startup in windows environment
 - configure 152
- startup script for our connector 407
- startup script on windows systems 407
- startup scripts
 - create 224
- status
 - event 95
- status for delete verb processing
 - outcome 70
- status for retrieve verb processing
 - outcome 64
- status for RetrieveByContent processing
 - outcome 66
- status for update verb processing
 - outcome 68
- steps for the remote agent connectivity
 - configuration 156
- store
 - content of an event 93
- stores
 - enabling the feature for connectors with JMS event 137
 - enabling the feature for connectors with non-JMS event 140
- subflow 855
- subscription to business object 92
- System Manager 336, 381, 400, 414, 451, 514
 - add supported business objects 414

T

- target
 - properties for JMS 626

- tasks in the WSDL editor 641
- tenant ID 197, 309, 312, 421, 492, 521
- terminate() method 39
- terminating the adapter agent 252
- test
 - adapter service using test client 689
 - extended business process 778
 - MDB and EJB 803
 - proxy and adapted process model 812
- test environment
 - set up 217
- the event object 105
- the event store object 102
- top-level object 340
- top-level value 480
- troubleshooting start up errors 412

U

- unit test 367, 412, 539
 - flow 492
- unix environment
 - configure startup in 153
- unsuccessful event processing
 - handling 131
- up the communication between remote agent and a broker
 - setting 155
- update verb
 - handling the 66
 - standard processing for 62
 - standard processing for an 67
- update verb processing
 - outcome status for 68
- updateEventStatus() method 102, 119, 122
- updating the hubadmin profile 573
- using JMSAdmin 623

V

- validate
 - JMS configuration 628
- validation of the JMS and MQ configuration 628
- values from a business object
 - extracting attribute 265
- values in a business object
 - saving attribute 266
- verb
 - behavior in business object handler 59
 - branching on active verb 260

- event 95
 - handling create 60
 - handling delete 69
 - handling retrieve 62
 - handling retrieve by content 65
 - handling update 66
 - obtain the active 257
 - standard processing for create 60
 - standard processing for delete 69
 - standard processing for update 62, 67
- verb action
 - perform the 59
- verb during mapping
 - setting 136
- verb in the business object
 - setting the 135
- verb in the event record
 - setting the 135
- verb operation
 - implement 267
 - our 267
- verb processing
 - outcome status for delete 70
 - outcome status for retrieve 64
 - outcome status for update 68
- verifying the connection with the application 117

W

- Web_tenant
 - XMLDeclaration 869
- WebSphere Application Server 167
- WebSphere Business Integration
 - adapters product 397
 - Message Broker 470, 520
 - Modeler 829–831
 - Monitor 557, 829
- WebSphere Business Integration Adapter development kit v2.4 184
- WebSphere Business Integration Adapter framework v2.4 173
- WebSphere Business Integration Connect
 - create
 - JMS gateway 624
 - target 626
 - update profile 611, 613
 - update the hubadmin profile 573
- WebSphere Business Integration Message Broker

165, 828
 See also Message Broker
WebSphere Business Integration Monitor
 administration utility 828
WebSphere InterChange Server 164
WebSphere MQ 156, 315, 362, 413
 JMS
 configure JNDI 622
 JMS provider 315
 JMS provider entry 320
 native 156
 object 399
 object name 402
 queue destinations table will 323
 queue manager 318
 reason code 413
 using JMSAdmin 623
WebSphere MQ JMS provider 316
windows environment
 configure startup in 152
Workflow Dashboard 827, 857
WSIF
 long-running business process 808

X

XML data handler 351
XML declaration 351, 453
XML version 350, 480, 552
xsd 346, 415, 460, 481, 553, 829, 869



Redbooks

WebSphere Business Integration Adapters: An Adapter Development and WebSphere Business Integration Solution

(1.5" spine)

1.5" <-> 1.998"

789 <-> 1051 pages



Redbooks

WebSphere Business Integration Adapters: An Adapter Development and WebSphere Business Integration Solution

**Incorporate
business-to-business
exchanges using
WebSphere BI Connect**

**Use adapters with
Message Broker and
Server Foundation**

**Develop and deploy a
custom adapter**

WebSphere® Business Integration is the IBM® business integration solution for process integration, workforce management, and enterprise application connectivity. WebSphere Business Integration helps you to create and deploy new business processes, synchronize business information in multiple business applications on diverse platforms, and transform message formats en-route between applications.

This IBM Redbook takes you through the full life cycle of an adapter development project, from design considerations, building, and testing through deployment and implementation on multiple broker types (using both an out-of-the-box technology adapter and the custom adapter for our development project).

For this redbook, we designed a scenario that mirrors many of the issues that real-life integration projects can face. The scenario starts by integrating custom enterprise applications. It then integrates those applications into the business-to-business world by extending the infrastructure. Finally, it adds a human interaction component which determines whether to take the internal route or external route (via trading partners) to application integration. Using many of the components within the WebSphere Business Integration family of products, this book includes a range of integration options that are available to implement this scenario.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks